FSDB

Technical documentation

Sébastien Rastegar – Guillaume Estocq – Ludovic Laguerre – Michel Riedel – Loïc Doyen

EPITECH INNOVATIVE PROJECTS

www.fsdb.fr
www.epitech.eu

About this guide :

The purpose of this documentation is to allow developers to understand how FSDB works and to continue to work on this project.

To do this, it explains, using diagrams, the various interactions between the system and FSDB. As an addition to it, a user guide is available and directed to users.

Release history :

Release	Date	Modification
1.0	04/04/2008	Creation
2.0	21/10/2008	Recast graphics and update content
3.0	05/01/2009	Add limits and bugs, referential and recast total

Table of contents

I – Project presentation:

FSDB is a command translator. This project mainly aims to able a beginner in the UNIX world, to manage a database after determining or copying its structure, only with the shell commands this person learned to deal with the file system.

The main thing about this project is its conceptual aspect. Its goal is to ease the use of databases, while offering a performance level at least equal to what is currently used.

The database will be mountable as a filesystem, and managed as is by the user, with all the advantages that this implies:

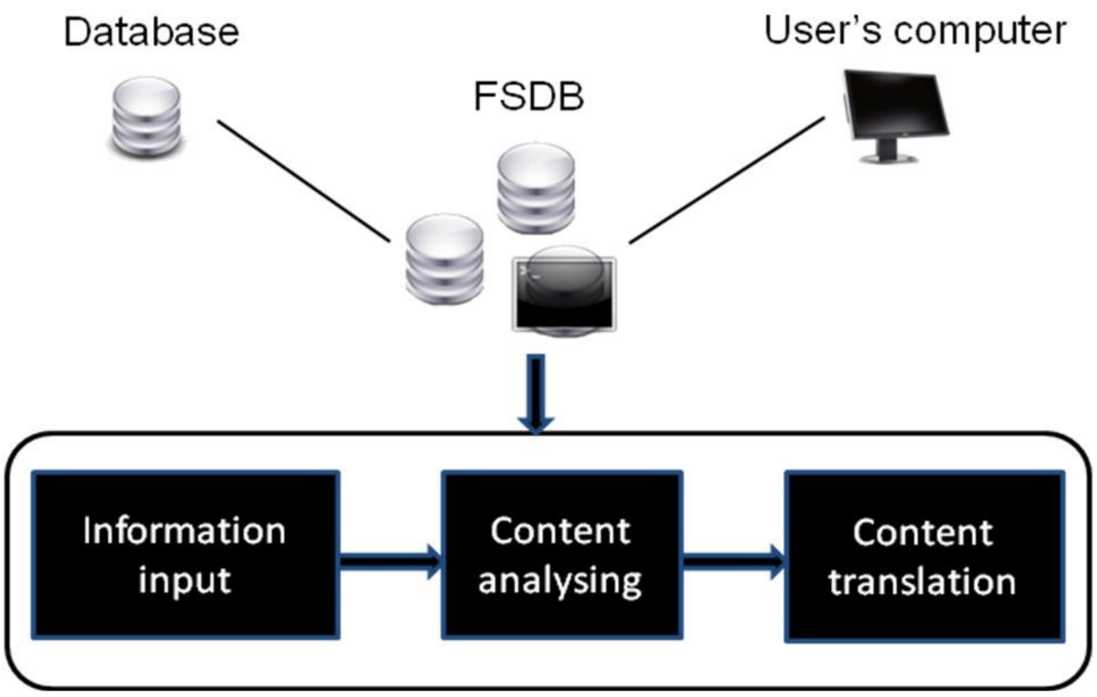
· Time savings

· No need to learn a new language

· No need for a programmer to deal with the communication part with the database in his programs, as it will be available through the shell

Every user able to use a Filesystem* under UNIX and to understand databases fundamentals is concerned.

II – Diagram of operation :



This diagram shows the various steps necessary to FSDB to process a database.

Information input	This step is the passing of the parameters the application needs to run correctly
Content analysing	This step is the information given by the user on the type of database
Content translation	This step concerns program execution

III – Prerequisite :

Our application is usable in a Unix / Linux environment but is compatible with any type of database (MySQL, Postgres, Oracle), as soon as a library is implemented to ensure interoperability between FSDB and the type of database used.

