Grand Unified Model for Control and Analysis Systems

A.Götz[1] (ESRF) and N.Hauser (ANSTO)

Introduction

All scientific data acquisition systems have control system and analysis programs. While many control systems try to offer similar functionality they are often implemented in very different ways. This is equally true for control systems used to run synchrotron and neutron facility beamlines. There are almost as many control systems as there are synchrotron and neutron institutes in the world. At first sight these control systems look different but a deeper look reveals them to be variations on the same theme. Drawing on the combined experience of the authors this paper explores what is common in scientific control systems and how this commonality can be used to set up a common development framework. It makes a proposal for a Grand Unified Model for control systems and analysis programs. A common integrated graphical user interface for control systems based on the GUM theory is presented as an ideal topic for international collaboration.

Improving the Past

The past is filled with many home made control systems for synchrotron and neutron beamlines. In the very early days in-house experts built most system from the ground up. The main reason for this was that computer control systems were still in their infancy and therefore there was no big market for them. Consequently there were few commercial systems to choose from. As control systems started to become ubiquitous (nowadays many simple home appliances have a control system) industry caught up with the demand and started offering a wider range of off-the-shelf components. But because the needs of scientific data acquisition are extremely varied and demanding there are not any complete solutions available off-the-shelf yet. What has instead become common is to mix commercial solutions with home grown solutions. The home grown solutions have also evolved. With the arrival of Internet there is more and more software freely available. The quality of this software has been increasing steadily so that today it is possible to get high quality compilers, language interpreters, web servers, frameworks, and class libraries for doing a plethora of tasks.

One of our goals as system integrators is to continuously improve the past. We therefore think that the increased availability of high quality free software should be leveraged to build what industry still does not provide – a flexible beamline control and analysis system capable of meeting all the demands put on it. If possible it should be done as a collaborative open source project. The resulting software should be freely available for other institutes to use and improve.

Philosophies & Paradigms

Control systems are built around philosophies or paradigms. Control systems can be classified according to the philosophies and paradigms they implement. We have identified the following philosophies:

1. Simple control

[1] On sabbatical leave at ANSTO from the ESRF

2. Data acquisition control
3. Data acquisition and analysis control

We have identified the following communication paradigms:

A. Client-server e.g. SICS, TACO, ICE
B. Distributed database e.g. EPICS
C. Master-slave e.g. SPEC
D. Fully centralized e.g. OpenGenie
E. Web distributed e.g. DANSE
F. Peer-2-Peer e.g. GUM

We have identified the following task distribution paradigms:

I. Mono-task e.g. SPEC
II. Two tasks e.g. SICS
III. Many tasks e.g. TANGO, DANSE

Some control systems maintain they do not try to follow any of these philosophies and only use commercially available components. Commercial components can also be classified into the above categories. This kind of classification system has the advantage that it is compact and universal. Systems, which implement 3FIII paradigms, are the ones with the highest level of integration and symmetry. GUM systems are 3FIII.

Grand Unified Model

Very few of the known control systems around today treat the problem of control and analysis as a combined problem. Most of them treat only the control problem or the analysis problem. In order to unify the two we propose what we have called the Grand Unified Model (GUM) for control systems. The Grand Unified Model for control systems states:

The control and analysis parts of a scientific experiment must be treated as part of one system with input and output being readily exchanged between all parts of the system. There must be a single integrated graphical user interface from which all aspects of the control and analysis system can be accessed. There is a basic set of building blocks that all control and analysis systems should have. All building blocks should have a well defined interface.

