

Dialogue expert system at command line interface – DES – CLI Ryahovetz

Iv Vasilev¹, N Nenkov^{2*}

¹University of Veliko Turnovo "St.Cyril and St.Methodius", Veliko Turnovo, Bulgaria

² Konstantin Preslavsky University of Shumen, Shumen, Bulgaria

*Corresponding author's e-mail: naydenv@gmail.com

Received 1 May 2015, www.cmnt.lv

Abstract

The article describes the construction of a dialog expert system that supports the work of the system administrator. In its operation it uses the command line, which greatly improves its functionality and flexibility.

Keywords: Expert system, Command Line Interface - CLI, Data Hoard - DH, Logical Unit - LU, Dialogue Interface - DI

1 Introduction

We are witnessing continuous development of new information and communication technologies, continuous improvement of computers and the emergence of a variety of "smart" devices that facilitate everyday life.

Many commercial companies and existing open source communities compete in attracting more customers with a variety of innovative hardware and software developments. The race is to achieve simpler and easier to use devices and programs that guarantees them a larger share of the market. But despite the strong development of these technologies, we can not ignore the fact that all these decorated, aesthetically appealing and easy to use interfaces obscure the existing functionality of the products.

When additional flexibility and support are needed they do not always do well, and even violate the operating system on which they are installed.

2 Exposition

This is the place to mention the well known from the past command prompt, shortly called CLI, which reliably serves professionals in this area. For the inexperienced user it is difficult, but there are strong advantages for professionals compared to the graphic environments. Generally this environment is used actively and will be used actively at server level, especially in **UNIX** and **UNIX like** platforms and even in the **Windows** server platforms in all business environments, banks and others. It is not outdated, practically the command environment is the foundation and the GUI is the add-on, consuming additional resources, which the business at server level does not want, cuts and pays well to specialists to work exclusively on **CLI**. We must also mention, that, in many server platforms, it is even impossible to launch **GUI** because the architecture does not allow it: namely "**RISK Reduced instruction set computing**" which relies on productivity and security and they have not even written a graphical interface or if they have, it will strongly tend to the command view with the purpose of stability. Each layer adds extra risks to the

security and potential crashes.

The table shows a part of the advantages and disadvantages of this type of environments.

TABLE 1 Advantages and disadvantages of CLI

Advantages	Disadvantages
Extremely stable and fast interface for communication with the operating system.	Extremely scary interface leading to total denial for working with such an environment
Extremely low consumption of system resources	Thorough knowledge of the particular operation system.
Full control over hardware resources and the operation system, having the rights needed.	Thorough knowledge of the commands and the ways to work with them.
Full control over the running processes, having the rights needed.	Often made mistakes by typing the different commands, arguments and compilation of strings of commands
Ability to perform complex tasks.	High level of risk if there are unlimited rights, to delete significantly damage individual programs or even the operation system itself.
Extremely fast communication, connection, transfer, configuration and control from one environment to another	The presence of large amounts of string information on the screen leads to confusion and missing valuable information. Especially if the operating person does not know how to retrieve it again.
Easy possibility of scripting and automation of routine tasks.	The need to work primarily with the keyboard and in very few and limited cases – with the mouse.
Full control over standard streams stdin, stdout, stderr; opportunity for easy routing and localization in logs formatting for subsequent filtering and monitoring.	

Certainly the topic on the advantages and disadvantages of one and the other environment is not limited to that described in Table 1.

The purpose of the described system is in three main directions:

1. To offer advices and expert solutions.
2. To assist the work with command interpreter with key information.
3. To teach itself and learn from the experience of the users.

3 Architecture Of The System

The system will consist on the principle of three-layer model with clearly identifiable modules. Namely **DH** – **Data Hoard**, **LU** – **Logical Unit** and **DI** – **Dialogue Interface**.

