An Overview of IRMIS

Introduction

This document describes an ambitious collaborative effort for defining and developing a relational database and associated applications to comprehensively document the large and complex EPICS-based control systems of today's accelerators. Although the wording of this document and the near term applications that are discussed suggest that the approach is limited to documenting control system components, work is underway to extend the approach to all components of the entire accelerator (e.g. mechanical, vacuum, power supplies, etc).

Description/Purpose

A typical accelerator control system is a complex integration of sensors, modules, instruments, cables, controllers, software, process variables, and ancillary application programs all working together in unison. A change to any element typically requires a coordinated change in other elements (e.g. adding an input module also requires adding field wiring and databases). Likewise, failure of an element will cause other related elements to not operate properly (e.g. a power supply failure can affect numerous devices).

Typical documentation schemes utilizing "revision controlled drawings" are quite limited and not practical for managing such a complex system. Defining the relationships between the thousands of elements is much more suited to a relational database implementation of documentation rather than a drawing-based method. The *Integrated Relational Model of Installed Systems* (IRMIS) is a project that undertakes this formidable challenge. The vision is simple; define every entity of an accelerator control system and its relationships with other entities. Once defined, numerous *views* of these relationships can be implemented that will answer the routine questions posed of a system during operation and maintenance. Some typical examples include:

- What process variables are associated with this device?
- What process variables were added, changed, or removed since the last run?
- Where does the other end of this cable go?
- What components do all of these non-functioning devices have in common?
- Which module type in this system has the worst reliability history?
- How many devices of a particular model number are installed?
- Where are all the devices of a particular model number installed?
- What application software will be affected if this device is removed?
- What equipment will be affected when this breaker is locked-out?

For physical components, there are three key relationships between devices that must be defined to answer the above questions:

- Where does each device get its power? [*the Power hierarchy*]
- What does each device physically plug into? [*the Housing hierarchy*]
- How does this device contribute to the passing of information through the Control System? [*the Control hierarchy*]

Figure 1 illustrates these three hierarchies for a typical rack mount GPIB instrument. The significant amount of information that can be mined from these three hierarchies is quite evident.

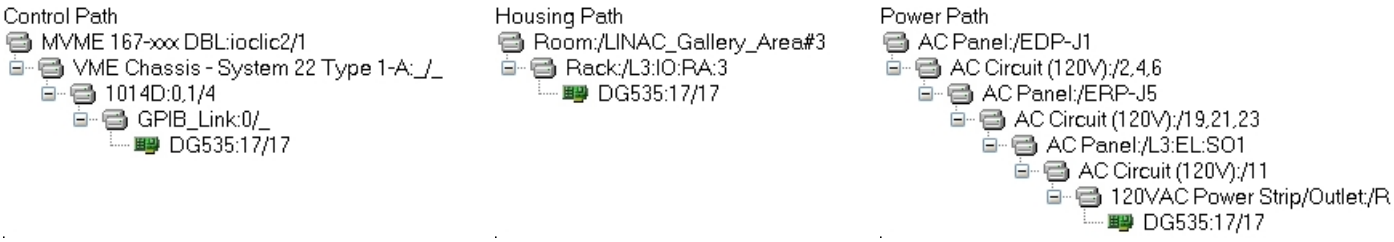

Figure 1

For non-physical entities (an "IOC", PV names, software applications, device support modules, etc), other appropriate relationships are defined so they can be associated with the corresponding physical components.

Implementation Approach – Descriptive vs. Prescriptive

IRMIS is a documentation tool, not a design tool. The information in IRMIS *describes* what is currently *installed* in the control system; it does not *prescribe* what is *to be installed*. This allows each facility to choose the design, programming and configuration tools most suitable to them yet still use IRMIS to understand the currently installed system.

IRMIS::Base and IRMIS::Extensions

It is recognized that each facility will prefer certain ancillary information to be stored in the database that is unique to that facility's requirements. Therefore, the implementation separates requirements that are common to any facility and those that are unique to a particular facility. The term *IRMIS* will be used to refer to the entire implementation of a site's relational database that describes the installed systems. The term *IRMIS::Base* will refer to the tables and tools that are designed to be site independent and beneficial for any EPICS-based control system.

The exact structure of IRMIS::Base versus site-specific tables as shown in Figure 2 is still evolving. The remainder of this document will not distinguish between the two, it will simply describe our vision of a comprehensive IRMIS tool.

