EDOC093

EUROGAM PROJECT

NSF DATA ACQUISITION SYSTEM

Resource Sharing in Eurogam

Edition 1.0

November 1991

NSF Software Systems Group

UK Science and Engineering Research Council

Daresbury Laboratory

Resource Sharing in EuroGam

David Brightly

Edition 1.0
November 1991

1 Introduction

This document explains the machinery supporting the sharing of data acquisition resources in the EuroGam system. The main aim of the system is to enable multiple workstations to share in the set up and monitoring of the data acquisition hardware in a controlled but flexible way. A secondary aim is to permit the operation of multiple EuroGam hardware systems or partial systems within a common computing environment consisting of a central file server and discless client workstations.

2 Resources

For the purpose of this discussion a *resource* is an RPC server process that provides access functions to EuroGam hardware components. As a general rule, the only way users can affect the data acquisition hardware is by making RPCs to these server processes. Most servers provide an *abstract interface* that hides the messier details of the underlying hardware. For example, each EuroGam VXI crate contains a *register server* that works in terms of named abstract objects called *registers*. Usually a register object represents a bit field within some actual hardware register on some data acquisition card within the crate. Similarly, the EuroGam histogrammer card is supported by a *spectrum server* process that controls the card and manages histogramming memory by operations on abstract *spectrum* objects.

Each data acquisition crate can contain one or more servers (though usually at most one of each type) and a EuroGam system consists of one or more such crates. These crates can be complemented by other computers running RPC servers. For example, the EuroGam system requires a data *storage server* that manages the flow of event-by-event data onto tape and one or more additional spectrum servers tailored towards the management of histograms held in disc files.

A resource thus has the following attributes:

- a *name* which identifies it;
- a *host* (represented by an IP address) in which it runs;
- an *RPC protocol* which it recognises;
- a UDP *port* number on which it listens;

Given this information a program running in a workstation (usually referred to as a *client* program) can contact a resource and request it to perform operations on the objects it manages.

3 Capabilities

All EuroGam RPC is based on the *connectionless* UDP transport protocol so an explicit protection mechanism is required in order to safeguard against accidental interference with one another's equipment by unrelated users. In an essentially *single-user* environment this would not be necessary. In EuroGam this protection is achieved through *capabilities*. These are access tokens which resource servers use to identify and authenticate users. Each RPC must carry a valid capability else it will be rejected by the server. Capabilities are awarded in response to **ClaimResource** RPCs and at any time may be voluntarily given up by a **FreeResource** RPC. Resources may be *exclusive* or *shared*. An exclusive resource has no more than a single capability awarded at any moment, and hence supports just one "user" at a time. All resource servers running in EuroGam data acquisition crates are exclusive resources. A shared resource can have several capabilities outstanding at any time and thus supports multiple "users", each identified by a distinct capability. The EuroGam disc-file spectrum server represents a shared resource. Its users must provide a valid unix login id and password as authentication in order to obtain a capability. Thereafter the server can identify the user issuing each RPC by means of the capability contained therein.

4 Experiments

A resource user at a workstation can store his capabilities in a disc file and thus make them visible to other workstation users, who can then embed them in their own RPC calls. Servers will be unable to distinguish two sources of RPC bearing the same capability. Hence two distinct workstation users submitting RPCs with the same capabilities will appear to a resource server as if from a single source. This provides the basic mechanism for multiple user workstations to share access to a common set of resources.

To avoid the proliferation of capabilities in files scattered all over the filestore (or users scribbling them down on scraps of paper!) the EuroGam resource sharing software formalises the concept of a *capability file*, and expects such files to be kept in certain directories only. A capability file is an array of `structs` of the following form:

```
struct resourceentry {                              /* records a resource */
   char resname [16];                                 /* name of resource */
   char reshost [16];                /* hostname of system its running in */
   int  resport;                                  /* UDP port it listens on */
   char resrole [16];                                         /* its "role" */
   char restype [16];            * ie, which RPC protocol it recognises */
   char resapparatus [16];        /* name of "apparatus" it belongs to */
   int  resIPaddr;                                               /* unused */
   CAP  rescapability;            /* 8 byte opaque capability, or null */
   int  resflags;                     /* status flags for this resource */
   };
```

recording the attributes of a set of resource servers and possibly a set of corresponding capabilities.

