A Real-Time *Mathematica* Database

A proposal for designing real-time processing *Mathematica* tools to be used in distributed control systems

by Kristina Kumbulla

August 2013

Introduction

Increasing the role of *Mathematica* in control systems

Motivation

Control theory is usually concerned with the analyzing and synthesizing of systems described by ordinary differential equations that often represent physical laws of motion. Large-scale control problems rely on computational tools. Few software tools enable users to write out such calculations in a timely manner, with *Mathematica* being one of them. The upcoming sections will present some of the ways *Mathematica* is being used in a controlled system and other ways that it can be used by proposing creation of tools that process and control large-scale control systems, often seen in the experimental industry. Such systems require the accumulation of large sensor or controller data. The focus is to increase the role of *Mathematica* in control systems used in the experimental industry, by proposing tools that automate large-data processing.

Experimental Physics and Industry Control Systems

This proposal does not aim the development of all tools needed to implement a control system software application. It rather focuses on utilizing some tools that have already been created. The proposal relies strongly on utilization of the Experimental Physics and Industry Control System (EPICS), a software environment used to develop and to implement distributed control systems to operate devices such as particle accelerators, telescopes and other large experiments.

EPICS is one of the most widely deployed control system infrastructures in the experimental physics community. The EPICS tool is designed to help develop systems which often feature large numbers of networked computers providing control and feedback. EPICS uses client/server and publish/subscribe techniques to communicate between various computers. One set of computers (the servers or input/output controllers), collect experimental and control data in real-time using the measurement instruments attached to it. This information is given to another set of computers (the clients) using the Channel

Access (CA) network protocol.

We propose to utilize EPICS, as an intermediate tool between the Input/Output Controller (IOC) hardware and higher level software, to create *Mathematica* tools that process data incoming from IOCs through the Channel Access protocol.

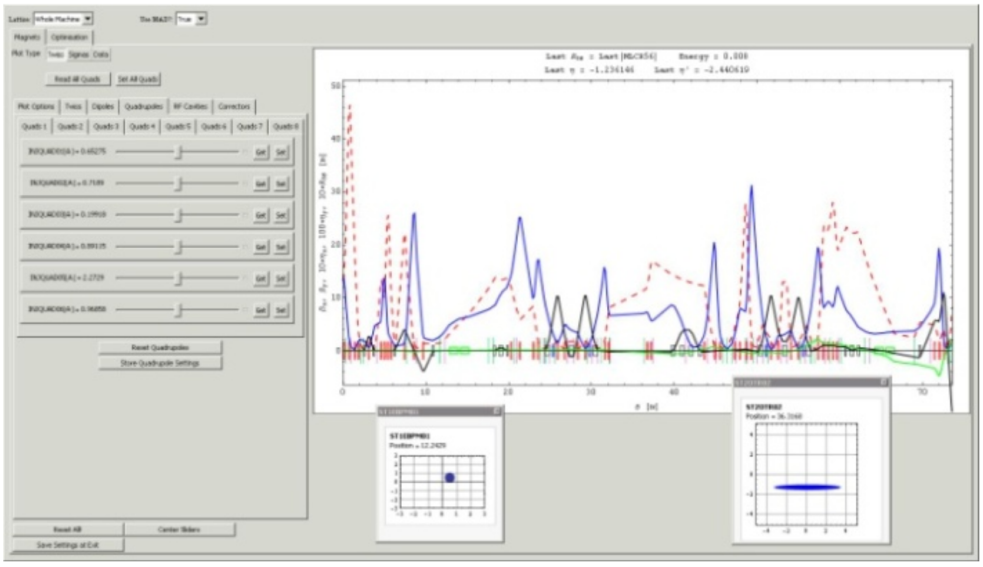
Literature Review

GUI-driven and procedural experimental work in a control system

The need for EPICS and high level programming languages

An article by Jones et al.[2] is a great reference for apprehending the overall design of a system running EPICS channel control system tools that uses high level software implementation for deeper data analysis. The article examined a set of high level software applications used in the ALICE accelerator control systems: *Mathematica* being one of them. Jones et al. emphasizes the need for high-level software to provide abstraction from the control system to perform tasks such as, degaussing, emmitance measurement, visualization and data acquisition, encouraging physicists to take the role of a software developer and build routines that manage such tasks on accelerators.

First, ALICE runs on EPICS and VxWorks. Among of the programming languages used to write software applications to control ALICE was *Mathematica*, Matlab and LabVIEW. Because of the ease of handling large datasets, MATLAB was the chosen language for a program that interfaced to the frame grabber and captured beam images for analysis in the ALICE accelerator. *Mathematica* was used to implement a standard model to analyze the machine in terms of standard lattice parameters. Analysis was performed either through the linear lattice code MLC, coded in *Mathematica*, or through the external MAD-8 code. Other applications built in *Mathematica* were image analysis routines that can automate the processing of images and its interaction with the control system, via EPICS. Overall, such routines utilized the advanced GUI building capabilities of *Mathematica*.



An exampe of a GUI used to drive controls of the ALICE accelerator through EPICS

The author concluded his argument with an emphasis in the advantages of EPICS control system enabling the use of so many different programming paradigms to create software that is relevant to the task at hand. Jones et al. presents us a very good example of the current usage of *Mathematica* in a GUI driven control system that uses the EPICS environment.

The need for real-time procedural automation

Even though the trend of using GUIs is clearly desirable for routine operations, meanwhile, for experimental work, is a serious impediment when used exclusively. The deficiency of an exclusively GUI driven control system is the correct motivation for Borland et al.[3] to present an article about few automated experiments in an EPICS environment using UNIX scripts and SDDS tools.