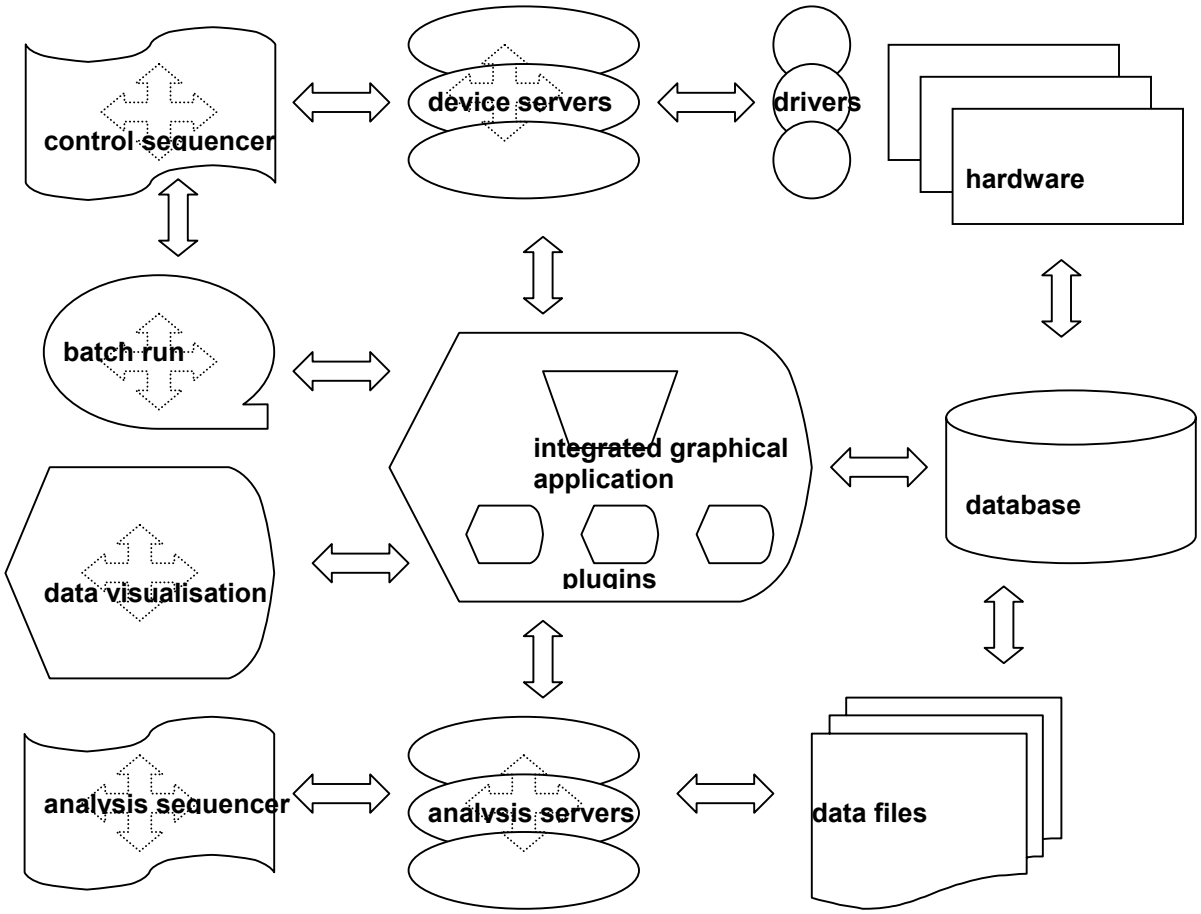
This is a ***model*** because it doesn't exist and is more like a blueprint for building a GUM control and analysis system. It is ***grand*** because it goes beyond what most systems are doing today and its aim is to make doing science much easier. It is ***unified*** because it can be used to control almost any kind of scientific experiment. The single integrated graphical user interface strengthens the concept of a single system by giving the scientist a single entry point for all parts of the system. The building blocks are identified below. GUM is also an ideal topic for international collaboration. The well defined interface to the building blocks is what enables a GUM system to support different implementations of the same building block.

Building Blocks

Philosophies alone do not a control system make, we also need building blocks. The building blocks of a control and analysis system are the basic elements needed to take and analyse data. We identify 13 software elements as being essential for solving the control and analysis problems:

1. **device drivers** – the operating system dependant software needed to access control hardware via the local bus e.g. CCD camera driver to access a CCD camera via a fast serial link via a local PCI card.