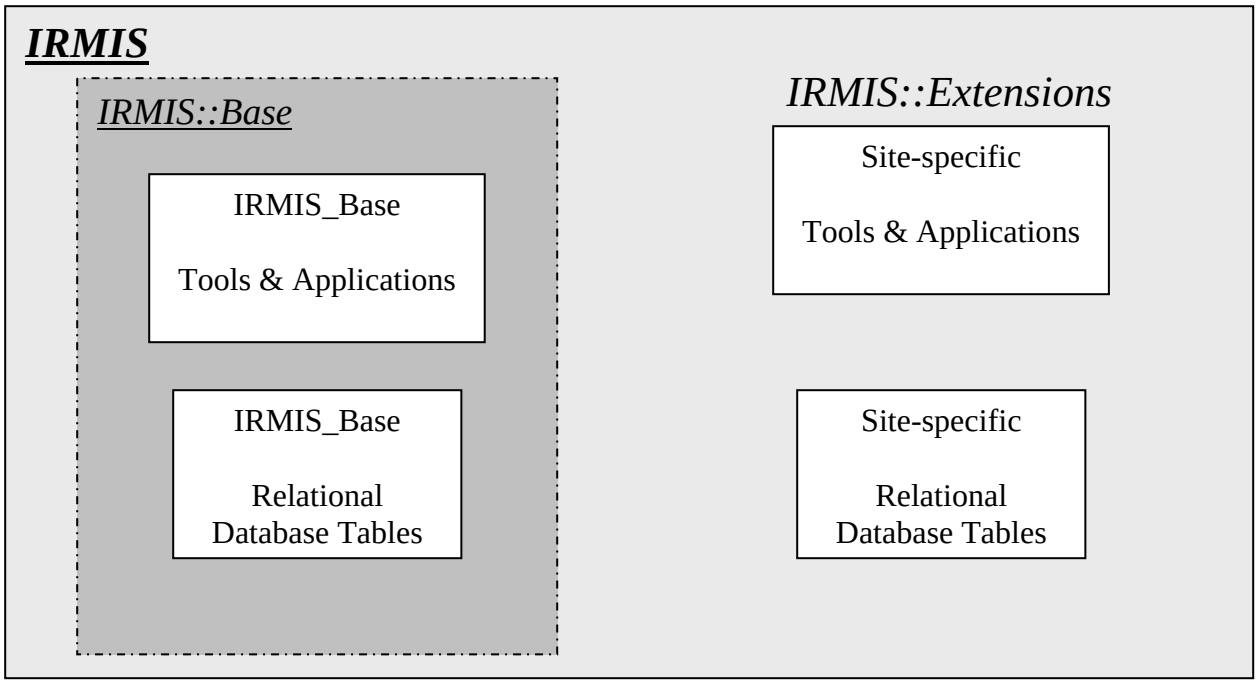

Figure 2. IRMIS Base and Extensions Structure

IRMIS – Integrated Relational Model of Installed Systems

Listed below is an overview of the information viewed as important to fully document an EPICS-based control system and therefore accommodated in IRMIS. The varying requirements are grouped by schema, where a schema is made up of a set of database tables and defined relationships.

Process Variable Database

A key attribute of an EPICS control system is the widely distributed real-time database of software components that reside in numerous CA server instantiations throughout the facility. The Entity Relation Diagram describing the fundamental EPICS record/field structure is shown in Figure 3. An EPICS record of a given name and type is loaded into a specific IOC and has a set of fields -- name/values pairs that are defined in a .db record instance file. The record type is considered to be an attribute of a given record, i.e. there are no type specific structures in the software relational database related to the record type. The record structure is defined in an external database definition (.dbd) file.

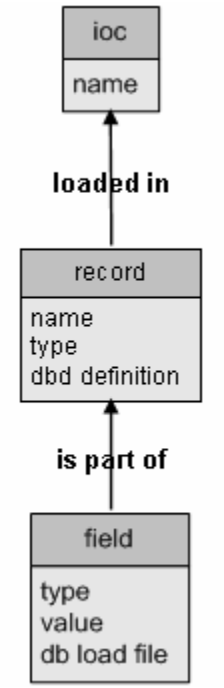

Figure 3. EPICS real-time software component Entity Relation Diagram
The physical implementation of this ERD is shown in the Process Variable Schema (Figure 4)

The majority of CA servers are those running iocCore, but other CA server applications must also be accommodated. One of the most fundamental needs for operating and maintaining an EPICS-based control system is to have the awareness of every process variable that exists in the system. IRMIS provides a comprehensive database of all process variables that exist in all the IOCs (and other CA servers) defined to be an active part of the control system. For each process variable (represented by a *record.field* entry in an IOC's EPICS database) IRMIS contains the full definition of the *record type*. Because of the flexibility of EPICS, it is possible that two *record types* in different IOCs may have varying definitions, even if the *record type* name is identical. The IRMIS Process Variable Schema (shown in Figure 4) accommodates this by tracking record definitions for each IOC.

As stated earlier, IRMIS is a descriptive documentation tool, so the information that it contains in the Process Variable Schema must match the process variables currently loaded in the IOCs (i.e. IRMIS is not involved in the definition of the iocCore databases). To ensure this synchronization, IRMIS is made aware of every IOC reboot that occurs and updates the Process Variable tables according to the freshly loaded process variables. This auto-population of the IRMIS Process Variable tables is a key aspect to ensuring data consistency. It is carried out by a non-invasive 'PV-Crawler' application that parses the .db instance and .dbd definition files used by the IOC at boot time. The PV crawler application is part of IRMIS::Base.

Updating the Process Variable table after every IOC reboot also provides the benefit of a complete history and chronology of defined process variables. Reports can easily be implemented which describe process variables "as they were defined" in addition

to the current definitions. Discovering which process variables were eliminated during a recent shutdown is a simple yet extremely informative query. The software portion of IRMIS is more than just an inventory of EPICS records and fields – it is a time-stamped 'snapshot' of the entire software system, containing all the IOC-specific record type definitions, the IOC boot history as well as the Process Variable instances and inter-processor links.

