ENVIRONMENTS FOR SIMUL_R

(Graphical Input, Transputers, Discrete Simulation)

Ronald Ruzicka
Austria

ABSTRACT

The conception of SIMUL_R has designed this language as an open system. Therefore several kinds of environment tools have been developed for SIMUL_R.

This paper presents a software and a hardware tool: the graphical input system **SIMDRAW** for models, which supports the development of SIMUL_R-models using a block-oriented view of dynamical systems, and the transputer system **SIMUL_TR**, which uses a PC as communication-part and offers real parallel computing of SIMUL_R-submodels.

At the end of this paper the discrete-process tool **PROSIMUL_R** is presented, which gives the possibility of working with discrete models (e.g. using continuous models as submodels) in SIMUL_R (there exists a transputer-version of PROSIMUL_R, too).

INTRODUCTION

This paper continues the last years paper (1) about SIMUL_R (2), which presents SIMUL_R as a new continuous-systems-simulation language.

SIMUL_R is a compiler-oriented language (with C (3) as host-language) with special features for
- *modeling* (several models in one program, automatic solution of algebraic loops, macro-library),
- *analyzing* (loop- and recursive programming within the runtime-interpreter, special commands for table-function and data-file computations, computation of models in sequence or in parallel),

- *documentation of results* (3D-plots, parameter-lines, niveau-lines, special plot commands for the solution of partial differential equations, moving pictures).

Though the SIMUL_R-sytem contains several features, it is necessary to develop environment tools for a better support of users' wishes. The first shown tools aim to improve the acceptance of SIMUL_R by two user-groups (amoung others): first the engineer and control-designer and secondly the big group of people, whose models take much time to be computed.

SIMGRAPH - A GRAFICAL DEVELOPMENT SYSTEM

The grafic-system SIMGRAPH is a block-oriented surface for the equation-oriented SIMUL_R-language. It has been implemented under GEM and is fully menu-driven.

The user draws a model-picture by dragging predefined operational-blocks into his model window. The blocks can be selected and linked together by lines. Each line - leading from one port of a block to one port of another (or the same) destination block - can be named. All lines can be accessed at simulation time by their names as normal model-variables (see Figure 1 for the SIMDRAW-desktop).

Fig. 1 The SIMDRAW-desktop.

Blocks

There are three kinds of blocks: standard, functions and transfer-functions.

Standard blocks are sumation, subtraction, multiplication, division, integration, comparison; a switch-block (connects one of two input-values to the output-port, depending on a logical input-value), a logical negation.

A constant-block delivers constant-values. The extern-object gives the possibility to access extern variables; the intern-block is a kind of watch-point for specifying the output-ports in an open system.

Two blocks are known from analog-computers: the track-store and the hold. So easily analog- or hybrid-computer-models can be imitated.

A special one is the BLOCK-block. It is something like a grafical macro and facilitates the structuring of models. With a double-click of the mouse-button onto a block-icon special values of blocks can be set: constant numbers to constant-blocks, external names to extern.
A double-click onto a BLOCK-object opens another niveau of your model - you somehow look one step deeper into it. Here you may define a sub-part of your model (using extern- and intern-blocks). The extern-blocks and intern-blocks here are the input- and output-ports of the BLOCK one niveau above!

Deeper niveaus are whole models themselves; and therefore it is easy to test your model by first testing your submodels.

Function blocks offer a lot of useful functions: harmonic oscilator, positive- or negative-part of a value, ramp- and step-functions, pulse-functions, random-values.

The last category are the **transfer function blocks**. That are blocks especially designed for control-engineers, which symbolize transfer functions (lead-leg, real-pol, complex-pol).

The SIMDRAW-desktop

The SIMDRAW-desktop only shows a part of your model (it might be 10000 times larger!). You can move your watch-window across the model, where you want. Naturally there is a *Show all*-mode, where the whole model can be seen in the look-window (see Figure 2).

Fig. 2 The SIMDRAW-Show All-mode.

SIMDRAW contains many tools for the preparation of models: deleting, moving and copying of parts of the picture; including of other pictures; saving, reading and plotting.

The SIMDRAW-translator

Nevertheless we not only want to draw but to simulate!

SIMUL_R is a compiled language, because compiled languages are much faster than interpreted ones. Therefore it fits into system-philosophy to translate and compile the picture: it is first inspected (error-messages are displayed, if something is not built up correctly, and the wrong block is inverted) and then translated into a readable SIMUL_R-program.

From this point on it can be automatically compiled to a executable program or be changed by hand for special purposes.