2. **device servers** – these are software processes which make hardware accessible via the network e.g. embedded controllers, TANGO device servers
3. **network protocols** – needed to communicate with remote devices, device servers and analysis servers e.g. TACO, TANGO, EPICS channel access
4. **database** – indispensable for any software system for storing configuration data, intermediate results and keeping an archive of changes to the system
5. **scanner** – scientific data is usually taken by scanning, scanning involves changing one or more parameters, taking data and repeating the sequence until sufficient data has been acquired
6. **sequencer** – in addition to scanning a scientific experiment also needs a sequencer in order to adapt the control and analysis system to the needs of the experiment, examples of sequencers are python, SICS, SPEC, ICE
7. **batch** - the batch software enables experiments to be setup over long periods of time in unattended mode
8. **command line interface** – a command line interface is required as front end to the sequencer
9. **graphical user interface** – a command line interface can be daunting for non-expert users, it is necessary to supplement it with a graphical user interface where the main control information is made easily accessible
10. **data file format** – the main aim of a scientific instrument is to produce data, data is commonly stored in files, the files need to have a well-defined format
11. **analysis servers** – once the data has been generated it needs to be analysed to provide feedback on the quality of the data, analysis servers are needed to crunch through the data
12. **analysis sequencer** – users need to sequence the different analysis programs in order to suit their needs
13. **data visualization** – raw and analysed data needs to be visualized so that the scientist get feedback on the progress of the experiment and make a decision on what is the next best step in the experiment.



Implementing the Building Blocks

How to implement a GUM control and analysis system today? Clearly there is no unique solution. Depending on the implementer's background and preferences different base technologies will be used to build the system. This is not necessarily a bad thing because variety is the basis of evolution. Having different implementations of the GUM control and analysis system will enable future implementers to compare them and make the right choice. It is more important today to discuss and agree on the basic building blocks of GUM and what their functionality should be. Having said that we still need to start somewhere if we want to build a GUM system. Here are some tips on implementing the building blocks of GUM based on the authors own experience and preferences:

1. **device drivers** – use a well-known operating system which is stable and has sufficient hardware and software support for the hardware you want to use. The authors recommendations are:
   - ***Linux*** – popular, stable, easy to manage, does not support a very wide range of hardware yet but this is changing, relatively easy to write drivers for
   - ***BSD*** – similar to Linux but supports even less hardware and software, reputed to be more stable and secure than Linux
   - ***Windows*** – wide range of hardware and software supported, however not easy to manage, expensive and drivers are not easy to write
2. **device servers** – use a well debugged device server model for writing your servers. The authors recommendations are:
   - ***TANGO*** – developed by the ESRF, Soleil, Elettra + ANSTO, supports a wide range of functionality for distributed components are implemented in C++ or Java, clients can be in C/C++, Java, Python, Matlab, Igor or Labview
   - ***TACO*** – also developed by the ESRF, FRMII + HartRAO, has less functionality than TANGO, distributed objects are implemented in C or Python, clients can be in C/C++, Python, Matlab, or Labview
   - ***SICS*** - drivers can be accessed via the SICS socket
   - **EPICS** – widely used, not designed from the ground up as a wrapper for single pieces of hardware, this functionality was added later on
3. **network protocols** – use one based on TCP/IP or UDP/IP. Where possible insist on network access instead of serial lines. Use the device server protocol when no network access is provided. Virtual LANs can improve network response and robustness in the case of broadcast storms.
4. **database** – make sure you design your system with a database from the beginning. *A database should accompany any control and analysis system throughout its lifetime*. The authors recommendations are:
   - ***MySQL*** **-** the cheapest and most popular choice. MySQL provides an efficient implementation of SQL. Simple to setup and manage. For improved reliability use the Innodb instead of the MyISAM format for tables.
   - ***Postgres*** **-** another free database which implements SQL. Has a good reputation for being a complete implementation of SQL. Postgres is used by research groups working on SQL.
   - ***Oracle*** **-** a well-known and widely used commercial database. Expensive and not easy to manage but very reliable and has an extensive set of functionality.
5. **sequencer**[2] – this is a fundamental brick in your system. It allows you to interpret and execute commands in order to debug, commission and prepare experiments. It should be a simple interpreted programming language which can ideally be used to access any part of the control and analysis system. The main advantage of a sequencer is its extensibility. It is strongly recommended to have a debugger for the sequencer. This can make your life much easier. There are many examples of sequencers, some of the popular ones are:
   - ***Python*** - a popular interpreted programming language which can also be used as a sequencer. It can be extended by linking it with modules written in C or C++. Has been chosen by Caltech for DANSE and NIST for ICE.