Borland et al. makes some really good points about the world of experimentation when confined by routine operations:

- Experimental work is often procedural, whereas GUI applications are user driven.
- Experimental work is by definition open-ended, and GUIs almost by definition confine the user to preconceived choices.
- GUI applications cannot be synchronized as the user is required to provide directions to each GUI.

One shouldn't need to run a GUI interface in order to use a particular algorithm (e.g., orbit correction). Borland et al. saw Unix scripting as a route towards implementing automated data processing sequences with each stage reusing the routines invoked in earlier stages.

This proposal aims to encourage usage of the procedural and algorithmic capabilities of *Mathematica* in an automated experimental IOC system such explained by Borland et al., to supplement GUIs such presented in the article by Jones et al. The proposal aims to make *Mathematica* the preferred tool among scientists that need automated processing of large data acquisitions in IOC systems that use Channel Access protocol.

Documented needs for Channel Access protocol in *Mathematica*

Any programming language or tool that supports Channel Access protocol has the capability to communicate with EPICS.
At Argonne National Laboratory Ben-Chin Cha and Bob Daly have developed a *Mathematica* package[10], which consists of a special set of channel access functions, to provide the *Mathematica* users with easy and flexible access of channel information across the IOC networks. DevMath, an external *Mathematica* link program for Device Access Library, was also available for users at Argonne National Laboratory. DevMath could interface with CaMath. This tool is not supported any more by the developers.

In any case, CaMath provided operational, monitoring and general debugging functions on one or more channels. For a brief idea, such functions included but not limitted to:

- query numerical value of a channel
- add or remove the value
- set/unset channels to be monitored

- set numerical value of a channel
- get most current monitor status of the specified channel from IOC

Then, one operational scenario that could be performed with CaMath and *Mathematica* would be:
-- get numerical value of current orbit from channel “ORBIT”;
-- set the orbit status of the channel for monitoring;
-- when orbit does not match its predicted course, run orbit correction algorithm;
-- set numerical value of the “ORBIT” channel, to typically control magnet movement.
Many aspects of this experimental system are highly volatile, and unable to catch up with the user demands, so, such a scenario should be automated. To automate this event with existing tools, the *Mathematica* ‘Dynamic’ functionality can be used. However, Dynamic is designed to collect real time data for a value that is on-screen. For a large-scale control system, the automation of thousands of processes is requested. Such processes are often requested to be automated in certain periodic or event scans as well as requested to be halt and re-animated at any time. Output of some event updating data may be used as input for periodically automated data. Proper scheduling algorithms to minimize system over-loads and overheads are necessary. While any *Mathematica* user may be capable of creating a Channel Access package with Mathematica, they would be less than able to provide synchronization and effective automation of large data capturing/processing/archiving that can update from minutes to milliseconds.

EPICS and its architecture

The understanding of the EPICS architecture will supplement the reader with a better insight of the *Mathematica* tools proposed for implementation.

The Experimental Physics and Industrial Control System (EPICS) is,

- an architecture for building scalable control systems
- a collection of code and documentation
- a collaboration of major scientific laboratories and industry

EPICS is primarily the work of the Accelerator Technology group at Los Alamos National Lab and the Advanced Photon Source at Argonne National Lab, now with more than 70 participants from laboratories, universities to industrial facilities. These sites include physics accelerators and detectors, telescopes and various industrial processes.

EPICS is distributed in source form, to ensure that every site has full control of its future locally. Current EPICS systems range in scale from single VME crate/single work-station sites with a few hundred “channels”, up to 175 VME crate/30 work-station sites with 30,000 channels.

At its core is the concept of a ‘process variable’, which presents either an input (e.g. a readout from a digital input) or an output (e.g., a reference setpoint for an analog output).

Physical layer

EPICS is fully distributed, thus it requires no central device or software entity at any layer. EPICS compromises three physical layers and five software layers. The physical front-end layer, referred as the Input/Output Controller is typically build from VME/VXI hardware crates, CPU boards, and I/O boards etc. The physical back-end layer is implemented on popular workstations from Sun, H-P, and

others, running UNIX; or on PC hardware running Windows NT or Linux. These layers are connected by the network layer, which is any combination of medias (Ethernet, FDDI, etc) and repeaters and bridges supporting the TCP/IP or UDP/IP Internet protocol.

Client layer

The client layer usually runs in the workstation/PC physical layer and represents the top software layer.Typical generic clients are operator control screens, alarm panels and data archive/retrieval tools. Other generic clients can be *Mathematica*, Display Manager, TCL/Tk graphical scripting language, etc. A very useful client is the Sequencer which uses a high-level language called the State Notation Language to implement cooperating finite-state machine.

EPICS clients are very high in performance. For example operator screens with 1000 objects are brought up in less than one second and can update dynamically about 5000 objects/sec. The alarm panel can react to 1000 changing alarm conditions/sec. The operator screens provide both synoptic displays, tabular data, simulated meters, buttons and sliders.

Channel Access layer

The second software layer connects all clients with all servers. Channel Access (CA), the backbone of EPICS, hides all the details of the TCP/IP network from both clients and servers. CA mediates different data representations (ASCII, floating types, etc) and allows high performance of data transactions between clients and servers. In EPICS, every client and every server makes connections with no ‘relay’ entities, so there are no bottlenecks beyond the physical limits of the medium. CA also uses a technique called notify by exception or callback, also known as publish/subscribe. Once a client has expressed an interest in certain data to a server, the server notifies the client only when the data changes. All data carry time-stamps, validation information based on quality of connection.