Many SIMDRAW-blocks (such as trackstore or the transfer-function-blocks) use the SIMUL_R-macro-library.

Example:

Figure 1 and 2 show a sine-cosine-system, Figure 3 contains the file, generated from this pictures by SIMDRAW.

```
#include'SIMCOMAC.DEF'
COSSIN {
  CONSTANT tend=1;
  CONSTANT dx0=1;
  CONSTANT omega=1;
  CONSTANT x0=0;
DYNAMIC {

DERIVATIVE {
  dx=INTEG(hdx,dx0);      x=INTEG(hx,x0);
  gdx=x*omega;
  hx=dx*omega;
  hdx=-gdx;

}

  TERMINATE t>=tend;

}

}
```

Fig. 3 Sine-cosine program.

SIMUL_TR - the transputer version

Historical aspects

The PC is a very suitable and worth-the-money tool for simulation. In comparison to many more expensive (and sometimes faster) computer-systems it has many advantages considering grafical input and output, man-machine-interface and spreading all over the world.

Nevertheless simulation often reaches the bounds of the PC because of memory- and time-reasons.

A few years ago new processors have been developed - the transputers (4) -, which on one hand are fast as standalone-processors and can easily communicate with one another on the other hand. Nowadays several cards are available, which subdue the interface PC to transputer.

Therefore it seems to be clear, that a simulation-language for transputers has to be developed or a transputer-version of an existing language must be implemented.

Development-problems of a transputer-language

Before such a project (the development of a computer simulation language on transputers) can start, some decisions have to be made:

- should a new language be written?

- if an old one is used, which parts should work on the PC, which parts on the transputer?

- should a compiled or an interpreted language be developed?

- is it necessary for the user to know that he works on a transputer (or might he use the system as a black box)?

- who should implement such a system: a simulant or a transputer-expert?

Answering the first question, it must be said, that it costs too much time to create a new system. We will take SIMUL_R, because this language posseses many features for parallel-computing even in its PC-version.

Therefore the third question can be answered easily: because of philosophy reasons a compiled version is used, and why should a fast processor be slowed down by interpretation?

The second question can be answered by saying, that all things, which have to deal with input and output, should be done by the PC and the computation-work should be performed by the transputer.

The question, whether the user should know, how its system works, can be decided by a philosophical view (information hiding - yes or no); but the experiences in this direction have to be considered, too (they have shown, that total ignorance on the system reduces effectivity).

SIMUL_TR has been developed by both sides of experts: simulants and transputer-engineers.

The SIMUL_TR-system

First the installation of the system takes place (the transputer-system configuration is specified or automatically detached) .

It has been decided, that from now, the user knows his system parts (different transputers or different processes on one transputer) and can refer to them as *process-numbers*. He can decide which submodels are computed in parallel on which processes. The SIMUL_R-mstart-command (which specifies the models to be computed in parallel, as known from the PC) contains this process-numbers as extensions.

Example:

```
mstart model_A:0, model_B:1, model_C:0;
```

This command decides model_A and model_C to be run on process 0 and model_C on process 1.

The user is not involved in all the problems and methods, which arise and are necessary with the link-communication of the transputers.

He only works at his PC, with its SIMUL_TR-surface as known from the SIMUL_R-PC-version - and is pleaseed about the velocity of computation.

Differences between SIMUL_R-PC and SIMUL_TR

There are some differences between PC-SIMUL_R and SIMUL_TR.

As decided above SIMUL_TR is as well compiled as the PC-version, but: not the PC-part has to be recompiled, but the transputer-program.

This offers a lot of advantages:

- only the model-part is compiled and linked with the numeric-library -> translation-time decreases,

- the user need not leave the runtime-environment for recompilation -> this is more comfortable

- the SYS-command enables the user to edit his program from the runtime-environment

Recompilation is done by using the new **COMPILE**-command:

Example:

```
COMPILE 'test -a';
```

This command compiles the SIMUL_R-program test.sim with the flag for automatic solution of algebraic loops. It generates the C-files for the transputer and a system-configuration file for the PC. The transputer compiles and links the C-file and starts the simulation-program.

The system-configuration file is read by the SIMUL_TR-PC-part and the internal structure is built up for the new system.

The simulation-program on the transputer waits for a start-command. If it receives such a message, necessary system-parameters and variable-values will be read and computation is started. During computation prepared-variables are sent to the PC. At the end of the simulation-run model-variables are written back to the PC.