IV – Technologies used :

a. Languages used :

- C/C++: First the project was developed in C, then for the sake of modularity it was decided to migrate the code in C + +
- SQL: Language to manipulate a database.
- Perl/Gtk 2: Whole libraries to achieve a GUI .

b. Tools used :

In order to realize our project, we will need tools to design, test and make it available to others:

- Eclipse: An Intelligent Development Environment (IDE), to develop the project itself

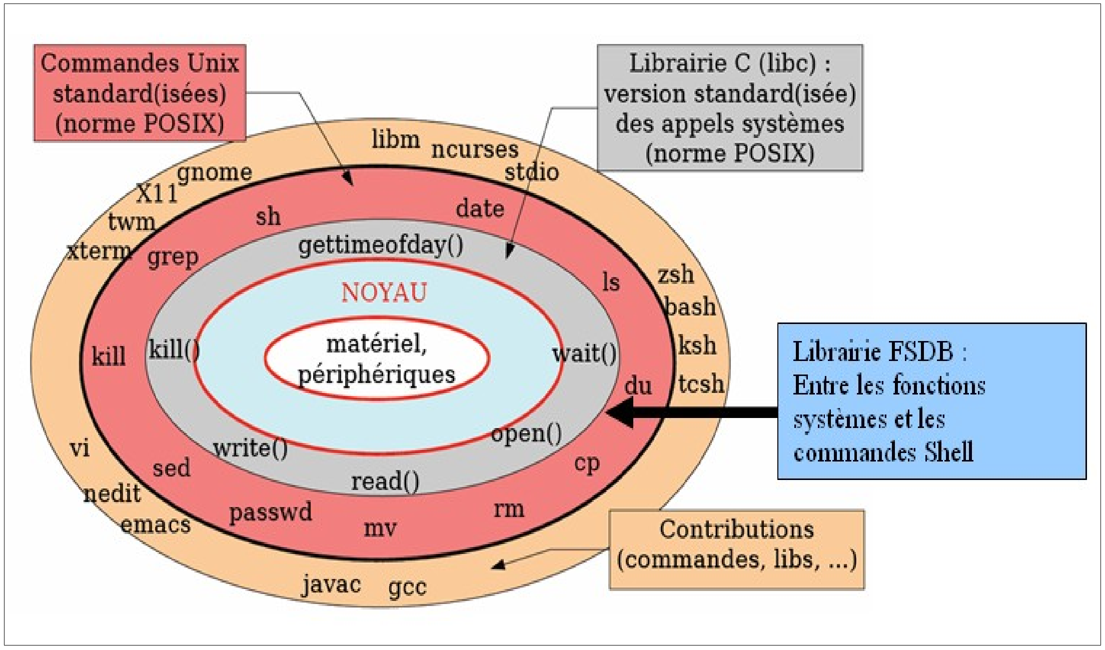
- AutoMake: A generator to compile our program that will end up with time savings when testing and eventually correcting the project.

- Esvn: An SVN client which will permit to share the FSDB source files and documentations.

V – FSDB in its work environment

1. FSDB within the operating system

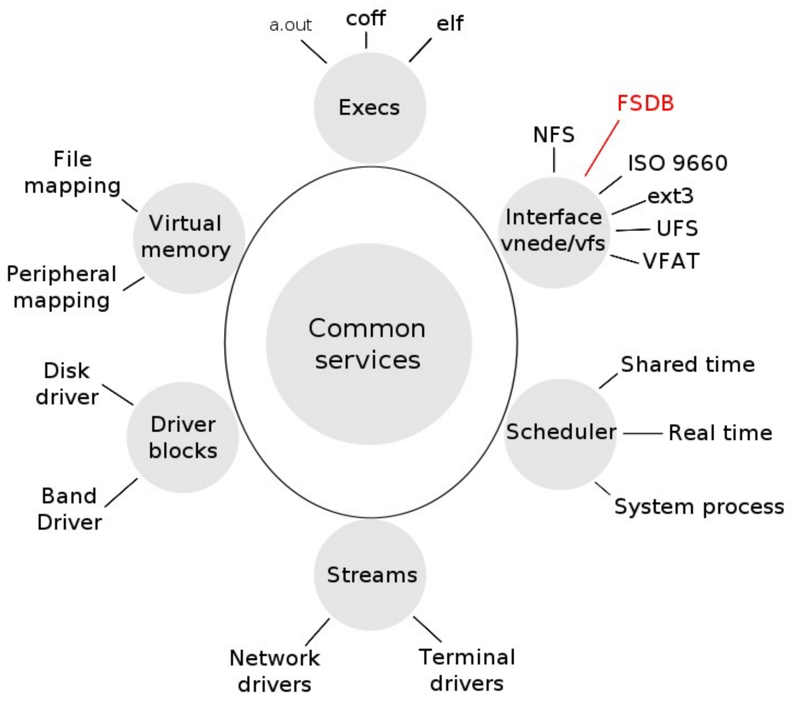
FSDB is a program that interacts directly with the system, so it is important to know the different layers of its Unix operating system to understand the operation of FSDB.


