AN ACTIVITY ORIENTED VISUAL MODELLING LANGUAGE WITH AUTOMATIC TRANSLATION TO DIFFERENT PARADIGMS

Luís M. Silva Dias
A. J. M. Guimarães Rodrigues
Guilherme A. B. Pereira

Department of Production and Systems
University of Minho
Braga, 4710-057, PORTUGAL
<http://www.dps.uminho.pt>
< {lsd,agr,gui}@dps.uminho.pt >
Phone: +351253604740/+351936271733 Fax: +351253604741

KEYWORDS
Visual Simulation Modelling, Activity Cycle Diagrams, Automatic Programs Generation, Arena

ABSTRACT

The traditional approach for discrete event simulation modelling includes visual support diagrams for modeller-client communication purposes (model interpretation and validation) and also to act as the basis for simulation language program construction.

Although modern simulation packages use powerful graphical interfaces for programming and animation purposes, these packages still require enormous simulation expertise to construct a simulation program. This work suggests the use of the Activity Cycle Diagrams-ACD (activity based philosophy) concepts for modeller-client communication, but also to act as an automatic generator of simulation programs under different paradigms - event scheduling (Basic Simulation Facility – Simulation Library) and Process Flow (ARENA – Simulation Environment) philosophies, thus eliminating any programming effort and expertise.

1 INTRODUCTION

The use of visual support diagrams to help the programming step of a simulation project is very common. Even when generic programming languages were replaced by specific purpose simulation languages the use of paper diagrams remained as a previous step to programming (Tocher 1963) (Pidd 1984). These diagrams were conveniently abstracted serving as support to the communication between the simulation client and the modeller (simulation expert), but also helping the construction of the corresponding computational programs (Clementson 1982) (Rodrigues 1987).

Modern simulation languages introduced new powerful graphical interfaces, but these interfaces are clearly programmer oriented, raising the difficulty in communicating with the client and still requiring enormous simulation expertise to use them (Dias and Rodrigues 2002b) (Harrell et al. 2003) (Kelton et al. 2004).

In this paper, we still suggest the use of a (simple) graphical support as a representation of what the client formally needs, but these diagrams will also act as the source to the automatic generation of simulation programs.

The visual language chosen is the Activities Cycle Diagrams (ACD), for its simplicity and efficiency in representing real operating systems. (Dias and Rodrigues 2002b) (Pidd 1990).

This mechanism also implied the construction of translation grammars. These grammars were written according to a modular specification of visual languages, based on attribute grammars (Henriques 1992) in MASOViLa (Modular Attribute-based Specification Of Visual Languages) (Rocha 1995) (Dias 1997) (Varanda et al. 1997). Our translation engine uses a pattern matching rewriting mechanism.

Using Activity philosophy for modelling, then generating simulation programs based on event philosophy and in process philosophy, three major simulation approaches (Bennett 1995) (Sargent 2004) (Overstreet 2004) were explored and linked.

Proceedings 19th European Conference on Modelling and Simulation
Yuri Merkuryev, Richard Zobel, Eugène Kerckhoffs © ECMS, 2005
ISBN 1-84233-112-4 (Set) / ISBN 1-84233-113-2 (CD)

2 LAYOUT AND ANIMATION ORIENTED SIMULATION ENVIRONMENTS

As already referred, appropriate diagrams were in use for many years to support the communication among people interested in a particular simulation. The simulation expert would then translate these models into a simulation language or even a general purpose programming language. As far as graphical support became available, an enormous variety of simulation environments emerged (Swain 1991-2003). **Graphical facilities** were then used to essentially represent a **system layout for animation purposes**.

Animation is recognized as an important aspect of simulation. However **when** the modelling process is **focused on animation**, several disadvantages may arise:

1. The **model may be overwhelmed** by many modules and accessory configurations.

2. Such a model, with increased complexity, will be **difficult for the client** to understand.

3. The analysis of a static model (in the first stages) based on the layout will not add much over its photograph or scheme. Thus the **semantic validation** will be left for the **animation** phase.

3 PROJECT MAIN STEPS

The main steps of this research work are summarized below:

1. The **choice of** an **easy** to use and widely spread visual language : Activity Cycle Diagrams (**ACD**).

2. **Formalization of ACDs** (keeping it simple), allocating to each graphical object the information required by the model.