[2] a sequencer interprets and executes commands

- ***Jython*** - is a variant of Python running in a Java Virtual Machine. It has the same syntax as Python. It supports calling Java classes directly from Python instead of modules written in C.
- ***SICS*** - a sequencer written at PSI specifically for doing neutron experiments. It is based on Tcl. Has driver support for a range of hardware. It is based on a client-server concept and supports multiple clients. SICS has also been chosen by the RRR reactor project in Sydney as the sequencer. It does not have a debugger.
- ***SPEC*** - a commercial sequencer from Certified Scientific Systems mainly used at synchrotrons. It has driver support for a wide range of hardware support. It is written in C and the language has its roots in awk. It recently supports a server mode for accessing some of the functionality via the network. It does not have a debugger.
- ***Tcl, Perl, JavaScript***, ... - in theory any interpreted language which can be extended could be used as sequencer. In practice it is advisable to choose one with a clean syntax, a good debugger and tailor it by adding control and analysis specific extensions.

6. **scan**[3] – Although it is in theory possible to make up any scan using a sequencer the authors think it is better to implement scanning as a special case. This has the advantage of having a robust implementation of common (if not all) scan types. It is possible to have a separate device server just for scanning or to use the sequencer as your scanner. The main differences are:
   - ***scan in sequencer*** **-** scanning done by the sequencer is by far the most common solution. It usually consists of a set of sequencer scripts that implement the scanning logic of setting actuators e.g. motors, and reading sensors e.g. detectors. Although this solution is common it does have some disadvantages. Because it it written in a flexible language it does not have a well defined interface.
   - ***scan server*** – the scan server is a server which implements the scanning logic. It has a well-defined interface which the user cannot change on-the-fly. It has the advantage that it can be easily integrated in a multi-client environment.
7. **batch**[4] - Scientists need to sleep. Batching is the key to this activity. It allows the scientist to setup a series of scans which will run in unattended mode. In the ideal case it should be possible to interrogate the status of the batch, interrupt it, insert new scans etc. while the batch is running. Batch should also recover from minor errors to continue with the next acquisition
   - ***batch in sequencer*** - most beamline control systems support the execution of scripts as so-called batch runs. Although these run in unattended mode they are not strictly speaking a batch because they do not support interrupting the batch and changing it on the fly.
   - ***batch server*** - a better solution would be to have a real batch run concept implemented in a server (similar to the Ruenbuffer in SICS).
8. **command line interface** – the command line interface allows you to type commands in a window and send them to the sequencer. Possible solutions are:
   - ***ascii terminal*** - this is the most basic solution and is often the most common one. It has the advantage that it requires no implementation. It suffers from being too basic. It offers no context sensitive help to the user.
   - ***graphical CLI interface*** - this is the evolved version of the ascii terminal. It includes mouse support, an online history, short cuts, graphical display of status etc. This solution implies the sequencer can be contacted remotely e.g. via a server interface or a telnet socket.
9. **graphical user interface** – the logical extension of the graphical command line interface is a fully graphical interface with buttons, status displays, and widgets for setting up the system. There are numerous examples of graphical user interface, in fact as many as there are control systems. However because GUM does not exist yet there is no solution which will fit on top of more than one control and analysis system. Attempts at rectifying this e.g. Gumtree, are still in their infancy. The choice is therefore either limited to deciding on which language is best suited to your needs and implement your own graphical user interface or join something like the Gumtree collaboration. Here are some tips on choosing a language and framework:
   - ***Gumtree*** - is a new project which is a graphical user interface for multiple control systems. Actually Gumtree is closely related to this paper and is a first attempt at implementing the GUM

[3] a scan iterates a variable from an initials to a final state and acquires data
[4] a batch is an arbitrary list of scans

theory. Gumtree is implemented in Java using the Eclipse framework. The framework provides all the infrastructure necessary to build a fully functional and professional graphical interface. It uses the SWT set of graphic widgets. Gumtree is still a sapling though and it is not complete. Gumtree is described in more detail below and in other papers presented at this conference.