Commandes Unix standard(isées) (norme POSIX)
Librairie C (libc) : version standard(isée) des appels systèmes (norme POSIX)
libm
ncurses
stdio
gnome
X11
twm
xterm
grep
sh
date
gettimeofday()
NOYAU
matériel, périphériques
ls
zsh
bash
ksh
tcsh
kill
kill()
wait()
du
Librairie FSDB : Entre les fonctions systèmes et les commandes Shell
vi
sed
write()
open()
cp
read()
nedit
emacs
passwd
mv
rm
Contributions (commandes, libs, ...)
javac
gcc

As stated in the general presentation FSDB is a command translator that it's whyhet is not surprising to find the library FSDB between UNIX commands layer and layer with C-library. Therefore it is possible to link a shell command at functions FSDB which use the C-library

2. Interaction with the services of operating system

An operating system provides number of services that allow the interaction between it and the programs. Therefore we see that FSDB works with the vfs interface (Virtual File System).



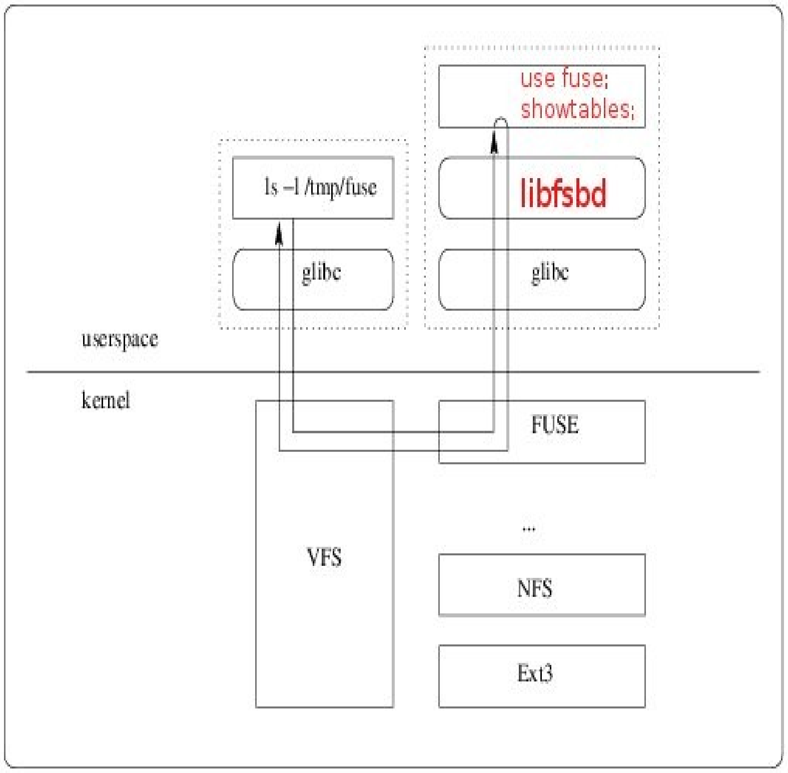
3. FSDB and libraries

We have seen that FSDB is located along the C library, so to set up a database, FSDB uses call provided by API of different database. However, to mount a database FSDB uses FUSE.

We defined the main system calls in a pre-defined format given by FUSE's API. We can change the properties of those commands in SQL request by using the libsql.