That means, that variable-changes between two simulation-runs are computed locally at the PC and are reported to the transputer at the start of a simulation run.

On the other hand prepared data is sampled and displayed on the PC (and its disk).

There is one situation, when recompilation of the SIMUL_TR-PC-part is necessary: when the feature of SIMUL_R, that new commands can be added to the system, is used (but this in general not often takes place).

SIMUL_TR - not only transputers

The conception of SIMUL_TR makes it possible to develop parallel simulation systems with the powerful runtime-features of SIMUL_R on any other parallel computer as well.

Nevertheless: the combination transputer and PC is a modern and promising way of simulation.

PROSIMUL_R - A DISCRETE ENVIRONMENT

The idea of PROSIMUL_R

The PROSIMUL_R-system is an extension to the SIMUL_R-system. That means, that it totally fits into the SIMUL_R(-continuous-systems) system. One of a SIMUL_R-program's submodel is a discrete-model. It uses the same syntax and has a similar body as the continuous models, but naturally other commands are available.

A discrete-model can start and stop continuous models. PROSIMUL_R offers the well-known discrete-commands, like create, wait and delay, but has a compareably small amount of new commands, because a lot of commands are implemented as macros (using the powerfull SIMUL_R-macro-meta-language).

The discrete-model, also called **PROCESS**-model may describe the course of production in a factory as well as the time- and causal-dependency of a set of continuous models.

Entities

The course of the discrete model is determined by the way so called *entities* path through it. Such an entity can be seen as work piece or as well as one of the program counters in processes using independendly the same program.

Entities are generated and/or started at special points of the PROCESS-submodel called **CREATE**-commands. At all those points the course of computation starts simultaneously at the begin of a simulation run.

Stations

A SIMUL_R-PROCESS-model is devided into **STATION**s, which denote each a logical unit in the course of the paths of the entities (e.g. a station may describe the loading-, working- and unloading-phase of a machine in a factory, a conveyor belt, a transporter).

Each station can be *activated* or *deactivated*. Entities in a deactivated station are frozen (their time-delays are frozen, too).

Movement of entities

If an entity reaches the end of a station, it continues its path in the next station (as written in the program). If this is the last station, it is released. There are two other ways of moving an entity to another station: by a **MOVE**- or a **CMOVE**-command.

The first one moves the entity to a station by using a specified station as *transporter* (the entity leaves the current station, goes through the transporter-station - e.g. being delayed by a **DELAY**-command or waiting for a continous system to be finished - and then continues at the destination station). Only one entity at a time can be moved by the transporter and so has the ability of controlling the transporters movement.

The **CMOVE**-commands move the entity to other stations by using another station as *conveyor*. A conveyor-station is not controlled by a special entity, but moves continuously. Many entities can be moved in this way simultaneously (e.g. belt or bucket conveyors). The movement and the relation between conveyor and entities are specified by the conveyor's velocity and length and the entitities'

length.

There are many macros, which build up both movement systems.

Seizing, queueing, freeing

In general, stations (e.g. machines, transporters, conveyors) have to be *seezed* before they can be used. That means that the entity tries to get the right to use a station, because many stations only have limited capacities. So it may happen, that some entities compete for a station and there is not the appropriate amount of free capacities (entities may need different amounts of capacities).

Therefore the entity has to *queue*. That means in general that it is moved to another station, where it stays until the needed number of free capacities is available. If there is enough capacity for more than one waiting entity, that one with the highest priority will get the station first (FIFO-rule with equal priorities).

A seezed station has to be *freed*, when the entity leaves the station.
An entity may seize several stations (e.g. a machine and a tranporter, which moves it to this machine), but only paths through the commands of one station at one time. Therefore many stations will be more seized, than really used (that is the reason, why each station has one attribute for the number of its units, which are seezed, and another for the number of units, which are really active - that means used).

Seizing and freeing is done by macro-commands.

Attributes of entities

Each entity has a lot of attributes (type, priority, locality, destination during movements, length during conveying), which are stored in SIMUL_R-variables (each entity is automatically assigned a number, which can be used as index into the arrays of the attributes). Nevertheless these indices need not be used, because special macros make it possible to directly access the attributes of those entities having reached a special position.

Conveyors

Conveyors and entities have a special length, conveyors a velocity (constant-*"time-conveyors"*, not constant-*"event-conveyors"*), too. A conveyor is built up from one or more stations, which each denote a part of it. Seen from a more abstract view these stations (e.g. belts) lead from one station (e.g. machine) to another.