3. Specification of a **file format** to represent the referred graphical objects (XML).

4. Implementation of a **graphical editor** to draw the models.

5. Implementation of a compiler's **compiler** for visual/graphical 2D languages.

6. Specification of a **grammar** for the ACD language.

7. Implementation of **two compilers** - the first compiler generates the simulation program code in **JavaBSF**, and the other one generates the **Arena** program (using both Arena modules and new developed modules in VBA code).

3.1 Programming Tools

The main programming language used was **Java** (Martins 1998) (Campione et al. 1999). Several sets of classes were implemented (corresponding to about nine thousand lines of code). Visual Basic for Applications (**VBA**) was also used, both in ARENA and in Microsoft VISIO. A template was built in **ARENA (**Professional Edition**)** (Kelton et al. 2004) with activities and queues to implement an activity-based executive on a process-oriented environment.

3.2 About the use of Graphics in Visual Modelling and Programming

The idea behind this work is to enhance the **utilization of graphical facilities in modelling**, making it a great contribution to **automatic generation of simulation programs**, keeping it simple and portable.

Graphical facilities are more helpful when they **support model semantic** than when it is based on system layout and animation.

Furthermore the utilization of a **simple** axiomatic set may be **accessible** to more potential users and clients.

The use of an **activity-based philosophy** seems to be semantically richer and more simulation oriented than process flow or event scheduling (Pidd 2004).

The strategy was to create a completely **open system**, since the graphical editor creates a text file in XML (eXtended Markup Language). This is compatible with any graphical editor using the established syntax. The compiler uses that XML file and a grammar to build a program in an object simulation language.

3.3 Translation

Our first **ACDs compiler** generates a Java program according to the **event scheduling** philosophy. Although this is computationally highly efficient it is harder to program. In other words we could say that it is more computer friendly than programmer friendly. The complex compiler developed allows an easier translation by the computer.

By developing a template with new blocks for **Arena (process oriented)**, we were able to simulate ACDs using Arena objects built according to an activity based philosophy.

Using these two compilers/translators, we explored deeply on three major modelling philosophies (Rodrigues 1987) (Pidd 1990) (Overstreet 2004), making automatic translation mechanisms between them.

4 PROJECT MAIN TASKS

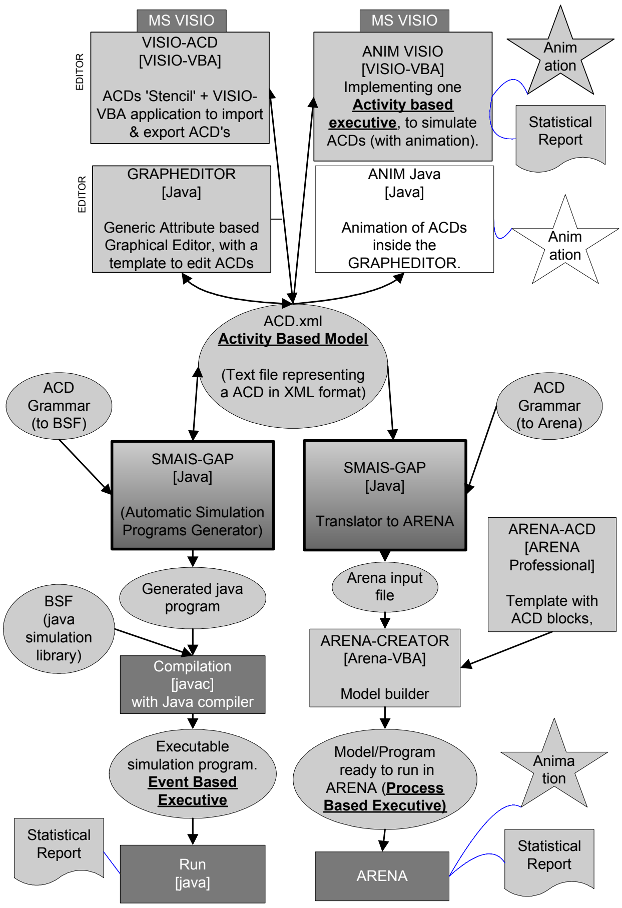


Figure 1: Project Map Diagram

The above diagram illustrates the interdependencies between the main tools and contributions of our project:

■ **ACD.xml** «*in the centre of diagram*» represents the chosen format to physically support the models: a text file in XML. (See also Figure 13 and Figure 14)

⇖ **GRAPHEDITOR** «*upper left zone of the diagram*» is the graphical editor that was specifically built for this purpose (see also Figure 9). Microsoft Visio was also customised to deal with ACD XML files – **VISIO-ACD**.

