UNIVERSAL DOOCS SERVER BASED ON THE SCRIPT LANGUAGE

J. Szewinski, WUT-ISE, Warsaw, Poland; SINS, Swierk, Poland
K. Korzunowicz WUT-ISE, Warsaw, Poland

Abstract

This document describes the design and implementation of the universal DOOCS[1] server based on the script language for the FLASH accelerator in DESY (Hamburg, Germany). Server works with the DOOCS, which is used in FLASH for machine control. The typical usage of this application is to communicate with the measurement equipment and control small facilities of the accelerator. The aim of the project is to provide a tool which can make the server creation easy for non-programmer users (typically physicists). The heart of the server is the script language parser which has been done using well known UNIX tools: bison and flex. The complexity of designed language is comparable with complexity of the Matlab language. Application has additional features like possibility of attaching external dynamic libraries or possibility of defining the state machines (more sequencer like). Server has been tested at FLASH and currently is used by people who wish to control their equipment via DOOCS, with the minimal effort of software development.

INTRODUCTION

Distributed Object Oriented Control System[1, 2] is developed at DESY (Hamburg, Germany), and used at the FLASH accelerator for machine control. It supports various devices and facilities, but on-going machine development requires continuous support from the control systems software side. The requirements for programmer to support DOOCS applications are following: C++ and OOP[2] paradigm knowledge, DOOCS internals knowledge, multithreaded application development experience, UNIX system programming experience and knowledge on available external resources (VME bus or reference timing system). Even average programmer needs at least few months to collect required skills. In some cases it may be a bottle neck of the development, especially for non-programmer users such as physicists. On the other hand, scientists very often use Matlab as the most natural tool for they work. This tool is based on the platform independent programming language which has simple and intuitive syntax, and does not require compilations, linking, memory allocation, pointers etc. Simple conclusion can be drawn here: *A tool which can talk with all DOOCS infrastructure, with Matlab comparable flexibility and intuitive usage would be very helpful*.

Of course, the power of Matlab is in algorithms available in numerically optimized toolboxes, not in simplicity of the language, and there is no competition on this field here.

[1]Distributed Object Oriented Control System
[2]Object Oriented Programming paradigm

GOAL

The main goal was to provide DOOCS compatible and easy to use tool for users, in a such way, that user only takes care of his application specific part, and is separated from the system part of the server itself (which is more or less the same in every DOOCS server).This is well known technique, called *separation of mechanism and policy*, it is one of the fundamental design principle in computer science (this and others principles of good software design can be found in [3]).

Beyond the ease of use, the final solution must be powerful enough to let user implement such a features like complex algorithms, solve equations, access hardware, etc. To fulfill this, it was required, first to describe all the operations that should be performed, and then server must read this this description. To do this, a programming language has been designed and implemented using well known UNIX tools: *bison* and *flex*.

SERVER

A DOOCS server represents data as *properties* (often called also *process variables*). There are many different types of properties, but most common used are single numerical value (integer or float), and a sequence of numerical values (used for plotting spectrums). From the end-user point of view it is important what is happening when data is read from or written to the particular property. DOOCS server programmer may modify or extend functionality of a property by inheriting from the base property class and modifying selected methods.

In this case, four new property classes has been defined in server project (by inheriting from the existing property classes): two classes representing single numeric value, one for integer and one for float type, and two classes representing sequence of numbers (also one for integers and one for floats). The most significant modification in all classes, was addition of user supplied callbacks execution during the property read and write operations.

Except the modification of properties, callbacks executions has been also placed in all major global server routines, which are executed on particular events (startup, shutdown, interrupt and timer handling, UNIX signal handling, etc.) Server with built-in parser for parsing callbacks script, can be compiled once, and then user is able to change almost any server behavior by editing external text (script) file, without need for another compilation.

SCRIPT LANGUAGE

As mentioned above, parser for reading script files has been constructed using *bison* and *flex*. First, the attention should be paid at the fact, that the server application is not just a simple interpreter, which reads some file and immediately do whatever was written there. There are statements which are executed during script parsing (like creating properties objects), but major work is be done later, when properties are accessed. The callbacks are not executed online while the script is processed, but it have to be done during runtime. Also it is important to remember that callbacks will be executed multiple times during server lifetime.

To solve above issues, OOP and C++ comes with great help. The language grammar (bison input) is described as an *abstract syntax tree* (or just *AST*, see [4]). For language statements (AST nodes), which have to executed later, it fits very well to simply return instead of numeric value, a pointer to the object, which will keep all information about that particular language statement, and will provide method like *evaluate()*, which will analyze kept data and return proper numeric value on demand. Grammar rules may have also multiple definitions, and OOP polymorphism is very useful in this case, the following example will show it. The simple DOOCS property can be declared in designed language in the following way:

```
TEST_PROPERTY
{
    read
    {
        y = a * (b + 5);
        return y;
    }

    write
    {
        # no writting support
        return 0;
    }
}
```

Figure 1: Example of simple property declaration in designed language.

