Palantiri:
A distributed real-time database system for process control.

B.J. Tummers, W.P.J. Heubers
bas@nikhefk.nikhef.nl, wimh@nikhefk.nikhef.nl
National Institute for Nuclear Physics and High-Energy Physics, section K
P.O.Box 41882, 1009 DB Amsterdam, The Netherlands

Abstract

The medium-energy accelerator MEA, located in Amsterdam, is controlled by a heterogeneous computer network. A large real-time database contains the parameters involved in the control of the accelerator and the experiments. This database system was implemented about ten years ago and has since been extended several times. In response to increased needs the database system has been redesigned.

The new database environment, as described in this paper, consists out of two new concepts:

- A *Palantir* which is a per machine process that stores the locally declared data and forwards all non local requests for data access to the appropriate machine. It acts as a storage device for data and a looking glass upon the world.
- *Golems*: working units that define the data within the Palantir, and that have knowledge of the hardware they control.

Applications access the data of a Golem by name (which do resemble Unix path names). The palantir that runs on the same machine as the application handles the distribution of access requests.

This paper focuses on the Palantir concept as a distributed data storage and event handling device for process control.

I. INTRODUCTION

The National Institute for Nuclear- and High-Energy Physics (NIKHEF) operates for about ten years a linear medium-energy electron accelerator (800 MeV). Currently the accelerator is extended with a pulse-stretcher ring. At the same time the experimental facilities are renewed and according to the plans the first experiment with this new set up will start in the summer of 1992.

All these facilities are controlled by a number of computers running a home made real-time operating system in a point to point communication network and a number of Unix based machines (ref[1]). To prevent possible conflicts all hardware control is performed under the supervision of a device, which historically is called a database, in which all values are stored before they are sent to the hardware. This database was designed ten years ago, uses one (real-time) machine as central storage device. Although this system is functioning well and has proved its reliability in the past years, it has some disadvantages: it offers two different way to access data (by name, and by number), changing its layout is cumbersome and maintaining it appeared to be quite difficult.

Obviously, we wanted something new. This paper describes the aims, the concept and some details of the implementation of the new system. In the last section the current status of the project and plans for the future are discussed.

II. AIMS FOR THE NEW DATABASE SYSTEM.

Because we did built the current system ourselves, used it and maintained it for the past ten years, we had enough experience to define the aims for the new system. As the database system is meant for real-time, on-line process control it is clear that it should be fast enough to meet the needs for this kind of applications. Furthermore it must be possible to use the system in a heterogeneous environment, consisting of as well as real-time systems as Unix systems.

For the database itself we agreed that the new database should:

- Not duplicate data, i.e. data is only stored in one place to avoid inconsistencies.
- Be distributed without applications being aware of the distributed nature of the database.
- Be flexible and easily extendable.
- Not be aware of its intended use; i.e. not have any knowledge of the hardware it is intended to control.
- Offer as services at least: read, write, lock and subscribe.

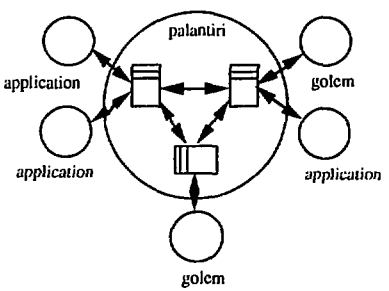

Figure 1. The Palantir database concept.

III. WHAT DOES THE NEW DATABASE LOOK LIKE ? PALANTIRI AND GOLEMS.

Palantíri [Quenya] 'Those that watch from afar', the seven Seeing Stones brought by Elendil and his sons from *Númenor; made by Fëanor in Aman.*
J.R.R. Tolkien, The Silmarillion, George Allen & Unwin, 1977, P 346.

Golem *n.* clay figure supernaturally brought to life (in Jewish legend); automaton, robot.
The Concise Oxford dictionary, 7th ed. 1982, P 426.

(These new names were invented to avoid confusion with the terms used in the existing system.)

On each machine that requires access to the database runs one specific process; not a server, not a daemon, but a *palantir*. This is indeed what the name suggests (to those familiar with Tolkien): a crystal ball giving access to all other *palantiri* in the system. On top of that, palantiri are able to store data that has been declared by their local agents/clients, which we call *Golems*.

A *Golem* is a process that typically, but not necessarily controls some hardware. The Golem has knowledge of this hardware, and receives the values to steer the hardware from its local Palantir. All relevant data that must be known to the outside world are sent to the Palantir by the Golem.

For the third type of process in our system we have not chosen a new name; processes that communicate with Palantiri, but do not own variables, are called Applications.

A Golem has a name which is made known to the Palantir, and owns a set of variables which are also made known to the Palantir. The Palantir allocates memory for these variables and maintains their values.