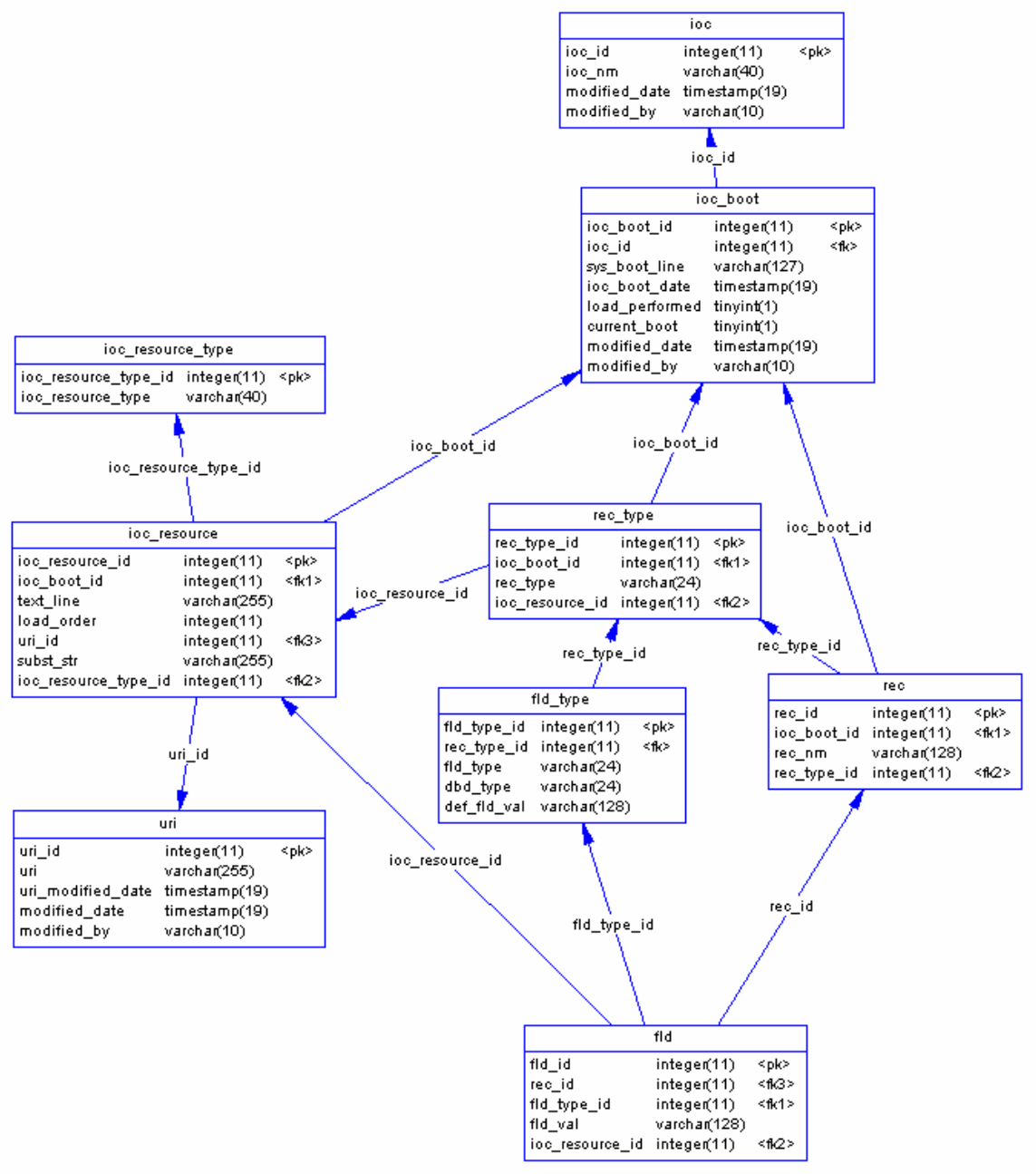

Figure 4. Physical Implementation of the Process Variable ERD.
The schema includes time-stamping of process variable instances and definitions.

The Process Variable table is central to the IRMIS "connection oriented" view of things. Applications are "connected" to process variables and process variables are "connected" to devices through ports and cables. Numerous relations are defined around the process variable database which ultimately connect software applications to accelerator components.

Although the vast majority of process variables represent standard records in an active iocCore database, PVs can also be served by custom channel access server applications. For the Process Variable data in IRMIS to be exhaustive, each site must appropriately manage any application that can "create" PVs and ensure they are appropriately loaded into the PV schema.

Component Database

The terms “component” and “device” are difficult to objectively and exhaustively define, yet they are a necessary fundamental concept for documenting an accelerator installation. Does it refer to “accelerator devices” (e.g. BPMs, vacuum valve, etc), control system devices (e.g. a VME card, an RS232 cable, etc), or both? Does one handle different device/component types with unique classes or is there a single concept of a device/component?

