GRUMPS Summer Anthology, 2001

Malcolm Atkinson[1], Margaret Brown[2], Julie Cargill[1], Murray Crease[1], Steve Draper[2], Huw Evans[1], Philip Gray[1], Christopher Mitchell[1], Martin Ritchie[1], Richard Thomas[3]

[1]Department of Computing Science
University of Glasgow
Glasgow G12 8RZ
U.K.

[2]Department of Psychology
University of Glasgow
Glasgow G12 8RZ
U.K.

[3]Department of Computer Science and Software Engineering
The University of Western Australia
Mounts Bay Road
CRAWLEY 6009
Australia

http://grumps.dcs.gla.ac.uk

Preface

by Steve Draper

This is the first collection of papers from GRUMPS [http://grumps.dcs.gla.ac.uk]. The project only started up in February 2001, and this collection (frozen at 1 Sept 2001) shows that it got off to a productive start. Versions of some of these papers have been submitted to conferences and workshops: the website will have more information on publication status and history.

GRUMPS decided to begin with a first study, partly to help the team coalesce. This involved installing two pieces of software in a first year computing science lab: one (the "UAR") to record a large volume of student actions at a low level with a view to mining them later, another (the "LSS") directly designed to assist tutor-student interaction. Some of the papers derive from that, although more are planned. Results from this first study can be found on the website. The project also has a link to UWA in Perth, Western Australia, where related software has already been developed and used as described in one of the papers. Another project strand concerns using handsets in lecture theatres to support interactivity there, as two other papers describe. As yet unrepresented in this collection, GRUMPS will also be entering the bioinformatics application area.

The GRUMPS project operates on several levels. It is based in the field of Distributed Information Management (DIM), expecting to cover both mobile and static nodes, synchronous and detached clients, high and low volume data sources. The specific focus of the project (see the original proposal on the web site) is to address records of computational activity (where any such pre-existing usage might have extra record collection installed) and data experimentation, where the questions to be asked of the data emerge concurrently with data collection which will therefore be dynamically modifiable: a requirement that further pushes on the space of DIM. The level above concerns building and making usable tools for asking questions of the data, or rather of the activities that generate the data. Above that again is the application domain level: what the original

computational activities serve, education and bioinformatics being two identified cases. The GRUMPS team is therefore multidisciplinary, from DIM architecture researchers to educational evaluators. The mix of papers reflects this.

Table of Contents

Description of GRUMPS Software

Prepared by Julie Cargill

13 April 2001

Introduction

This document describes the two software systems developed by the Grumps team, namely the User Action Collector and the Lab Support Software. These systems have been developed and implemented as separate entities and as such are delivered independently. This document will also provide details of the steps taken by the grumps team to minimise any risk involved in carrying out an experiment using the grumps software, and will outline the procedures in place to manage the experiment.

The User Action Collector

This section describes the technical details of the two programs built by Huw Evans for the initial Grumps experiment based in the level 1 lab of the Computing Science Department at Glasgow University.

The two pieces of software are Huw's Action Collector (HAC) and the Action Concentrator (ACon). The rest of this section is organised as follows: firstly, there is a description of how the HAC and ACon fit into the overall Grumps architecture, then there is a description of the HAC and the ACon programs, which contain a description of their respective lifecycles.

HAC and ACon's Place in the World

The HAC is the program that runs on each student machine in the level 1 lab. It is a C program that makes use of the Microsoft Hooks API defined for Windows 2000 and NT

operating systems (http://msdn.microsoft.com/library/psdk/winbase/hooks_9rg3.htm). This API allows a programmer to register C functions which will be called by the operating system when pertinent events happen on the machine. Such events are split into a number of kinds, and those that HAC is interested in are: keyboard events, mouse events and window events.

When HAC is started, it contacts the ACon process via a C-level socket to register its existence. As the user uses the machine, these three kinds of events are collected and stored locally, in memory, in a buffer. When the buffer fills up, information on these events is sent down the socket to the ACon process. The buffer is then filled up with more events and this cycle of emptying the buffer, sending the data across the network and filling the buffer up again continues, until the program is terminated by the user or it is automatically terminated by the ACon process.

The Action Concentrator (ACon) is a server process, running on `kona` which is written in Java. Its job is to collect the event information sent by each HAC process, and send it on to the TLC Database. When a HAC process contacts ACon it is registered in a list of known processes. The ACon also opens a socket to the HAC process so that the HAC process can be remotely controlled from the single ACon instance.

When the ACon process is started, it displays a graphical user interface (GUI) that lets an administrator do two things: terminate all the currently executing HAC processes; and see the information coming from each HAC as a single line of text per event.

Thus, viewed from the bottom up, the HAC process is at the bottom as one instance of it is running on each machine in the level 1 lab. Conceptually above this is the single ACon process, which each HAC connects to, in order to send the user's events. Above the ACon process is the TLC Server and database which receives and stores event data from the ACon.

The HAC

The HAC program consists of a DLL and a single application, currently called `hooks32` for historic reasons. When `hooks32` is started, a window is displayed which is automatically minimized, to get it out of the way of the user. They can maximize it in the normal way. The GUI consists of four options: `All Data`; `No Keys`; `Change Focus`; and `Off`. Each of these is described next and is then followed by a description of the HAC lifecycle and measures that have been taken to make HAC as resilient as possible.

All Data

If `All Data` is selected, the key the user has pressed, the window X/Y coordinate of where they have clicked their mouse and the identity of their window on a window focus change is logged. If this option is on, it is possible for the software to see everything the user has typed, it is possible to know which window it was typed in and where in that window it was typed.

Hide Keys

This option is the same as `All Data` except that the information on the exact kind of key typed by the user is obfuscated. The software only knows the kind of key, not the exact information. This hides what has been typed, but collects information on whether it was an alpha key, a numeric key, an extended key (e.g., a punctuation character), or if it was the delete key. Mouse click and window focus changes are logged in the same way as for `All Data`.

Focus Change

This option only logs information on the name of the window that has been currently clicked in to gain the focus. No keyboard information nor any mouse information is logged.

Collection is Off

If this option is selected no information is collected, although the program is still running. This gives the user the opportunity to temporarily switch off event logging if they want to. The user can then switch it back on again by selecting any one of the other three kinds.

HAC Process Lifecycle

After the user logs in, the HAC program is started automatically as it is registered in each user's start up list. Thus, HAC is not running when the user enters their Windows 2000 password. The program minimizes itself automatically and selects `Hide Keys` logging. The part of the program that performs the logging is written as a DLL and so is lazily mapped into the address space of each process that the user uses. It should be stressed that there is only one copy of the DLL in memory and that the operating system ensures that it is mapped into the host address space correctly. The buffer that each event is written into is currently 139,288 bytes. This buffer is written into by the mapped DLL from each application and the `hooks32` application periodically consumes the information. HAC is essentially a multiple producer, one consumer process.

As the user is using the machine HAC is collecting information on the events that the user has given permission to collect, via the GUI. This information is periodically sent to ACon. This collection proceeds until either the user logs out, the user terminates the HAC, or an error occurs somewhere in the HAC and/or ACon architecture.

Logging Out

If the user logs out, the HAC program is terminated in the same way by the operating system as for any other Windows program and this instance is removed from the list of registered processes at the ACon.

User Termination of the HAC

The user can explicitly terminate the HAC by clicking on the X control in the top right part of the window. When they do this, the currently buffered set of actions is sent to

ACon and the program terminates. This causes the operating system to remove each mapped instance of the DLL and to remove the `hooks32` application. From this point on it is not possible for the user to launch another version of the HAC unless they log out and log in again and then the lifecycle proceeds as above.

Error Handling

Errors are inevitable in any distributed system and the HAC and ACon have been designed to impact each level 1 machine as little as possible. For example, if for any reason the HAC cannot contact ACon, the HAC terminates itself. For example, if ACon were to crash when running on kona, the next time each HAC tried to send its set of events, the socket connection would be invalid, this case would be detected and the HAC would gracefully exit. For diagnostic purposes, when the HAC terminates a file is written to, called `c:\temp\huwdebug.txt` with one line describing why the termination was necessary. This file will be, at most, 100 bytes long.

It is also possible to terminate all the HAC instances in one go by clicking a button on the GUI displayed by the ACon server.

HAC Resilience

As the HAC is collecting potentially sensitive information entered by the student, a number of measures have been taken to ensure that the HAC process cannot be interfered with by a rogue third party process.

When HAC starts up, it contacts the ACon process which is running on kona, on a well known port. Both the hostname and the port value are hardwired in the source code and the source code is stored under Huw Evans's Unix account with read and write file permissions for the owner only. Thus, only someone who knows the superuser password or can log in as Huw can get access to this information.

When the HAC contacts ACon on the well known port it has to send an identifying string that ACon checks to ensure it is a valid version of HAC contacting it. If this check

succeeds, ACon tells the HAC instance the name of the machine and the port number to which the events should be sent. Currently this is kona on another port. This redirection facility is there to ensure load balancing can be performed in the future, if we decide we need more than one ACon in a network working concurrently. If this check fails, ACon just closes the socket.

When the HAC contacts ACon on the port to which the events will be sent, HAC sends a particular string. This is checked by ACon and if it is successful, ACon send back a particular string to HAC which it checks. In this way, both ACon and HAC know they are communicating with an instance of another and not some rogue third party. If either of these checks fails, the side that has detected a failure immediately closes the socket.

Once the HAC knows it is communicating with a valid version of the ACon, HAC opens a server socket itself, on a well-known port. This is used by ACon when it wants to shut the HAC down. In order to do this, the port number the HAC server socket is listening to and the termination string needs to be known. Without this information, the HAC process cannot be terminated as it ignores all strings other than the termination string.

At this stage in the communication lifecycle the HAC and ACon processes mutually trust the other. The event data that is sent from the HAC to ACon is not encrypted as this represents a significant processing overhead in terms of setting up encryption keys and performing the encryption and decryption on either side of the communication.

Thus, as the source code for this system is as securely held as it can be, the only option open to someone would be to run a packet sniffer on the network to determine the ports that are used by the programs and the strings that are exchanged. They could then write a program that could shut down a HAC remotely or they could impersonate a HAC. Shutting down a HAC just means no events are sent to the ACon from that machine; it does not interfere with the user of that machine. Impersonating a HAC would allow someone to generate events that the HAC would log, assuming they had the event string layout correct, which presumably they do as they have been sniffing packets on the

network. If they get the string layout incorrect, ACon just ignores the event. However, it is possible for someone to generate events that look as if someone else generated them.

This kind of attack is obviously undesirable, but given the precautions outlined about it would take a large amount of work to determine everything necessary to do this. It is believed that the amount of work required is sufficiently high to act as a deterrent in addition to other Departmental and University wide rules on abusing computing resources.

The ACon

As described above, the ACon process manages the N connected HAC processes and collates this input and sends it upstream to the TLC database. The current design of ACon is a multiplexer. The N HAC connections are multiplexed down a single connection to the TLC database. The HAC events are read in and placed into a buffer. When this buffer fills up, the events are sent to the TLC database and more events from the HAC processes are placed into the buffer. Once the event has been sent to the TLC database, this is the end of the involvement for the HAC and ACon processes.

The ACon to TLC communication is Java to Java and is performed by writing Java serialized objects down a socket connection obtained from a URL connected to a servlet running on the TLC side. Currently, each time ACon needs to empty its buffer, a new connection is opened to the servlet. Eliminating the need to open a new connection each time is an area for future work.

As the data from ACon to the TLC database is written using Java object-serialization, the event text is not easily read. In order to eavesdrop on this data, someone would have to know the layout of a Java serialized object and be able to impersonate the TLC servlet running on venus. This would require them to install a process on venus that understood the URL and servlet protocols and TLC's protocol as well. This requires a high level of sophistication and, as with the HAC, we believe the amount of effort required is an adequate deterrent so that no one will bother to do it.

The ACon Process Lifecycle

The ACon process is started on kona before any of the HAC processes are started. The ACon process then receives all the HAC events and passes them onto the TLC database as outlined above. Thus the ACon process needs to be left running on kona for as long as there is a HAC process so that no events are lost. If at any stage the ACon is terminated, each connected HAC will automatically terminate the next time their buffer fills up and they try to communicate with the ACon. Once the experiment has been finished, someone will terminate the ACon and any outstanding HAC processes by using the ACon user interface.

Tutor-Student Help System

Architecture

The tutor support system comprises of two components: the client which runs on the student machines and the backend database. These components are managed using ColdFusion scripts. The client is a web page running on Internet Explorer. This web page is refreshed every 30 seconds to ensure that the student is presented with the most recent information. The web pages contain forms which allow the students to pass information to the database which also force a refresh of the pages. The tutors will have access to information about their group's students using similar web pages running on Internet Explorer on hand-held devices.