Server layer

The third software layer is the server layer. The fundamental server is the channel access server that runs on the target machine. This server cooperates with all channel access clients to implement the callback and the necessary synchronization mechanisms to be sent to the database layer. Although channel access clients are typically independent host programs, the channel access server is a unique EPICS task: just one copy of it runs on each network.

Database layer

The database layer is the heart of EPICS. Using a host tool, the database is described in terms of function-block objects called records. About 50 record types exists for performing such chores as analog input and output, binary input and output, building histograms, performing calculations, driving timing hardware and more. The fundamental entity in the EPICS database is the channel. A channel is a path to one or more attributes of a record in the database. Besides its name, a record has a value and perhaps other attributes such as units, maximums/minimums, etc. The record name is the only way that CA connects a client to a server.

Now, record activity initiates in many ways: from I/O interrupts, to software events generated by clients, or periodic scan rates. Data can flow from the hardware level to the software level or vice-versa. Most record parameters can be dynamically updated by channel access clients. Full customization is possi-

ble by defining new record types. New EPICS implementations require toolbox configuration rather than C programming. All of the EPICS database layer is coded in C/C++. Linkages between records are not restricted to one IOC, but can span multiple IOCs.

Device layer

The fifth bottom layer of the software is the device driver layer. Large rapidly growing list of drivers has been written for many popular VME, serial devices and Industry Packs. Drivers are written in C.

Procedure

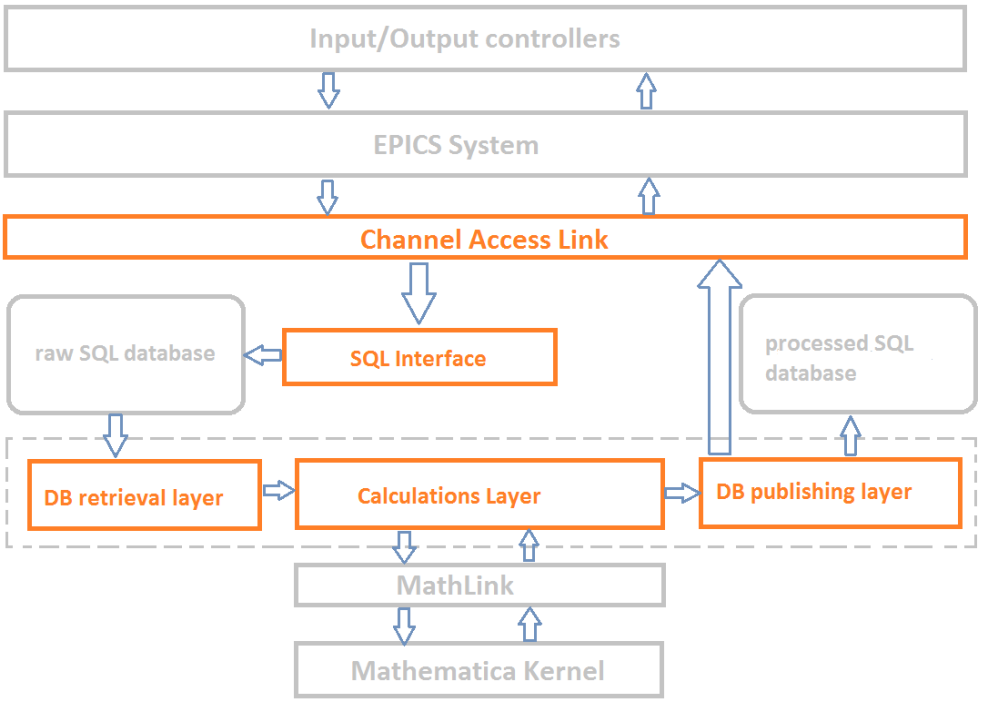
Implementation of Channel Access Protocol and *Mathematica* database

Introduction

This part of the document proposes the designing and creation of *Mathematica* tools that interface with EPICS to be used in experimental control systems. Constructing a *Mathematica* set of tools that is capable of automatically retrieving, calculating and updating data in a channel controlled system can be modulerized if each tool's main functionality is designed as a layer of operations. So then the following sections propose a layered approach towards incorporating *Mathematica* tools that as a whole can automate a user defined process. Such tools shall be able to perform the following tasks in this automated logical order:

- Retrieve raw record data from IOCs.
- Store raw data in a relational database along with a network 'quality' specified on a threshold criteria defined by the difference of the stamped time and arrived time.
- Archive raw data.
- Process raw data on a user pre-defined algorithm and user pre-defined order, which matters in periodically scanned channels, by retrieving data from the raw database
- Store processed data in a new relational database; assign each data the inherited network quality from the raw database and inherited scan type.
- Archive processed data.
- Push record data updates to IOCs (through EPICS) at a frequency as defined by scan type

A 'database utility manager' shall enable users to configure channels for data processing and subscribe channel data for a scan type: periodic or event based.



The proposed layered real-time Mathematica database

The unprocessed database

Maintaining a built-in real-time unprocessed or raw 'Channel Access (CA) *Mathematica* database' that replicates the EPICS database is necessary for optimizing real-time processing by reducing traffic bottlenecks that could potentially lead to large number of transactions missing their timely deadlines. Also, this design provides users with a *Mathematica* accessible large data storage, as well as enables them to reorganize the dataset to be used for processing in *Mathematica* in their preferred fashion, such as lists or sparse arrays. To supplement operators of EPICS control systems with a real-time raw CA *Mathematica* database that mirrors the EPICS database, we need to provide *Mathematica* Channel Access protocol functionalities. Such functions would be able to grab data from the EPICS IOCs and store it in tables of the raw CA *Mathematica* database in an automated fashion.