Once again, IRMIS::Base defines a basic component schema with sufficient hooks for each site to extend that definition to fit their particular needs. In IRMIS, a fundamental distinction is made between ‘component’, which has an objective ‘function’ (e.g. convert an analog signal to a digital value; convert a DC current into a magnetic field) and a ‘device’, which has a more subjective ‘role’ (e.g. a vertical steering corrector, horizontal beam position monitor). The device ‘role’ is often used as the basis for the process variable naming convention. Every component has a well-defined function, whereas it is not always clear what the role of a device might be – it may depend on the user’s perspective. A device may be made up of a set of inter-connected components.

A *component* is <coarsely> defined as any unit-replaceable physical entity associated with the accelerator (e.g. modules, power supplies, instrumentation devices, mechanical components, etc). Physical spaces where other components may be housed (e.g. a *room* because it houses other components, like racks) are also considered components. Although there are thousands of components that make up an accelerator facility, there are significantly fewer *component types* (unique types of components). In IRMIS::Base, there is a single class definition for all *component types*.

The attributes of *component types* are sufficient to:

- uniquely identify that *component type*
- provide basic information about it,
- describe how that *component type* connects with other *component types* in the three primary hierarchies (control flow, housing, and power).

The major attributes (accommodated in IRMIS::Base) of a *component type* are:

- Component Name
- Component Description
- Manufacturer
- Link(s) to Further Documentation

- Form Factor

  This attribute describes the physical form of the component (e.g. A-size Eurocard, C-size Eurocard, Allen Bradley 1771 I/O, IP module, rack mount, etc). This attribute allows queries to narrow their search to components with specific form factors.

- Function

  This attribute describes the generic function of the component (e.g. motor, cardcage, CPU, ADC, instrumentation, etc). This attribute allows queries to narrow their search to components with a specific function or keyword.

Each *component* in the facility is of a given *component type*. The distinguishing feature of the IRMIS component database is that as well as recording the existence of each component, its relationship with other components is also captured. Three types of inter-component relationships or 'connections' are recorded, as follows:

- Any component is 'housed' in another component. A simple example is a card-cage component housed in a rack, which serves as its 'housing parent'.
- Any active component is 'powered' by another component in the database – its 'power parent'.
- A control system component is 'controlled by' its 'control parent' component.

These deterministic parent/child relationship types each imply orthogonal component hierarchies. The multi-hierarchical nature of the IRMIS schema can be generalized to other hierarchies – e.g. accelerator physics hierarchy, vacuum system hierarchies, etc.

- To distinguish sibling components of the same type that have the same housing parent, a 'physical descriptor' attribute is required. A common example occurs when 2 identical I/O modules are housed in a single card-cage. The physical descriptor used to differentiate these siblings in this case is the card-cage slot number.
- Driven by the requirement to relate control hardware components to EPICS process variables, each component instance has a 'logical number' – this is the logical variable used by the EPICS software to address the specific component. When a process variable has failed, the logical number used in the EPICS record is required to locate the failed hardware. In addition, each component instance has a place-holder for a serial number.

The multi-hierarchy approach to modeling components conveys much more useful information than simple attributes. Queries such as 'what other components are on this common electrical circuit' or 'what other components are nearby' are made possible by the hierarchical nature of the database.

The Entity Relation Diagram illustrating the component and component-type entities is shown in Figure 5.

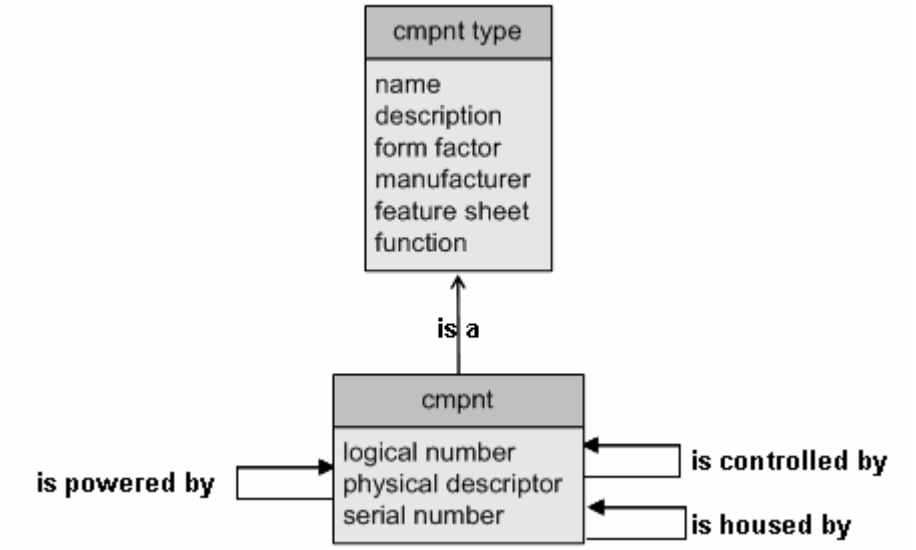

Figure 5. Component/component-type ERD, illustrating the multi-hierarchical nature of inter-component relationships.