Functionality

The functionality of the system can be broken down into two areas: student functionality and tutor functionality.

Student Functionality

1. Overview of their lab group.
   - This view is only available during their lab session.
   - Can be presented textually (as a list) or graphically (as a map of the lab).

- Provides information about all the students in the lab group
    - Whether or not they are using the system.
    - When they were last seen.
    - Whether or not they have asked for help.
    - How long they have been waiting for help.
    - Where they are sitting.

2. Students can request help from their tutor/demonstrator.
    - Students can only lodge one request for help at anytime.
    - Requests for help only persist for the duration of their login.
    - Requests can be cancelled.
3. Students can record personal memos
    - Students can lodge multiple personal memos which are not visible to other students.
    - Memos persist between sessions.
    - Memos can be deleted at any time by the student.

Tutor Functionality

The tutor functionality is similar to the student functionality. This section highlights the differences between the student and tutor functionality.

1. Overview of the lab group
    - The tutors can determine which cluster colour(s) they view.
    - The tutors can modify the way the lab map is displayed (alignment and rendering).
    - The lab map indicates whether there are other students sitting in their group's cluster during their lab time.
    - The group list allows the tutor to indicate that they have seen a student (independently of whether the student asked for help or not). A student can only see information his or her self. This allows, for example, a demonstrator who does not have a hand held to indicate that they have seen a student.
2. Tutors cannot request help or log memos.
3. Tutors can view information about individual students.
    - A history of help requests made.
    - Number of times the student has been seen.

Risk Minimisation Preparation

This section describes the technical approaches we have taken to minimise any risk involved when using the software in the lab. This has consisted of extensive testing and code to handle shutdown and selective startup. These are considered in order, followed by information on additional testing that has not yet been performed, but will be before the software is deployed.

Extensive Testing

The software has been extensively tested in F091 and tested in the level 1 lab by the System's team and by members of the Grumps team.

Shutdown

When the action concentrator is run on kona, it displays a GUI for use by an administrator. This gui displays in textual form the events that are being generated by each machine. It also contains a single button which when pressed, cleanly terminates each action collector running on the Windows machines in the level 1 lab. This enables us to remove the process from every machine remotely, without disturbing any of the students.

If the server should crash, thus removing the GUI from the screen, the next time an action collector on the student's machine tries to communicate logged events, it will discover that the server is no longer available. This was described in detail above.

Selective Startup

As described above, an action collector will terminate itself if the server is not running. Thus, if the server is not running and the student logs in, the action collector will be started from the start-up list, it will discover there is not server to talk to and it will terminate itself in the same way above. In this way, the software will not disturb the student when they log in if the server has been taken down for any reason.

Errors within Action Collector

If an error is detected within the action collector, e.g., if function calls fail inside the program, the action collector terminates itself in a clean fashion, as described above. This is in addition to the termination that will occur if communication with the server is not possible. This makes the action collector as unobtrusive as possible.

Help System

The risks presented by the system have been minimised in the following ways:

- *username/password authorisation is required for both students and tutors.*
- *Uses standard Internet technology (`htaccess`)*
- *Because the IP number of the machine is required to determine the user's location in the lab the system cannot be used outwith the level 1 lab.*
- *The client is a standard web page thus provides no additional risk to the client machine beyond that which exists when a browser is used.*
- *A second ColdFusion server has been ordered to avoid overloading the Department's web server.*

Prepared by Julie Cargill, julie@dcs.gla.ac.uk
13 April 2001

The GRUMPS Architecture: Run-time Evolution in a Large Scale Distributed System

Huw Evans , Peter Dickman and Malcolm Atkinson
Department of Computing Science, Glasgow University
17 Lilybank Gardens, Hillhead, Glasgow, Scotland, UK, G12 8RZ

Abstract

This paper describes the first version of the distributed programming architecture for the Grumps[1] project. The architecture consists of objects that communicate in terms of both asynchronous and synchronous events. A novel three-level extensible naming scheme is discussed that allows Grumps developers to deploy systems that can refer to entities not identified at the time when the Grumps system and application-level code were implemented. Examples detailing how the topology of a Grumps system may be changed at run-time and how new object implementations may be distributed during system execution are given. The separation of policy from mechanism is shown to be a major part of how system evolution is supported and this is made even more flexible when expressed through the use of Java interfaces for crucial core concepts.

[1] Grumps is an acronym that stands for Generic Remote Usage Monitoring Production System.

1 Introduction

This paper describes the first version of the run-time architecture for the Grumps research project [4]. The Grumps research project is developing techniques and software to automatically collect, manage and analyse large collections of user actions. The project has four interrelated goals: to make it easier to organise experiments that efficiently and as unobtrusively as possible collect and store traces of remote users' actions; to make it easier to analyse the stored data to test ideas about the users' activities and the facilities provided to support them; to discover whether such an approach is effective in improving the quality of distributed information systems; and to trial these issues in specific application areas, such as education and bioinformatics.

The rest of this paper is organised as follows. Section 2 states the main system requirements that have driven the initial design; section 3 describes the major components of a Grumps system and gives its architecture; section 4 describes the three-level naming scheme; section 5 discusses, with examples, how a Grumps system is evolved at run-time; section 6 details some related work; section 7 briefly describes future work and section 8 concludes the paper.

2 System Requirements

The main requirements for the Grumps architecture are: the system should be able to operate across the Internet which requires support for communication through firewalls and via proxies; the topology should be changeable at run-time; and the operations performed by the deployed objects should also be subject to change at run-time. Support for run-time evolution in Grumps is provided in the two main areas of the naming scheme and in the approach to the design of object deployed into the system. The deployed objects, called Grumps Units (GUs), are containers for the objects that perform application-level processing over the events that flow around the system. Grumps allows these contained objects and the GUs to be evolved by sending a GU a control event. The

control event contains code to inspect the GU and call its public methods to alter its behaviour and the behaviour of the objects it contains.

A Grumps system consists of a graph of network-deployed GU objects. A GU object has a location in the network of machines. Into this GU object are placed the objects that perform some part of the application semantics. These contained objects are connected together at run-time by sending connection information to a GU. During system execution, its objects move through a number of well-defined object lifecycle states. For example, the contained objects recreated, deployed into a GU, connected to other objects, have their implementations changed during computation, are removed from the deployed system and are then finally archived or destroyed. In order to be able to support such a system requires a naming scheme that can refer to such entities (in terms of their location and functionality) and a GU architecture that supports evolution as one of the most common operations performed in the system. The common operation is to evolve the objects that a GU contains, although the architecture also supports the ability to add and remove GU objects and also to evolve a GU's implementation.

The Grumps architecture builds on the ideas contained in the DRASTIC project [2, 3].

3 Major Components

3.1 Input and Output Channels

In Grumps, events are communicated via input and output channels. An input channel is defined to be the receiving end of a communication line, which can be read to receive events. An output channel is the sending side of the communication line. Events that are written at the sending side are read at the receiving side and one output channel is connected to exactly one input channel. The sending side and the receiving side can be in the same process or in different processes, on different machines. The phrase ``input channel'' refers to the receiving end (server-side) of a communication line and ``output

channel" refers to the sending side (client-side) of a communication line. The phrase ``event channel" refers to a whole channel, and does not distinguish between either end.

3.2 Grumps Units and Grumps Events

The major component in the Grumps architecture is the Grumps Unit (GU) (figure 1). A Grumps Unit is an object that encapsulates a number of event processing objects. Each event processing object is referred to by a number of input channels and the processing object may refer to a number of output channels. Each GU object can be communicated with via its single control event channel. Input and output channels communicate in terms of Grumps Events (GEs) which are sent asynchronously. The control channel is used to change the GU at run-time. Control events are sent to the control channel synchronously[2]
.

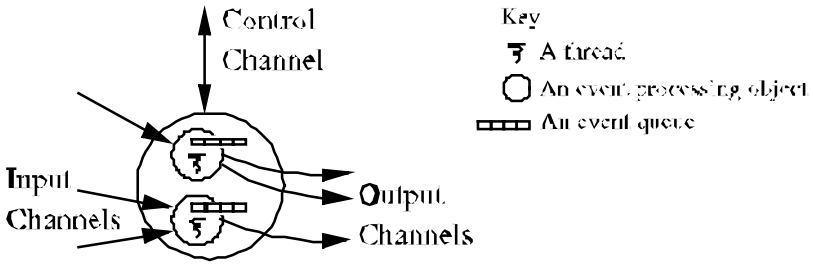

Figure 1: An Implementation View of a Grumps Unit

A Grumps event carries with it information on when the event occurred, which GU originally sent it and some event specific information. Each event arriving on an input channel is placed into a FIFO queue, one queue per channel. This queue is managed by an object that is responsible for reading the objects from the queue, processing them in some way and (possibly) sending them out on an output channel. GUs can be combined into graphs of GUs and it is the intention of the Grumps project to be able to treat a particular graph of GUs as a single GU. This will allow users of the system to reuse components, building sophisticated experiments from a collection of GUs with well-known behaviour.

---

[2] Communication is synchronous to the control channel as it is generally useful to be able to get a result object back as a result of sending in a control event.

A number of specialised GUs have been identified. Some of these are: autonomous logic, which takes events and emits control events; operational store, which makes events persistent; and compress, which takes a single event or a stream of events and compresses it or them into another form. By combining these GUs into graphs it is hoped that an experimenter can quickly and easily build experiments.

3.3 Experiments and Environments

One intention of the Grumps architecture is to be able to make it easy to organise experiments. A Grumps experiment consists of a number of computers that are connected via a network. Each machine runs a number of environments and within an environment there are a number of GUs. A Grumps experiment and its environments are named entities in Grumps, e.g., the environments are named operating system processes. An example of this part of the architecture is given in figure 2.

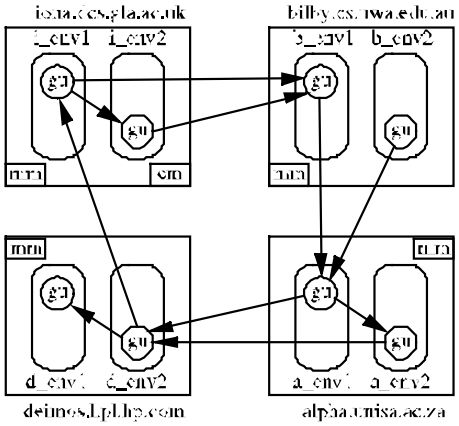

Figure 2: An Example Grumps Distributed Architecture

Here we have four machines: `iona.dcs.gla.ac.uk` is in Scotland; `bilby.cs.uwa.edu.au` is in Australia; `alpha.unisa.ac.za` is in South Africa and `deimos.hpl.hp.com` in on the west coast of America. Each machine is hosting two environments, `i_env1` and `i_env2` in the case of `iona.dcs.gla.ac.uk`. To keep the diagram simple, each environment has a single GU (the control channel is not shown) which has a number of input and output channels.

`iona.dcs.gla.ac.uk` supports another kind of GU, called em on the diagram, which is the `ExperimentMonitor`. This GU is responsible for providing facilities that are common to the whole experiment. This includes serving Java class files and providing a name service for the objects involved in the experiment[3]. When the `ExperimentMonitor` is executed it is given an experiment name, which is a simple string allocated by the administrator of the system. Each environment that is started with that experiment name is considered to be a part of that experiment.

On each machine a GU, called mm on the diagram, is running. This is the `MachineMonitor` and it is responsible for controlling the environments on that machine, e.g., such as starting a new one. mm is running inside an environment called `MachineMonitor` which has been started within the context of an experiment.

4 Naming in Grumps

In setting up a graph as described in figure 2 the software needs to be able to refer to the entities in the system. For example, the GU that is running in `i_env1` must be able to refer to the GU that is running in `b_env1` in order to establish a remote reference to it. This is accomplished with the novel Grumps physical naming scheme.

In Grumps, the experimenter needs to be able to name many different kinds of entity, for example, GUs, environments, machines, files, data collection devices and so on. An extensible physical naming scheme is defined that allows the experimenter to name such entities and be able to create new naming schemes to refer to entities that are yet to be identified.

[3] Given that this experiment involves computation on four continents the `ExperimentMonitor` facilities will have to be federated at each site. This is an area for future work.

4.1 Extensible Physical Naming Scheme

The general form of the Grumps extensible physical naming scheme consists of four parts: a naming scheme identifier; a naming scheme interpreter; naming scheme information; and freetext to allow for expansion within the naming scheme. The naming scheme identifier tells the naming-system what kind of name this is, e.g., whether it refers to a GU or a file. The naming scheme interpreter is information on which language-level code should be used at run-time to interpret this naming scheme, e.g., how to get hold of the machine name if the naming scheme is for a GU name. Next is the information peculiar to the naming scheme which consists of information to uniquely identify the entity in question. The freetext is an open-ended string that can be used to add extra information to the name that is not directly catered for in the naming scheme with a given interpreter.

