Jagellonian University Cracow
Institute of Computer Science

A Continuous System Simulation Language for IBM Personal Computers

Jacek Kuraś Jacek Lembas Marek Skomorowski

Department of Computer Science
Jagiellonian University

Abstract

A continuous system simulation language for IBM personal computers and compatibles is described. The use of the language, and its evaluation is presented. Comparison of the language with other continuous system simulation languages is discussed.

CSJU - 5/92
May 1992

Institute of Computer Science, Nawojki 11, 30-072 Cracow, POLAND

1 Introduction

A number of continuous system simulation languages have been developed, for example: CSMP III, CSSL-IV, ACSL, and others ([1], [2], [3]). These efforts eventually led to a standard programming language proposed by the Simulation Council Inc., called CSSL (Continuous System Simulation Language) ([4]). This paper describes GODYS-PC - a CSSL-type language for IBM personal computers and their clones. The language is an improvement of GODYS (an acronym for Graph Oriented Dynamic System Simulator) - a continuous system simulation language for ICL 1900 and IBM 360 computers ([5], [6], [7]).

2 Language elements

Elementary entities of the language (lexical atoms) are as follows:

identifiers - names of objects denoted by strings of one to eight alphanumeric characters (A, ... , Z, a, ... , z, 0, 1, ... , _), the first of which must be a letter,

constans - integer or real numbers denoted by digits, . (decimal point), E, +, - (constants generally conform to rules common in high level programming languages),

special atoms - operators, separators, parentheses such as: =, +, -, *, /, (,), ; and so on.

The above elements are written in free form coding format. The # character in each record denotes the end of a record. All characters followed by # are interpreted as a comment. The @ character in the end of a line denotes that the next line is the continuation of the current instruction, declaration or directive. The meaning of the language identifiers is as follows:

reserved names in GODYS-PC - the key words (for example **model**, **prepare**, **dynamic**, **end**), names of functions and operators (for example **lt**, **or**, **integ**, **step**, **sqrt**),

names defined by the user - constants, observed variables, external parameters, internal parameters, and other variables.

An array is a set of variables identified by a single variable name. Arrays can be used as parameters in some functions. Operators may be classified as either arithmetic, relational or logical.

Arithmetic operators are as follows: ** (to raise to a power), * (multiplication), /(division), + (addition), - (subtraction). Relational operators are defined as follows: less than (**lt** or <), less than or equal to (**le** or <=), equal to (**eq**), greater than or equal to (**ge** or >=), greather than (**gt** or >), not equal to (**ne** or <>). Logical operators are defined as follows: logical **and** (**and** or &), logical **or** (**or**), complement (**not** or ~), logical **exclusive-or** (**exor**).

All constants and values of expressions are numbers. If the result of an operation is logical then its value is 1.0 for true, and 0.0 for false. The arithmetic result is true if its value is greater than 0.0, and false in the other case.

The basic unit for the language compiler is a record (a line). One record contains one declaration, instruction or directive. The structure of the model description is shown in the next part. Each instruction is denoted as an equation. The left side of the equation must be a variable or an internal parameter. The right side should be an expression. The expression in GODYS-PC is a sequence of terms, operators and parenthesses, and generally conforms to rules common in high level programming languages. Each term may be a constant, an identifier or a function call (name of a function with a list of arguments and a list of parameters in parentheses). The list of arguments and the list of parameters are separated by a semicolon. Arguments and parameters are separated from each another by a comma. GODYS-PC provides 75 functions (functional blocks) and operators. A few of these functions (used in an example in section 5) are the following:

- **integrator**:

$$y = \mathbf{integ}(x; y0) \qquad y(t) = y0 + \int_{t0}^{t} x(\tau)d\tau, \qquad y0 = y(t0)$$

- **largest value**:

$$y = \mathbf{max}(x1, x2) \qquad y = max\{x1, x2\}$$

- **step function:**

$y = \mathbf{step(x)}$ $\qquad y = \begin{cases} 1 & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases}$

- **limiter:**

$y = \mathbf{bound(x; a, b)}$ $\qquad y = \begin{cases} ab & if \quad x \geq b \\ ax & if \; -b < x < b \\ -ab & if \quad x \leq -b \end{cases}$

The user can also define his own functions (in FORTRAN 77).

3 Job processing

The basis for the program construction in the GODYS-PC language is a mathematical model of a system being under examination. A program written in GODYS-PC consists of the following two parts:

- the model description,
- the sequence of directives which control simulation carried out on the model.