If an entity enters a conveyor it is put on it (its back at the beginning of the conveyor). When its front reaches the end of the conveyor-part and the station there is the destination, it is removed from the conveyor; so it has only been moved on the conveyor for a length of
length of conveyor - length of entity.

The time it needed for this length is used as conveying-time by the conveyor-macros.

If the entity has not reached the destination station after the first conveyor-part it has to travel on using the next conveyor-part station. But it takes some time to get from one conveyor-part to another (that time, from the first touch of the entity's front on the new conveyor to the last touch of the back of the entity on the old conveyor). This process is modeled by the **CMOVE_**x**DELAY**-commands and -macros.

Conveyors can be built up by using special macros.

The PROSIMUL_R-program

The PROSIMUL_R-sourcetext consists of an arbitrary number of continuous models and one process-model with the command-sequence shown in Figure 4.

```
PROCESS model_name, n {
  ... initial
  DYNAMIC {
    STATION stat_name {
      ...
    } ;
    ...
    STATION stat_name {
      ...
    } ;
  }
  ... terminal
}
```

Fig. 4 The discrete-model-syntax.

Example:

The example in Figure 6 describes roughly a little factory (Figure 5) with five stations: a generation-station for entities, two machines (computed in continuous models) and two stations for the 2-part conveyor. The entity is created and then processes at one of the two machines depending on its type (system attribute #SI(0)).

Fig. 5 A little factory.

```
machine_1 {
  ...
}

machine_2 {
  ...
}
```

```
PROCESS factory, 30 {

  CONSTANT len1=5, len2=7, vel=1;
  CONSTANT tend=100, ent_count=50;
  CONSTANT ent_length=0.2;

  STATIONS start, station_1, station_2,
        belt1, belt2;

  DYNAMIC {

STATION start {
  CREATE_DIST
          unif_dist(0,1),  "creation time"
discr_dist(0.5), "type"
          ent_count;      "number of entities
                       to be created"
  #SF(3)=ent_length;     "length of entity"
  #dest =              "destination"
 SWI(#SI(0),station_1,station_2);
  #SEIZE  ((#dest))
  #TSEIZE (belt1,len1,vel)
  #CMOVE  ((#dest),belt1,len1,vel)
} ;

#TCONVEYOR (belt1,station_1,len1,vel,
             belt2,station_2,len2,vel)

STATION station_1 {
  start 0;
  #FREE   (station_1)
  LEAVE;     "entity leaves system"
} ;

STATION station_2 {
  start 1;
  #FREE   (station_2)
  LEAVE;     "entity leaves system"
} ;

TERMINATE t>=tend;

  }
}
```

Fig. 6 PROSIMUL_R-model of a factory.

Special features

SIMUL_R gives the opportunity of optimizing a system and of saving the whole system-state. If you consider a special system (maybe the model of a factory, with continuous models for machines), some parameters (velocities, distribution-rates of jobs to different machines) can be optimized under special conditions (necessary production-output, cheapest configuration, ...) and stored. So a

parameter-library is built up.

If such conditions occur later on, the system-parameters can be loaded and the system can be used immediately with the optimal parameters.

As conveyors can be modeled with not-constant velocities, *starting processes* of systems can be simulated.

There are several computation features for statistical analysis (minimum, maximum, mean, variance, histograms).

CONCLUSIONS

The grafic-environment SIMDRAW is an easy-to-use input-tool for SIMUL_R-models. It supports program-development by its feature of making submodels. In the future many other block-primitives will be available.

SIMUL_TR - the transputer version of SIMUL_R - opens up a new method of simulation: fast computation and parallel simulation with a PC as the input-output-medium.

In the near future another facility of parallel-computing will be ready to use: not only different models in parallel, but also one model computed in parallel (with automatic partitioning of the parallel parts) and the use of parallel numeric algorithms.

The discrete-language PROSIMUL_R is a consistent extension to the SIMUL_R-system, which can use all the features of SIMUL_R. In connection with the transputer-version SIMUL_TR an animation-tool has been developed that delivers a combination of 3D-movements, sprites and reality-photographs.

REFERENCES

1 R. Ruzicka:
SIMUL_R - A Simulation Language with Special Features for Model-Switching and Analysis, Proc. of the ESMulticonference 1988, Nice, 429 pages

2 Dr.Ruzicka-Simulationstechnik:
SIMUL_R - A user's guide.
Vienna, 1988

3 Kernighan, Ritchie:
The C Programming Language
Prentice-Hall, 1977

4 Inmos:
Transputer reference manual
Prentice-Hall, 1988