The naming scheme is a physical one because information is built into the names that describes the physical location of the entities that they refer to. For example, for a GU's control channel, the physical information is the name of the machine and the port number on which it is listening.

4.1.1 Referring to an Event Channel

The particular physical naming scheme described here as an example is used by the run-time system to refer to the event channels that are contained within a GU. An event channel has an output side and an input side. These two sides may be in the same environment, they may be in different environments on the same machine or they may be in different environments on different machines. In addition, the input side may be within the same administrative boundary, or it may be in a completely different domain, behind a firewall or only reachable after contacting a proxy first.

To be able to refer uniquely to an event channel at run-time, the Grumps communication system needs to know two pieces of information: the name of the machine and the integer value of the port on which the EC is listening for incoming events. Three other pieces of

information are added to this; the experiment and environment names[4] are added for human users, and a protocol identifier is added to distinguish the kind of inter-GU communication being performed. These five pieces of information constitute the naming scheme information for an event channel which is written thus:

/FW/MyExperiment/(130.209.240.35:20000):NameServer/

Figure 3: EC Naming Scheme Information

This name describes the receiving end-point of an event channel (which may or may not be active) on the machine whose IP address is `130.209.240.35` (iona), listening at port 20000 in the environment called `NameServer`, which is part of the `MyExperiment` experiment. The `FW` tag says that in order to talk to this GU, the communication from the sender needs to pass through firewalls. The above name is prefixed with a Grumps extensible name of the following form:

grumps::ec.name.physical:(java:ECPhysicalNameParser)/

Figure 4: EC Extensible Physical Naming Scheme Prefix

`grumps::ec.name.physical` is the naming scheme identifier for the physical event channel. grumps:: indicates that this is a Grumps name and `ec.name.physical` says that the rest of this string is a physical event channel name. The string `(java:ECPhysicalNameParser)` indicates that the whole physical name was generated by an instance of the Java class `ECPhysicalNameParser` and that to interpret the name an instance of that class should be used.

[4] Future versions of Grumps will consider whether a GU and its contained objects can participate in more than one experiment at a time. If this is supported, the design of this naming scheme will have to be revised.

The extensible physical name scheme allows Grumps programmers to provide naming schemes that are tailored to the specifics of the entity being referred to and to refer to object-kinds that are yet to be identified.

4.2 Logical Names

The above physical names have one serious drawback. If the GU with the event channel that is running on iona is stopped and, after a time, restarted, it may not be able to be use port 20000. Another EC may have been started while our GU was down and this other EC could have been allocated port 20000. If this is the case, our GU could run its EC on another port. However, the name given in figure 3 is now useless as it refers to the wrong event channel. Grumps defines a logical naming scheme to abstract over the information at the physical level.

A logical name is a pair consisting of the experiment name and a 1024 bit value. Associated with a logical name is at most one physical name. The `ExperimentMonitor` (em) in figure 2 contains a name server which can allocate a logical name given a physical name. Once allocated, the logical name is considered to exist forever, i.e., it cannot be reused. The logical name is always associated with the service provided by the entity referred to with the physical name. This service would be one of the fundamental kinds of GU identified in section 3.2 or it may be one of the `MachineMonitors` or it may be a file or a machine. When a programmer has a reference to a particular logical name they need to be confident it refers to the service they think it does, and not some other (potentially completely different) service. For this reason, the logical name cannot be reused. Therefore, it is safe for it to become part of a GU's persistent state, should that be required. This use of logical names in Grumps is akin to the use of `ServiceIds` in Jini [1].

When a logical name contains an event channel physical name (for which there is an input channel), the logical name can be considered to be the output channel. In practice

the logical name is presented to the communication layer, the physical name is extracted and a socket[5] connection is opened to the input channel described in the physical name.

4.2.1 Using Logical Names

There are three main operations performed using logical names: associating a physical name with a new logical name; using a logical name to gain access to the physical entity; and re-associating an existing logical name with a new physical name. To discuss these cases within the context of a physical event channel name, assume that the event channel is referred to with the EC naming scheme given in figure 3 and that it is associated with the following logical name object:

(MyExperiment:1330515575...)/(130.209.240.35:20000)/

Figure 5: Using a Logical Name with a Physical Name

where `(MyExperiment:1330515575...)` is the logical name component and `(130.209.240.35:20000)` is the (abbreviated) physical component. To allocate a new logical name, the following occurs. When the GU on `iona` was started, its EC was allocated the port number 20000 by Java. The em name service was then contacted to allocate a logical name and associate the physical name with it. A copy of the logical name/physical name association was kept at the name server and a copy was passed back to the object that represents the event channel that is running on `iona` which is listening on port 20000 for incoming events.

This kind of logical name is then used to communicate with an event channel. Assume the logical name has been handed out to another GU, which is running somewhere else in the network. To open a connection to the EC on `iona`, the logical name is presented to the Grumps communication facilities. The physical name is extracted and is used to establish a socket connection to `iona` on port 20000. The logical name is passed down

[5] Grumps is using sockets for event channel communication as they are independent of any other transport protocol, e.g., RMI or Jini, and they provide a convenient building block onto which future communication technology can be built, such as that required to traverse firewalls and proxies.

the socket, together with the event being transmitted. In the EC object on `iona` the logical name it has been passed is compared with its own. If they are equal[6] the communication can proceed, otherwise the socket is closed and no more communication takes place. This check ensures the client knows it is talking to the EC for which the logical name was allocated, and not a different channel which just happens to be listening on the same port.

We now assume that the GU and its EC are stopped and restarted on a different machine, in a different environment, and the EC is listening on a different port. When the GU is started in the new environment, the EC object must first of all contact the em name server and re-associate its new physical name with the logical name stored in the name server. However, the GU running on the other machine has a copy of the logical name with the old EC physical name in it. When it tries to open a communication socket to the old address, it will either be explicitly rejected as described above or it will fail as no EC is listening on that socket. This client-side error is presented to the programmer as an exception and the name server is contacted, passing in the logical name. The experiment name and the 1024 bit quantity inside the submitted logical name are used to lookup the previously registered version of the logical name, which now contains the new physical name. The result of this lookup is passed back to the client. This logical name is then presented to the communication layer and a successful connection, via the new physical EC name, is made to the remote EC running on the new machine.

4.3 Finding Logical Names

Given the description of the naming mechanism above, the only way to gain access to a logical name was to have been passed a copy of one from another piece of code. As the logical names are logical, they are meaningless to a programmer and so are not suitable to be used as keys to the name service's logical name lookup service. It does not make sense to allow logical names to be looked up on the basis of their currently defined physical

[6] Two logical names are equal if the experiment name Java strings are equal and the two 1024 quantities are equal.

name as the logical name is meant as an abstraction over the details of the physical location of an entity. Therefore, the Grumps system defines another way of looking up logical names based on `Attribute` objects.

An `Attribute`[7] object is an object that provides human interpretable information to be associated with a particular logical name. An `Attribute` object may be something like a simple name, such as `Configuration Information` or `Data Collection Device`. When a logical name with its associated physical name has been allocated by the name server, it can have the logical name exposed via a set of `Attribute` objects. For example, a logical name could be exposed via the single named attribute object called `Data Collection Device`. Some other piece of code on the network can then perform a lookup in the name service, passing in, via an array, an attribute named `Data Collection Device`. The result of such a lookup is an array of logical name objects that match this array of attributes.

5 Evolution

This section describes how a GU and the objects it contains may be evolved at run-time. This is explained by considering how a new input channel may be created inside a GU (by sending in a new event processing object) and then how this event processor may be replaced with a new one. During system execution, the GU may be receiving control events while also handling a number of concurrently executing event processing objects. During this activity, code in another environment can send into the GU a control event. This control event can affect the operation or state of the GU itself, or by calling methods on the GU's interface, can gain access to one of the event processing objects and alter its implementation or data.

[7] The approach to using `Attribute` objects in Grumps borrows a lot from both the ideas of using attributes in Jini [1] and the general idea of a trader as defined by ANSA [5] and is therefore not described in great detail.

5.1 The GU Control Interface

This section describes the control interface to a GU in more detail to show how the topology of a GU graph and the semantics of a GU can be updated at run-time. A GU has a single control event channel and makes available a single public method for incoming control events:

```
void receiveCtrl(ControlEvent e) throws GrumpsException;
```
Figure 6: GU method for Receiving Control Events

`ControlEvent` is an interface and instances of classes that implement this interface are passed to the GU along its control channel[8]. A `ControlEvent` object defines two public methods:

```
void apply(GU gu);
void setSocket(Socket socket);
```
Figure 7: ControlEvent Interface

The `setSocket` method is used by the receiving-side of the communication facilities, so that the received `ControlEvent` object can send information back to the client-side that originally sent the `ControlEvent`. The apply method is the cornerstone for supporting the ability to change a GU at run-time. When the GU receives a `ControlEvent` instance (e in figure 8), it is passed to the GU's `receiveCtrl` method and the GU executes the apply method[9]

---

[8] The control channel has been registered in the em name service using a set of attribute objects that uniquely identify it. This is currently implemented using an instance of the class `ControlChannelAttribute` that indicates the kind of channel this is and several named attributes that describe the experiment, environment and machine name and port number on which the control channel is listening.

[9] Although security is not a primary goal in Grumps, some checks can be performed by the GU on reception of a control event. For example, `receiveCtrl` could reject all control events that were not

```
e.apply(this);
```

Figure 8: Applying the ControlEvent to the GU

Thus, the reference to the GU object is passed to the apply method that is defined on the `ControlEvent` object. Whatever code has been provided in the implementation of the apply method is then run, and together with its reference to the GU, the public methods of the GU may be invoked. The GU defines a number of methods that allow for control of itself directly or give access to the event processing objects that it contains. For example, when replacing one GU with another, the new GU may require access to the old GU specific state and the new GU may want to ensure all the queued events are flushed out. The GU interface is required to expose its own specific state and it provides an API to gain access to the individual event processing objects, which in turn provide an API to flush their queues[10].

In this way we have decoupled the policy for affecting a GU at run-time from the mechanism used to accomplish it. It is impossible to foresee all the operations that could be required to be performed over a GU. Therefore, it does not make sense to put any policy defining operations into the code for a GU. Once the GU had been deployed, it would be very difficult to change that policy. With this mechanism, the policy can be defined in a class that implements the `ControlEvent` interface and we can transfer this policy object to the GU in question. The GU then executes the code in the `apply` method which affects the GU's operation via the public methods it defines.

5.2 Providing a New Input Channel

A GU in Grumps can support a number of input channels. To create a single new input channel in a given GU a `ConnectionRequestControlEvent` is sent to the GU to

---

instances of a particular type. Considering the security issue in terms of cryptographically secure control events or some other mechanism may be an area for future work.

[10] Further developing the GU interface and that of the event processing objects is an area of future work.

which the new input channel should point. This kind of control event defines the apply method given in listing 1 (constructor not shown that assigns to `po` and `exposer`).

```
public class ConnectionRequestControlEvent extends
ControlEventImpl {
  ThreadedEventProcessingObject po;
  Exposable exposer ;
  public void apply (GU gu) {
    if (!( gu instanceof GUContainer))
      return ;
    GUContainer grumps unit = ( GUContainer) gu;
    GUConnection connection = new GUConnection(0, po,
exposer );
    grumps unit . addConnection(connection );
    po. start ();
    connection . start ();
    writeObject ( connection . getLogicalName ());
  }
}
```

Listing 1: Providing a New Input Channel

GU is the interface all GU implementations must implement, `GUContainer` is the top level class that implements the GU interface. We first of all check that the object passed to apply is the kind of object we are after. This is so the assignment to the grumps unit object is successful. The next line then creates an object called `connection`. This is the object that manages the new input channel within this GU. The `po` object defines how incoming events on this channel are processed. The `exposer` object defines how this new event channel will expose itself to the em name server, using a set of attributes, logical name and physical name as described in section 4. The value 0 means the Java system should allocate the port number on which this connection will listen. The new connection is then added to the GU in the `addConnection` call. The next two lines cause the `po` and `connection` objects to start their threads (see below). The last line returns to the client code (that sent the control event) the logical name of the connection object that was just created. This logical name is then used at the client end to communicate with this connection. Once the logical name is returned to the client side, the new connection has been added and events can be sent to it. These events will arrive

in the po object which processes them and (possibly) sends them on an output channel, or to some other destination, such as a disk.