Additional support for applications using the 3 component hierarchies are provided by a set of ancillary tables. These attributes provide hierarchy dependent filtering mechanism that is used in the maintenance of the component database:

- Control IFR (Control **I**nter**F**ace **R**equired)
This attribute describes the set of "interfaces" that the component type *requires* in order to connect to the control system. It answers the question "What kind of component does this component type plug into in order to communicate with the control system?". For example, a component type that had a GPIB interface would have a Control IFR of "GPIB Slave", because it must be "plugged into" a parent that accommodates GPIB slaves (in this case a parent GPIB link) to communicate with the control system. A typical VME card would have a CIFR of "VME Slot". This attribute can be used to narrow the search for devices that are to plug into other devices (e.g. one would never plug in a GPIB instrument to an RS232 link).

- Control IFP (Control **I**nter**F**ace **P**rovided)
This attribute describes the set of "interfaces" that the component type *provides* to other (child) components. It answers the question "What interfaces does this component type provide for other component types to allow them to connect to and communicate with the control system?" For example, a VME Chassis would have a Control IFP of "VME Slot", because it provides several locations where VME Slot component types can be plugged in. Some component types may provide several control interfaces, such as a VME CPU that has serial ports as well as IP slots. A GPIB component type must be "plugged into" a GPIB link to communicate with the control system. A typical VME card would have a CIFR of "VME Slot". This attribute is critical for supporting a Control Hierarchy of the installed devices.

- Housing IFR
This attribute describes the set of "interfaces" that the component type *requires* to be physically mounted or housed. It answers the question "What does this component type *physically* plug into?". For example, a rack mount instrument would have a Housing IFR of "Rack space" and a rack would have a HIFR of "Floor space". Note that a VME module has a HIFR of "VME Slot", which is identical to its CIFR. This attribute allows accommodation of component types that do not specifically communicate to the control system. While many components will not be part of the Controls Hierarchy, it is difficult to identify any installed component that does not have a *housing parent*.

- Housing IFP
This attribute describes the set of "interfaces" that the component type *provides* for housing other component types. It answers the question "What can *physically* plug into this component type?". For example, typical VME chassis would provide numerous VME Slots as well as a physical location for a pluggable replaceable power supply.

- Power IFR
This attribute describes the set of "interfaces" that the component type *requires* to be appropriately powered. It answers the question "What type of power does this component type *require*?". For example, a rack mount instrument would typically have a Power IFR of "120VAC". A VME module has a PIFR of "VME Slot", which is identical to its CIFR and HIFR.

- Power IFP
This attribute describes the set of "interfaces" that the component type *provides* to power other component types. It answers the question "What type of power does this component type *provide*?". For example, an AC panel provides numerous AC circuits, so its Power IFP would include "120VAC_Circuit". "Wall-warts" would have a PIFR of "120VAC" and a PIFP of, for example, "24VDC".

The above attributes support a comprehensive capability to fully describe how every component fits into the accelerator facility - with respect to the way it communicates to the control system, the way it is housed, and the way it is powered (Figure 1). One can easily envisage the huge source of information that this approach provides, yet with a very simple schema of defining component types. Site-specific attributes about components will be defined in an ancillary table with appropriate cross references. These might include inventory, spares location, test procedures, or a cognizant engineer. The physical implementation of the component/type database is shown in the schema in Figure 6.