⇗ **ANIM VISIO** - «*upper right zone*» simulates and animates ACDs inside MS VISIO. The **ANIM-Java** tool will animate ACDs in the GRAPHEDITOR (is not yet completed, as signalled in the diagram by white background).

⇙ **SMAIS-GAP** is the major tool of this project «*lower left section*» allowing the translation of an ACD (activity-based) into a program (event-based) (see also Figure 15, Figure 16 and Figure 17). It uses the **ACD Grammar (to BSF)** (see also Figure 10, Figure 11 and Figure 12).

⇘ **SMAIS-GAP** using the **ACD Grammar (to Arena)** «*lower right section*» refers to the translation to **Arena input file**. **ARENA-CREATOR**, using that input file, builds a model with blocks from **ARENA-ACD** template (see also Figure 18, Figure 19, Figure 20 and Figure 21).

5 ACTIVITY CYCLE DIAGRAMS (ACD)

ACDs were widely used mainly with older languages (e.g. HOCUS, ECSL (Clementson 1982)) to schematically specify the system's behaviour, through each assumed entity cycle diagram. These entity cycles explicitly refer the active states into **activities** and passive states into **queues**. This graphical language just requires the use of three types of graphical objects: **Rectangles** (activities), **Circles** (queues) and **Arrows** (links).

For an activity to start, it is necessary that entities exist in the preceding queues in the required number and with the adequate attributes. When these conditions hold, it is possible to start the activity. When the activity ends, the entities involved are moved to consequent queues.

The **complete model** consists of the Activity Cycle Diagrams of all the classes of entities, together. Interactions between entities take place at activities. Figure 2 represents the basic activity concept, with one activity in the middle that starts when each precedent queue has one required entity. When the activity ends, the entities are moved to the consequent queues.

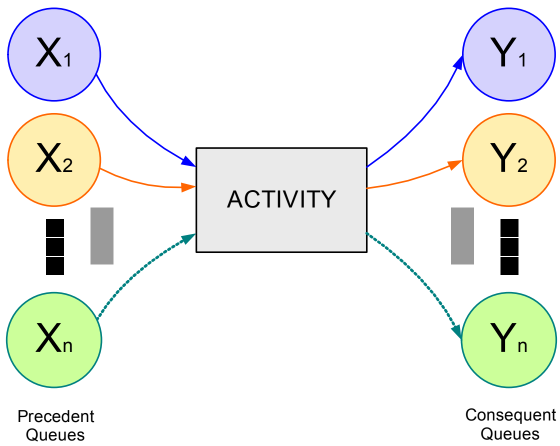


Figure 2: Activity concept

The simplicity inherent to the activity concept and to the ACD facilitates its easy understanding for **validation and teaching purposes**. It has been advocated that the ACD is useful for **research** discrete event simulation studies.

5.1 ACD Example: the Bartender Problem

In this illustrative example we have a barman that serves customers in a bartender (Clementson 1982) (Rodrigues 1987).

Entities of class **CUSTOMER**, are initially OUTSIDE. They ARRIVE and then WAIT for activity POUR. When served they are READY to DRINK. After that, if they NEED to drink more they go to queue WAIT, otherwise they leave to OUTSIDE. The ACD of CUSTOMER is described below (Figure 3).

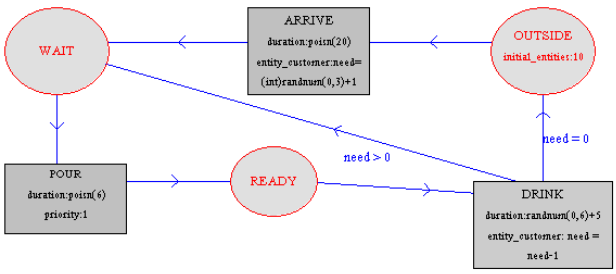


Figure 3: CUSTOMER ACD

Entities of class **BARMAN**, are waiting in the queue IDLE and they can either participate in the activity POUR or WASH:

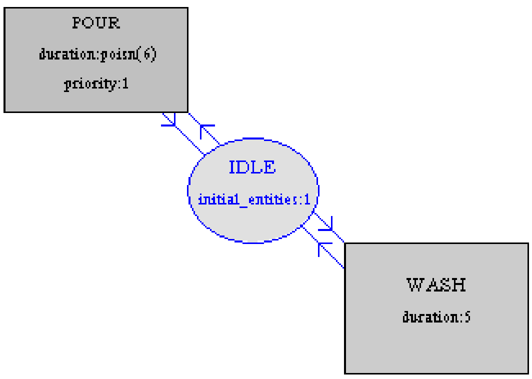


Figure 4: BARMAN ACD

Entities of class **GLASS** are initially in the queue DIRTY. After being WASHed (in batches of size 3), they wait in the queue CLEAN. When there are 1 customer waiting, 1 barman idle and 1 glass clean, then POUR activity begins. Once FULL, the glass goes to activity DRINK. The ACD of GLASS is in Figure 5.

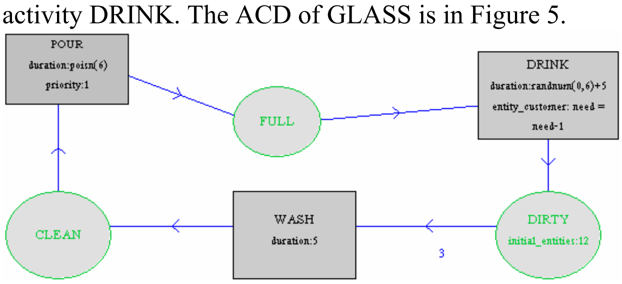


Figure 5: GLASS ACD

This problem instance parameters are:

Activities duration:

- ARRIVE: Poisson distribution, average=20.
- POUR: Poisson, average=6 [poisn(6)].
- WASH: Fixed = 5.
- DRINK: 5 + Uniform distribution between 0 and 6 [5+randnum(0,6)].

Entities initial allocation:

- CUSTOMER: 20 in queue OUTSIDE.
- BARMAN: 1 in queue IDLE.
- GLASS: 12 in queue DIRTY.

Entities class setup:

Figure 6: ENTITIES setup

- CUSTOMER: have one attribute: NEED.

Attributes:

- ARRIVE: Customer attribute NEED is initialized with : (int)randnum(0,3) + 1 → {1,2,3,4}
- DRINK: Customer attribute NEED is decremented. After this activity, customer attribute NEED is evaluated to decide customers destination.

Simulation setup:

- DURATION: 1000 time units.
- WARM_UP: 120 time units.
- SEED: 123543.

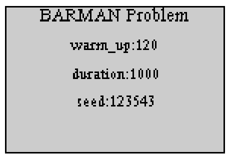


Figure 7: Simulation setup

The following ACD (Figure 8) includes all system:

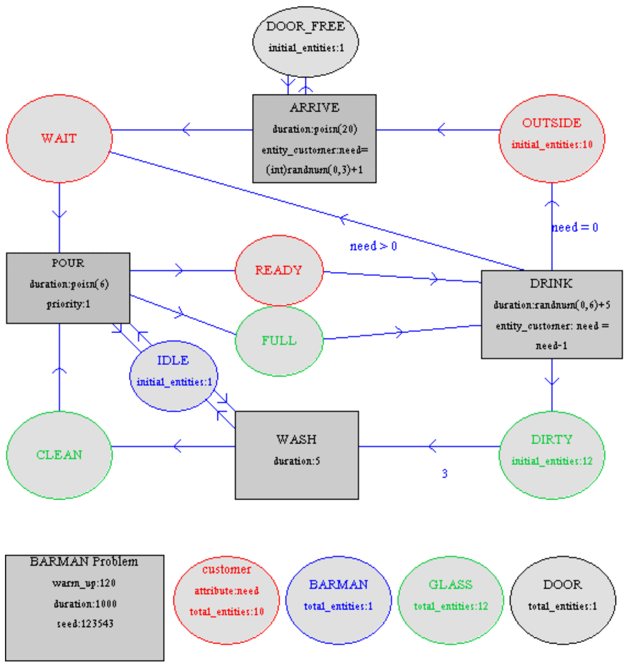


Figure 8: Global ACD

DOOR is an auxiliary entity used to control the Customer's arrives (one at a time). Described in (Rodrigues 1987).