5.2.1 Thread Programming

When a new `connection` is created, two threads are started. The connection thread in listing 1 is listening to the port for incoming connections. When one is received the event is read from the socket and passed to the `po` object. The `po` object places the event into a queue and calls its own `notify`() method. This causes the thread portion of the `po` object to wake up from a previous call to its `wait`() method. The thread then takes the next event out of the queue, processes it according to the code defined in the `po` object[11] and (possibly) sends it out on an output channel. Access to an output channel is achieved in the `po` object by gaining a (number of) logical name objects and opening communications with them, sending the processed event, or possibly a new event to the receiving EC.

As the code for interacting with the GU is held in the control event object, the policy for how to create the thread objects and start them can be changed over time, by providing new classes that implement the `ControlEvent` interface. Thread classes in Grumps implement the interface `GrumpsThread`. This provides additional control over the thread, specifically it provides a method void `exit`() that instructs the thread that it must shut down. The `po` object above overrides this method to provide management of its queue of events. For example, when `exit`() is called on `po` it may want to drain its event queue. The code to do this can be provided in the `exit`() method.

5.3 Updating an Input Channel

The `GUContainer` implementation of the GU interface also defines a method `getConnection`() which takes a single logical name as an argument. This method returns an object of type `GUConnection`, an example of which is the connection object in listing 1 above. This is how the semantics of a GU and its processing of events on a

[11] Processing the event may generate other events, which can be sent to output channels.

particular event channel is updated at run-time. This is achieved by sending an instance of the `UpdateConnectionControlEvent` class to the control channel. The `apply` method on this control event does the following:

```
public class UpdateConnectionControlEvent extends
ControlEventImpl {
  LogicalName logical name;
  ThreadedEventProcessingObject po;
  public void apply(GU gu) {
    if (!( gu instanceof GUContainer))
      return;
    GUContainer grumps unit = ( GUContainer) gu;
    GUConnection connection = grumps unit . getConnection
(logical name );
    connection . setProcessingObject (po );
    po. start ();

    writeObject ( null );
  }
}
```

Listing 2: Updating an Input Channel

This time a reference to the `GUConnection` object contained inside the grumps unit is retrieved using the EC logical name object which is part of the state of this control event. The `GUConnection` object then has its `setProcessingObject` method called, passing a new `po` object in as a parameter. The old `po` object that is inside the connection object has its `exit`() method called and the new `po` is then assigned and its `start`() method called. In this case no information needs to be sent back to the client side as the connection object has not been changed, thus its logical name remains the same.

6 Related Work

There have been many systems that allow the topology of a distributed computation to change at run-time, such as the ongoing work on Darwin [6] at Imperial College, London. However, the Grumps architecture allows the semantics of the system to change, without having to change its topology. The closest related work in terms of architecture is Jini [1].

However, the Jini work supports highly dynamic networks of services. The Grumps work is focussed on building graphs of interconnected components and evolving them at system run-time in an environment where communication through firewalls and via proxies has to be supported.

7 Future Work

The Grumps project is a three year project and the first six months have elapsed. The basic communication and GU evolution infrastructure is complete, including the three-level naming scheme, the name server and the event mechanism used for communication and code replacement. The object lifecycles of the GU, event processing objects and other classes, such as `LogicalName` and `GrumpsThread` need to be considered in terms of the definition of their interfaces, their default implementations and documentation. Other members of the Grumps team will be providing tools to allow an experimenter to define the topology of their experiment, and then deploy, execute, and tear it down again.

8 Conclusions

This paper has described the first version of the run-time distributed programming architecture for the Grumps project. The architecture and its main components have been described as well as its three-level extensible naming scheme. How the system can be evolved, both in terms of its topology and in terms of its semantics have been described. The separation of policy from mechanism is of crucial importance when providing an environment in which evolution can be supported and this is made even more powerful by the provision of a core set of interfaces that capture the main mechanism. A default policy for this core set of interfaces is provided by a set of classes that implement those interfaces. `ControlEvent` objects contain the policy for updating a GU at run-time.

For more information on the Grumps project, see the Grumps website http://grumps.dcs.gla.ac.uk.

9 Acknowledgements

The authors would like to thank the members of the Grumps team for providing the necessary environment in which to perform this work and the EPSRC (GR/N38114) for providing the funding.

References

[1] W. K. Edwards. Core Jini. P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1999.

[2] H. Evans and P. Dickman. DRASTIC: A run-time architecture for evolving, distributed, persistent systems. In M. Aksit and S. Matsuoka, editors, Proceedings of the European Conference on Object-Oriented Programming (ECOOP '97), volume 1241 of LNCS, pages 243--275, Jyv˜askyl˜a, Finland, June 1997. Springer.

[3] H. Evans and P. Dickman. Zones, Contracts and Absorbing Change: An Approach to Software Evolution. In Proceedings of the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '99), volume 34 of SIGPLAN Notices, pages 415--434, Denver, Colorado, USA, Oct. 1999. ACM.

[4] The Grumps Project. http://grumps.dcs.gla.ac.uk/.

[5] A. Herbert. An ANSA overview. IEEE Network, 8(1), Jan./Feb. 1994.

[6] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. Lecture Notes in Computer Science, 989, 1995.

Investigations Into Privacy and Other Aspects of a Real-Time Distributed Marks Collection System

Richard C. Thomas[1], Paul Cashman[1], David Edwin[1] and Martin Ritchie[2]

[1]richard@cs.uwa.edu.au
Department of Computer Science and Software Engineering
The University of Western Australia
Mounts Bay Road
CRAWLEY 6009
Australia

[2] ritchiem@dcs.gla.ac.uk
Department of Computing Science
17 Lilybank Gardens
University of Glasgow
GLASGOW G12 8RZ
Scotland

ABSTRACT

The real-time collection of laboratory marks gives the opportunity to save lecturer time and, as a side effect, possibly improve the early detection of students at risk. However there may be adverse consequences for student and demonstrator privacy. This paper presents a distributed marks-collection system we have built at The University of Western Australia. It captures marks in situ in the laboratory from almost 700 students enrolled over two units. The web-based system, constructed using the Teachers' and Learners' Collaborascope software, has proved highly robust. We also discuss our investigations into privacy concerns.

KEYWORDS

Privacy, student marks, distributed information system, performance prediction, user monitoring, TLC, servlet, non-functional requirements.

1. INTRODUCTION

There is pressure on academics to become as efficient as possible when they teach large classes. It is not uncommon to have 300 students in a unit (lecture course), and then even tasks like the collection of marks can be extremely onerous. The potential and pressures to automate are huge.

One of advantages of automation is that lecturers can have access to better information more quickly. For example in a unit where there are laboratory classes spread over the week, the lecturer might know after the Monday whether that week's assignment is too hard or too easy. Another possibility is that the students most at risk could be detected early and thus given extra support to help them catch up.

However there are also disadvantages in automation. Much collateral information can be gained through the collection of marks in situ, for example which students sit next to each other and whether a demonstrator is biased in whom they mark, e.g. a male marking mainly women. Thus there are issues sometimes falling within the legitimate domain of the lecturer controlling the teaching, while others could impinge on the privacy of the demonstrators and students alike.

Privacy is controversial. We adopt a conservative, research-based view of privacy and how it might affect student perceptions. More radically, Ahituv (2001) argues society is moving rapidly towards a more open information society because the cost of protecting privacy will become too high.

In this paper we first discuss the motivation for our new real-time distributed marks collection system, then review concerns and investigations into its privacy effects. After

an overview of the system architecture our initial findings are presented. We conclude with plans for future developments.

2. MOTIVATIONS

In the Department of Computer Science and Software Engineering at The University of Western Australia the Software Engineering 104 unit has over 320 students. There are weekly labs comprising 5 exercises to be completed and marked by the demonstrator *in situ*. The advantages of this approach are that the students have to show competence to the demonstrator in a series of practical programming exercises. There is rapid feedback and demonstrators can check whether the student appears to understand the material. The drawback is that there are over 1500 pass/fail marks to be collated each week.

Another major motivation was to detect as early as possible those students likely to be at risk from failing the laboratory component and hence the entire unit. It has been shown by Thomas (1998) that there are wide variations in activity to achieve simple milestones in first year lab exercises for computer science students. It is likely that some of this can be attributed to general skill levels at using computers and exploring new interfaces, while the rest is due to specific domain problem solving effort.

We hypothesise that a crude indicator of trouble can be obtained from timing when students complete their exercises and tracking changes in their position in the cohort over the course of the labs. For example students with good problem solving abilities but poor computing skills would likely start off amongst the slowest in the class but progress relatively well during the semester. Real-time collection of marks in the lab would give us the raw data to investigate this.

3. PRIVACY ISSUES

In recent work on systems to support students and demonstrators in the laboratory, Draper (2001) has suggested that the design of privacy safeguards needs to be considered at the outset of the project rather than grafted on at a later stage. In view of this it was

decided to investigate the privacy issues with the lab marking system while it was being designed.

Privacy has been of interest to psychologists, sociologists, philosophers and lawyers. Accordingly there are many definitions, but for the present purposes we draw on that used by Newell (1994). We take privacy to be a state of separation from the public domain, in which the public domain does not intrude. It is a voluntary and temporary separation that can be psychological, physical or informational. People perceive privacy to be invaded when: personal data is disclosed rather than performance data; when there is no permission for the disclosure; and when unfavourable consequences can occur from the disclosure (Tolchinsky *et al* 1981).

We interpret the work of Block and Stokes (1989) to suggest that people fear being observed and evaluated by others and that this fear may be justified depending upon the type of information being disclosed. Privacy can increase job satisfaction (Newell 1994; Block and Stokes 1989; Duvall-Early and Benedict 1992), although both positive and negative effects on job performance have been reported, see also Kirchner (1966). Privacy is also important for creativity, concentration and decision making (Edney and Buda 1976).

The above research highlights the importance of a sensitive approach to privacy issues when monitoring students – as well as in the design of information systems generally. We hypothesize that the following are important concerns for the privacy of student information:

- The degree of control available to people over their disclosed data
- The eventual use of the disclosed data
- Relevance of the data to the application area
- The specific type of data collected (personal versus performance data)
- The disclosure of information to unknown people.

An aim of the current research is to check these findings as far as practical. We are investigating this for students and demonstrators using: a questionnaire; a marks viewing tool for students; and a viewing tool, hopefully, a linked data privacy control tool for demonstrators; and an exit questionnaire. These tools are being monitored down to the mouse click level in some instances. Thus it is hoped, for example, to analyse thinking time for answering each part of the questionnaire, possibly using some of the ideas from the usability study of McElhaw and Hammond (2000).

4. SYSTEM ARCHITECTURE

The system is based on the Teachers' and Learners' Collaborascope (TLC) developed at Glasgow University (Smith 1999).

Students access the TLC via a marks applet, one for each of the Software Engineering and Java units. Another application for changing demonstrator passwords uses JSP rather than an applet for security reasons. These connect to the TLC servlet, which performs common services for the applications, e.g. authenticating users and dispatching SQL queries to the database. With hindsight the marks application should have used JSP as it is more secure. All these applets, queries and JSP have been written for this project. Other software will enable lectures to change marks and obtain summary data.

The questionnaire is delivered via another TLC application called QDT, written in Glasgow by Nightingale (1999). Our version runs as an applet.

The servlet runs under Tomcat on a linux machine using the Apache web server. The servlet spawns a server which handles connections to the database via JDBC. The TLC database runs under Microsoft SQL Server 2000 under Windows 2000.

In order to keep disruptions to the production system to a minimum, we have a parallel development system. It has been helpful but there are complications when cutting over

new functions to the production version, especially in the database. These could be rectified in the future.

5. INITIAL FINDINGS

The marks collection system has been very successful. It is currently used by about 320 students and 25 demonstrators weekly. There has been a high degree of acceptance by students and demonstrators. The latter remarked how much better it was than the manual system in use for another course.

There have been few errors and much less paperwork. Occasionally marks have not been transmitted to the database, but it has been clear when this has occurred and remedial action has been taken. The GUI design was less than perfect due to time and system development stresses, but trials with students suggested that it was adequate for logging on, and support was available for first time users from the demonstrators if required. This judgement has proved sound. This seems to have been an example of “good-enough GUI design”. People have commented favourably on the clean, functional design.

In the weeks following cutover to the first marks client, QDT and other services are being installed. So far QDT has had over 110 hits.

Occasionally there has been a total loss of the marks system from the student perspective. For example once they could not log on and this was traced to a fault from a minor upgrade in the campus-wide student records system, used for authenticating students. The other half dozen or so outages have all been due to development work impinging on the production system. The prognosis for reliability, therefore, is excellent.

We have realized that the TLC database schema could be improved, and the TLC server software could be streamlined for ease of programming and maintenance. We have discovered that these real-time applications require much higher reliability levels than has been the case for departmental systems up to now. Moreover it is clear the non-

functional requirements such as privacy, security and reliability are important drivers in the architectural design process. The pivotal role of non-functional requirements in the acceptability of systems is emphasized by Sommerville (2001).