- ***Java/Eclipse*** - is a Java framework developed by OTI for and distributed by IBM as open source. It provides a very good framework for developing graphical user interfaces. If you are planning on inventing something new then at least consider using Eclipse as your framework. The Eclipse framework is ideal for developing integrated applications. Everything in eclipse is developed as a plugin. Refer to the eclipse web site for more information.
- ***Java/Swing*** – Swing is the standard widget library for Java. Therefore a large number of Java graphical application use it. Swing implements a common look and feel for all platforms. One inconvenience of this approach is that Swing applications react differently compared to native applications. Because Swing is implemented entirely in Java it is slower than the SWT widget library of Eclipse which uses the native libraries on each platform. Swing however has the advantage that it runs on all platforms where Java runs.
- ***Python/Qt*** – Python is a popular language and a fair amount of graphical applications have been written in Python. There is no standard Python graphical library. Qt it quite widely used. Others are wxPython, tkInter etc. One disadvantage of Qt is it needs a platform specific build.
- ***Labview*** – one of the stalwarts of laboratory automation. Almost all institutes have a bit of Labview in their control system. Labview has the advantage that it is easy to build a quick graphical application. However larger applications are difficult to maintain. Some institutes have attempted to develop their entire graphical interface in Labview but users generally complain about the lack of flexibility. Labview lacks a scripting language.

10. **data files** – the result of acquiring data for a scientific experiment is data. Data needs to be stored and managed. It also has a format. We strongly suggest you adopt a common format for all your data. If you are considering developing your own format think many times before doing. It is often an expensive and futile exercise. Take a careful look at existing data formats and try to figure out why it doesn't fit your needs. Try to use it and adopt or adapt it. There are too many data formats around which one institute only uses and which only serve to create work to convert them into other data formats. Some possible formats are:
    - ***Nexus*** - In the field of neutron and xray scattering Nexus is currently one of the obvious choices. If more institutes decide to use it then it will help fight the proliferation of new data formats.
    - ***CIF*** - the Crystallographic Information File is a format used for storing crystallographic diffraction patterns. It is sometimes seen as an alternative to Nexus but it does not have the same flexibility as Nexus for storing almost any kind of raw and processed data. Its main advantage is seen in the fact that it stores data in ascii format. This however is not adapted to large and complex data sets.
    - ***raw data*** – the debate is still on in the community whether to store raw data during processing in an open format like Nexus or whether to store it in a proprietary format. Proprietary raw data formats are usually simpler and more efficient to manipulate but they are often sorely lacking in information.
11. **analysis servers** – once the raw data has been acquired it needs to be reduced, processed, and analysed to produce a scientific result. This is where analysis programs kick in. Until recently analysis was considered to happen offline during or after the experiment. It was not considered to be part of the control system. With the increase in CPU power, the improvements in software, and the increasing number of non-expert users this assumption is not valid anymore. Indeed the GUM theory states that control and analysis should be joined together in one system. How to do this? It is clear we first need to make sure that the analysis programs exist. Then we need to incorporate them in the control system. There are various ways of doing this:
    - ***analysis servers*** - just in the same way hardware can be made available via a device server over the network so analysis programs can be converted to servers and be made available over the network. The same technology can be used to do both e.g. TACO or TANGO just to mention two the authors are familiar with.

- ***DANSE*** - is another interesting way of integrating analysis programs. It uses a web interface and farm of computers to execute the analysis programs. But how to connect this to the control system? Refer to the DANSE paper at this conference. DANSE is grid enabled.
- ***Grid*** - is one of the buzzwords of this decade. It could also be an answer to the problem of where and how to run analysis programs. Time will tell. Currently the grid technologies are still in their beginnings and are complicated to put in place and maintain. Hopefully this will become simpler in the future and it will be easy to farm out analysis tasks to a grid.

12. **analysis sequencer** - once analysis programs have been converted to servers they need to be orchestrated the same way the hardware needs to be sequenced. The best solution here is to use the same sequencer for doing control and analysis sequencing. This way the analysis is automatically integrated with the control and can be started as soon as the acquisition is finished. Refer to the sequencer section above for a list of possible sequencers.
    - ***OpenGenie*** - developed and used by ISIS. This is a sequencer for doing analysis on spallation and neutron source.
13. **data visualization** – the old age adage "a picture is worth a thousand words" is very true in science. For example a graph is often much easier to interpret than a table of numbers is. Trends can be seen immediately on a graph while a list of numbers need more mental processing in order to interpret them. Which data visualisation package to use? Here again there are as many choices as there are institutes just about. Some guidelines for choosing are:
    - ***ISAW*** - a free package written in Java for visualising 1D and 2D data sets for neutron scattering. Available from Argonne National Laboraties.
    - ***VTK*** – a 2D, 3D and 4D visualisation toolkit written in C++. It is very powerful and offers a large number of features. One of the best free packages around.
    - ***commercial*** – a number of commercial packages abound for visualising data like *Matlab*, *Igor*, and *IDL* to mention a few. Their advantage is they are usually professionally finished products, which perform well when used as stand-alone packages. Their disadvantage apart from the fact that they cost money is that they cannot be easily integrated into the local graphical user interface. Here the free open source toolkits are better because they can be easily integrated, modified and distributed for free.