Therefore, I propose to construct a client layer that consists of a Channel Access interface to execute functions on a selection of channel records of IOCs. I also propose to use a database to store data records. The user should be able to configure which databases, tables or channel records should be mirroring EPICS database, tables and records, preferably through a GUI database utility manager. The idea is that this new database shall contain the same data information as that of the EPICS database, as well as a network client layer that updates the raw CA *Mathematica* database on a periodic or prioritized event scanning basis. So then the raw CA *Mathematica* database is not just a storage but also a network interface. This real-time database should be self maintainable and should be able to communicate the data to the Kernel through the 'calculations layer' only when values are set for processing. The data processing will be discussed in the next section.

Interface to an open relational database is more readily supported by *Mathematica*. The following *Mathematica* scripts demonstrate some operations that can be done with *DatabaseLink* on a previously created database:

```
Needs["DatabaseLink`"];
```

```
conn = OpenSQLConnection["demo"]
```

```
SQLConnection[demo, 1, Open, TransactionIsolationLevel → ReadCommitted]
```

```
SQLTables[conn]
```

```
{SQLTable[RTTEST, TableType → TABLE], SQLTable[SAMPLETABLE1, TableType → TABLE]}
```

```
data = SQLSelect[conn, "SAMPLETABLE1"]
```

```
{{1, Test1}, {2, Test2}, {2, Test3}}
```

Now that we have a *Mathematica* list, we can use the *Mathematica* functions and patterns to process the data to obtain desired outputs. *DatabaseLink* allows one to also use raw SQL commands:

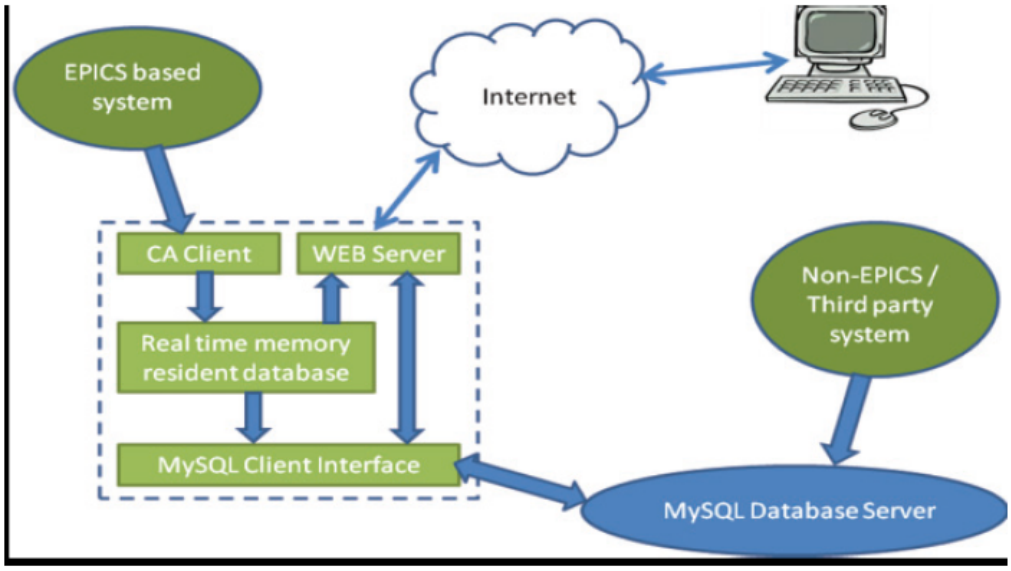
```
SQLExecute[conn, "SELECT * FROM SAMPLETABLE1"]
```

```
{{1, Test1}, {2, Test2}, {2, Test3}}
```

```
CloseSQLConnection[conn];
```

I propose to research ways to incorporate this already existing functionality to communicate with the CA layer in an automated fashion to store data in the CA *Mathematica* database.

Other functionalities that can be provided is archiving. Archiving timed experimental data is neccessary for analyzing processes, avoiding redundant experimental methods, and pin-pointing potential past errors of the system. The performance of a particle accelerator is measured by its beam stability which is achieved through fine tuning of the system parameters. In an article by Roy et al.[5] the requirement of archiving accelerator system parameters is used as a motivation to create an EPICS MySQL Archiver. The architecture of this system uses very similar structure to our proposed functionality of the CA *Mathematica* database.



The system design of archiving EPICS database as presented in an article by Roy et al.

Using this architecture as a motivation to design the CA *Mathematica* database, I am inclined towards using a relational database model to store real-time and archived data; however, whether we use a

database functionality that already exists or we create one from scratch shouldn't be decided at this time.

The processed database and the calculations layer

In an article by Borland et al. an example of a time-constrained data processing is presented. I will use that example to walk through the steps of an implementation of another layer that will be proposed.