The following image (Figure 9) shows a screenshot of the GRAPHEDITOR, editing the bartender problem.

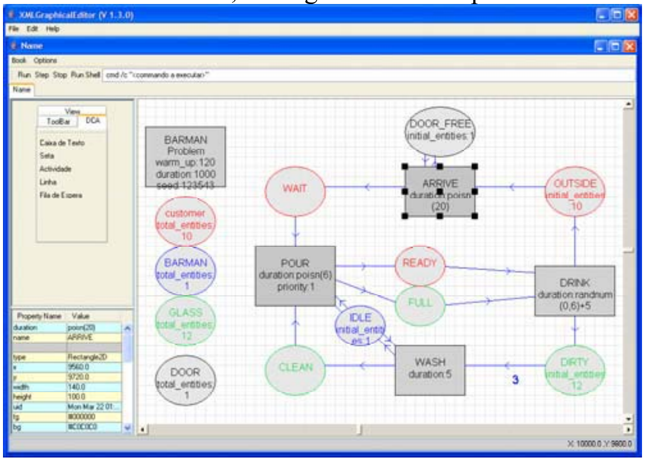


Figure 9 GRAPHEDITOR Screenshot

5.2 ACD Language Formalization

Some authors suggest modifications in ACD language, e.g. (Filho and Hirata 2004), but in our opinion that compromises the ACD simplicity.
A significant contribution of our work consisted on the development of the ACD language formalization embedding in the diagrams all the information required for the simulation.

1) We added attributes to the visual objects:

- '**duration**' and ['**priority**'] in **activities**. ('priority' is a value setting the activity priority over other activities. Higher value means higher priority).
- Entities **attributes** changing in activities. (EX: In the DRINK activity of the example model, the customer 'need' attribute is decremented by one. (activity attribute 'entity_customer' = "need=need-1")).
- ['**initial_entities**'] in **queues**.

2) In order to include global information on the simulation a rectangle alone (called **simulation setup**) is used with the following attributes :

- '**model_name**' – Model_Name_String
- '**duration**' - **Simulation time**
- '**warm-up**' - initialization time.

3) For global attributes and features of each **entity**, one ellipse alone is created with:

-'**entity_name**' – name of the entity

-'**total_entities**' – number of total entities of this kind that will exist in the simulation.

- ['**attribute**']* - used zero or more times to declare entity attributes. (EX: customers have one attribute: 'need').

- ['**sort**'] –expression establishing the queue sorting rule. Value is defaulted to 1. If defined, the queue will not be FIFO, this attribute must have an expression, elements are sorted in ascending order based on the evaluated expression over each arriving entity. May be used to create virtually, multi-queues.

4) **Arrows**

- Arrow ending on activities:

'**label**' have the number of need entities (entering throw this arrow in the activity) to start the activity (this implements batches). Default value is 1 (if omitted). (EX: three is the number of 'glasses' to start 'WASH').

- Arrow leaving from activities:

'label' have conditions to decide which destination queue will be chosen (usually based on attributes values). (EX: 'need' attribute is used to decide if the 'customer' go OUTSIDE or WAITING after drinking).

6 THE GRAMMAR AND TRANSLATION

To create the AIMS compiler, we wrote a set of rules in Visual MASOViLa notation (Dias 1997). Each rule synthesizes one new symbol. We developed rules for *queue (3),* for *activity (9),* for *entity (3),* for *input_link (1),* for *output_link (1)*, for *simulation (5)*, and also for syntactical and semantic *error* detection (10) – see graph in Figure 12.

We include bellow one expression example, with textual explanation, visual representation and the generated Java code (portion of the compiler code).
The next graph (Figure 10) is the rule that transforms an *arrow* in an *output_link,* when it is connected from an *activity* to a *queue***.** The new *output_link* symbol, receives all attributes from *arrow* symbol (*). Furthermore it synthesizes the *output_link*'s attribute *origin* from the *activity's* attribute *name*, the attribute *destination* from the *queue*'s attribute *name* and also the *condition* attribute from *arrow's label*.

