PROGRESSES ON !CHAOS DEVELOPMENT*

Luca Gennaro Foggetta#, Claudio Bisegni, Simone Calabrò, Paolo Ciuffetti, Giampiero Di Pirro, Giovanni Mazzitelli, Alessandro Stecchi, INFN-LNF, Frascati (Roma), Italy
Luciano Catani, Domenico Di Giovenale, Federico Zani, INFN-Roma II, Roma, Italy

Abstract

!CHAOS (Control System based on Highly Abstracted and Open Structure), the new control system architecture proposed by INFN is in development and some parts of it are now under test on the DAØNE and SPARC complexes. Although the main goal of the !CHAOS project remains the accelerator-based research facility proposed for the Cabibbo Lab and the SuperB accelerator, other applications are under study in order to adapt this new design to the needs coming from different fields, with a growing interest from many companies. Recent developments, tests results, potential applications and future project's plans are presented.

INTRODUCTION

Nowadays, high speed communications having low number of handshaking and status words are the focus of new internet community technologies.

Having taken care of the obvious differences, this "distributed cybernetic systems" is one of the technologies we took as reference in identifying a new strategy for building a new paradigm for a distributed control system (DCS) [1]. It consists of different levels of abstraction in between the hardware and the human endpoints, ensuring the independence of the DCS [2] remaining part from the single controlled element. This is, obviously, quite difficult to achieve if a large number of unhomogeneous elements has to be integrated in different and dynamic configurations. This problem can even grow enormously if the whole system has to be maintained for years. To reduce complexity and diversity of interfaces to hardware, we focus on Ethernet as the !CHAOS preferred bus due to its unquestioned robustness and flexibility: standard !CHAOS I/O hardware will include embedded computing and networking capabilities. Moreover, to ensure seamless device integration, we will avoid proprietary busses, software and technologies. Instead we are interested in emerging technologies borrowed from cutting-edge Internet services. So far, our main effort has been dedicated to designing an accelerator control and data acquisition system (DAQ) based on a synergic combination of network distributed object caching (DOC) and a non-relational key/value databases (KVDB), these software technologies matching the requirements of our innermost abstraction layer. The performances we expect from these technologies are quite intuitive: speed of data storage and retrieval for the distributed caching, data throughput and queries execution time for the database and, especially, how much this performances can benefit from their inherent scalability. The conjunction of the overall hardware and the software performing the control is achieved by defining our custom dataset (DS) for each actor (endpoint) in the !CHAOS framework. In the DS there is a unique identifier of each instantiation that belongs to one element of every hardware or software classes. The Metadata Server is the logical element that ensures the system configuration, the startup communications among endpoints; it also hosts the syntax and semantic of DS.

!CHAOS

!CHAOS intends to provide a solution that naturally allows: redundancy of all its parts, intrinsic scalability, minimization of points of failure, hardware hot-integration and auto configuration. It will be suitable for the slow control of a large number of apparatuses of different size and complexity. In !CHAOS the strategy adopted for the communication between components might be compared to the rules followed in the social forums: if someone has something important to say, it is on its own duty to send it to a generalized cache of information. Vice versa, if some component wants to gather information about the status of another endpoint, it is again their duty to fetch the information. Practically speaking, we distributed the endpoints inter-processes, relying on their intention. These requirements suggest organizing the data flow as an autonomous (and asynchronous) data pushing and retrieving to and from a central fast collector (Fig. 1).

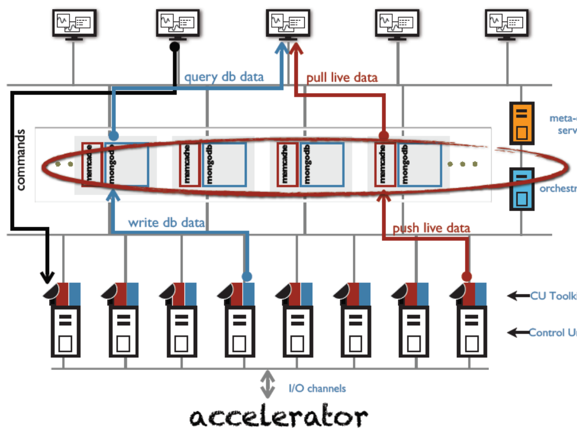

Figure 1: Schematic of !CHAOS.

