DATA BASE SERVER ON THE VEPP-4 CONTROL SYSTEM*

D. Filimonov, S. Karnaev, B. Levichev, A. Nikiforov,
I.Protopopov, S. Smirnov, A.Veklov,
BINP, Novosibirsk, Russia

Abstract

The VEPP-4 control system was designed almost 20 years ago as distributed CAMAC-based control system [1]. The CAMAC-embedded 24-bit Odrenok computers are used in the system as the main control machines. The main problem of the control system further development is the change of the obsolete computers on PC. For ensuring the access to equipment and data for programs working in PC computers, the host-based database was designed.

The communication software is Linux-based and includes the Application Server (AS) in the host machine and Device Servers (DSs). The AS provides channel and database access for user applications. DSs provide interaction between AS and executive programs working in executive computers, connected to electronics. The backend database server is based on PostgreSQL.

This paper describes a structure of the VEPP-4 control system database server and the database access procedures.

1 INTRODUCTION

The VEPP-4 Control System includes 14 home-developed Odrenok computers [1] integrated in the local Ethernet domain. PCs under Linux are used as operator consoles in the Control Room. The following programs are running in the PCs:

- Odrenok terminal windows,
- Odrenok file-boot server,
- Graphical applications for visualisation of the different data received from Odrenoks [2],
- New beam diagnostic applications and applications for providing experiments on J/Ψ and Ψ '-mesons mass measurement [3].

Further development of new programs in PC was restrained by lack of the united database that includes the data about structure of the facility, control and measuring channels, etc. This data is distributed between different files in Odrenok's file system and it is very difficult to get them from PC.

At the present time, the united database is designed. It consists of about 30 PostgreSQL tables. The database system provides the following functions:

- remote access to the data from PCs,
- remote access to control and measuring electronics,
- exportation of the data from PC to Odrenok and back for providing functioning of the Odrenok's software.

* This work is supported in part by Russian Fund of Basic Research (N02-07-90108)

2 DATA BASE SYSTEM ARCHITECTURE

All the data about the operational facility is divided into two parts:

- static data - description of the facility's parts, electronics, control and measuring channels, saved operation modes, etc.,
- dynamic data - present state of control and measuring channels, i.e. values describing a present operation mode of the facility.

The static data in the database system is distributed between relative PostgreSQL tables. The dynamic data is in electronics and in special programs Device Servers (DSs). All communication with electronics are realised via DSs.

The data base system architecture is shown in Fig.1.

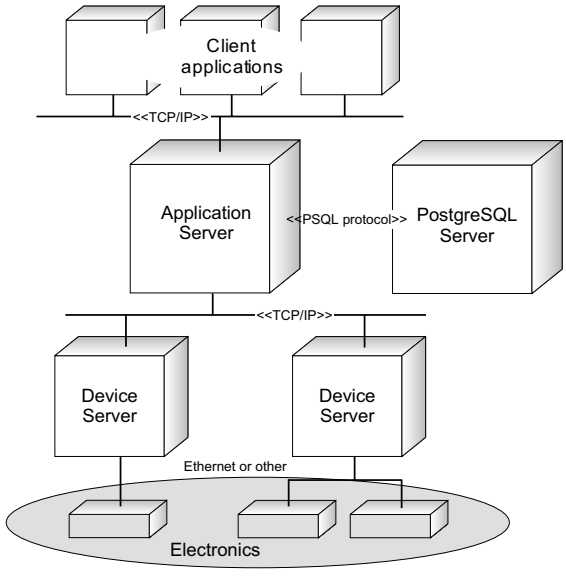

Figure 1: The data base system architecture.

The kernel of the database system is the Application Server (AS). It provides the following functions:

- handling of the requests from client applications,
- interaction with DSs,
- interaction with PostgreSQL.

AS handles the requests from client applications by reading information from PostgreSQL and sending in the own turn requests to DSs.