The abstract layer

"Any problem in computer science can be solved by adding another level of indirection" Alan Kay.

How will the integrated graphical user interface interact with a GUM control system? The GUM theory says you need an integrated application to do so. The integrated application should be able to support multiple control systems. Because GUM does not prescribe how each of the building blocks should be implemented we need an abstraction layer which sits between the integrated application and the GUM control and analysis system. The abstraction layer is common for all control systems. Gumtree, or any framework which supports multiple control systems, will be interfaced to the abstract layer. Adapters will make the link between the abstract layer and individual control and analysis systems. The adapters can be implemented as local classes or as servers on the network. Which solution to adopt depends on how the abstract layer is implemented i.e. as a local interface or as a network proxy class.

What should be in the abstract layer? The following interfaces have been so far identified but the list is not complete. More work is needed to extend this list to cover all building bricks of GUM. The best way to do this is to implement the graphical user interface of GUM and interface to working control and analysis systems. This is exactly what Gumtree is doing.

Interfaces

One way of adding an abstraction layer which supports GUM is to define interfaces. The interfaces are binding contracts on which applications can be built. The advantage of this approach is that it does not prescribe how to implement the building blocks. Different implementations of building blocks can be supported as long as they support the common interface defined below. Due to space restrictions this section has been kept short. A real implementation needs many more methods to be defined. Anyone interested in collaborating on this topic is invited to join the Gumtree project and/or refer to the website.

ControlSystem

The control system interface abstracts the entire underlying control and analysis system. It can be browsed to return a description of what objects are supported by the system. The underlying control components are assumed to be objects. All objects share a basic common interface (e.g. name, type etc). Specific types are introduced to deal with common types of objects like motors, detectors etc. New types of objects can be introduced as long as their interface is well defined.

connect(id)	Connect to the control system with the given id
getDeviceList()	Browse the control system for a list of devices
getDeviceTypeList()	Get the list of device types in control system
getDeviceOfType()	Get a list of devices of a specific type

Sequencer

The sequencer executes commands. It should ideally be a server. I should support at least the following commands:

executeCommand()	Send a command to the sequencer to be executed
subscribeInterest()	Register interest in events from the sequencer
unsubscribeInterest()	Unregister interest in events from the sequencer

Scan

The scan iterates one or more parameters from an initial state to a final state and acquires data at each trigger. The trigger is defined by the type of scan e.g. time, position, external etc.

get/setActuators()	Get/Select actuators to scan (e.g. motors)
get/setSensors()	Get/Select sensors to acquire during scan (e.g. detectors)
executeScan1D()	Execute 1D scan
executeScan2D()	Execute 2D scan
executeScand3D()	Execute 3D scan

Motor

Motors are very common devices on beamlines. Their job is to move the various optical and sample elements into the right position for the experiment.

getMotorList()	Get list of motors
get/setMotorSpeed()	Get/Set motor speed
abort()	Abort motor movement
get/setMotorAcceleration()	Get/Set motor acceleration
get/setMotorPosition	Get/Set motor position
getMotorState()	Get motor state

moveMotorRelative()	Move motor to relative position
moveMotorAbsolute()	Move motor to absolute position

Detector

Detectors are the sensors of a beamline. Their job is to acquire the data.

get/setIntegration()	Get/Set integration
get/setSize()	Get/Set dimensions
startAcquisition()	Start acquisition
getState()	Get state

CommandLine

Command line is the terminal interface for the sequencer. It is the graphical or ascii window where the scientist types her commands to send them to the sequencer.

setStatus()	Set status of CLI
addStatus()	Add info to status line
clearHistory()	Clear history window
get/setOutput()	Get/Set text displayed in output window
get/setInput()	Get/Set text displayed in input window