So, our effort was to create an infrastructure, in someway similar to an OS, where the standard of sending/retrieving data, the endpoint proxy software and

* Work partially supported by AIDA grant no. 262025 and SuperB coll.
# luca.foggetta@lnf.infn.it

the information collecting/distributing service, are well defined and consistently providing a communication solution. Also, we choose existing software products (i.e. RPC, serialization, data storage KVDBs, …) on the base of their community support and reliability.

As for the OSs, we divided !CHAOS in abstraction layers. They are completely developed in C++, with low level standard and POSIX calls both to ensure cross-compilation in different operating system and require the minimal effort to maintain it in different kernel versions. We have successfully compiled !CHAOS on MAC and Linux OS's and a minimal custom distribution (starting from Debian) is under debugging to be installed in our ARM test boards. The RPC message flow to and from endpoints is accomplished by using key-value paradigm, for now implemented with BSON[3] specification. We have to point out that !CHAOS employs a distributed object caching for the real-time data access (Live Database) based on MEMCACHED[4] and a KVDB for data archiving (History Database) based on MONGODB[5]. These two components provide the caching memory service of all the system, where each information comes from a single endpoint that tags the information itself. This one can be retrieved by any of the other endpoint.

First Abstraction Layer (FAL) - Data Message

This abstraction layer works on key-value pairs. The information are exchanged between two nodes, accordingly with these general concepts:

- a node is an internal addressable entry registered into a MessageBroker (a kind of message router that abstracts the client, the server and the dispatcher to every classes that want to export a method)
- each MessageBroker publish its services in a tcp port [ip:port]
- a node is an entity that registers some actions using MessageBroker and it is identified by a string (alias or first part of uuid)
- an action receives and returns an RPC object managed by MessageBroker
- Software tools (i.e. RPC controller, databases, input/output from hardware…) are independent

This layer acts as Data Proxy Service, implementing the following operations on data:

- push live/history data for a device
- create index on device::attribute::{action}
- delete index on device::attribute
- retrieve data with logical operation on attribute and index

We use two kind of data forwarding:

- Message, the information are sent to a remote host
- Request, the information are sent to remote host, and an answer is awaited

In either types of data forwarding, a result of submissions is sent to the client when a server has received the data. A message can bring a sub message, which is dispatched after the main message has been successfully executed.

The data serialization protocol adopted for !CHAOS is BSON, a binary-encoded JSON (JavaScript ObjectNotation) documents, optimized for fast storage performance. We choose BSON for the wide community support and for its intrinsic simplicity. In the duty of the innermost abstraction layer, BSON well suits in, taking the key-value couples mediated by intrinsic information of data-type (letting some fundamental possibility in customization of the latter). BSON has an overall structure extremely clever and suitable for the different needs related to the message delivery that can arise in an accelerator machine complex.

We have the intention to customize some properties of the BSON specification to introduce new kind of data type, increasing the uniformity on data transfer. The BSON protocol is now under stress test, even though it remains a possibility to use other serialization methods with minimal effort, given the existence of this abstraction layer.

Second Abstraction Layer(SAL) - Common Tool

Having defined the messaging layer, we generalized the way !CHAOS interfaces to clients and GUI on one side, and front-end to the other side. For the human readout and data representation we developed the User Interface Toolkit (UI) while for diagnostic/DAQ/control hardware we developed the Control Unit (CU). In addition we introduced the concept of Execution Unit Toolkit (EU) intended for computational software service (as dynamic apparatus control and data calculation service). These toolkits divides essentially in three parts, but using the FAL, their synergic work is assured by the Common Toolkit C++ software.

The Control Unit (CU) abstracts the !CHAOS resources to the device drivers developers. It completely manages data, controls scheduled operation and commands flow, back and forth from the !CHAOS (live and histo DB's) to the hardware processes. The device's programmer is only asked to develop the driver for the specific controlled hardware using the CU API's (DMP = Device Management Plugin). Once the device driver is integrated with the CUToolkit, the CU will be installed directly on the HW, if it hosts an embedded computational unit, or in a host that controls HW with an external bus. So, CU works as local process owning six elementary C++ methods, well described in [2], implementing the six standard operations in the CS. The principal methods are the "defineActionAndDataset", where the CU is instructed from !CHAOS on which hardware is attached to and the "run" method, which is called automatically from the CU for controlling and polling the HW. This method also collects the Live data and send it MEMCACHED via the FAL. One or more instances of

CU can run simultaneously, though completely independent, in the front-end controller. Each of them will be dedicated to a particular device or a family thereof, specialized for that particular component by means of the unified DMP.