All useless commands are desactivated.

We are using FUSE as a library. It allows one to create a mount point. In this mount point we setup FSDB. FUSE delivers an API which provides different tools for the definition of the system calls.



Therefore the flow graph shows libraries traveled when the user makes a LS which is a show tables in SQL.

VI – Limits and bugs know

A series of tests were conducted to test the stability of FSDB, its limitations and identify potential bugs.

Test 1: Displaying a table

The test consist to repeat the command several times. It shows if FSDB is tough to repeat the commands.
During this test the same script was launched from another terminal and this allows the if there are problems of access.

Result: No problems found

Test 2: Create a table

The objective of this test is to show the stability and reliability of the program with a script, he will create the 400 tables 2 columns in the database.

Result: OK, noted that treatment can be long, depending on the hardware of the server or the method used to host the server (Example : VM).

<u>Test 3: creation / deletion of a table</u>

We test the performance of two identicals script launched in parallel, he remove the table and create her. This is a test to see if FSDB is well managed access.

Result: OK, it is impossible to create or delete the same table when a user is currently works on

<u>Test 4: Connecting to the databases</u>

We will try to connect to the database with a username that does not identify by the base.
Result: OK, shell returns our an error 'Permission not granted', only root can access to the folders where databases storage.

VII – Multiple databases

Our project runs when installing MySQL and Postgres, however, a shared library system is set up to ensure the compatibility of the most types of databases as it has a C++ API.

This system allows to make our project evolve, as everyone can implement its library for FSDB, and therefore manage of its preferred type of database.

In Annex 1 is provided the API to interface its library with FSDB.

VII – Referential

Document	Acronym	Status
schedule of conditions	CDC	OK
Study of the existing	EDE	OK
Study Report detailed	RED	OK
Architecture assesment	AA	OK
Technical documentation	TD	OK
User guide	UG	OK
Summary of activity	CRA	OK
Quality insurance plan	QIP	OK

A portion of these documents are available on our website: www.fsdb.fr . To grant access to other documentation or any other suggestion thank you for contacting us from the website.

ANNEXE 1 : API to implement a shared library for database

```
/*
 * CDBlib.h
 *
 *  Created on: 7 oct. 2008
 *      Author: ludo
 */

#ifndef IDBLIB_H_
#define IDBLIB_H_

#include <map>
#include <string>
#include <vector>

#include <syslog.h>

#include <SQLQuery/SQLQuery.h>

typedef struct s_off{
 unsigned long off_row;
 unsigned long off_str;
}t_off;

typedef std::map<std::string, std::string > ConnDB;
typedef std::map<std::string, std::string> colDesc;
typedef std::map<std::string, colDesc > tblDesc;
typedef std::vector<std::string> tabString;

using namespace std;

class IDBlib {
public:
 virtual void connectDB(const ConnDB&) = 0;
 virtual void closeDB() = 0;

 virtual void getTblDesc(const string&, tblDesc&) = 0;
 virtual void getColDesc(const string&, const string&, colDesc&) = 0;
 virtual bool dropColumn(const string&, const string&) = 0;
 virtual bool dropTable(const string&) = 0;
 virtual bool addColumn(const string&, const string&, const string&) = 0;
 virtual bool createTable(const string&) = 0;
 virtual bool renameColumn(const string&, const string&, const string&) = 0;
 virtual bool renameTable(const string&, const string&) = 0;

 virtual void showTables(tabString&) = 0;
 virtual void showColumns(const string&, tabString&) = 0;

 virtual unsigned long fileSize(const string&, const string&) = 0;
 virtual unsigned long fileSize(const SQLQuery &query) = 0;

 virtual bool tableExist(const string&) = 0;
 virtual bool columnExist(const string&, const string&) = 0;

 virtual int  readColumn(const string&, const string&, char *, size_t, off_t, t_off*) =
0;
 virtual int  readColumn(const SQLQuery &query, char *, size_t, off_t, t_off*) = 0;
};

typedef IDBlib* (*create_t)(const ConnDB&);
typedef void (*destroy_t)(IDBlib*);

typedef struct s_dblib {
 void  *handle;
 create_t create;
 destroy_t destroy;
}fsdb_struct_dblib;

#endif /* IDBLIB_H_ */
```