Batch

Is a sequence of scans which execute in unattended mode.

new()	Start a new batch run
pause()	Pause the current batch run
abort()	Abort the current batch run
insertBatch()	Insert extra run(s) in the current batch
get/setBatch()	Get/Set the batch buffer
continue()	Continue a paused batch run
delete()	Delete runs from the current batch buffer
getScanNumber()	Get the scan number of the current batch run

DataVisualiser

newPlot()	Create a new plot window
addPlot()	Add a plot to the existing plot window
deletePlot()	Delete a plot from the plot window
get/setAxes()	Get/Set axes on plot
get/setColorMap()	Get/Set the color map
get/setSymbols()	Get/Set plot symbols
get/setColor()	Get/Set color of plot(s)

FileBrowser

openDirectory()	Open a directory

openFile()	Open a file
get/setFile()	Get/Set the file name
get/setFormat()	Get/Set the file format
get/setInfo()	Get/Set file info
get/setData()	Get/Set data

Analysis

startAnalysis()	Start analysis program
get/setData()	Get/Set data to be analysed
getState()	Get state of current analysis
saveData()	Save data

Grid

newGrid()	Create a new
get/setData()	Get/Put to/from data to grid
startAnalysis()	Start analysis program on grid
stopAnalysis()	Stop analysis program running on grid

Gumtree

Gumtree is a graphical user interface being developed at ANSTO for the new Replacement Research Reactor (RRR). Gumtree adheres to the principles of the GUM theory and is currently the only project known to the authors doing so. It is being designed from the ground up to support multiple control system. Although it is being developed for the neutron beamlines at the RRR it could be used at the Australian synchrotron in Melbourne, and for running lab experiments which can consist of only a single device.

Gumtree has chosen the Eclipse/Java framework as it base technology and is developing all modules as Eclipse plugins. ***The plugin architecture is ideal for developing many graphical components separately and making them available in one single application***. The Eclipse runtime engine discovers all the plugins and makes them available to the user. The architecture of plugins is well defined. Different institutes can developed plugins separately and they still work together. In addition to the plugin framework eclipse offers classes for implementing a wide range of graphical interfaces e.g. wizards, property sheets, html-like pages, context sensitive help, update management etc. It is a very powerful framework and the Gumtree team believes that by adopting Eclipse they have saved themselves a lot of work, jump started their application and can provide a polished finished product by levering the Eclipse technology. Gumtree is open to collaboration. It uses the abstract interfaces above to interface to other control systems. If anyone is interested in collaborating on Gumtree or applying it to their control system please contact the Gumtree team via the Sourceforge mailing list.[5] Gumtree is an open source project under the GPL licence hosted at Sourceforge.

Future perfect

Is there a future after GUM? Of course! GUM is only the next step. We see trends in graphical computing going to fully 3D graphical interfaces where even the terminal console windows are 3D. Croquet is one example of this kind of user interface. Croquet is also pioneering the notion of collaboration tools being

[5] mailto:gumtree-developers@sourceforge.net

part of the operating system. One of the best uses of computers is for doing simulations. We are convinced that simulating experiments before running them will be part of the future. Gumtree already contains a basic control system simulator. Other people are doing much better simulations. In the future this could be extended to include scientifically meaningful data simulation. In the future we might actually perfect computers to serve science instead of the other way around, as is still often the case today.

Conclusion

In this paper we have tried to identify what constitutes what we think is an ideal control and analysis system for doing science today. We have defined the Grand Unified Model for control and analysis systems. We have introduced a classification system for control systems based on existing philosophies and paradigms. We have made a proposal for a common set of interfaces for interfacing to control and analysis systems. This is work in progress and needs to be implemented on a working system before it can be considered as adequate. We propose that other programmer's interested in this approach join us on the Gumtree project, the first attempt at implementing a graphical user interface for a control and analysis system which adheres to the GUM principles. Gumtree is based on the Eclipse Java framework and is hosted at Sourceforge.

Links

1. Gumtree– http://www.sourceforge.net/projects/gumtree
2. Eclipse – http://www.eclipse.org
3. TANGO– http://www.esrf.fr/tango
4. TACO– http://www.esrf.fr/taco
5. EPICS – http://www.aps.anl.gov/epics
6. Croquet– http://www.croquet.org