Beam motion characterization and source location is important due to the tight tolerance of the Advanced Photon Source (APS) for the electron beam oscillatory motion. So rather manually screening every power supply and girder (source of motion are typically magnet power supply ripple and magnet girder excitation) one can obtain information from the beam observations and experimentation, and use that information to make a calculated decision to modify these supplies. Then, first, raw beam position monitor (BPM) data is gathered. Such an event would be provided by the CA link. Now, the data from the beam histories needs to be automatically processed in several ways. To perform such an event in *Mathematica*, I propose to create a 'calculations layer'. Such a layer should perform algorithmic functions on selected raw data records that reside in the raw *Mathematica* database. This layer is an interface to allow the user to enter a set of algorithms applicable to a set of records that will be processed by the *Mathematica* Kernel. This layer should be 'bound' by a 'database retrieval layer' and a 'database publishing layer'. The primary function of the database retrieval layer is to grab data from the raw *Mathematica* database automated from a user-defined periodic or event scan basis. This data shall be handed over to the calculations layer, where user-defined algorithms are peformed on some user defined input records received from the raw *Mathematica* database. The database publishing layer will store the processed data in the processed *Mathematica* database configured by the user and push the processed data over to the IOCs on a user-defined periodic or event scan basis using the CA link. The calculation layer will be capable of not only processing data but also of automatically controlling and displaying it through GUI capabilities of *Mathematica* at real-time. A processed *Mathematica* database archive can be designed in the same fashion as the one discussed in 'the Unprocessed Database' section. The calculations layer shall provide options to retrieve archived data for real-time processing needed in rolling averages or parameter estimation in a regression model. So then, what would be achievable with this tool? In the example of the beam motion characteriztion calculation, with such *Mathematica* tool, an operator would be able to configure automated algorithms on data from slow beam histories:

- **calculate standard deviation** of each retrieved BPM history which is typically organized in a matrix and each column corresponds to one set of BPM of one specific time. In the proposed functionality, the BPM history data would be residing in an archive and will be available for calculating the standard deviation of each BPM. Store result in the processed *Mathematica* database.
- **remove slowly-varying components**, and create a new BPM parameter whose value is equal to the standard deviation of the column; store the results in the processed *Mathematica* database.
- **plot the quantity**: (standard deviation)/(square root of the horizontal beta function); these plots have the capability of animating real time data. Archive data will also be used to plot previous data to demonstrate a time varying quantity.
- **Fourier analyze the BPM raw data** and output a spectrograph plot of the power spectral density (PSD). The PSD within a particular frequency band is summed, and plotted as a function of BPM. When the power in the band from 5.5-7.5 Hz is normalized to 1/(square root of the horizontal beta function), the result reveals whether power supply sources need to be investigated or not.

All steps bulleted above will be processed and displayed to the user, while he or she observes the animated results with the ability to make a fast decision in manipulating the controlled system, in this

case, a decision on whether power supply sources need to be modified or left alone.

The specific design of the calculations layer is left for open discussion. Deciding on how to establish that communication should be left for researching, yet I will attempt to provide one direction of thought about the GUI that allows the users to configure algorithms that will process input and output records. One could design the configuration GUI of the calculations layer to behave like an 'un-evaluatable' (but configurable) *Mathematica* notebook, where 'input records' are channel records residing in the raw *Mathematica* database which are retrieved by the database retrieval layer and 'output records' are channel records residing in the processed *Mathematica* database that update the IOCs through the datatabase publishing layer. This GUI could even be accessible through the already existing Front End. The user shall be able to write *Mathematica* functions to configure algorithms to be applied on the raw input records. Input and output channel name aliases should be unique for ease of identification of IOCs.The user should be enabled to set the calculation layer in motion in a parallel or sequential order at a "click of a button". Only then, should this form of *Mathematica* notebook be evaluated at a periodic or event scan basis synchronized with the database retrieval layer and the database publishing layer.

Additional tools and functionalities

Complexity management

Incorporating such a complex system requires management and user assistance. For example, running multiple *Mathematica* databases over the same network can cause conflicts in concurrent data processing. While a careful user may be capable of preventing such events from not happening, there is room for error. Some tools that could be incorporated to maintain a safely operable and user friendly channel access database are:

- ***Mathematica* Network Manager**: to handle conflicts in database updating from multiple *Mathematica* tools running in one network, I propose to construct another layer that handles communications among all *Mathematica* databases running on different workstations over the same network.This tool shall only exist if multiple instances of the database application are allowed to run over a network.
- **Database Utility Manager**: to allow users to set and manage channel information to be retrieved by the raw *Mathematica* database and channel information to be updated to the processed *Mathematica* database and IOCs. The user shall also be enabled to subscribe/unsubscribe channels for scan events and types. Subscribing and Unsubscribing channels from scanning shall also be allowed to be performed without needing to halt the automated data gathering and processing event. This later functionality could be provided in the calculations layer.
- **Algorithmic Dependency Configurator**: this tool would be incorporated within the *Mathematica* calculations layer to enable the user to determine a sequential order in which certain algorithms need to be performed, neccessary when certain inputs are dependent on certain outputs, as well as to enable the user to increase real-time performance through the parallelization of functions over multiple Kernels, in cases where there is no input-output dependency. For systems with 30,000+ channels such a tool shall be GUI driven. A possible GUI: a user modifiable hierarchical tree of nodes, with each node type identified by an input, output or a function where the hierarchy identifies with the sequential order of functional performance. The function node, when accessed for editing, shall launch an 'un-evaluatable' *Mathematica* notebook (if this form of configuration is chosen as a design).
- **Real-Time Database Scheduler**: this functionality resides in the underlying functions of data retrieving and updating. Choosing the proper scheduling algorithm is important for achieving the best performance. A typical real-time scheduler assigns a higher priority to a transaction that has an

earlier deadline. Other algorithms that handle timely transactions and system over-load is the Highest Value First (HVF) algorithm. In case that multiple transactions are executed concurrently, a concurrency control mechanism maintains data consistency by scheduling shared data access. Choosing a good concurrency control algorithm is important to minimize significant overhead, often resulting from lock-based concurrency control algorithms. A very interesting paper by Yoshiaki Kiriha[11] describes some of his experiences with real-time databases and a proposal that reduces overload and overhead during data transaction scheduling.

