< prev contents next >

1 Overview

Chapter Index

1.1 Motives

In a basic design of the Java Control System at Fermilab, the data are provided by the Data Acquisition Engines (DAE). In most cases, the DAEs are running on a secure network, protected by the firewall. In order to read or set data, an application needs to access the DAE using Remote Method Invocation (RMI) protocol. The existing protocol designed for client-server connections is quite powerful and comprehensive, but does not provide enough security. Because of that, access to DAEs is controlled mostly by the firewall which applies strong constraints on the network traffic. This leads to some serious limitations for the client applications. Roughly speaking, no applications are allowed to access (read and set) the data from outside a very limited area. The similar problem exists with the database access.

On the other hand, most of the client applications designed in Accelerator Controls Department are supposed to be used by a broad community of people. Some of them do not have a chance to get data directly from the engines. Secure Controls Framework (SCF) was designed to address these security issues; namely, to provide a common protocol that would allow client applications to connects the servers in a secure way. Besides that, SCF provides a new data acquisition API and hierarchy of data objects, which (as I beleive) will be more convenient for development of the client applications.

1.2 Architecture

The architecture of Secure Controls Framework is quite simple:

On the client side, there is a set of classes that implements the server connection, data acquisition jobs, database access; and provides an API used in the applications to access this functionality. Besides that, there is a set of reusable custom GUI components related to the data acquisition: for example, an extention of `JLabel` that shows reading or setting values, device selector, and a parameter page.

A process running on the server side accepts the client connections, verifies credentials of the remote users, and establishes virtual secure channels to the clients. Then it works as a bridge between these trusted clients and the data sources (DAEs and the database). Variuos types of data requests from the clients are decrypted, verified, and converted to a format understandable by particular data sources. The data obtained from the data sources is converted, encrypted, and sent back to the clients. The client-server communication works through Java RMI protocol.

The client authentication is provided by Kerberos V5 protocol. Java's GSS-API implementation is ised to transfer client credentials to the server and establish a secure communication channel. In simple phrase, all the security is based on the user's Kerberos ticket: it must be obtained with a third-party authentication program (like *kinit* or *Leash32*) before using SCF. The only exception was made for the Main Control Room desktops: they are authenticated without Kerberos, by their IP addresses. See “Using Kerberos Authentication in Java”

for details.

As was said, SCF works with two kinds of data sources:

Database Repositories
: ACNET devices, parameter pages, and the Universal Repository of Serialized Objects (URSO). The first two databases are inherited from VMS and avalable only for reading. URSO can store arbitrary Java classes in the serialized form. It uses UNIX-like permissions and available for reading and writing. On the client, the data access is provided through JNDI.

Accelerator's Data Acquisition System
: Most of the data provided by Data Acquisition Engines (DAE) is avaiable through SCF. Both readings and settings are supported. Plotting data type is supported. Remote user privileges (account, node, and service) are translated accordingly in a “delegating” DAQ user, created by SCF. DAE policy for settings lock/unlock remains unchanged. On the client, the data access is provided through a custom API.

Additional data sources can be implemented as needed.

The client-side code is located in the following packages:

List Of Packages

Package	Description	Public API
`gov.fnal.controls.scf`	Server connection and one-shot data acquisition	Yes
`gov.fnal.controls.scf.acquire`	Basic data acquisition abstractions	Yes
`gov.fnal.controls.scf.example`	Examples of programs	No
`gov.fnal.controls.scf.items`	Implementation of data acquisition objects	Yes
`gov.fnal.controls.scf.naming`	Implementation of JNDI	No
`gov.fnal.controls.scf.remote`	Implementation of RMI objects and transport level	No
`gov.fnal.controls.scf.runtime`	Implementation of client-side data access	No
`gov.fnal.controls.scf.swing`	Reusable GUI components	Yes

Public API is located in the highlighted packages. Everything needed on the client side is packed in *scf-client.jar* archive. Kerberos Module *(krb5.jar)* is always required in the classpath. `ScfConfig` and `JobDataTable` classes also need to have Application Framework available.

1.3 Current Status

The system is fully operational. Users are allowed to connect SCF servers from any on-site location. Off-site access is not available so far, and it is a subject of concrete needs. By default, the system will connect to an operational server. If it does not, or if you need to have a list of current servers, please write to apetrov@fnal.gov.

The pilot implementation of SCF is **Data Browser** application:

[Start Data Browser]
you need to have the cached Kerberos ticket!

1.4 Pro et Contra

In order to decide what technology to use in a new application, you can consider the following arguments:

You can take advantage of Secure Controls Framework if...

- you are developing a stand-alone program on Java using J2SDK 1.4.2 or higher, *and...*
- the program is expected to be running mostly on on-site computers at Fermilab, *and...*
- the program needs to get some data from the data acquisition system, process it somehow, show this data to the user, and do some settings based on the user feedback, *and...*
- your users won't mind to start *kinit* or *Leash32* every time.

You should not use Secure Controls Framework if...

- you have to use an older or different version of Java or *may be* if this is an applet, *or...*
- the program is expected to be running mostly from outside Fermilab (use servlets instead), *or...*
- this is a server-side program (use DAQ instead), *or...*
- the program needs to process a large amound of data, or process fast data online, *or...*
- Kerberos is unavailable.

< prev contents next >
security, privacy, legal