Finally, the success of the marks applet in the Software Engineering unit has prompted another lecturer to use it in his first year Java 124 labs. Currently about 670 students enrolled in these two units use the system for a total up time of about 44 hours per week.

6. FUTURE PLANS

Substantial data mining will be carried out after semester to test the hypothesis that students at risk can be predicted early from timing data. Hopefully this will give a rational basis with which to decide whether a student should be given extra attention by the demonstrators in the lab.

It is intended to introduce a wireless palm top computer system for the demonstrators, using some of the software from the recent experiment at Glasgow (Crease *et al* 2001). In our case students will ask for help or marking and the palm top will display to the demonstrator a map of outstanding requests. The demonstrator will mark the work and enter the result through the palm top. The student display will be updated to reflect the new mark. Demonstrators will have the requests ordered according to those deemed in need of extra attention, minimizing waiting time and so on.

We also intend to use the Glasgow UAR software (Cargill 2001) for recording actions with the mouse and keyboard to see if basic computer literacy can be determined. Should this be possible, the prediction algorithm could be made much more sophisticated.

In conclusion, perhaps one of the most important insights we have had is that there appear to be plenty of applications waiting for reliable services to transport, store and mine data. The future adoption of user monitoring, together with its associated benefits,

may depend upon offering services and software that can readily meet the needs of application builders.

7. ACKNOWLEDGEMENTS

This work has been partially funded by the UK EPSRC (GR/N 38114). Thanks are also due to Julie Cargill, Murray Crease, other members of the GRUMPS team, Chris McDonald, Ashley Chew and Philip Cummins.

8. REFERENCES

Ahituv, N. (2001) The Open Information Society, *Communications of the ACM*, **44:6**, June, 48-52.

Block, L.K. and Stokes, G.S. (1989). Performance and Satisfaction in Private Versus Non-private Work Settings, *Environment and Behaviour*, **21:3**, 277-297.

Cargill, J. (13 Apr 2001) Description of GRUMPS Software, GRUMPS working document [html], URL http://grumps.dcs.gla.ac.uk/docs/GrumpsSoftware.html 20 August 2001.

Crease, M., Gray, P.G. and Cargill, J. (2001) Using Location Information in an Undergraduate Computing Science Laboratory Support System, submitted to UbiComp 2001 Conference.

Draper, S. (s.draper@psy.gla.ac.uk) (23 Apr 2001). *Ethics*,Email to grumps@dcs.gla.ac.uk

Duval-Early, K. and Benedict, J.O. (1992). The Relationships Between Privacy and Different Components of Job Satisfaction, *Environment and Behaviour*, **24:5**, 670-679.

Edney, J.J. and Buda, M.A. (1976). Distinguishing territoriality and privacy: Two studies, *Human Ecology*, **4,** 283-296.

Kirchner, W. (1966). A Note on the Effect of Privacy in Taking Typing Tests, *Journal of Applied Psychology*, **50:5**, 373-374.

McElhaw, M. and Hammond, J. (2000). Users to Consumers: The Relevance of Human-Computer Interaction to E-commerce. In Cecile Paris *et al* (eds) Interfacing Reality in the New Millennium, Proceedings OzCHI 2000, Sydney 4-8th December, 124-128.

Newell P.B. (1994). A Systems Model of Privacy, *Journal of Environmental Psychology*. **14**, 65-78.

Nightingale, W. (1999). Question Delivery Tool, M.Sc. Dissertation, Computing Science, University of Glasgow. URL http://www.revelation.gla.ac.uk/Projects/TLC/ProjectReports/TLC/Wendy.pdf 20 August 2001.

Sommerville, I. (2001). *Software Engineering*, Addison-Wesley, 6th edition.

Smith, C. (1999). The TLC Server, M.Sc. Dissertation, Computing Science, University of Glasgow. URL http://www.revelation.gla.ac.uk/Projects/TLC/ProjectReports/TLC/Cameron.pdf 20 August 2001.

Thomas, R.C. (1998). *Long Term Human-Computer Interaction – An Exploratory Perspective*, Springer Verlag.

Tolchinsky, P.D., McCuddy, M.K., Adams, J., Ganster, D.C. and Woodman, R.W. (1981). Employee Perceptions of Invasion of Privacy: A Field Simulation Experiment, *Journal of Applied Psycholog*, **66:3,** 308-313.

Using Location Information in an Undergraduate Computing Science Laboratory Support System

Murray Crease, Philip Gray and Julie Cargill
Department of Computing Science,
University of Glasgow,
Glasgow, G12 8QQ, UK
<murray,pdg,julie>@dcs.gla.ac.uk

Location is important to both students and tutors in teaching laboratories – in particular, it influences how they interact during help-giving. We have developed the Lab Support System (LSS) using mobile wireless-enabled computers to support tutor-student interaction in teaching. In this paper we discuss the ways that we represented and used location information in this system and our observations of the role of location during a field trial of the LSS. We also consider how additional location information might be exploited in future system developments.

1 Introduction

Location is important to both students and tutors in university teaching laboratories – in particular, it influences how they interact during help-giving. We have developed a system to support tutor-student interaction in laboratories and, in the course of developing and using the system, we have had to confront a variety of issues related to the role of location and location

information in the activity and in our system design. This paper reports on our experience and some of what we have learned from it.

The Lab Support System (LSS) is a web-based application, deployed on static workstations and wireless-enabled palmtop computers, that supports student-tutor interaction in computer laboratories, particularly the process of students asking for help and tutors delivering it. In developing the LSS our primary focus has been the tutorial process, especially as it exists in teaching labs in Computing Science at our university; issues related to location and mobility were very much secondary and emergent and not an initial central concern. Exploiting location information was not a prime factor in our design nor did we set out to change the locational aspects of the activity (e.g., reduce tutor movement). Nevertheless, our system did take location into account and we found some interesting interactions between location and system use, as will be reported below.

In this paper we describe the LSS, discuss the way it handles location information both in the current system and in possible future versions of the system, and report our observations of the role of location and location information during a field trial of the system. Section 2 provides an overview of the way tutors serviced student help requests prior to the introduction of the LSS. Section 3 gives an overview of the LSS and section 4 discusses the features of the system that relate to location information. Section 5 presents some observations on how location and location information influenced system use during a four week field trial. Finally, in section 6 we offer some conclusions and consider possible location-oriented enhancements of the system.

2 Locational Information in Teaching Labs

The LSS has been developed in the context of the GRUMPS project, investigating support for large-scale distributed experiments based on computer usage data [Atkinson et al, 1999]. The LSS was one part of a larger initial testbed for capturing data generated by student lab performance, both at the keystroke level and at an application level. Our development was

originally focussed on student-tutor interaction in a 1st year teaching lab in the Computing Science Department at the University of Glasgow (GUCSD) [Draper 2001]. Although our longer-term aims are more general, the work reported here refers to the system we developed for this 1st year lab.

The GUCSD 1st year laboratory consists of a single room with 80 student workstations deployed on benches. First year students carry out practical work in tutorial groups of around 20 students and a staff tutor. Groups remain fixed for a semester's work. Each tutorial group attends one two-hour laboratory each week during the year, working on practical exercises. Each tutorial group has assigned to it a cluster of workstations (a set of contiguous workstations in one area of the lab) intended to be used by the group during the lab session. Machines in the lab are divided into four clusters, each with a different colour represented by a coloured sticker on the machine's monitor

In GUCSD 1st year lab sessions, physical location influences help requests and help-giving in several ways. Students are static in the sense that, for most of a lab session, they remain at one workstation, although they may use different workstations during different sessions. Tutors have no fixed location but move freely around their tutorial group as well as occasionally visiting students from other groups and consulting with other tutors and demonstrators.

There is a relationship, albeit fuzzy, between physical location and group membership. Although a student may use different workstations on different occasions, they are supposed to choose from the machines belonging to their group's cluster, although this is not strictly enforced. Also, currently unoccupied machines in a cluster may be used by students not belonging to the group scheduled to use that cluster.

Tutor help is largely demand-driven. Students catch the attention of a tutor (usually the one belonging to their group) and the tutor must work out a strategy for handling the current set of

requests. The help itself, of course, is given when the tutor is near the student (standing behind, looking a the display, or sitting next to them). Occasionally, the tutor might make an announcement to the entire group or call a small group away for a mini-tutorial.

We can consider location and location information from both the students' and tutors' points of view. For a tutor servicing help requests is largely influenced by estimation of need. However, physical location plays a part. For example, some tutors take requests from a nearby student first, if they think it will not be long. Other tutors work their way systematically along a lab bench, using the physical layout to structure the help servicing. In general, tutors find it difficult to walk past a student in need of help to service the request of another student further away.

Finding a student who has requested help in a busy lab is not always easy. As mentioned, presence of a student in the assigned cluster is not a good indicator of group membership. Also, the shape of clusters makes a difference. For example, one cluster in the lab studied is T-shaped. Students at the end and head of the T were rarely seen.

From a student's point of view, getting the attention of a tutor is the most important part of requesting help. This clearly depends on where the tutor is relative to the student and what they are doing. The most popular methods of obtaining help before the introduction of the LSS were hand-raising (36% of students questioned stated that they used this method every lab. 52.8% used it in some labs) and attracting attention when a tutor is passing by (12.4% of students questioned stated that they used this method every lab. 53.9% used it in some labs). On rare occasions, a student would get out of their seat and approach the tutor directly (thus maximising their locational advantage). Some students appear to have consciously chosen to sit out of their cluster, as far away as possible so that they could work unimpeded by unsolicited tutor attention.

3 System Description

The LSS was implemented in Cold Fusion™ using SQL Server™ as the persistent data store. Cold Fusion™ was chosen because it required a minimal specification on the client machines and allowed the system to be run across different platforms. It also allowed the rapid prototyping and development of the system. The system proved to be very reliable; it failed only once, due to a failure of the web server (unrelated to the use of the LSS). Students ran the system on low-specification Windows NT4™ workstations (the standard lab machines they used for their practical work) and the tutors used Compaq iPAQ™ handhelds running WinCE 3.0™. In both cases, the system was run using Internet Explorer™ browsers. In this section the functionality of the two versions of the system – for students and tutors – is described.

The Student System

The primary aim of the student system was to allow students to request help from their tutor with a minimum of disruption. It was also hoped to prompt the students to think about their problem by requiring them to type in some keywords summarising their problem. The LSS also enabled a student to see who else in their group was requiring help and, if the student requiring help wished it, the keywords describing the problem. This would allow students, who perhaps had been stuck with a similar problem, to help each other. A secondary aim of the LSS was to allow students to record personal memos. These memos, which only the student can view, can be used to record anything but were primarily aimed at allowing students to record problems outside supported lab times to act as a reminder to raise the problem at the next scheduled lab. The interface to the student system can be seen in figure 1.

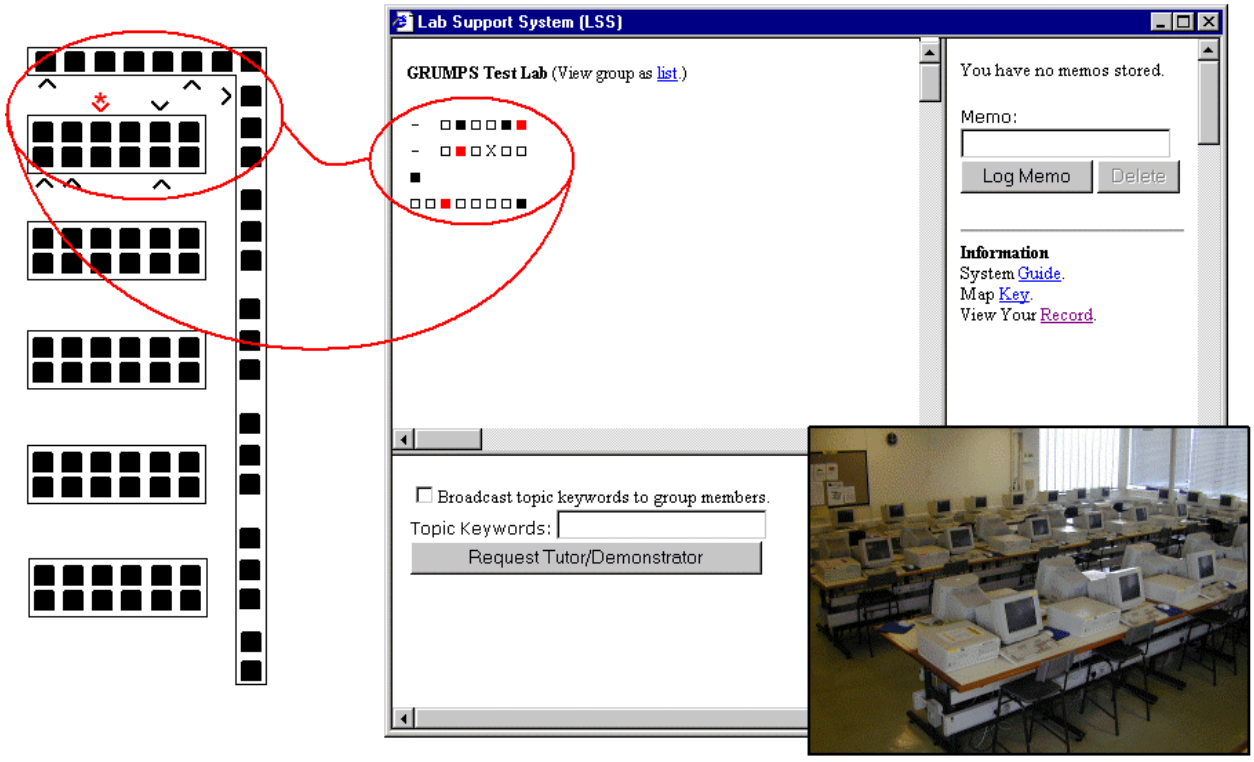

Fig. 1. **Student interface to the LSS (on the right) showing the three main areas of functionality: view group (in this instance as a map); request help (in the lower window); and create/delete memo. Correspondence between map and the actual workstation cluster in the lab is shown to the left. Note that map orientation is rotated 180^{o} from the student interface since the student using this interface would be facing the camera. The student is represented by 'X' on the interface and is marked with '*' on the map. The direction a student is facing in the map is given by the point of the arrow. A photo of the lab is inset at bottom right.**