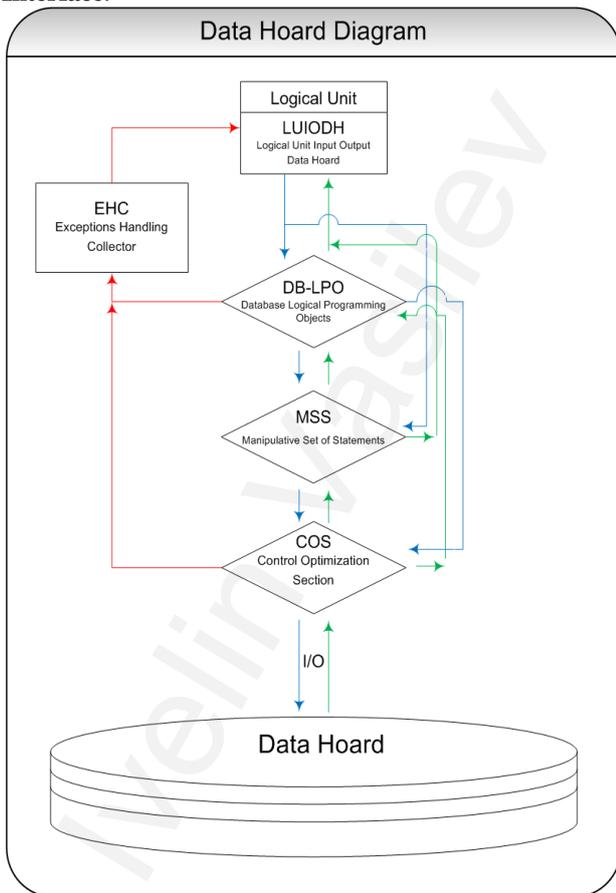


FIGURE 1 Module for storage and control of data

1) **DH - Data Hoard**. Module organizing storage of input - output data and control of information. This module can be divided into the following sub modules:

- Physical database structure. A set of **DDL** statements creating the necessary set of relative tables providing space for storage of information.
- **DB-LPO Database Logical Programming Objects**. Logic programming objects within the database. Each database offers its own language for writing and implementation of logic within the database for the purpose of fast operation, control and manipulation of input - output data. For example, *Oracle PL/SQL, MySQL Stored Procedures, PostgreSQL PL/pgSQL and etc.*
- **MSS – Manipulative Set of Statements**.

Manipulative set of statements serving input – output data. A set of predefined and optimized DML queries and their routine call.

- **EHC – Exceptions Handling Collector**. A segment for collecting and processing errors.
 - **COS Control Optimization Section**. A set of indexes, keys, triggers and others.
- 2) **LU – Logical Unit**. This module has the specific task to perform only and exclusively logical operations

- **LUIODH (LU – I/O – DH) – Logical Unit Input Output Data Hoard**. Segment for interconnection with the **DH** module. The connection with the **DH** module must be made through a main duplex controller **LUIODH** for controlling the input – output data, divided into two simplex sub-controllers – one controlling and manipulating the input data **Logic Input Data LID** and one controlling and manipulating the output data **Logic Gets Data LGD**.

- a) **LID – Logic Input Data**. It can only receive data from its own main controller and communicate with the data hoard module **DH**. (**LUIODH → LID → DH**).
- b) **LGD – Logic Get Data**. It can only receive data from its own main controller and transmit them to the other main controller within the logical unit **LU**. (**LUIODH → LGD → LUIODI**)

- **LUIODI (LU – I/O – DI) – Logical Unit Input Output Dialogue Interface**. Segment for interconnection with the **DI** module. Connection to and from the **DI** module is made similar to the segment before through one main duplex controller (**LU – I/O – DI**) **LUIODI** and two simplex sub-controllers **Logic Get Request LGR** and **Logic Response to Dialogue LRD**. Again there is a distribution, on which controller the data transfer to be performed.

- a) **LGR – Logic Get Request**. It can only receive data from its own main controller and transmit them to the other main controller within the logical unit **LU**. (**LUIODI → LGR → LUIODH**).
 - b) **LRD – Logic Response to Dialogue**. This unit can only receive data from its own main controller and communicate with the dialog interface. (**LUIODI → LRD → DI**).
- **SB – Scheduler Batches**: Segment for scheduled execution of routine tasks with the purpose of periodic processing of data with the purpose of updating, adding and modifying, reminding, triggering events and other.
 - **LEU – Logical Expert Unit**: Segment performing only complex logical operations. Connection to it is possible only through the two main controllers **LUIODI** and **LUIODH**.
- 3) **DI – Dialogue Interface**. Dialog interface for interconnection between end users and the system. This module has the main task to accept requests from

the end user and return the necessary information.
 - **I-face - Input Interface.** Input interface receiving requests from users. The main task is to lightly check for the validity of the requests, arguments and parameters.

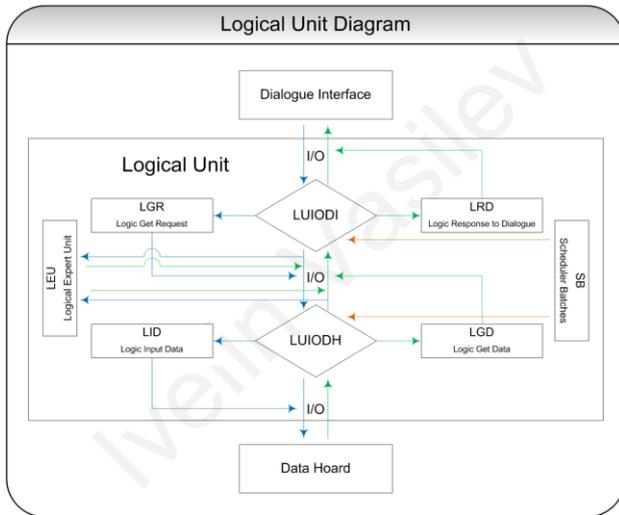


FIGURE 2 Logical unit

- **SU - Security Unit.** Security unit, which has the task to accept data from the **I-face** segment and make a thorough check in order to prevent blocking of the system due to improper or unauthorized use, attacks and others.