User-friendly features

- **Experimental Physics Package**: built-in *Mathematica* functions that model some commonly used algorithmic processes in Experimental Physics or, generally, large-scale control problems. A major source of large-scale control problems is the modeling of the physical process by PDEs. Techniques for solving large-scale control problems often involve discretization of the partial derivatives, reduced to problems that require solutions using modern sparse linear algebra techniques. *Mathematica* is very much capable of providing operations on sparse matrices, and built-in packages that analyze and synthesize large-scale control systems would be a nice addition.
- **Archive Simulator**: a functionality within the calculations layer that retrieves archived data, the start and end time of which is configured by the user, to run a simulation on a past experiment, necessary when the user wants to observe charts or processes to analyze data from the experiment that was carried out. Provided the correct configuration parameters, such functionality would also enable the user to test hypothetical scenarios through user-created archive data, hence enabling them to test configured *Mathematica* algorithms.
- **Display Navigator**: a user friendly display navigator to enable operators to navigate through displays, such as plots, graphs, and interactive dialogs as well as a tool that halts processing of certain images when not visible to the user. While the user has arranged for displays to appear on screen through the calculations layer, not all displays are relevant to be observed at all times. Adding to that also the processing time that is being used for an unwanted functionality, this tool would increase performance and user friendliness.

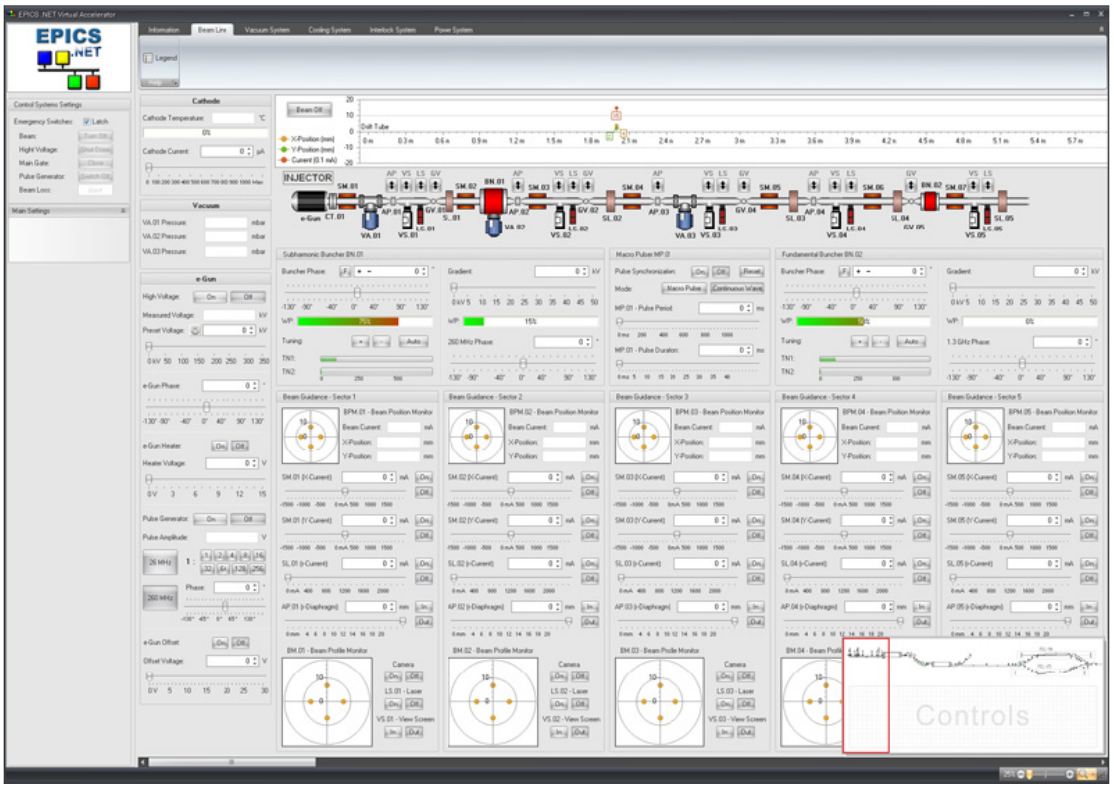
Decisions of additional tools can be made upon communication with operators of the Experimental Industry and an overall research of resources and time management.

Resources and Timeline

Materials/tools needed to carry out the project

Users of the EPICS environment are inclined to use a Linux-based operating system to run the high-level software applications; few also choose Windows. Ideally, both operating systems should be available while implementing the proposed tools. Other operating systems used in experimental control systems could be researched. The EPICS base environment should be downloaded and compiled on the workstations where *Mathematica* code will be developed and tested.

In experimental control systems, the target machine almost always runs on VxWorks real-time operating system. Using a target machine that runs on a VxWorks simulator would be preferable. However, using a Linux operating system is another option. This target machine should be able to simulate input/output controller data. Another feasible testing environment would be an EPICS simulator. An EPICS .NET library and simulator has been created for the Turkish Accelerator Center by Teoman Soygul[12].