Shared use of a set of data acquisition resources by a group of workstation users constitutes an *experiment*, and each experiment is effectively identified by its capability file. Any workstation user who has read and write access to such a capability file can access the resources detailed therein. Since users collaborating on a EuroGam experiment will need access to further files defining other aspects of the experiment the resource sharing software observes a naming convention for experiment-related files. Each experiment is given a name (at the users' discretion) and represented by the directory

`/eg/experiments/`*experiment*

All ancillary files required by the EuroGam software system for an experiment are found beneath this directory. In particular, an experiment's capability file is named

`/eg/experiments/`*experiment*`/capabilities`

Placing experiment related files beneath the fixed directory `/eg/experiments` also helps any VxWorks application software which needs access (via NFS) to files created by workstation users—compiled Event Builder and Sorter programs, for example.

Workstation application programs can assume the existence of an environment variable, `EXPERIMENTDIR`, which contains the full pathname of the user's current experiment directory, as in

`setenv EXPERIMENTDIR /eg/experiments/`*experiment*

In any session involving a EuroGam experiment this variable must be set before any attempt is made to access EuroGam hardware resources. To do this we envisage a user interface feature which offers the possibility of selecting an existing experiment directory or creating a new one. Typically the first option would be chosen when a user group wish to bring a second or subsequent workstation to bear on their experiment. The second option would be chosen at the start of the setting up process for a brand new experiment. The user-interface feature solicits a name for the new experiment, checks that the name is not currently in use, and then creates the experiment directory and populates it with an empty capability file and whatever other ancillary files are defined by EuroGam application software. At the end of an experiment a further user-interface feature releases all the experiment's resources and optionally destroys the experiment directory. All exclusive resources *must* be released at the end of an experiment else the following user group will not be able to claim them for their own experiment. An experiment directory (possibly even including a capability file recording active capabilities) can continue to exist after all workstation sessions accessing the experiment have ended, and will survive a workstation or file server boot. This enables a user-group to regard an "experiment" as extending over several episodes of scheduled beam-time. However, retaining active capabilities beyond the end of the current beam-time episode is not recommended since this can inconvenience other users. The system administrator must always be empowered to release such claimed resources either by seizing the capability file or, more violently, by restarting the held resource server processes.

5 EGlib

Clearly, any workstation application program for controlling a EuroGam experiment must exploit the information in some capability file. The preferred way of doing this is through a library of functions, EGlib, built around the capability file concept. EGlib provides functions for accessing all the object types supported by EuroGam resource servers. For example, the function

EGreadRegister (*name*) $\rightarrow$ *value* returns the current *value* of the register called *name*. The EGlib functions expect a capability file to be memory-mapped into the calling program's virtual address space so that it appears as an array of `struct resourceentrys`. **EGreadRegister** finds the `resourceentry` representing the server supporting register *name* by means of a naming convention for EuroGam objects and resources. It can then generate an RPC to this server using the host and port addressing information held in the `struct`. EGlib has further functions for creating and deleting capability files and for memory-mapping a selected capability file.

6 Naming objects

Each object supported by a EuroGam resource server has a full name in the form

/resource/name

where *resource* denotes the name of the server resource supporting the object and *name* is the resource's own name for the object. Within a resource names must be unique, but different resources can support objects of the same name. Objects are also distinguished by their *type* (e.g, **register**, or **spectrum**), so two objects of different type can have the same full name, though this may give rise to confusion in the minds of users! For example, the register object `Ge23.CFD.Thresh` supported by the VXI register server resource `vxi1` has the full name `/vxi1/Ge23.CFD.Thresh`. To locate a server for a given object an EGlib function just scans the capability file looking for a `resname` field matching *resource*. Alternatively, objects can be located by means of *aliases* in the form

/role/name

where *role* must match one or more `resrole` fields. In the case of matching by role, different resource servers can support the same *role* but should not support the same object *names*. The *role* concept can be used to permit access to (say) register objects without knowing which register server supports them. For example, a register with the name `Ge23.CFD.Thresh` could be found in one of at least two EuroGam VXI crates. If both register server resources in these crates support the role (say) `frontend` then the register can be located by using the alias `/frontend/Ge23.CFD.Thresh`.