The top left-hand area of the interface presents the current the state of a lab group. In this case, the group is represented in a map. The student is represented on the map as a black 'X'. Other members of the group who are using the system are represented as a black '■'. Computers not in

the student's colour cluster are represented as '-' and unoccupied[12] computers in the student's cluster are represented as '□'. If a student requests help then their representation on the map turns red. It can be seen, therefore, that in Figure 1 the computer cluster the student is sitting at is approximately rectangular (there are two computers from an adjacent cluster at the left represented by '-') and there are eight students logged in including the student whose interface is shown. Three students are asking for help. They are shown in red on the interface and are located two computers to the left of the student, at the end of the row facing the student and to the left and behind the student. It is also possible for the student to view the state of their group as a list. In this case all members of the group are shown regardless of their location in the lab or login status. If they are logged into the system, the name of the machine they are using is given. If they have asked for help this is indicated and, if they have permitted it, the topic keywords for the help request will be given also.

The area at the bottom of the screen allows the students to request help. The students must type in some keywords describing their problem before they are allowed to submit a request. If they check the 'Broadcast topic keywords to group members' box the keywords they enter will be shown to other group members. The area at the top right of the screen allows students to record and delete memos. These memos are only visible to the student. The student is also able to view their record. This allows the student to see the information that is provided to the tutor, such as previous help requests and when they have been seen by a tutor. There are also some links to web pages providing help in operating the LSS.

The Tutor System

As well as allowing tutors to service student requests for assistance, the tutor system also provided tutors with background information on students, thus enabling them to provide more

[12] In this context unoccupied means not being used by a student who is the lab group and is using LSS. Future versions of LSS will more accurately represent the presence of a student at a machine regardless of lab group and use of LSS.

meaningful help to individual students. Furthermore, by providing a high-level view of the entire group, it was hoped that tutors might be able to recognise problems common across their group and, for example, convene a mini-tutorial. It was not our intention, however, to impose any particular strategy for the handling of help requests. We wanted to give tutors more information during labs, but leave it to them to decide how to use that information.

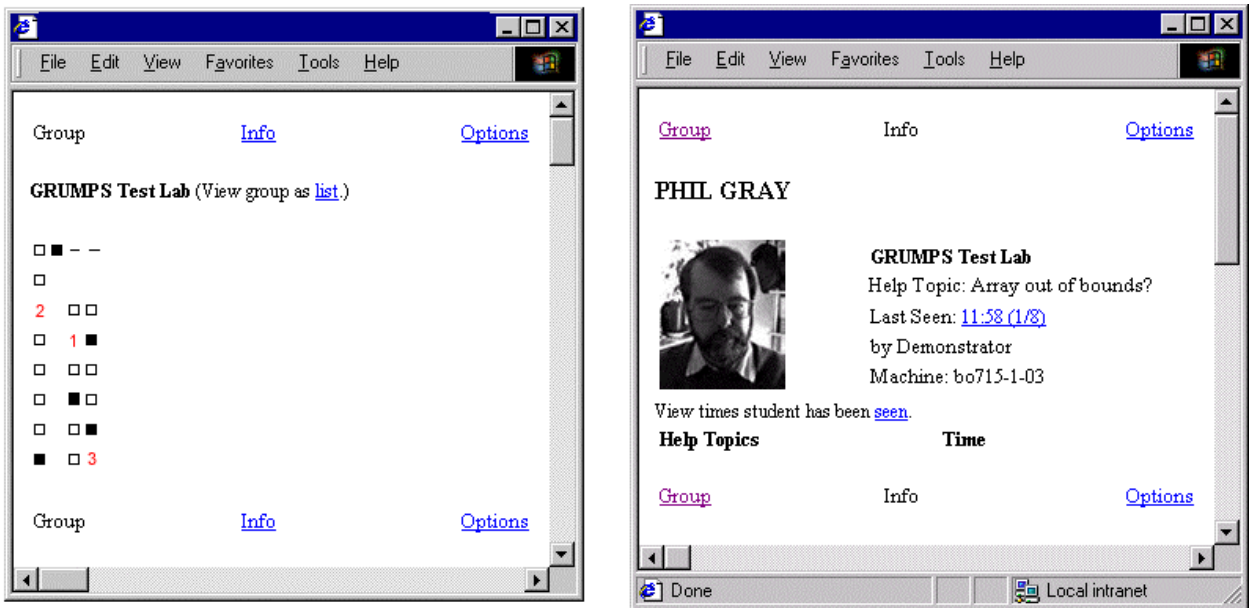

Fig. 2. **Tutor interface to the LSS. The map view on the left is shown giving the tutor a high level view of the state of his/her group. The right-hand image displays the detail associated with the student logged in at workstation represented by '1'.**

The tutor interface is shown in figure 2. The tutor has three separate windows, only one of which can be viewed at a time. In the options window, the tutor can select the lab group they are going to work with, how that group will be represented, either as a map or a list, and, if represented as a map, how the map is rendered: the orientation of the map and whether the whole lab is viewed or just the area with the group's students. Figure 2 shows the group view rendered as a map. The tutor view is similar to the student view, but students requesting help are allocated a number, based on the order they asked for help. In this case, there are three students asking for help. A

tutor can see a student's record by clicking on their representation on the map or their name in the list view. This brings up the information window which is shown on the right in figure 2. In this window, the tutor can see a photograph of the student, the student's help request, if any, as well as details of previous help requests and when the student has been seen by a member of the teaching team.

4 Locational Information in the LSS

Several forms of locational information are used in LSS. The entire system is based on data held in an SQL Server™ database. This database stores information describing the layout of a computing science teaching laboratory. As well as containing information about the x,y co-ordinate location of individual machines, the database stores higher-level locational information such as the orientation of the machine and the cluster to which the machine belongs. This information is used to generate maps for both the student and tutor interfaces.

In the student interface, the orientation of the map depends upon the computer the student is using. The orientation of the map is always such that what is in front of the student in the lab is above the student's location on the map. This is achieved by using the orientation information associated with the workstation as stored in the database. Fixed orientation is not possible with the tutor's interface, since it is used on a handheld device. Therefore, map orientation is configurable via a menu, including options presented in terms of four landmarks in the room (windows, printers, tutorial room, fire exit).

The students are not able to specify the area of the lab visible to them. If the student is sitting in the correct cluster for their lab group then they can only see that cluster. If they are sitting at the incorrect cluster for their lab group they can see the cluster they are sitting at, the cluster they should be sitting at and any clusters located between them and the cluster they should be at. Furthermore, to reinforce the fact that they are sitting at the wrong cluster, the student's representation on the map changes from 'X' to '*'. By minimising the amount of the map shown

to students it is possible to minimise the screen space used taken up by the LSS interface on the student machines. Tutors are able to choose either a view of the entire lab or just the area where their students are sitting. In the latter case, if one or more students are sitting outside the appropriate cluster for the tutorial group, the map view is resized accordingly to include their machines in the view. Although the tutors are using a handheld device with a limited screen-size, because LSS is the only application running and the full screen is being used (as opposed to the student situation where LSS is run as a background application) the full map of the lab can be displayed on the screen without the need for scrolling.

Different lab groups have different cluster(s) associated with them. This can be a single cluster or a list of clusters, perhaps covering the whole lab. Thus, the system is able to cover different perceptions of the importance of location. If students are to sit in an appropriate cluster for a lab their group has a specific cluster associated with it. If students are to sit anywhere (i.e. location is unimportant) the lab group would have all the clusters associated with it.

5 Observations on the LSS in Use

Both the LSS and its related keystroke level system were deployed in a 1st year computing lab from 24 April to 18 May 2001. A total of 27 tutors and 283 students used the LSS system during 87 two-hour lab sessions. During this trial, we collected data via the Grumps tools themselves, as well as via: direct observation during lab sessions; discussion groups (i.e., focus groups); questionnaires; diaries and interviews. The non-computer-based data collection was intended to identify critical incidents and to explore behaviours and attitudes that couldn’t be captured automatically. The comments that follow are based on this data.

Introducing our system tends to reduce the power of location to control the order in which help requests are serviced. We noticed that tutors sometimes felt compelled to use the numbering system even when they may have wanted to use some other servicing strategy. Direct observations of LSS and non-LSS lab sessions suggest that tutors find it uncomfortable to walk

past students who are further down in the queue of requests. 81% of tutors who commented on their LSS servicing strategy employed a combination of help request order and systematic circulation of the lab group.

During the trial, the use of the LSS changed the role of tutor location on student requests and on student perception of the response. When making a request for help, students located the tutor in the room and were more likely to use the LSS to summon help if the tutor was located at a distance or out of sight of the student; when asked to explain how they used the LSS, 26.3% of students who responded reported using this strategy. Similarly, students reported that tutors' strategies for responding to help requests were noticeably different when the LSS was in use. Students reported that tutors would respond to help requests as they appeared in the queue, rather than servicing student help request based on the students' proximity. That is, students were aware of the change in tutor strategy that we also recorded. A number of students commented on the fairness of the first-come, first-served strategy.

Tutors tended to keep map orientation fixed, setting it up at the start of a session. Occasionally, they were observed turning the machine to orient it to the room rather than changing the map orientation.

For students we anticipated that the map would give them information about the state of their group – who was there and where. We expected that some might use this to find nearby students with similar problems. We have little evidence that this information was actually used explicitly in this way. With respect to orientation, students would often get a different orientation each time they logged in. We found no evidence that this caused any problems. Observation of students in the lab suggested that they could locate other members of their group in the map.

As stated above, we anticipated that the map, if used in combination with publicly broadcast topic keywords, might encourage peer-to-peer help amongst the students, the pedagogical

benefits of which have been described in other studies [Greer et al, 1998]. There is no firm evidence that the LSS stimulated this kind of behaviour. The post-study student questionnaire suggests that users experienced difficulty expressing their problem in keywords, and the recorded archive of keywords used also supports this.

Probably because of the relationship between tutors and students, we observed no explicit negotiation to establish the ways that the new technology would be used, unlike the process reported in other studies on the introduction of mobile location-aware technology [Weilenmann 2001].

6 Conclusions

Perhaps the main finding resulting from our dealings with location in developing and using the LSS is the richness of issues in this application that are related to location. Even without introducing sophisticated location awareness, introducing mobility to some of the technology had a significant impact on how people performed their tasks. We benefitted from what we found out during early investigations of the domains and we were also fortunate that unexpected and unanticipated effects of introducing a mobile information source didn't cause the LSS to fail. It appears that the details of location matter and can have a significant influence on the success and appropriateness of information presentation and interaction techniques.

We intend to develop the LSS further, although the means of doing so is currently under discussion. There are a number of ways in which such an LSS enhancement might benefit from location, including:

- Proximity awareness of a tutor

  In the current system, a student must explicity complete the help session by pressing one of a set of help completion buttons on their interface (the buttons correspond to different types of help completion, e.g., tutor provided help, I solved the problem myself, etc.) and they cannot make another help request until this is done. Students often didn't bother to

complete this, until they needed help again. Proximity sensing could be used to provide automatic closure on a help request. That is, if a tutor is detected physically close to the student, their subsequent departure (after a suitable interval) could be interpreted as completion of the help session or could trigger a modal dialogue (like using the taking of a card in an ATM to trigger dispensing of money).