User interface that interacts with the simulator or the control system, created by Teoman Soygul[13]

Simulation of a multiple input multiple output (MIMO) system is neccessary for simulating the real-time data flow in an IOCS. John Fitzpatrick[14] presents a paper on a completed simulation project of a MIMO wireless system written in C++ and Matlab. Such materials may serve as point of reference for further research of tools to be selected for a portable testing environment. The ultimate acceptance testing should, ideally, be arranged to be run on an accelerator. Argonne National Laboratory could be a helping resource in this process.

MathLink as well as the FE design could be used to implement the calculations layer. Other tools include C/C++ compilers, a good number of *Mathematica* Kernels to test parallel performance of the calculations layer, network clustered workstations to simulate testing experimental environment, etc.

Data to be collected to carry out the project

Communication with operators and physicists of the experimental industry could be crucial in decisions about classification of tools that can be built. They can also be a good reference for knowledge of other software tools widely used in software development in experimental control systems.

Understanding the EPICS base design is important for building the Channel Access link. Research work in real-time applications is neccessary to understand the scalability of the design that we will need to implement. Reusing already implemented designs should be assessed. For example, utilizing the Dynamic design, which uses Earliest Deadline First (EDF) algorithm, could be a path to look into while achieving a design for the real-time CA processed database. However, the EDF algorithm will not be sufficient for a system with hundreds of concurrently updating dynamic objects. Other scheduling algorithms could be used on an upper interface before the Dynamic is used. Overall, aiming

to achieve the ideal real-time responsiveness and reliability should have priority over reusability. Appropriate research should be carried out.

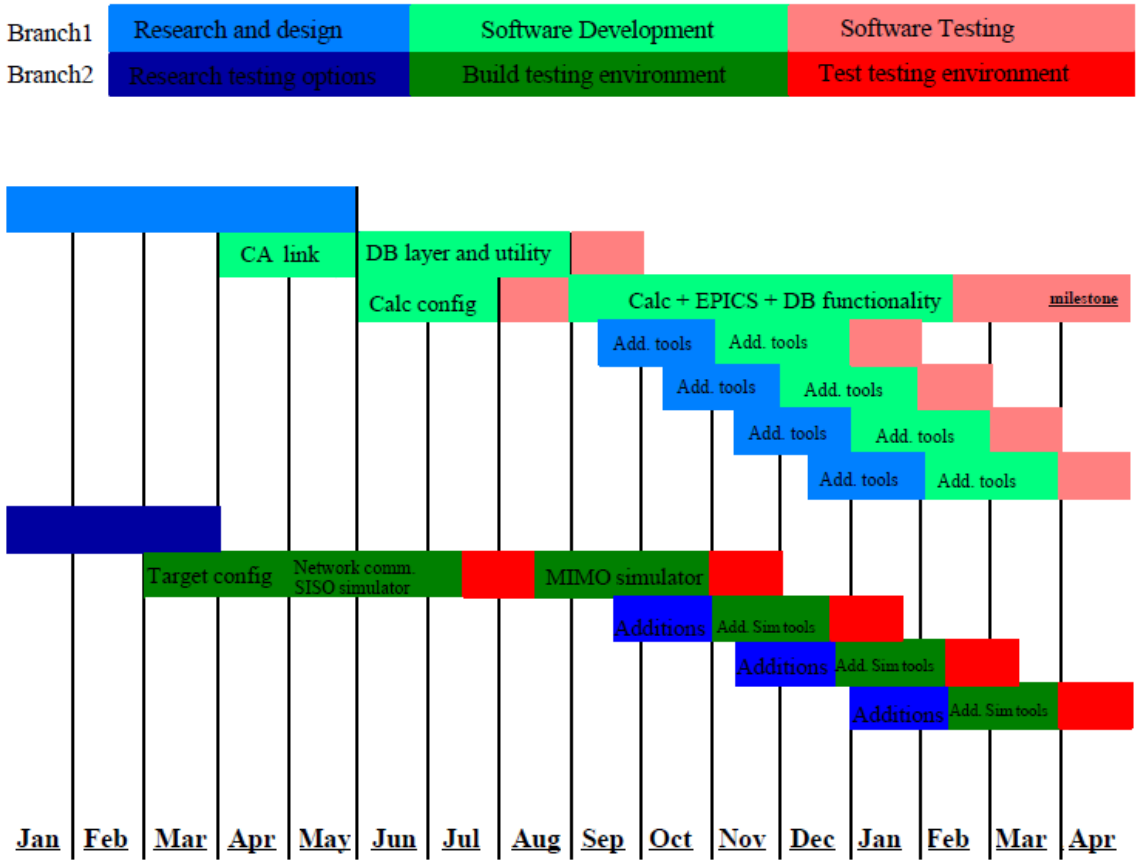
Another goal to keep in mind when designing and implementing each database layer is the possibility of their reusability over a number of other network protocols for possible future projects.

The programming language should also be assessed. With existing SQL database functions, and the Dynamic functionality, I can see both C++ and *Mathematica* being chosen for programming the layers.

Due to the nature of the project a documented design should be preferred to enable a team to work coherently.