DSs are the programs which are either directly connected to the electronics or communicate with low-

level programs connected to the electronics. DSs provide the next functions:

- receiving requests from AS and transfer them to electronics or to low-level programs (in our case to programs in Odrenoks),
- sending to AS Alarm signals.

DSs when started are registered on AS and report the area of their responsibility.

3 DATA BASE TABLES STRUCTURE

There are about 30 relative PostgreSQL tables for static data storing. All tables are divided into 4 groups:

- facility structure description,
- electronics description,
- control and measuring channels description,
- stored operation modes.

Full structure of the VEPP-4 data base tables is shown in [4].

3.1 Facility Structure Description Tables

The whole structure description of the facility is a hierarchy of objects of different types. There is a pointer to the up-level object in the description of each object. The root record in the structure is the object *'VEPP-4 facility'* (see Fig.2). Object types are stored in the separate table.

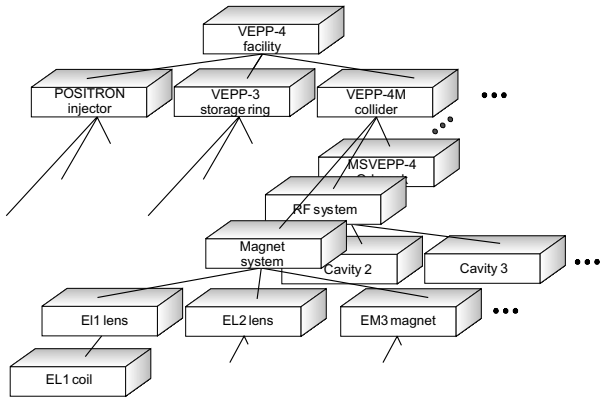

Figure 2: The objects structure.

For description of the different characteristics of the objects, the conception 'parameter of object' is used. Parameters provide qualitative and quantitative descriptions of any type of objects, for example: quadrupole type object has placement azimuth, length, magnetic field gradient, current/gradient factor, resistance parameters. Both parameters and parameter types are also stored in the separate tables.

3.2 Electronics Description Tables

For the description of the electronics, the following hierarchical structure is used: computer - *crate* - module. The *crate* is the logical assembly of modules (for instance, CAMAC-crate, MilStd-line) in which the module is uniquely identified with a simple characteristic (position, address, etc.).

The data about modules and *crates* are allocated into four tables: *crates* and modules description, *crate* types and module types.

The description of the *crate*-computer connection uses conception 'path to *crate*'. Path to *crate* includes a list of components and components' values which uniquely identifies the physical address of the *crate*. There are several types of paths in the Control System, i.e. there are several types of computer - *crate* connection. Also, there are several types of path components. The example of the path from Odrenok to CAMAC-crate with three components is shown in Table 1.

Table 1: The path to crate
(type of path is "DS24-CC24")

Path comp. number	Component type	Value
1	Position of DS24 interface module	12
2	Module channel	1
3	Number of interruption from crate controller	10

3.3 Channels and Elements Description

For the control system description, from the standpoint of intercoupling of the electronics and the controlled devices, the conception of *channel* is used. The channel description contains the information about the physical value/ module code factor, value range, etc. Logically, channels are united into elements.

For description of channels and elements the following main tables are used:

- the element types table,
- the elements description table,
- the channel types table,
- the channels description table,
- the multiplex elements description table.

There are different types of channels: control channels, measuring channels, switch (bite) channels, etc. In spite of heterogeneous data (table fields) for different type channels, the channel description table contains descriptions of all types of channels. This approach gives advantage in access time and simplifies the software.

There are multiplex elements (so-called *'handles'*) with one or two input/output values. I/O values of multiplex elements have matrix translation to/from values of channels included into these elements. Here is one-I/O matrix element example: the value of the storage ring 'energy', evaluated via coil current values in bending magnets.

3.4 Operation Modes Tables

The operation mode is a set of values of control and measuring channels, corresponding to the status of the part of the facility or group of devices.

