EPICS ChannelFinder – Enhanced Directory Service Design Document

prepared by: Ralph Lange

Scope

High level applications tend to prefer an hierarchical view of the control system name space. They group channel names by location or physical function. The name space of the EPICS Channel Access protocol, on the other hand, is flat. A good and thoroughly enforced naming convention may solve the problem of creating unique predictable names. It does not free every application from being configured explicitly, so that it knows all channel names it might be interested in beforehand.

The ChannelFinder tries to overcome this limitation by implementing a generic directory service, which applications can query for a list of channels that match certain conditions, such as physical functionality or location. It also provides a way to map several directory entries to one channel, allowing for different perspectives of the same set of channel names.

This document serves as the description and design specification of that service.

Overview

Directory Entry

Each directory entry consists of a channel name, and a arbitrary set of attributes (name-value pairs).

Basic Operation

The application sends a search request to the service, containing an expression that references attributes and their values.

The service returns a list of matching channels with their attributes.

Design

Directory Data Structure

The requirements for the directory data structure have been identified as follows:

- A channel name (string, unique, single-valued).
- An arbitrary number of attributes, as name-value pairs.
  (Name: string, unique. Value: string. May be multi-valued.)

While it eventually might be desirable to restrict the set of available attributes, or restrict some attributes to be single-valued, such restrictions should not apply during the development and commissioning phase of the service.

To allow for different perspectives of the control system, alias entries can be defined, with a separate set of attributes. An attribute in the alias entry will contain the Channel Access name of the “real” channel that the application will have to connect to.

Service Structure

Many sites are using RDB systems to keep track of installed devices and their properties (e.g. IRMIS). For a better separation of data domains, and to facilitate usage of the directory service for sites without such an RDB system, the service will be a separate, stand-alone application. The RDB systems, on the other hand, will be able to provide a large part of the directory entry data.

To provide a good separation between the service and the directory, the directory storage engine will be separate from the web service. All business logic will be part of the web service. This allows for switching to a different directory storage without changing the API or the business logic of the directory service.

Service Type / Application API

The application interface to the service will be a web service. The data (list of channels and their attributes) will be returned in XML (or JSON) notation.

All current programming and scripting languages have good support for using web services. This approach minimizes the implementation effort on both the application and the service side.

The service will be stateless, i.e. no session data will have to be kept between queries.

In addition to the http GET method for querying the service, the POST and PUT methods will be defined to create and update directory entries.

Robustness and Redundancy

Both the directory data storage and the web service can use well-established technologies to increase robustness and redundancy. The statelessness will help creating redundant and load balancing systems.

Performance Goal

The performance goal metrics can be described as:
With the service running on a standard server (~$6K machine), and a directory database containing ~150K channels with ~10 attributes each, a query with an arbitrary expression of attributes (within reason) that returns 2K channels should finish in under 1 second.

Implementation

Language and Framework

The directory service will be implemented using Java, developed within the NetBeans IDE, using Glassfish for deployment and running the service.

In accordance with BNL Controls Group policies, the directory server code will be public domain, hosted on SourceForge, using Mercurial as SCM system.

Database

A mySQL database will be used for storing the directory data.