So, the machine data flow is stable maintained and refreshed by CU's scheduled and/or asynchronous calls and the LiveDB and HistoDB should be used for the data displaying and machine controlling. The displaying feature is accomplish by UI, with the same working structures of the CU but devoted to manage the synchronization in the presentation and in the packaging of data. The latter is in charge of collecting data into usable and dynamically created logical groups of information. We have produced a minimal group of API for retrieving data with fast refreshing, obtained with a refresh service, where the data displaying processes submit a registered fetch action in MEMCACHED. For the graphical user interface, we use QT environment, even if a Data Display Plugin (DDP) is the next step to allow data display and analysis with the mostly used engineering and scientific software package (like ROOT, MATLAB®, LabVIEW®) in the same parallel way of the DMP. The most recent development has been dedicated to the development of the EUToolkit. By implementing the same abstraction rule of the CU an the UI Toolkit, the EU acts as distributed computational service, mostly used by UI for data information reduction to quality parameter and for the control service of the !CHAOS machine workflow. In a naïve way, the EU instantiation on a computational node (real or virtual) will work as a particular computational algorithm or will act as a node in the transfer function of a loop control one. Its functioning is exactly the same of the CU except for the hardware in its frontend, with ideally the same set of methods and DPMs. So, again, the EU programmer has only to develop the algorithm; he doesn't need to care on how !CHAOS fetches the input data and store the output.

Third Abstraction Layer (TAL) – Plugin Modules

This is the last interface layer connecting the external world to the !CHAOS framework: some example are the DDP and DPM. Even though these are hybrid application (i.e. they are logically different to each other since their constituents are changed as function of the different external application in touch with), their paradigm and low-level communication are intended to be the same. We have developed two kind of DPM at the CU layer: one that extends CU calls directly in C to the hardware attached to the host computer, the other very interesting CU extension works with its mirrored DPM created in LabVIEW®. The communication channel between the two sides of this DPM is provided by means of Unix pipes (tested on Linux and MAC), now under test. Pointing out that the CU extensions are different because of different prototypes and third party software, these TAL elements aim to make easy the designing of the instruments driver. The programmer has only to know the DS and the algorithm it has to develop, using the external side of the DPM that extends the previously described six basic methods. For the LabVIEW® example, he only needs to create six different VIs starting from templates that will include sub-Vis for serialization of LabVIEW data structures into BSON strings and vice versa we developed for this purpose. For the DPP, we wants to perform the same step, preferring the low level communication approach between The !CHAOS side and the external application one, as used in the DPM.

TEST OVERVIEW

In the previous articles we already mentioned that some !CHAOS components were under test in existing CS infrastructure. The CS presently driving the DAΦNE accelerator has been used for testing one of the core components of !CHAOS: the *live-data* caching. The MEMCACHED server, running on a Linux box with 100 Mbps Ethernet network interface, stores the data of the ICE (Ion Cleaning Electrodes) consisting of a data structure of 64 bytes. The dataset is pushed by the front-end controller with a frequency ~ 100 Hz. In one year of continuously running test, we report no fault on MEMCACHED. We have planned to extend the first operative version of !CHOAS to the Beam Test Facility area in the DAØNE infrastructure. The test also confirmed the independency of the performance on the number of fetching consoles (up to 7) and the MEMCACHED server CPU load never exceeded 1% and a memory usage of ~ 0.1%. Similar tests have been performed at SPARC where a MEMCACHED server is used for communication between components of the CS by means of a common dataset area. The first class of object migrated to the MEMCACHED server was the SPARC camera subsystem. We chose this class of elements because of its very large dataset; each camera is at least 640x480 with 8 bit. After this first installation we made some speed tests demonstrating an increase of the transfer speed compared to the previous client/server communication scheme. Moreover, they confirmed that data transfer is independent from the number of client consoles fetching the image from the MEMCACHED server. The success of this installation convinced us to adopt also for the rest of SPARC elements this communication solution that uses MEMCACHED as data server. Currently we are in stable running condition with the upgraded system.

REFERENCES

[1] G. Mazzitelli et al., "High performance web applications for particle accelerator control systems" IPAC'11, WEPC142, pp.2322.

[2] L. Catani et al., "Exploring a new paradigm for accelerators and large experimental apparatus control systems" ICALEPCS'11, pp.1557.

[3] http://bsonspec.org/; http://www.json.org/

[4] http://memcached.org/

[5] http://www.mongodb.org/