- Location awareness of student
  We currently determine student location by login to the LSS. This is problematic because a student might be sitting in the lab, not yet logged in, or logged into the workstation, but not the LSS.
  - Detecting locational features of help requests
    Another location-oriented issue is identifying sets of help requests that are spatially nearby and presenting this information to the tutor, allowing the tutor to create ad hoc tutorial groups. The map provides some support for this, but still requires cognitive effort to identify potential groups.

Acknowledgements

This work was supported by EPSRC under the Distributed Information Management Initiative (Grant Number GR/N38114/01). We wish to thank all our colleagues on the Grumps Project (http://grumps.dcs.gla.ac.uk) for their feedback on the ideas expressed in this paper and for their contribution to the design of the LSS. Section 4 of the paper is based on observational studies carried out largely by Margaret Brown.

References

Atkinson, M; Draper, S; Gray, P (1999). The Grumps Project Proposal.
http://grumps.dcs.gla.ac.uk/documents/DIMbid-v8-29feb.htm

Draper, S. (2001) Lab Support Software. Grumps Technical Report.
http://staff.psy.gla.ac.uk/~steve/grumps/lss.html

Jim E. Greer, Jim E; McCalla, Gordon I. ; Cooke, John ; Collins, Jason A. ; Kumar, Vive; Bishop, Andrew ; Vassileva, Julita . (1998) The Intelligent Helpdesk: Supporting Peer-Help in a University Course. Intelligent Tutoring Systems, (1998): 494-503

Weilenmann, Alexandra. (2001) Negotiating Use: Making Sense of Mobile Technology. Personal and Ubiquitous Computing (2001) 5: 137-145.

Electronically enhanced classroom interaction

By Stephen W. Draper, Department of Psychology,
Julie Cargill, and Quintin Cutts, Department of Computing Science,
University of Glasgow

.

Abstract

A design rationale for introducing electronic equipment for student interaction in lecture theatres is presented, linking the instructional design to theory. Prospective studies are outlined for exploring its use over new ranges of application. Rival views of the concept of interactivity are one way to organise the evaluation of this learning technology.

Keywords: Creative solutions, overcoming limitations, design rationale, niche, feedback.

Introduction: the design

This paper describes the design rationale for introducing electronic equipment for student interaction in lecture theatres, and the studies now in prospect of the use of this equipment.

The equipment is essentially that of the TV show "Who wants to be a millionaire?": every member of the audience (i.e. each learner in a lecture theatre) has a handset similar to that of a TV remote control, the presenter displays a multiple choice question (MCQ), each learner transmits the digit corresponding to their chosen answer using the infrared link, a small PC (e.g. a laptop) accumulates the answers and displays, via the room's

projection system, a bar chart representing the distribution (totals) of the responses to audience and presenter alike.

The main pedagogic types of use of the equipment are:

1. **Assessment**, both formative and as practice for summative assessment. Here the MCQs are meant to test content knowledge, and perhaps are drawn from a bank used for formal assessment on the course. The advantages of the equipment here are that "marking" is fully automatic, each learner can know immediately if they gave the right or wrong answer, how their performance on the question compares to the group as a whole, tailored explanations may be given by the presenter, and the presenter equally sees immediately how well the class measures up on that question (feedback from learners to teacher). The feedback cycle here takes about two minutes per item (somewhat longer if explanations are given). Any kind of MCQ may be used, provided the response is a single selection from a small fixed set: whether the usual rather shallow item, or one designed to probe understanding more than information retention (possibly by prior use of phenomenography (Marton, 1981; Marton & Booth,1997) to map the common misconceptions).
2. **Formative feedback on learning** within a class (i.e. within a contact period). Similar items might be used, but in order to discover and demonstrate what points should be focussed on during the class. Thus one or several such question items at the start of a class could be used to select a topic for detailed coverage, while the same or similar items at the end could demonstrate to what degree the group now understood the topic.
3. **Formative feedback to the teacher** on the teaching i.e. "course feedback". While the standard questionnaire at the end of a term, semester, or course has in general a very small effect on changing anything (Cohen, 1980) and takes a year to do so, a quick on the spot anonymous poll half way through a class (e.g. on whether the pace is too fast or too slow, the jokes too numerous or infrequent, the examples too many or few) can be used to change things immediately.
4. **Psychology experiments**: for the particular case of teaching psychology, a very considerable range of experiments can be directly demonstrated using the

audience as participants. For instance visual illusions may be displayed and the equipment used to show what degree of uniformity of response is found. Priming effects can be shown, where the perception of an ambiguous word or display is affected by what was shown before. The performance of witnesses to a crime (including the effects of some well known biases) can be explored by showing a short film, followed by various questions about what was shown. Social psychology effects, e.g. on conformity, could be demonstrated if responses to early questions were faked to see whether the class then changed their responses to later questions. In general, experiments that rely only on a stimulus and a forced choice response, but not on accurate measurements of reaction times, can usually be demonstrated in this way.

5. Possibly the most productive application, however, and the one with the largest body of existing research, is in using the equipment to **initiate a discussion**. Here, a carefully chosen MCQ is displayed and the learners register an answer, thus privately committing to a definite opinion. The presenter then, however, does not indicate the "right" answer but directs the class to discuss their answers with each other. Having to produce explanations and reasons is powerfully "mathemagenic" (conducive to learning), which of course is why researchers learn so much from giving talks and writing papers, and why teachers make their students write essays and answer questions. The equipment can be a significant help in introducing this, even into large classes.

Justification or design rationale

Although techno-enthusiasts, and indeed many government agencies or departments, have been pushing the use of computers and other technologies in education, and there are now many people whose job is essentially this and who are therefore necessarily aligned with this indiscriminately positive attitude, there is still very little good evidence of benefits. Perhaps this is not surprising: Landauer (1995) found it very hard to discover evidence of economic benefits for using computer technology in general. Besides suggesting that developing evaluation methods powerful enough to test this may be a

more important, if more difficult, research task than generating yet another application of technology to learning, this does mean that each application should be carefully justified. In a review of a number of applications (Draper 1998), I argued that most applications showed no significant improvements over what they replaced, but that the few striking positive exceptions were characterised by "niche-based design": by a good fit between a particular learning situation and a specific technical solution. They were projects that had been inspired by identifying a specific weakness in current delivery, and had focussed technology on solving that problem rather than on replacing what had been adequately done before. Can the use of the classroom equipment described above meet the implied standard of justification?

In considering large classes in large lecture theatres, the main problem is usually analysed as to do with the lack of interaction and the consequent extreme passivity imposed on the audience. In terms of Laurillard's model of the learning and teaching process (Laurillard, 1993, p.103), this situation fails to support the iterative interaction between learner and teacher that is one of her underlying principles, and more specifically does not support even activity 2: the "re-expression" by the learner of what the teacher has expressed. Actually, with very highly skilled learners and a teacher reasonably in tune with the group, this can nevertheless take place: for instance, where the learners take notes that are not mere dictation, but substantial re-formulations of what is being talked about. (This is a reasonable theoretical analysis of the considerable benefits I have often obtained from listening to talks at conferences where I have not asked questions, but have nevertheless learned something useful.) However this degree of skilled, silent interaction is not often present in undergraduate teaching, and large numbers usually prevent learners asking sufficient questions to repair the attunement between speaker and audience, from both a pragmatic (there isn't time for many people to ask questions) and a social (it just feels too embarrassing) viewpoint.

That, then, is the diagnosis offered here of the chief weakness of lecturing to large groups. The handsets and associated equipment offer a way of tackling that weakness by allowing each learner independently to generate an answer (at least a partial instantiation

of activity 2), and to register that answer, and in so doing to affect the course of what happens next. The summed responses are real feedback to the teacher, that naturally leads to adjustments and reattunement if required, and in fact do this better than questions and answers from any subset of individuals. Furthermore the equipment offers an anonymity of response that addresses the shyness that additionally inhibits any interaction.

Is the equipment really likely to be any better than the alternatives? The simplest alternative is getting students to give a show of hands. This equipment crucially offers more privacy (it's a secret ballot, and important for just the same reasons). Other rival technologies are to issue each student with a cardboard or plastic cube with a different colour on each face, to be turned to show their "vote"; or with a large sheet of paper divided into a few squares each with a digit in, that the student can hold up in front of their bodies and point to the digit they select. These methods allow only near neighbours to see a student's selection. Thus the electronic equipment offers somewhat better privacy, but the difference may only be crucial with new classes: it is quite possible that with a class grown comfortable with the electronic version, moving over to a cheaper but less private version might not destroy the interactivity. The electronic version also provides faster and more accurate counting of the results: most presenters will only estimate shows of hands to the nearest 20%. The accuracy may have a small but not negligible value in making all participants feel their views count, and are not just lost in crudely approximate estimates.

In scrutinising this instructional design rationale, note that it does not feature computers in a starring role (although actually one is crucial to tabulate the results): the instructional design mostly isn't in the equipment or software, but in how each teacher uses it. That is a lesson which perhaps the rest of the learning technology field should take more to heart if the aim is in fact to improve learning rather than to promote the glamour of machines. On the other hand, note too that this design does not fit with a simplistic interpretation of the slogan "learner-centered". Improved learning and the learners are the ultimate intended beneficiaries, but that end is achieved to a great extent by first serving the teachers better,

by giving them much better, faster, and more detailed information on what the learners are thinking now and where their problems are at each point.

Prospective explorations

There is a considerable history and community of practice in using such equipment in the specific area of promoting discussion (the last of the pedagogic uses listed above) and so improving student understanding in science and engineering at the school and early university levels (e.g. Hake 1998). The author and colleagues have obtained sufficient equipment for several lecture theatres, and are about to begin exploratory studies, particularly with a view to exploring the range of applications, and how far its utility can be demonstrated beyond its best established application area. We hope to trial its use in all five of the pedagogic modes listed in the first section, in two universities (Glasgow and Strathclyde), in at least two disciplines (psychology and computer science) in both universities together with several others as opportunities arise, at various levels (years) in undergraduate programmes, and in a range of group sizes from 300 students downwards. (The biggest need and the biggest potential gains are in the largest group sizes, but innovation is of course a lot "safer", i.e. easier to manage, in smaller groups.)

The exploratory studies should yield practical knowledge such as question banks for the participating disciplines, and how much support is needed for first time use (a new lecturer and students who haven't used the equipment) and for regular use. They will also yield evaluation results on what benefits can be demonstrated. We hope to use a version of the method of Integrative Evaluation (Draper et al. 1996) to address both these aspects.

Interactivity

Some of the most important evaluation issues can be organised around the notion of interactivity. Some researchers tend to an almost mechanical interpretation of interactivity e.g. counting the number and branching ratio of choice paths for users in multimedia learning software (Sims, 1997; Hoyet, 2000). With this equipment, that corresponds to the number of questions put to the learners for them to respond to,

regardless of their content. It also corresponds to the effects we may well see of novelty, of the perception that the teachers are taking special trouble over the teaching (the Hawthorne effect; Mayo, 1933), or simply of physiological arousal (the physical activity involved in pressing buttons i.e. mechanical interactivity) which has led to the heuristic rule of not lecturing for more than 20 minutes without a pause, having the audience move around periodically, etc. On the other hand, if we believe in the Laurillard model, then the important factor would probably be the amount of time each learner spends on activity 2 ("re-expression"): so using the handsets should be better than a non-interactive monologue, but not as good as time spent in peer discussion (open-ended verbal responses rather than selecting one of the digits on the handsets). On the other hand again, if what is important about "interactivity" is actually changing what happens by visibly affecting the teacher (i.e. genuine human-human interaction with the actions of one party being contingent on those of the other), then it will be changes to what the session is used for as a result of responses to questions near the start that predict the largest learning gains. Varying approaches in classes, and taking independent measures of both learning and enjoyment or alertness should eventually allow such questions to be decided.

Other technical details

There are some further detailed issues that arise, and could be investigated. The particular equipment used transmits not only a digit to signal the learner's selected response to the question, but also a confidence level (high, medium, or low), and an ID for that handset which may or may not have been arranged with a known mapping to the student's identity. Furthermore the number of attempts each learner makes at the question before the cutoff time may be recorded. The GRUMPS (2001) project is interested in exploring data mining of records of such student interactions, though that involves negotiating issues of privacy and data protection with the students. We are writing software to smooth the integration of the equipment with other lecture facilities (e.g. the use of PowerPoint presentations), and with keeping records of the interactions.

Conclusion

The studies in prospect with this equipment should eventually allow us to pronounce on the validity of the design rationale presented in this paper.