Time frame

Two main branches of work can be performed simultaneously: the development of the to-be-released software and development of the testing environment. The testing environment should simulate the input/output scale and frequency of an accelerator. Ideally, the acceptance testing should be run on an accelerator, however, it might not be possible. Therefore, the testing environment will be the main calibre of the quality of the application to be build. Every milestone of the developed software shall be reached when successfully tested against the target machine. Therefore, an agile development of both work branches is required, where a testing capability is functioning per developed software functionality. Parts of the target machine development cannot be agile, however, it is expected that the hardware research and design timeline will be shorter than that of the application's timeline, so the target machine hardware will be set up -- and a simple SISO simulator shall be functioning -- before the testing of the CA link and DB retrieving/updating layer. The chart below presents an estimated timeline of the project.



Team size

2-3 Software Developers -- will learn to use EPICS system, develop software applications, develop network tools, test against the target machine. They may need to research tools and algorithms to be deployed in development of the proposed *Mathematica* tools. Knowledge of C++ or/and *Mathematica* may be needed.
1-2 Electrical/Hardware Engineers -- will be creating the testing environment, which could consist of any or all options listed in the budget table below. They may be required to develop a MIMO simulator, VxWorks simulator, or/and put together a target machine. They should, preferrably, be familiar with problems in control systems and assist with development of the Experimental Physics package, which envolves control system problems.

Budget

Materials and tools selected to build a portable testing environment should be properly researched. The table below presents three options of a target machine system that behaves as an Input/Output Control system to be used for testing. Pricing of materials do not reflect actual product pricing but rather an estimate of what I expect and look for when setting up the environment. Feasibility and complexity of each option does not neccessarily contrast or compare with the pricing.

Materials or tools	*Price*
Option 1: VxWorks simulator	
Target machine: any machine that can run VxWorks OS	$200-3000
I/O simulating tools/system	$0-1000
EPICS base system	$0
VxWorks Operating System	$1000-300
Networking tools (to troubleshoot and communicate)	$0-1000
Total	$1200-8000
Option 2: Linux simulator	
Target machine: any machine that can run Linux OS	$200-3000
I/O simulating tools/system	$0-1000
EPICS base system	$0
Linux Operating System	$0
Networking tools (to troubleshoot and communicate)	$0-1000
Total	$200-5000
Option 3: EPICS simulator	
EPICS base system	$0
Create I/O simulating system	$0
Total	$0

A future prospect

Mathematica image and signal processing algorithms will have a new meaning when supplemented with a real-time and archiving database. For example, image processing algorithms can be run on video frames accumulated in a frame grabbing database through a network communication to automate image analysis observed at real-time. Meanwhile, signal processing and data mining algorithms can be run on say real-time and archived analog radio signals to scan and piece together relevant information over time to solve a real-time problem. Overall, a good foundation of a real-time database system opens some doors for future projects. Utilizing a layered design to implement this project enables us with the flexibility to plug and play with other types of network layers. Most real time processes observed in environments such as power plant monitoring and controlling, robotics surgery, accelerator physics or mobile laboratories such as the Curosity rover run on automated systems that control large data processing over a network. This project is a good start in concurring such markets.

References

[1] EPICS - Experimental Physics and Industrial Control System. www.aps.anl.gov/epics/

[2] J. K. Jones et al. High-Level Alice Software Development. In Proceedings of PAC09, Vancouver, BC, Canada

http://accelconf.web.cern.ch/accelconf/pac2009/papers/fr5rep028.pdf
[3] M. Borland, L. Emery, N. Sereno. Doing Accelerator Physics Using SDDS, UNIX and EPICS. Argonne National Laboratory. ://www-bd.fnal.gov/icalepcs/abstracts/PDF/th3be.pdf
[4] K. Zagar et al. Evolution of the EPICS Channel Access protocol. In Proceedings of ICALEPCS2009, Kobe, Japan.
[5] Anindya Roy et al. EPICS MySQL Archiver - Integration between EPICS and MySQL http://accelconf.web.cern.ch/accelconf/pcapac2012/talks/thcb02_talk.pdf
[6] Stephen A. Lewis. Overview of the Experimental Physics and Industrial Control System: EPICS. Lawrence Berkeley National Laboratory. http://csg.lbl.gov/EPICS/OverView.pdf
[7] Martin R. Kraimer et al. EPICS Application Developer's Guide. EPICS Base Release 3.14.11. http://www.aps.anl.gov/epics/base/R3-14/11-docs/AppDevGuide.pdf
[8] DatabaseLink User Guide. http://reference.wolfram.com/mathematica/DatabaseLink/tutorial/Overview.html
[9] Peter Benner. Solving Large-Scale Control Problems. Sparsity and parallel algorithms: two approaches to beat the curse of dimensionality. Numerical Awareness in Control. http://www-user.tu-chemnitz.de/~benner/pub/B_CSM_2004.pdf
[10] Ben-Chin Cha and Bob Daly. CaMath User's Guide. http://www.aps.anl.gov/epics/EpicsDocumentation/ExtensionsManuals/CaMath/CaMath.html
[11] Yoshiaki Kiriha. Real-time Database Experiences in Network Management Applications. NEC Corporation. C&C Research Laboratories. Stanford University. ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/95/1555/CS- TR-95-1555.pdf
[12] Teoman Soygul. EPICS.NET, Virtual Accelerator. GitHub https://github.com/soygul/EPICS.NET/blob/master/Virtual%20Accelerator/MainForm.cs
[13] Teoman Soygul. EPICS.NET, Virtual Accelerator. CodePlex http://www.soygul.com/projects/epics/
[14] John Fitzpatrick. Simulation of a Multiple Input Multiple Output (MIMO) wireless system. Dublin City University. http://performance.ucd.ie/jfitzpatrick/files/MIMO%20Simulator%20FYP.pdf