Variables have a name, a type and access rights attached to them, all of these are specified by the Golem. The variable name looks like a Unix path name. The first element is the name of the golem, all other elements are given by the golem. But there is a direct relation between the variable name and the structure of the data. The

variable called *should* in the following C-structure, that has been declared by a Golem with the name *A13*

```
struct magnet
{       long is;
        long should;
} Q1;
```

may be accessed by the path */A13/Q1/should*. The *magnet structure* can be read at once by accessing */A13/Q1*.

Access rights are different for the golem that owns the variables and for the rest of the world. The access may be any of read, write, lock and subscribe. Now, read and write will be clear; lock will at least give some idea, but what does subscribe mean?

Any application or Golem that has an interest in the value of a variable (either simple, or compound), can subscribe to this variable. This means that whenever the variable is written to a report is sent to each subscriber stating the name of the variable and the new value. This allows our magnet steering Golem to get a report of the new *should* value of its magnet without continuously polling the Palantir. Actually, it is possible to specify that reports are to be sent only when a write action results in the data being changed.

A lock is a flag that can be set on a variable, again simple or compound, reducing write access to that variable to the locking process. It may be used either to keep data unchanged over certain actions, or to keep all other processes from meddling with variables you are assumed to have full control over. A lock exists until it is removed by the locking process.

IV. A BIT MORE DOWN INSIDE A PALANTIR.

A Palantir is a process that forever waits for a message to arrive. Each message is processed in order of reception, depending on the type of request and the name of the variable with which the request is concerned the Palantir decides to handle the request locally, or to forward the request to another Palantir. Requests that can be handled locally are fully serviced before looking for a next request. Forwarded requests are maintained in a number of queues for future reference when the response from the remote Palantir arrives.

All this is quite straightforward, and Palantiri would be very simple indeed but for exception handling. Machines may become unreachable through network failure, or because the machine itself is down. There is no simple distinction between the two. Or a process having an outstanding lock may disappear, thereby creating a possible deadlock situation.

Palantiri have an elaborate 'are you there' mechanism, both to all other Palantiri and to their local Applications and Golems. An application or Golem that does not exist anymore results in all its locks and subscriptions being removed. Note, that a Golem's variables will remain valid, though any action performed on them will result in a warning message 'Golem dead' to the requester. The dead of a Golem is also reported to all processes having subscribed to any of the Golem's variables. When the Golem comes alive again (this is possible) the subscribers will be notified again.

When a Golem, and its data, is no longer needed, the Golem may be removed. All processes having a subscription or lock on the Golem's data are notified of the removal of the data, and the subscription or lock ceases to exist.

When a Palantir becomes unreachable, again all subscribers to any variable on that Palantir are notified. Locks on that Palantir stay in effect, but obviously, any action to that palantir must result in an error message. When the Palantir becomes reachable again, it is possible (well usually) to differentiate between network failure and machine failure. When the unreachability was the result of network failure, it is assumed that locks and subscriptions are still valid. When the remote machine has been down, locks and subscriptions are reestablished as soon as the Golems they are concerned with come to life again.

Other nice problems came into existence by requiring Palantiri to be transparent. In a homogeneous environment this is no problem, but we do have

computers of different types, each having its own data representation. Several solutions were considered, but in the end data is stored in the Palantir's local format, and is sent over the network in the requesters local format. This has one large disadvantage: When a new type of machine enters our environment, we will have to recompile all Palantiri after adding the appropriate conversion routines.

V. CURRENT STATUS AND PLANS FOR THE FUTURE.

The implementation of Palantiri on the Unix systems has been completed. They have been tested under various conditions and seem to be running fine. Documents are available with the functional specifications of the Palantir itself and the Palantir access library (ref [2,3]).

In a makeshift setup though, performance tests have been done, and even in a situation with many variables (20000) spread over 200 Golems running on few machines (4) and a high load in subscription reports (>100/second) the system kept running nicely.

As a nice side effect, it seemed possible, even simple, to create a C shell environment in which the entire Palantir database can be accessed very much like the Unix file system tree. Hence we now have commands like *pls*, *pcd*, *pget* and (slightly more difficult) *pput*.

The control system of the new experimental facilities will be based on the Palantir concept.
The control system for the linear accelerator and the stretcher ring is based upon the 'old' database concept. Porting this system with all applications involved to the Palantiri database is a large project. Because of operational and manpower aspects, it is impossible to do the port in a short period of time. To be able to convert and test existing applications, golems have been made that map the existing control database onto the new Palantir domain. Doing this, the two domains are connected and it is possible to convert first all applications and then replace the 'old' database by the new one.

VI. REFERENCES.

[1] W.P.J. Heubers and R.G.K. Hart, A Workstation-based Operator Interface to Control MEA. Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems, Vancouver 1989.

[2] B.J.Tummers, Palantir Functional Specifications, internal NIKHEF document.

[3] B.J.Tummers, Palantir Library Functional Specifications, internal NIKHEF document.