For stored operation modes, the following tables are used:

- the operation modes groups table, which allow us to aggregate stored modes, providing for different processes at the facility (acceleration, magnetic reversal, etc.),
- the operation modes description tables containing the operation mode's attributes: date and time of recording, comments, etc.,
- the table of values of channels.

For the all stored operation modes, the same table of values of channels is used. It is supposed that the number of records in this table will not exceed several hundred of thousands. In each record of the table *id*s of an operation mode and a channel, and channel's value in the operation mode are included.

4 DATABASE CONNECTIVITY LIBRARY

4.1 General Description

The database connectivity library consists of four kinds of entities:

1. A set of storable classes representing database structure with each class implementing ASStorable interface.
2. A set of meta-storable classes describing properties, those are common for any certain storable class, such as list of filters, etc.
3. A set of filters. Filters are used for selecting a subset of storable items.
4. Storages. There are three kinds of storages implemented:

- ASDbStorage: directly uses database to keep storable objects,
- ASMsgStorage: is used by client applications via Application Server,
- ASCacheStorage: is used to cache requests to underlying storage, either ASDbStorage or ASMsgStorage.

This architecture makes it possible to use the same database connectivity interface both for AS and client applications.

4.2 Interfaces of database related classes

ASMetaStorable. This interface contains all useful information and allows us to work with *ASStorable* classes without knowledge of class type at compiling time. *ASMetaStorable* is an abstract factory [5] for creating corresponding subclasses of *ASStorable* and *ASStorableFilter* (methods *getFilter(Q_UINT32)* and *create()*). *ASMetaStorable* can create the objects from *QDataStream* as well (method *create(QDataStream&)*). It requires *ASMetaStorable* to provide obtaining an instance of *ASMetaStorable* by its type *id*; it's easy since each subclass of *ASMetaStorable* is a singleton [5] i.e. there is only one instance in the system.

ASStorable. All concrete subclasses of this interface are able to get itself from and to put itself into *QDataStream* (methods *get(QDataStream&)* and *put(QDataStream&)*, to store themselves in storage and to obtain themselves from storage by *id* (methods *get(ASStorage*, Q_UINT32)* and *store(ASStorage*)*). Each implementation of *ASStorable* has a link to appropriate *ASMetaStorable* instance that keeps all meta information.

ASStorableFilter. Classes which implement this interface contain information about fields involved in the filtration process including fields used for sorting result set. Filters are able to put themselves into and get themselves from *QDataStream*.

ASStorage. This interface describes all methods on single database records and methods for obtaining several records at once:
(*getStorables(QPtrList<ASStorable> *, const ASMetaStorable*), findStorables(QPtrList<ASStorable>*, const ASStorableFilter *)*).
This interface does not provide complex multitable queries because there is no common need of such requests for all types of classes. Moreover, such requests cannot be easily coded in all current implementations of *ASStorage* (*ASDbStorage*, *ASMsgStorage*, and *ASCacheStorage*). There is a different way to implement such requests: an extensible set of messages which AS and client applications use to communicate with each other. Actual implementation of *ASMsgStorage* is based upon these messages.

5 ODRENOK - PC DATA ENCODING

Before starting the data base system, we need to fill up the database tables with data located in Odrenok file system. Also we need to support running the existing Odrenok software. In order to gain this goal we implemented two converters. The first program parses Odrenok-encoded files and fills up database tables with the information. The second creates files for Odrenok programs from the database.

6 REFERENCES

[1] A.Aleshaev, et al., 'VEPP-4 Control System', ICALEPCS'95, Chicago, USA

[2] A. Bogomyagkov, et al., 'Data acquisition and handling in the VEPP-4 Control System', PCaPAC'02, MO-P6.

[3] V. Blinov, et al., ' Linux-based Toolkit in the VEPP-4 Control System', ICALEPCS'01, San-Jose, Ca., USA, Proceedings http://xxx.lanl.gov/pdf/physics/0111147

[4] http://vepp4-pult1.inp.nsk.su/~vepp4/db

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns. http://hillside.net/patterns/DPBook/DPBook.html