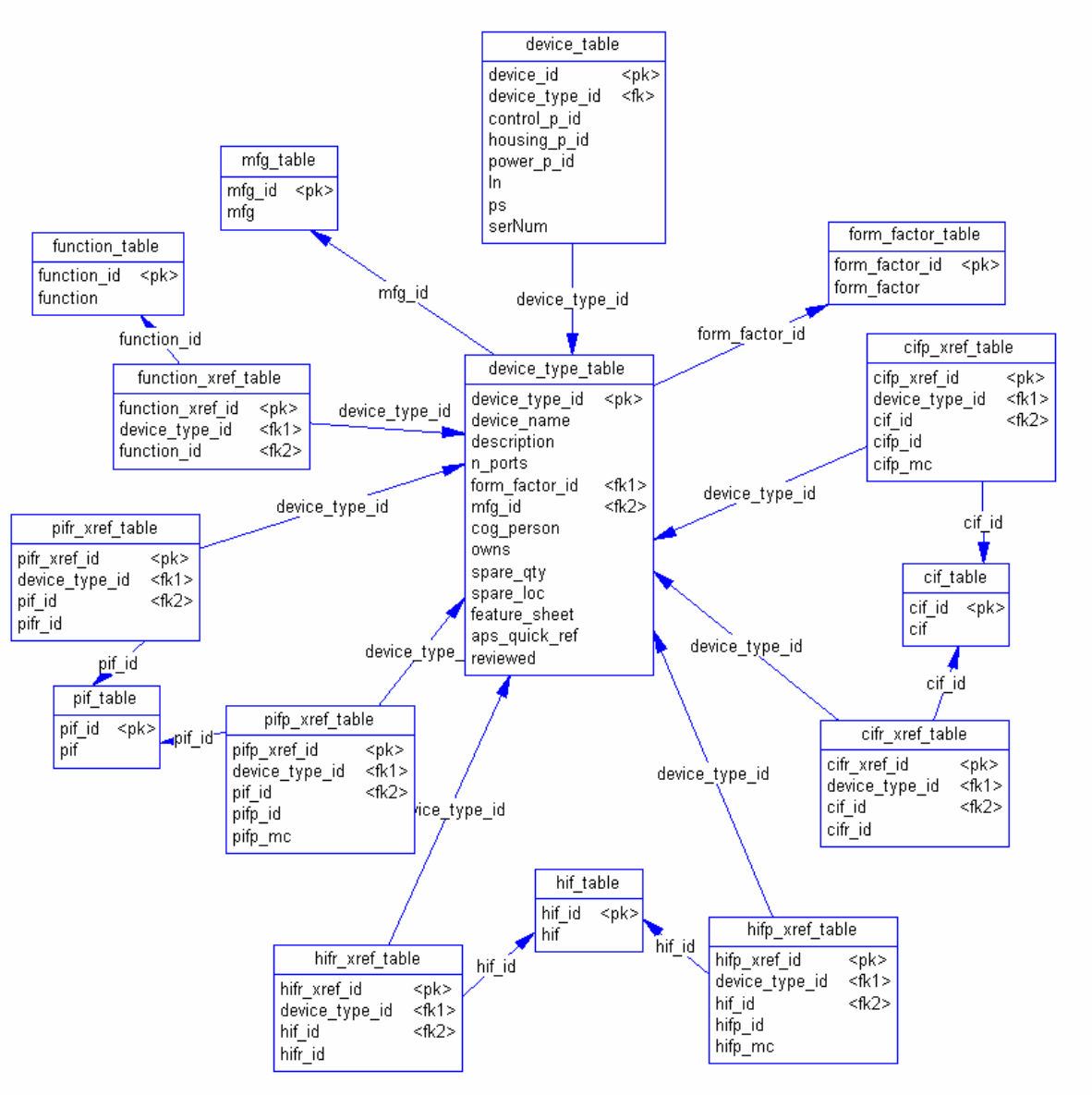

Figure 6. Physical implementation of the Component/Component-type ERD. Additional APS-specific component attributes are included.

Port/Cable Entity Relation Diagram

All component types have “ports” (usually the connectors on a component type) that are the physical connection to other components. A ‘port’ table contains the definition (e.g. port name, connector type, etc) of every port for every component type. With the component and port tables in place, with the inherent connectivity these tables imply, a cable database schema is surprisingly simple to implement. Rather than a simple inventory of cables, this approach also maintains the “connection oriented” scheme in which a cable is defined as a connection between to different component ports. To take care of the vast variety of cable and connector types that are available, and the situation in which the installer may either split, or ‘pigtail’ selected conductors from within a given cable/bundle, the ERD is extended to include the individual conductors. For most cases, the user interface will provide interaction with the cable database at the port level, only resorting to the pin/conductor level for the special cases described above. The cable schema, combined with the component hierarchies, will allow data and signal flow to be traced beyond the control equipment and on to the field devices. The ERD for the conductor/cable schema is shown in Figure 7

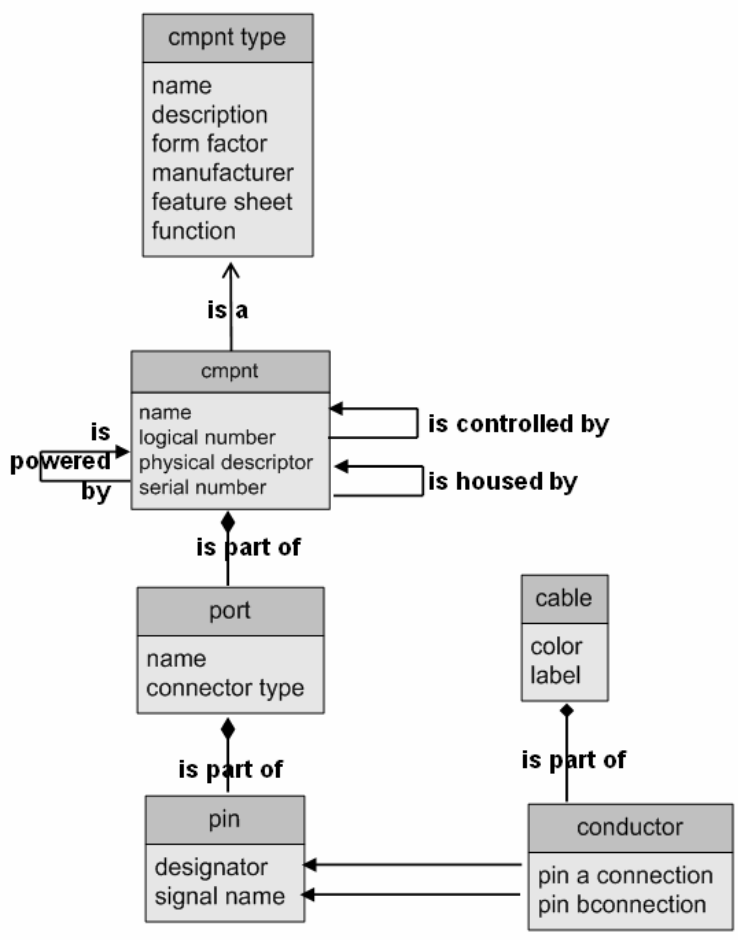

Figure 7. Extension of the component ERD to include ports, pins, conductors and cables

PV Client Schema

To properly manage Process Variables, it is invaluable to record which software programs (i.e. PV clients) are using any given PV. The clients of interest are a mixture of popular EPICS extensions (e.g. MEDM, EDM, ALH, etc) as well as numerous applications and custom programs at each specific site. IRMIS::Base will provide the table schema and a method of populating this table for certain popular clients. For the Process Variable Client data in IRMIS to be exhaustive, each site must appropriately manage any application that uses PVs and ensure they are appropriately loaded into the PV Client schema.