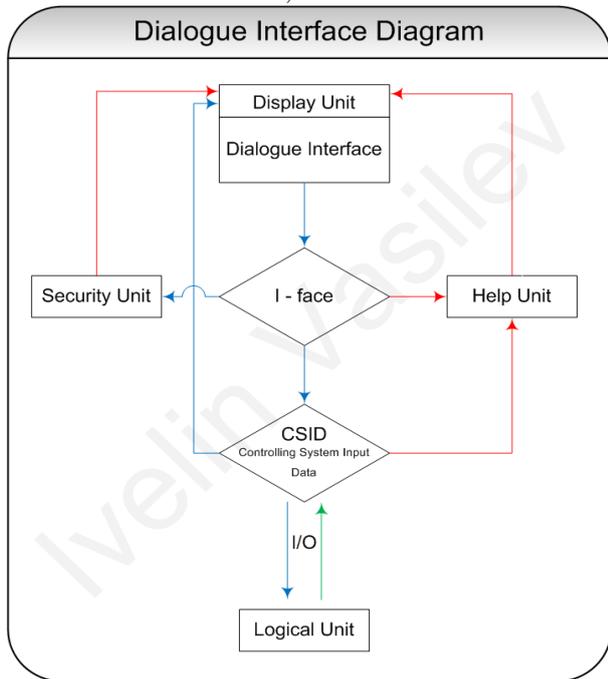


FIGURE 3 Dialog interface

- **HU - Help Unit.** In case of incorrect input data detected by **I-face** or **SU**, this unit is triggered to suggest or indicate the basic rules for using the system.
 - **CSID - Controlling System of the Input Data.** Based on a single main controller that accepts requests passed by the **SU** unit and transmits them with almost no change to the logical unit **LU**.

- **DU - Display Unit.** Unit for presenting information to the end user by simply printing text on the screen or loading a set of templates for more complex queries.

4 Functioning Of The System

The operation of each system is based on the occurrence of certain events. In this system the events mainly occur from the dialog interface **DI** or **SB** segment for scheduled tasks.

We assume that an event has occurred from the dialog interface, ie a user has given a command to the system and expects a response. We assume that the command is correct and everything is fine at security level, it passes to the main controller of the dialog interface "**CSID**". As described above, it turns with the information submitted, almost unchanged, to the logical unit "**LU**" to its main controller "**LUIODI**". The controller is duplex, because it works both ways, it can receive and transmit, the task of the controller is to determine where the request came from and what is its purpose. In this case, for example, we have an incoming request from "**DI → LU**" and thus the data is passed on the simplex sub-controller **LGR** (logical unit receive data)

Received here, data are reviewed and according to the type of operation and type of data, logical operations can be triggered to transform data and interpret them in a form convenient for the next lowest module "**DH**". After formatting the data if needed by more complex logic operations, then the main controller "**LUIODI**" turns to "**LEU**" for their implementation.

Data passed on this conveyor are ready to be put on the next main controller "**LUIODH**", serving to connect to the module storing data. The task of this main module is to determine what operation is expected – retrieve information or input information. If the event is data entry, it passes on the sub-controller **LID** for input data. Its task is to check the input data for consistency, to ensure that data is not duplicated, whether the form consists with the one of the base and other routine operations. When you enter a large amount of data and if its nature allows, it can be formatted in several stacks and submitted asynchronously for faster performance, otherwise it is passed synchronously. In cases of data dependency, the asynchronous method fails and the synchronous transaction method is used. So prepared data is submitted for entry to the module "**DH**" and a record, change or deletion is performed.

Generally the implementation of the modification may be performed directly or may be passed through "**DB-LPO**" in need of further logical processing. The need of this logic at the lowest level is for achieving fast performance when processing large volumes of information and when it is necessary to perform multiple manipulations on the data in the database. This avoids unnecessary iterations with the "**LU**" logical segment.

In the optimization of the product, the most popular requests are implemented in the section "**MSS**" and are routinely invoked when matching events. The purpose of this segment is to avoid unnecessary accumulation of similar statements even if they are well optimized. The idea is one statement to be re-used as many times as possible and in need of change, either modification or

optimization, to affect the entire system. For additional control or its reduction, is used "COS" control optimization section of the module "DH", which creates, prohibits, permits or deletes a set of constants, triggers, indexes and other control-optimizing techniques within the selected database.