The example code in Fig. 1 declares a property, which on the event of read calculates the following equation:

$$y = a * (b + 5) \tag{1}$$

The AST of above equation is following is shown in Fig. 2.

The grammar for parsing this example is shown in Fig. 3 (simplified bison code).

The grammar item representing the expression (`expr`) has multiple (alternative) definitions (for different arithmetic operations, variables, etc), and it is also recursive rule, because is use itself (`expr`) in own definitions. In this case, it is convenient to provide a base abstract class

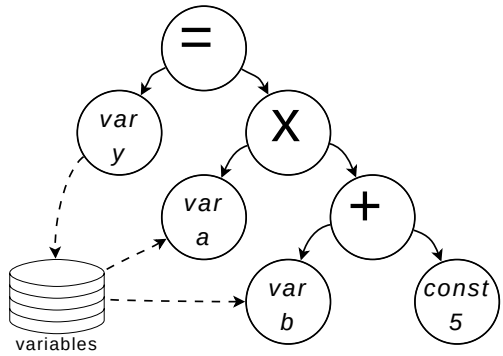

Figure 2: Example syntax tree.

```
assign: text '=' expr { $$ = new c_assign($1,$3);   };

expr: '(' expr ')'   { $$ = $2;                     }
    | number         { $$ = new c_expr_const($1);   }
    | text           { $$ = new c_expr_var($1);     }
    | expr '*' expr  { $$ = new c_expr_mult($1,$3); }
    | expr '+' expr  { $$ = new c_expr_add($1,$3);  };
```

Figure 3: Grammar rules for parsing equation (1).

`c_expr` with an pure virtual method `evaluate()`, which will be implemented in all derived classes `c_expr_var`, `c_expr_mult`, etc. Regardless, which definition of `expr` will match, always an object of class derived from `c_expr` will be returned, which can used without wondering of which class it really is.

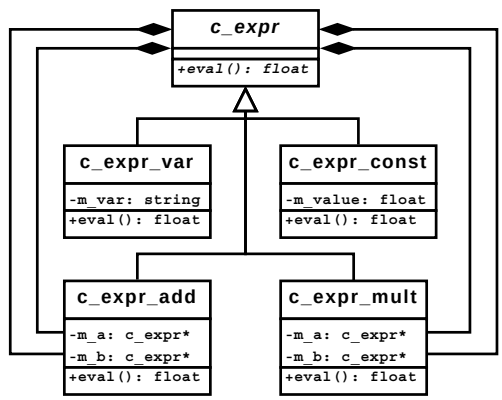

Figure 4: Class diagram for the `c_expr` class family.

Each descent class, which object is returned in answer to a recursive grammar rule (`c_expr_add`, `c_expr_mult` in the example), has member pointer(s) to one (or more) objects of base class `c_expr` (again without knowing what kind of object it will be finally). This approach dynamically builds trees of objects during script parsing, where each object represents exactly one node from AST of given input. Objects are linked (via pointers) in such a way, that evaluation of the root node will cause recursive evaluation of all objects in the tree, which will result in taking particular actions or performing calculations.

FEATURES

Designed language has most of fundamental elements of programming languages and own specific features:

- most ANSI C operators available
- variables (local and global scope)
- support for vectorized operations (like multiplying whole table by a value)
- flow control statements (conditionals and loops)
- procedures (functions)
- most of functions from `math.h` available as an built-ins
- DOOCS client functions available as an built-ins (communications with other servers)
- execution time measurement routines (based on `gettimeofday`)
- support for defining finite state machines and sequencers
- support for loading plugins (custom user dynamic libraries)

RESULTS

Presented solution has been developed at DESY (Hamburg, Germany) and used for various applications in FLASH accelerator, especially in the prototyping phases when there is lot of changes done in the short time. It was used for LLRF and beam control mostly. Created servers had their script files of over 1000 lines, and has been created also by people with zero experience in DOOCS C++ programming, in the time much shorter than time need to learn and understand DOOCS in traditional way.

CONCLUSIONS

This method of software automation, may be used with different technologies (not only DOOCS, not only control systems) to minimize hard-coded implementations, maximize flexibility by applying functionality (even complex) from external source during runtime and protect users from falling into unnecessary platform specific details and complications. It helps to describe the user application on a higher level of abstraction, independently from hardware and platform issues, which results in faster development and more understandable code, focused only on investigated problem.

REFERENCES

[1] `http://doocs.desy.de/`

[2] K.Rehlich et al, “DOOCS: an Object Oriented Control System as the integrating part for the TTF Linac”, Proceedings ICALEPCS 1997, Beijing, China

[3] Eric Raymond, ”The Art of Unix Programming” Addison-Wesley 2003, ISBN 0-13-142901-9

[4] Joel Jones ”Abstract Syntax Tree Implementation Idioms” Department of Computer Science, University of Alabama