During the translation process, the model description is translated into an intermediate language of a problem oriented abstract machine. This machine is implemented by an efficient interpreter which runs under control directives mentioned above. The results generated during the simulation experiments are written into a file, and next they are taken out in the way determined by output directives.

4 GODYS-PC statements

A program written in GODYS-PC has the following structure:

model model-identifier

The section of declarations which may comprise the following declarations: **paramt**, **const**, **prepare**, **declare**, **map**, **nomap**, **rename**

initial

The initial section (optional)

end
dynamic

The model description

end
load model-identifier

The sequence of directives which may comprise the following directives: **data**, **execute**, **continue**, **hdr**, **print**, **prplot**, **graph**, **plotxy**, **dump**, **save**

finish

The first statement of every GODYS-PC program must be the **model** statement in the form : **model** model-identifier. A model-identifier is used to identify the program.

In the section of declarations external parameters of the model are declared. During the subsequent experiments values may be assigned to these parameters. In this section identifiers of constants used in the model description are also declared as well as identifiers of the model variables values which are observed during the simulation. The section of declarations may comprise the following declarations:

paramt - declares identifiers of parameters of the model,

const - defines identifiers of constants of the model,

prepare - declares observed variables, values of which are to be collected and saved during execution for subsequent plotting and (or) printing purposes,

declare - specifies a number of arguments and parameters in functions defined by the user,

map - is used to print a full map of the model,

nomap - disables printing a map of the model,

rename - is used to change the identifier of the independent variable **T**.

The **initial** statement (optional) introduces a block of code which is related to simulation initialization. This block must be terminated by the **end** statement. In this block initial values are assigned to internal parameters.

The description of the model consists of a sequence of instructions within the **dynamic** and the **end** statements. The model description instructions may not be executed in the sequence they are written. The sequence of operations in the object program is determined by the compiler on the level of an abstract machine language.

The operation of the executive system is controlled by simulation directives. They allow, among others, to initiate the individual experiments, which may be autonomous or may be a continuation of a previous experiment. For every experiment, simulation parameters may be determined (for example: numerical integration method, integration step size, and so on). The execution of a given simulation experiment may be forced even if there was an execution error in a previous experiment. It is also possible to trace values of the model variables within a defined segment of simulation time. The sequence of directives may comprise the following directives:

load - loads the program into the memory,

data - is used to assign values to external parameters,

execute - initiates execution of the simulation experiment,

continue - is used to restart execution of the simulation.

Directives **execute** and **continue** may comprise the following parameters:

tmin, tmax - specify the beginning and the end of simulation time,

method - is used to specify an integration method. The executive system has got six procedures for the numerical integration: five fixed step methods and one variable step method,

abserr, relerr - are used to specify absolute and relative tolerances in the variable step integration method,

comdel - an interval size for printing and plotting simulation results,

begtr, endtr - specify the beginning and the end of simulation time for tracing purposes,

clktime - real time limit (in seconds) for the simulation,

force - is used to force the next simulation experiment irrespectively of errors in the previous one,

opt - is used for the parameter optimization (in the case of parameter optimization a finite number of parameters has to be determined such that a cost function of these parameters is minimal).

Output directives take out a subset of values of the observed variables in the determined form. The scaling of diagrams is provided automatically or the scale factor is determined with the use of boundary values given prior to the simulation, by the user. Output directives automatically print the value of the independent variable **T**. There are following output directives in the language:

print - causes the values of the specified variables to be printed at each interval size,

prplot - causes the values of the specified variables to be plotted at each interval size (all variables specified in the **prplot** directive are plotted on the same graph),

graph - causes the values of the specified variables to be printed and plotted at each interval size (one variable on a graph)

plotxy - draws a graph of a relation of the two specified variables.

The **hdr** heading directive causes a print line consisting of the heading to be printed at the top of each table or graph. The **save** directive causes the simulation results (the values of the observed variables) to be saved into a file. These results may be further processed by other programs (for example by statistical programs). The **dump** directive causes the content of the abstract machine memory to be written into a file. The execution of the dumped program may be continued by means of the **continue** directive. The last statement of every GODYS-PC program must be the **finish** statement.