Acknowledgements

This work is being supported in part by the EPSRC funded grant to GRUMPS (GR/N38114), and also in large part by the University of Glasgow, both directly and through the TLC project. Many thanks to Prof. Jim Boyle of Strathclyde University, whose rich existing experience with this equipment gives us confidence in proceeding, and many practical tips. We look forward to collaboration with him, and with David Nicol who has done a pioneering evaluation of Jim's work.

References

Cohen,P.A. (1980) "Effectiveness of student-rating feedback for improving college instruction: a meta-analysis" *Research in Higher Education* vol.13 pp.321-341

Draper, S.W. (1998) "Niche-based success in CAL" *Computers and Education* vol.30, pp.5-8

Draper,S.W., Brown, M.I., Henderson,F.P. & McAteer,E. (1996) "Integrative evaluation: an emerging role for classroom studies of CAL" *Computers and Education* vol.26 no.1-3, pp.17-32

GRUMPS (2001, May 30). The GRUMPS research project [WWW document]. URL http://grumps.dcs.gla.ac.uk/ (visited 2001 June 1)

Hake,R.R. (1998) "Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses" *Am.J.Physics* vol.66 no.1 pp.64-74

Hoyet,H. (2000) "Graphing Interactivity in technology-based training" *TechTrends* vol.44 no.5 pp.26-31.

Landauer,T.K. (1995) *The trouble with computers: Usefulness, usability, and productivity* (MIT press; Cambridge, MA)

Laurillard, D. (1993) *Rethinking university teaching: A framework for the effective use of educational technology* (Routledge: London) p.103.

Marton,F. (1981) "Phenomenography -- describing conceptions of the world around us" *Instructional Science* vol.10 pp.177-200.

Marton,F. & Booth,S. (1997) *Learning and awareness* (Mahwah, New Jersey: Lawrence Erlbaum Associates)

Mayo,E. (1933) *The human problems of an industrial civilization* (New York: MacMillan) ch.3.

Sims, R. (1997) "Interactivity: A Forgotten Art?" *Computers in Human Behavior* vol.13 no.2 pp.157-180.

PRS Support System – A Summer Project

Chris Mitchell

Preface

This report is based around work undertaken for the GRUMPS[2] research project over the summer period 2001 between the third and fourth years of my computer science degree. I was hired to develop software for an interactive teaching tool and to integrate this software into the existing GRUMPS toolset.

PRS

A system has been developed by Hong Kong University of Science and Technology, which uses infra-red handsets to allow an audience to interact with the speaker. The system, called Personal Response System (PRS) [3], is composed of the following parts:

- **Infra-red handsets.** A numbered keypad allowing the user to select an answer
- **Infra-red receiver**. Interface between the handsets and the computer
- **Computer.** Runs the PRS software which displays responses from the audience
- **Projector or video screen**. Allows the audience to see that their response has been accepted and displays the results in a graph.

The use of this system is primarily based around multiple-choice questions where a question is presented to the audience along with several numbered options. Each member of the audience can then select their answer by pressing a key on the keypad. Once all responses have been received, the PRS software can automatically produce a graph of the results.

In the context of teaching this tool is primarily aimed at indicating students' understanding of material being covered in a lecture, giving the lecturer information that might change the way the lecture proceeds. Other uses of the system have been suggested including:

- **Attendance.** Students can be identified by their handset and consequently class size can be monitored
- **Assessment** The system could be used to provide instant marking (and therefore feed back to students on their understanding).
- **Discussion** Questions can be asked to provoke discussion of a particular topic.
- **Experimentation** The system could be used in the context of a psychological experiment. Some form of stimulus is given to the participants and their response is recorded. Timing of the responses is saved so this could also be explored.

Problem Definition

The problem to be addressed over the summer was that the PRS software did not meet the requirements of the GRUMPS project or the needs of the identified user group, lecturers on the Level 1 computing science course. Therefore my task was to build an application to run with the PRS software and hardware that would implement the functionality that was not available in the original software.

The three main pieces of functionality that were needed but not included in the PRS system were:

- On-screen display of question and possible answers
- Storage of the question and possible answers
- Integration of instrumentation software to allow data collection from PRS on user behaviour.

Further requirements were identified such as simplifying the process of using the PRS software by, for example, prompting the user for input.

Instrumentation software has been developed by the GRUMPS project to allow data about a user's interactions with a piece of software to be collected and stored centrally. This allows either real-time retrieval of the data or data mining at some later stage.

System Users

The users of the system were viewed as fitting one of two categories: lecturers who already use a computer and projector to display lecture notes; and those that used some other, non-electronic method. For those who did not use computers to display their notes, the task was more straightforward: produce a system that shows both the PRS system software and the question and possible answers. Users who already use computer systems, namely Microsoft PowerPoint, pose a greater challenge. Either they would require two computer systems (one for their lecture notes and one for PRS) or an easy method whereby the user could switch between their notes and PRS. The latter was chosen since the use of two computer systems would often be limiting

(availability of equipment, lecture theatres too small, etc). The intended user base included many users who were familiar with Microsoft PowerPoint therefore the idea of integrating PRS into Microsoft PowerPoint using Visual Basic for Application became the chosen solution. This would also allow automated switching between lecture notes and the PRS system.

The PRS Support System

System Overview

A Microsoft PowerPoint presentation that is to be used with the PRS system is only slightly different to a standard presentation. The user can prepare lecture slides beforehand and insert PRS questions at specific points in the lecture, or they can prepare a lecture with no preset PRS questions and insert and present a PRS question during the lecture. Slides are prepared in the normal way but when a PRS question is to be inserted into the lecture, it is done with a menu option “insert PRS question”. The application then asks the user for the question text and up to ten possible answers. The text of the question and answers is stored on a slide which is inserted into the set, allowing further editing. Each time a PRS question slide is reached during a slideshow, Microsoft PowerPoint is hidden and the PRS software is displayed. The screen is split in two with the PRS software in the top half, and the in the bottom half. The lecturer then starts a timer and waits for the student responses. After the responses are received and the results displayed, the lecturer can continue the slideshow and the next slide in the set is displayed. The slideshow continues as normal until the next PRS question slide is reached. Provision is made for asking a question “on the fly” during a slideshow.

Users who are not proficient in Microsoft PowerPoint or who prefer another method of lecture presentation are able to prepare question and answer sets using a text files. The question and answers and then imported into the system for use during the lecture.

The Component Parts

Visual basic

Two main flavours of Microsoft Visual Basic exist: Visual Basic (VB) and Visual Basic for Applications (VBA). VB is a rapid application development language. It has a graphical Integrated Development Environment (IDE) and uses an event-based model for Graphical User Interface (GUI) development on Microsoft Windows. VB can be viewed as a slightly limited programming language as not all Windows system functionality is immediately available, although provision is made for extending its capabilities: system calls that are normally only available to C programmers can be coerced into VB. VBA is a version of VB specifically tailored to development of Microsoft Office applications. An editor is provided in each Office application. Code can be saved in two main ways. Firstly it can be saved as part of a document, which allows the user to see the code (and perhaps alter it). Or secondly it can be saved as an add-in, which separates the code from the document and makes the code available throughout the application rather than a single document. The main advantages of add-ins are that they can be easily loaded or unloaded, they don't allow the user to see the code and they provide two methods, `auto_open` and `auto_close`, which are called by the application when it opens or closes. Any initialisation that is required by an add-in is therefore done in `auto_open` and any tidying up is done in `auto_close`.

Microsoft PowerPoint 97

Although Microsoft PowerPoint 2000 was available most of the intended users would only have access to Microsoft PowerPoint 97. This version was therefore selected for development. Microsoft PowerPoint 2000 would have provided a more advanced environment to work with since it provides a set of events that can be used to execute code. One of these events deals with the transition from slide to slide during a slideshow. This would have been ideal for developing the code that interacts with the PRS system. Microsoft PowerPoint 97 does not provide these events so an alternative was necessary. A separate application, a Dynamic Link Library (DLL) written in VB, was developed to run behind Microsoft PowerPoint to periodically check if it is time to swap between the slideshow and PRS. The DLL achieves this through use of the

system timer functions. Though not an ideal solution, this does not impact too greatly on performance as very few resources are used until it is time to swap. The rest of the code for the system is written in Microsoft PowerPoint VBA as an add-in.

Discussion of the Implementation

Two forms are used in the system: one for editing the question and answer text and one for displaying this text. The editing form presents the user with eleven text boxes: one for the question and ten for the possible answers. The details are read from the form then saved to a slide. An unfortunate draw back of the VBA system is that forms are always modal. That is, they are displayed in such a way as to prevent the user from interacting with the application until the form is dismissed. This means that during editing of the text the user cannot rely on being able to return to Microsoft PowerPoint, where they may want to view the content of their presentation while creating their question. The form which presents the PRS question to the students reads the text from the slide and displays it. The display of the text is complicated by several factors:

- The size of the display area is limited.
- All answers need to be visible at the same time.
- Not all answers will be of the same length.
- Some answers may consist of multiple lines.
- Not all questions will have the same number of answers.

To address some of these problems, the display form automatically adjusts the positioning of the text according to the current question and answers.

During the project my idea of how the text of the question and answers should be store changed in response to user testing. Originally, in order to avoid malformed question and answers, the text was stored in such a way as to prevent the user from editing the text other than by using the modal form to enter the details as mentioned above. Initial tests with a user suggested that lecturers might want to edit the question and answers on a slide. Both forms mentioned above use the methods of a shared class to get and set the question and answer text, making code changes easier to

implement. An apply method is called to save form contents to a slide. In order to allow the user some editing facility within Microsoft PowerPoint, the question text is now the title of the slide and the answers are textboxes, which allow previously created text to be corrected or changed entirely. The editing form is still used for inserting new question slides (including on-the-fly questions) and altering the number of possible answers to a question.

Several other pieces of code go together to make up the completed system, mainly dealing with setting up menu commands and reading text from files.

Comparison with other Interactive Lecture Systems

A company called Socratec [4], in conjunction with Educue[1] (the distributors of PRS), has produced an interactive teaching system called Respondex based around Microsoft PowerPoint. As various systems of handsets currently exist, Respondex provides the ability to work with several (though not at the same time). In the case of PRS it collects keypad responses directly, completely removing the need for the original PRS software. The system is based around a set of templates that are inserted into the presentation and filled with the question and answers. The various templates provide for using the system in various ways. Types of questions that are provided for include Yes/No/Unsure, Averaging and Likert scale. Each template can be set to display various statistics derived from the answers received (simple percentages, bar charts, totals, etc). Although the system also has the ability to take multi-digit answers it is unclear whether all handset can do this. The system can also be used to pick volunteers who gave a particular answer. The answers that the audience gives are stored in a database for future investigation. The database also allows reports to be produced showing information about the responses given.

While this system provides some functionality that my system does not, the converse is also true. For instance, the ability to read questions and answers from a text file, while not revolutionary, puts the system within reach of a wider set of users.

The original PRS system is currently at version 2.47 but the makers of the system, Educue, are releasing version 3 in the near future.

Future Work

The PRS support system as it stands will allow lecturers easier and more comprehensive use of interactive teaching based around the original PRS system. The lack of the source code or receiver-computer protocol limits the scope of the project greatly. Several ideas that have been proposed have been infeasible due to this factor. Such things as allowing the audience to enter a sequence of digits as their answer would require several separate questions (i.e. please enter digit 1, please enter digit 2, etc). Other suggestions that either have not been addressed yet or have been dismissed as currently unfeasible are:

- **Alex Johnston Grid Style Questions** The user is given a question that they work on over an extended time period and submit answers at various points in working through the question. These questions allow a depth equivalent to an essay to be tested whilst at the same time provide instant marking.
- **Graphical Questions** Where a picture is presented with areas labelled with numbers representing the answers to a question
- **Database storage** Using database technology to store application data
- **Graphing** Custom graphing of responses would allow more flexibility than is currently available. Also in combination with the database, two sets of answers to a question could be displayed to an audience for comparison. A question could be asked at the beginning and end of a lecture to see if the lecture has improved understanding of a particular topic.
- **Automation of PRS** The software could be automated using windows system calls.

More work is necessary on what lecturers would like to do with the system and also on how they would actually use the system. The latter of these two could be discovered by instrumenting the support system itself or interviewing lecturers and examining the gathered data to understand usage.

Conclusion

The project has been an enjoyable experience although VBA has not been an easy language to work with. This is probably due to the fact that it is intended for writing more simple macro-style code. Although other systems have come to our attention, this system provides integration of the GRUMPS instrumentation software. The quality and scope of the system would be improved if the source code was available.

References

[1] Educue – http://www.educue.com/

[2] GRUMPS – http://grumps.dcs.gla.ac.uk

[3] PRS site at HKUST – http://www.ust.hk/celt/prs/

[4] Socratec – http://www.socratec.com/