The algorithm used in EGlib for locating an object with full name */resource/name* and type *type* is thus quite simple: RPC in sequence to all resources found in the capability file whose `resname` or `resrole` fields match the *resource* component of the object name and whose `restype` matches *type* until one is found that recognises the *name* component. In the case of functions that operate by pattern matching on *name*, (**EGwriteRegisters**, for example) the scan continues until all resources whose `resname` or `resrole` fields match *name* have been RPCd.

7 Apparatuses

A viable EuroGam system consists of an inter-related set of co-operating hardware and software components. This is formalised in the EGlib concept of an *apparatus*—a named set of resource servers comprising part or all of a EuroGam configuration. Editable ASCII descriptions of possible configurations exist in the form of so-called *apparatus files*. These are in a simple keyword-value property list format that can be readily compiled into `resourceentry` structs. For example, a simple EuroGam configuration consisting of two crates each running register servers and one running a spectrum server might appear thus:

```
((name vxi1)
 (host nnvd)
 (role frontend)
 (type register)
 (ipaddress 123456)
 (port 10220))
((name vxi2)
 (host nnvk)
 (role frontend)
 (type register)
 (ipaddress 123456)
 (port 10220))
((name vxi2)
 (host nnvk)
 (role hist)
 (type spectrum)
 (ipaddress 123456)
 (port 10230))
```

(Please note that the *ipaddress* field has to be present even though the value is ignored—this will be corrected!)

EGlib provides functions for constructing capability file `resourceentry` structs by compiling an apparatus file (**EGattachApparatus**), and for claiming all the resources in a capability file belonging to a given apparatus (**EGclaimApparatus**). There are also converse functions for freeing all resources belonging to an apparatus and for destroying the corresponding structs. More than one apparatus may be attached or claimed at any one time.

8 Worked Example

The following C fragment shows the code needed by a typical workstation program that accesses a spectrum. The program's first argument is the full name of the spectrum to be read (assumed 1d), and it follows the conventions on the use of the EXPERIMENTDIR environment variable and the naming of capability files. All error handling is omitted for clarity.

```
char specname[128];
int counts[4096];
char exptdir[128];
int i;
strcpy (specname, argv[1]);
strcpy (exptdir, getenv ("EXPERIMENTDIR"));
i=EGsetExperimentPath (exptdir);
i=EGconnectExperiment ("capabilities");
i=EGsetSpectrumPath ("");
i=EGread1dSpectrum (specname,0,4096,counts,5)
```

The program might be invoked with the command

```
demo-prog /hist/ge23.4mev.q0
```

The next example shows how capability files are constructed, assuming the existence of a suitable experiment directory, with a sub-directory called `apparatus` containing apparatus files:

```
char exptdir[128];
char apparatusdir[128];
char value [128];
int i;
strcpy (exptdir, getenv ("EXPERIMENTDIR"));
i=EGsetExperimentPath (exptdir);
i=EGcreateExperiment ("capabilities");
strcpy (apparatusdir, exptdir);
strcat (apparatusdir, "/");
strcat (apparatusdir, "apparatus");
i=EGsetApparatusPath (apparatusdir);
i=EGattachApparatus ("demo-apparatus");
i=EGclaimApparatus ("demo-apparatus");
i=EGsetRegisterPath ("/frontend");
i=EGreadRegister ("Ge23.CFD.Thresh", value);
```

References

[1] *Eurogam Project: System Specification*, Pucknell et al, Edition 1.0, May 1990.

[2] *EGlib (3L)* manual pages

[3] *EuroGam Frontend Crate Server*, EDOC024, D.Brightly, September 1990

[4] *Distributed Systems—Concepts and Design*, Coulouris and Dollimore, ISBN 0–201–18059–6