5 An example

As an example of a program written in the GODYS-PC language let us consider a simple model of the following inventory control system ([8]). Production order backlog (pob) depends on production order rate (por) and production rate (pr). Production rate depends on production order backlog and a time constant t1. Inventory (inv) depends on consumption (cons), which is exogenous, and production rate. Average consumption (avcon) depends on consumption and averaging period t2. Desired inventory (dinv) depends on average consumption and weeks cover desired - a constant r. Production order rate depends on inventory, desired inventory, average consumption and an inventory correction - a constant t3.

The model of the system can be written mathematically as follows:

$$\frac{d(inv)}{dt} = pr - cons$$

$$\frac{d(pob)}{dt} = por - pr$$

$$\frac{d(avcon)}{dt} = \frac{cons - avcon}{t2}$$

$$pr = \frac{pob}{t1}$$

$$por = avcon + \frac{dinv - inv}{t3}$$

$$cons = f(t), \qquad f(t) \geq 0$$

$$dinv = r * avcon$$

Let us consider that consumption (cons) is a **step** function. Real inventory (realinv) should be greater or equal to zero. Production rate (pr) should be bounded from above.

The GODYS-PC program for the model is written as follows:

```
model invent
prepare cons, relinv, pr
paramt t1, t2, t3, inv0, pob0, avcon0, a1, a2, t0, b, r
dynamic
inv = integ (pr - cons; inv0)
relinv = max(inv, 0)
pob = integ (por - pr; pob0)
avcon = integ ((cons - avcon)/t2; avcon0)
pr = bound(pob; 1, b)/t1
por = avcon + (dinv - inv)/t3
cons = a1 * step(t) + a2 * step(t - t0)
dinv = r * avcon
end
load invent
hdr "a simple inventory control system"
hdr "cons - consumption, relinv - inventory, @
pr - production rate"
data t1 = 4, t2 = 4, t3 = 4, inv0 = 1000, @
pob0 = 400, avcon0 = 100, a1 = 100, a2 = 25, @
b = 600, r = 10, t0 = 5
execute (dt = 0.1, tmax = 40, comdel = 1, @
method = trapez)
prplot(cons = (0, 400), relinv = (0, 1500), pr =(0, 200))
finish
```

The simulation results are presented in Figure 1.

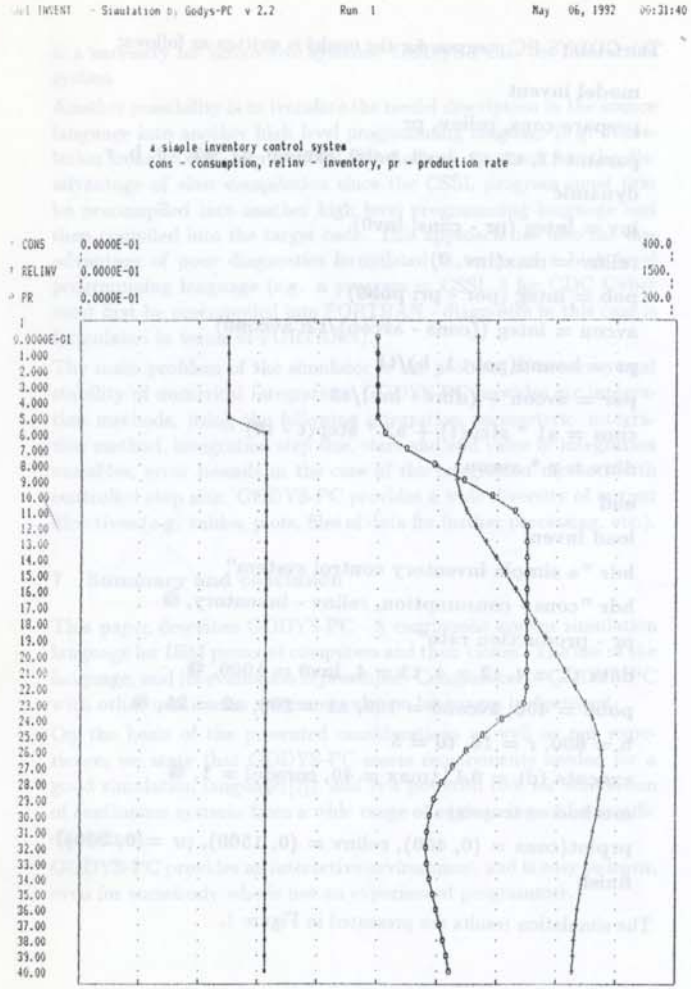

Figure 1. The simulation results of the example.