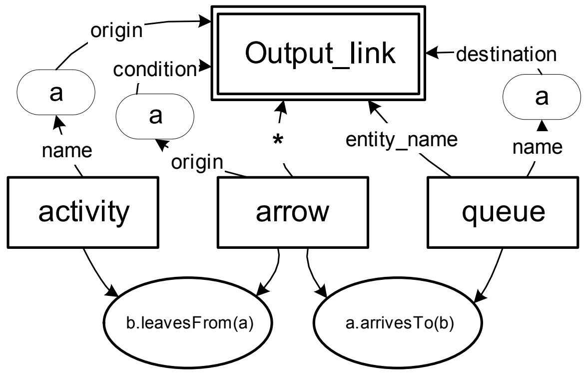


Figure 10: Output_link rule written in Visual MASOViLa

```
public static Symbol output_link(Vector args){
  Symbol activity = (Symbol) args.get(0);
  Symbol arrow = (Symbol) args.get(1);
  Symbol queue = (Symbol) args.get(2);
  if(((Arrow)(arrow.get("container")))
    .leavesFrom((Container)
    (activity.get("container"))  ) &&
  ((Arow)(arrow.get("container")))
    .arrivesTo((Container)
    (queue.get("container")))       )
  {Symbol res = new Symbol(arrow);
    res.put("origin",activity.get("name"));
    res.put("destination",queue.get("name"));
    res.put("condition",queue.get("label"));
    res.put("entity_name",queue.get("entity_name"));
    return res;
  }else return null;
}//& output_link
```

Figure 11: Java code for Output_link rule.