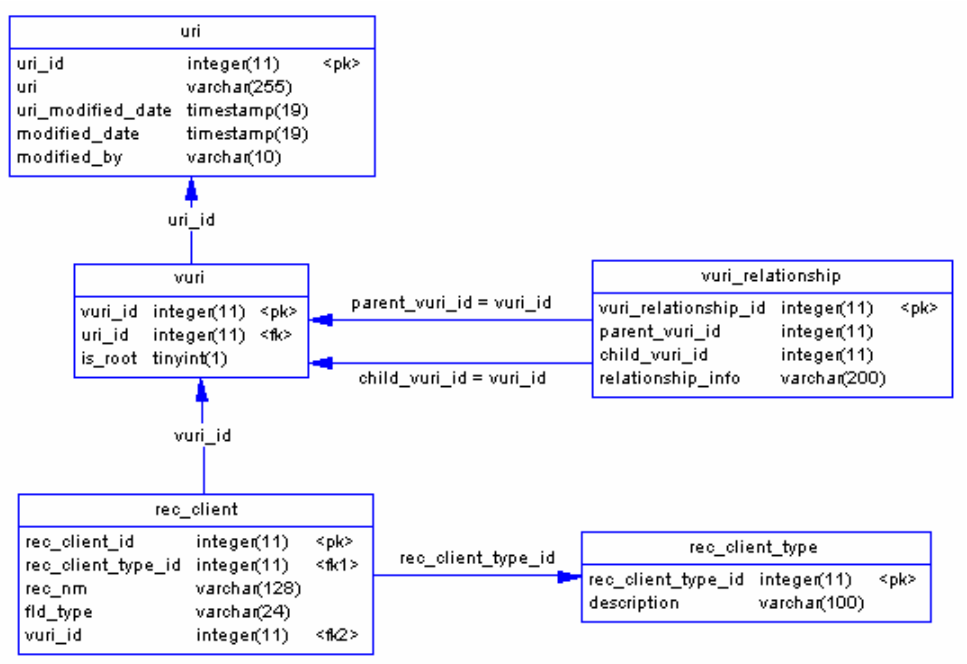

Figure 8. Generic PV Client schema. This schema shares the uri table defined in the PV schema (Figure 4).

Additional Tables
Other tables that have been considered that may become part of IRMIS::Base include:

- Control Applications/Subsystem
It would be useful to associate modules, process variables, cables, etc with the control system application for which they were installed. For example, given a process variable name, one could determine that it was part of the Low Level RF System in the LINAC.

- Failed Component Table
Tracking each failed component according to its component type and serial number would provide important reliability information over a long time scale.

- Component of Interest Table
Some components have unique characteristics that are of further interest to track, such as calibration factors or characterization data. Since this is a subset of all components, a separate table is needed to hold such information.

IOC Schema

Although physical IOCs are simply a combination of installed components (e.g. VME crate + CPU + Ethernet connection, etc), they are also a unique entity in an EPICS-based control system that requires separate tables to more fully describe them. The 'ioc' table in the Process Variable schema (Figure 4) holds an entry for every IOC in the system. Additional attributes for each IOC, including IP #, boot directory, IOC criticality, IOC operational status, cognizant developer, Ethernet connections, etc are captured in two additional tables, shown in Figure 9. The IOC entity is defined to be the top of the "controls hierarchy" that describes control data flow through the system (See Figure 1; Control Path).

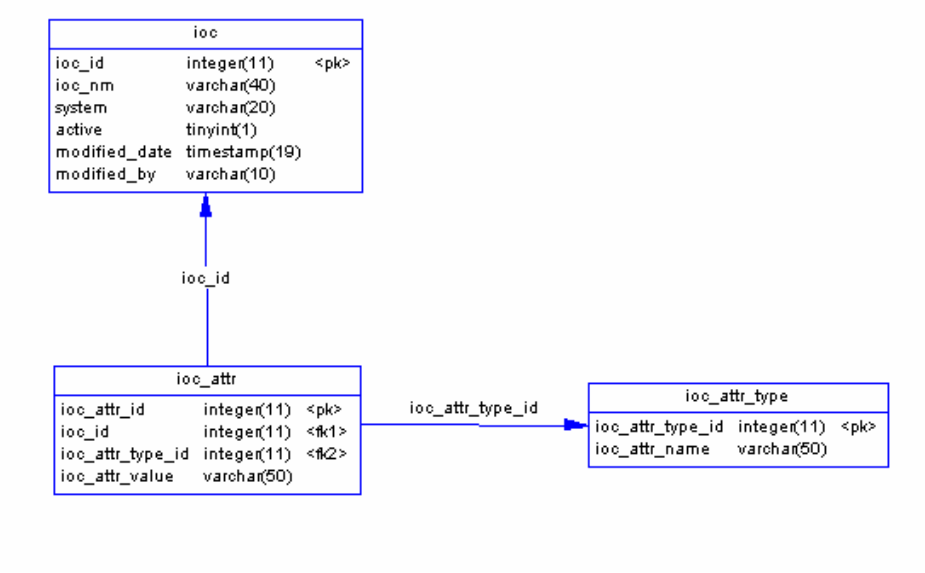

Figure 9. IOC schema. (The 'ioc' table is previously defined in the PV Schema).