6 Evaluation and comparison GODYS-PC with other continuous system simulation languages

Any system for a digital simulation of continuous systems consists of a simulation language, a processor, and a set of functions (blocks). All existing continuous system simulation languages (CSSLs) are block oriented. There are two different ways of describing such a block structure: block statements (e.g. Pactolus, CSMP/1130) and system equations (e.g. DSL/90, MIMIC, GODYS-PC). The early CSSLs have all been of the block statement type. From the first, simple block statement languages to the modern system equations languages, various forms of increasing complexity have been developed. Syntactically we can classify all existing CSSLs into the following five bascic types ([2]):

type 1 - simplest possible form, block numbers instead of block names, fixed numbers of inputs (e.g. Pactolus).

type 2 - simple, strictly block oriented, free format, block numbers or symbolic names (e.g. MIDAS).

type 3 - algebraic operations as equations, functions as blocks (e.g. COBLOC, DES-1).

type 4 - algebraic form, blocks defined in the usual mathematical notation of functions (e.g. DSL/90, MIMIC, CSSL 3).

type 5 - algebraic form like type 4 but with ' notation for derivation (e.g. ANAGOL, SIESTA, DARE).

GODYS-PC is of **type 4**.

The processor of a simulation system is a program consisting of a few modules. The more specific parts are the translator and the simulator. The translator of GODYS-PC translates the model description in the source language into an intermediate language of a problem oriented abstract machine. This machine is implemented by an efficient interpreter. Such an approach has the advantage of fast compilation and excellent diagnostics, formulated in terms of source program references. The translation into an intermediate language of a problem oriented abstract machine provides such a possibility which

is a necessity for interactive systems. GODYS-PC is the interactive system.

Another possibility is to translate the model description in the source language into another high level programming language (e.g. translation from a CSSL into FORTRAN). Such an approach has the disadvantage of slow compilation since the CSSL program must first be precompiled into another high level programming language and then compiled into the target code. This approach has also the disadvantage of poor diagnostics formulated in terms of a high level programming language (e.g. a program in CSSL 3 for CDC Cyber must first be precompiled into FORTRAN - diagnosics in this case is formulated in terms of FORTRAN).

The main problem of the simulator is the problem of accuracy and stability of numerical integration. GODYS-PC provides six integration methods, using the following integration parameters: integration method, integration step size, start and end value of integration variables, error bounds in the case of the integration method with controlled step size. GODYS-PC provides a wide diversity of output directives (e.g. tables, plots, files of data for further processing, etc.).

7 Summary and conclusion

This paper describes GODYS-PC - a continuous system simulation language for IBM personal computers and their clones. The use of the language, and its evaluation is presented. Comparison of GODYS-PC with other continuous system simulation languages is discussed.

On the basis of the presented considerations as well as our experience, we state that GODYS-PC meets requirements needed for a good simulation language([2]), and is a powerful tool for simulation of continuous systems from a wide range of engineering and scientific disciplines.

GODYS-PC provides an interactive environment, and is easy to learn, even for somebody who is not an experienced programmer.

References

[1] Catalog of simulation software, Simulation, volume 51, 1988, 136 - 156.

[2] Giloi W.K., Principles of continuous system simulation, B.G. Teubner, Stuttgart, 1975.

[3] Gordon G., System simulation, Prentice Hall Inc., 1978.

[4] Strauss J.C., Augustine D.C, Johnson B.B., Linebarger R.N., Sanson F.J., The SCi continuous system simulation language (CSSL), Simulation, 9, 6, 1967, 281 - 303.

[5] Jakubowski R., Król J., Implementation of the simulation language based on functional graphs, Podstawy sterowania, 1972, tom 2.

[6] Król J., Kuraś J., Lembas J., Ślusarek M., Implementacja języka do symulacji układów ciągłych ze zdarzeniami dyskretnymi, Podstawy sterowania, 10, 1980, 57-58, (in Polish).

[7] Król J., Kuraś J., Lembas J., Język symulacyjny dla komputerów ODRA 1300, Informatyka, 11, 1979, 16-17, (in Polish).

[8] Coyle R.G., Sharp J.A., System dynamics, problems and cases, University of Bradford, U.K., 1976.

[9] Continuous system simulation language III (CSSL 3), Reference Manual / User's Guide, Control Data Corporation, 1974.

[10] IBM Corp., CSMP III program reference manual, 1972.

[11] Chu Y., Digital simulation of continuous systems, McGraw-Hill Book Company, 1969.