To explain the reverse action, we assume that an event to extract data from the system is invoked by the dialog interface. The request itself runs along the channel for input data until reaching the main controller **DH**. From there the necessary data are returned to the main controller "**LUIODH**", which distributes things to happen on the channel for retrieving data **LGD**.

The logic controller takes the raw data and makes primary processing of data and then transmits the data to the controller **LUIODI**, which in turn distributes them to pass on the sub-controller **LRD**. This sub-controller accepts the semi-processed data and applies formatting methods and tools in order for data to be presented to the user in an appropriate and understandable format. In complex dialog templates, data is sliced into stacks and returned asynchronously to the dialog interface and loaded at the locations indicated on the template. In simple consoles the data is returned synchronously.

In the description until this point, we have not mentioned the element performing scheduled execution of routine tasks **SB - Scheduler Batches**. For the proper operation of the system, certain processes must be run in the background. Such tasks can be required to calculate data in a certain period of time or in case of reaching deadlines to apply some logic; to routinely extract data from third-party systems and update the system and to output data from the system and others.

This element's task is to create an event to one of the two main modules **LUIODI** or **LUIODH**. After the event is invoked, the control shall be taken by the respective controller and "**SB**" only marks that at the appointed time a process is started and the controller took the implementation. From here on, the corresponding controller takes care of the task until its implementation and thus registers the results in the base and logs information on occurred errors or successful implementation.

5 Interaction With Third-Party Programs

Every good software should offer an application programming interface for connection to third-party programs to itself, as well as support such interfaces for connection from itself to third-party programs. These are the so-called **API Application Programming Interface**. The need for such interfaces is required by many factors such as the following:

- Encapsulation and stability;
- Establish a strict protocol for communication;
- Simplified method for access to complex systems;
- Overcoming platform and other differences;

References

- [1] Кочан Ст 1993 *Запознаване с операционната система UNIX* перевод СТОЯНОВА Р Издателство "Парафлю" ООД
- [2] Kroto H W, Fisher J E, Cox D E 1993 *The Fullerenes Pergamon* Press Oxford
- [3] Randal K M 2003 *Mastering Unix Shell Scripting*
- [4] Randal K M 2008 *Mastering Unix Shell Scripting: Bash Bourne and Korn Shell Scripting for Programmers System Administrators and UNIX Gurus*
- [5] Tallman D E, Wallace G G 1997 *Synth. Met.* 90 13
- [6] Kochan S G, Wood P *Unix Shell Programming*

- Fast performance;
- Control over input – output data;

Application Programming Interface is the fourth independent module that can be turned off and on as needed and does not affect the operability of the system if it is not active. It can be regarded as an auxiliary or extension module for expanding the scope and functionality of the described here expert system. Its more detailed description is not under review here.

6 Conclusion

The described system solves everyday tasks of administration and management of different platforms. The quality of the resulting advices depends on the knowledge entered into the system base. Further development would be in the direction of adding new modules and increasing the relevancy of its recommendations.

7 Abbreviations

DML – Data Manipulation Language
DDL – Data Definition Language
CLI – Command Line Interface
GUI – Graphic User Interface
DH - Data Hoard
DB-LPO - Database Logical Programming Objects
MSS – Manipulative Set of Statements
EHC – Exceptions Handling Collector
COS - Control Optimization Section
LU – Logical Unit
LUIODH (LU – I/O – DH) – Logical Unit Input Output Data Hoard
LID – Logic Input Data
LGD – Logic Get Data
LUIODI (LU – I/O – DI) – Logical Unit Input Output Dialogue Interface
LGR – Logic Get Request
LRD – Logic Response to Dialogue
SB – Scheduler Batches
LEU – Logical Expert Unit
DI – Dialogue Interface
I-face - Input Interface
SU - Security Unit
HU - Help Unit
CSID - Controlling System of the Input Data
DU - Display Unit
API - Application Programming Interface

Acknowledgments

This development was funded by Project RD-08-306/12.03.2015 to Shumen University "Konstantin Preslavsky", Shumen, Bulgaria.

Authors	
	<p>Ivelin Vasilev, 1975, Gorna Oryahovitsa, Bulgaria</p> <p>Current position, grades: Head of department "Installation & Deliveries" at Codix Bulgaria University studies: IT in the legal and executive power, St. Cyril and St. Methodius, University of Veliko Turnovo, Bulgaria Scientific interest: Unix Environments Publications: 2 Experience: 17 years</p>
	<p>Nayden Nenkov, 1957, Novi Pazar, Bulgaria</p> <p>Current position, grades: Vice Dean, Faculty of Mathematics and Computer Science, Shumen University University studies: Masters of Computer Science, PhD Scientific interest: Artificial Intelligence Publications: 35 Experience: 29 years</p>