Table Population & Data Validation

Even a cleverly devised schema is useless unless the tables are exhaustively and accurately populated. This is a daunting task for large existing installations. However, because EPICS applications are primarily implemented by "configuration" rather than by custom programming, a significant amount of the information required to populate the tables can be "mined" by scripts that know the internals of EPICS.

Below is a description of how each table can be populated.

Process Variable Table
Since IRMIS is a descriptive view of the installed control system, the Process Variable Table is 100% automatically populated to represent exactly what has been loaded into the channel access servers. The "pv crawler" is a sophisticated script that interprets the IOC st.cmd files in the same way iocCore does and develops a completely consistent set of PVs as to those loaded in the IOCs. The crawler is made aware of any IOC reboot in the system and updates the IRMIS table appropriately.

Additional measures must be implemented at each site to accommodate PVs that have been instantiated within a custom written channel access server.

Component Schema
By mining the vast configuration information within st.cmd files, databases, the dbior reports from running IOCs and also applying some generic assumptions and rules, significant information can be deduced about the installed components. A large subset of VME modules, I/O subnets, and GPIB devices can be identified by such methods. At APS, we estimate that 70-75% of our installed control system components were "discovered" by such methods, leaving less than half to be interactively entered using the Visual Connection Configuration Tool (VCCT).

IOC Schema
Most facilities already have some mechanism for tracking the majority of information that is required in the IOC Table. For large numbers of IOCs, gathering this data could be automated. For smaller installations, hand entering the IOC table is not a daunting effort.

Port/Cable Schema
This is a nasty one. Most cables will have been documented in drawings using various tools and formats. Unfortunately, most of this data is quite useless for anything other than looking at it [which is another strong argument for IRMIS]. If spreadsheets have been used to document cables, it may be possible to export and extract a portion of this data and populate the Cable Table. Significant effort would be needed to properly identify the component instance in IRMIS to the one in the spreadsheet. Likewise, port names would have to be consistent. It is likely that the vast majority of entries into the Cable Table will be manually entered. This is why substantial effort should be put forth into an easy to use Cable Editor that makes this effort as simple (and fun) as possible.

PV Client Schema
Another crawler will certainly be developed which understands the configuration files of popular EPICS tools. With appropriate version control and disciplined directory usage, most PV Client data can be automatically populated. Additional measures must be implemented at each site to accommodate channel access client programs whose PV usage is not as easily determined.

Tools

The success of IRMIS will depend almost entirely on the usefulness and ease of use of the tools provided for the staff. Below is a <growing> list of tools that need to be developed:

- PV Search/Viewer
- Component Type Search/Viewer/Editor
- IOC Search/Viewer/Editor
- PV Client Search/Viewer
- Installed Component Search/Viewer/Editor
  This application is a sophisticated graphical user interface application to view and define the installed components and their connections with other components. Because of this "visual connection" function, it is often referred to as the *Visual Connection*

Configuration Tool (VCCT). This tool must provide views of these connections in the three main hierarchies described throughout this document: Control Flow, Power, and Housing. It must be intuitive, easy to use, and assist in maintaining data integrity across the database tables.

- Port/Cable Search/Viewer/Editor
  This application is an ambitious effort to allow easy interaction with the cable and port tables. It may be an integral part of the VCCT application described above. Another observation is that it has a lot of similarity to VDCT (defining links between blocks). Is there any benefit in leveraging off of that effort?

The current plan is to implement these tools under a Controls Framework that has been derived from XAL. This will not preclude the development of other applications (e.g. PHP, PERL, etc), but a common look and feel for the most used tools is strongly desired. To support that effort, some toolkits/API's that will be developed include:

- Java Database Access Layer (API)
  The API will provide a common set of methods for finding, creating, and updating IRMIS entities such as "Record" (PV) and "Component". This database layer will be based on a design pattern known as Data Access Objects (DAO). The DAO are in turn implemented using Hibernate, a popular open-source Java Object-Relational Mapping (ORM) tool. The use of Hibernate brings with it database independence. The same code can run against MySQL, Oracle, Postgres, etc.

  The API will be used to support thick clients written in the application framework derived from XAL. This same API can also be re-used to support thin web client applications, developed as Java JSP pages.

- XAL Design Patterns
  A set of techniques for developing IRMIS tools in XAL will be established and documented via a sample application. It is important that applications are well-behaved in the face of potentially slow SQL queries and multiple database update requests from application threads.

- Security
  All thin and thick applications will require authentication and authorization to conduct any activity other than read-only queries.