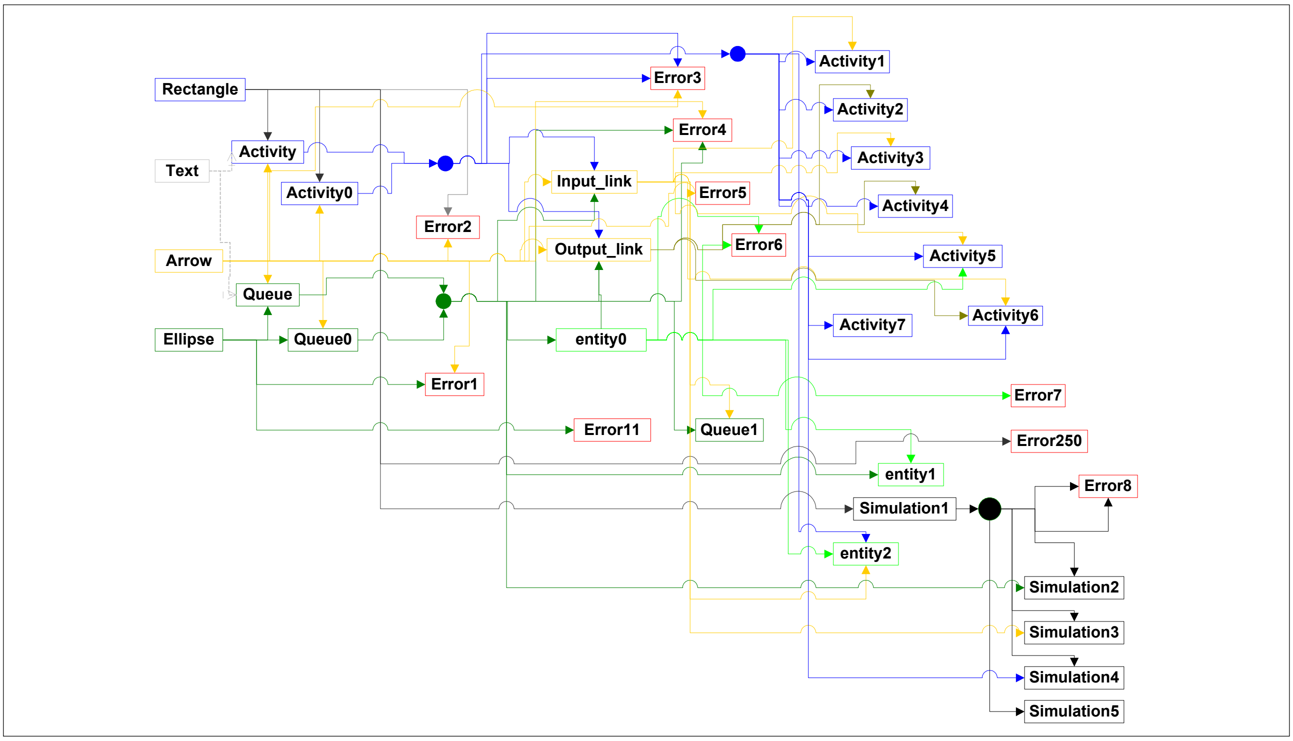


Figure 12: Graph with the grammar rules interfaces, and interdependencies

The Output_link rule is just one example that can be found in the central region of Figure 12:
The translation is archived by an engine (the SMAIS-GAP) that successively searches patterns in the Diagram under analysis to try to apply each grammar rule (in a specific order). Applying a rule, means rewriting or creating one symbol in the Diagram (some rules may also remove symbols from the diagram).

To the Output_link rule, the engine must know that this rule needs: one activity, one arrow and one queue. The engine then picks all combinations of symbols from the diagram under analysis, submitting different sequences with (activity, arrow, queue), to the rule. The rule will then return *null* if the symbols are inappropriate, or return a new symbol to the diagram. Attributes of each new symbol are “richer” than previous (collecting pieces of code). For this grammar, after several rule modifications, the generated program is represented within an attribute of a synthesized symbol (*simulation*).

7 XML FILE INTERCHANGE FORMAT

The objective was to give birth to a proven visual language (an example of a complete program written in this language can be found in Figure 8). This program is not dependent on the translator/compiler used. We defined an XML format (eXtended Markup Language) making possible to store it with all the attributes and to transfer it to any application in a text file (since to keep it in bitmap format (as a photo) would not obviously be appropriated).

In Figure 13 we can find the DTD corresponding to the defined XML format. In Figure 14 we can see a portion of one XML file corresponding to the barman problem. Each file is a book: a collection of sheets. Each sheet contains one ACD diagram. (Books can have one or more sheets).

```
<?xml version='1.0' encoding='UTF-8'?>
<!--
<?xml version="1.0"?>
<!DOCTYPE Book SYSTEM "ACD.dtd">
<Book>
...
</Book>
-->
<!ELEMENT Visible (PCDATA)>
<!ELEMENT Value (PCDATA)>
<!ELEMENT Property (Visible|Type|Value)*>
<!ATTLISTProperty
  Name CDATA IMPLIED
 >
<!ELEMENT Foreground (PCDATA)>
<!ELEMENT Background (PCDATA)>
<!ELEMENT Type (PCDATA)>
<!ELEMENT Y (PCDATA)>
<!ELEMENT X (PCDATA)>
<!ELEMENT Shape
  (Property|Foreground|Background|Type|Height|W
  idth|Y|X)*>
<!ATTLIST Shape
  Type CDATA IMPLIED
  Uid CDATA IMPLIED
 >
<!ELEMENT Height (PCDATA)>
<!ELEMENT Width (PCDATA)>

<!ELEMENT WorkSheet(Shape|Height|Width)*>
<!ATTLIST WorkSheet
  Order CDATA IMPLIED
  Name CDATA IMPLIED
 >
<!ELEMENT Book (WorkSheet)*>
<!ATTLIST Book
  Name CDATA IMPLIED >
```

Figure 13: DTD file (XML specification)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <!--
 Last Saved on Sat Jan 08 22:28:37 BST 2005
 -->
- <Book Name="Bartender">
- <WorkSheet Name="Global ACD" Order="0">
  <Width>860</Width>
  <Height>580</Height>
- <Shape Uid="001" Type="Rectangle2D">
  <X>9280.0</X>
  <Y>9900.0</Y>
  <Width>140.0</Width>
  <Height>80.0</Height>
  <Type>Rectangle2D</Type>
  <Background>#C0C0C0</Background>
  <Foreground>#000000</Foreground>
- <Property Name="name">
  <Value>POUR</Value>
  <Type>String</Type>
  <Visible>true</Visible>
  </Property>
- <Property Name="priority">
  <Value>1</Value>
  <Type>String</Type>
  <Visible>true</Visible>
  </Property>
- <Property Name="duration">
  <Value>poisn(6)</Value>
  <Type>String</Type>
  <Visible>true</Visible>
  </Property>
  </Shape>

  ...
  </WorkSheet>
  </Book>
```

Figure 14: Part of a DCA in XML (with one object).

8 JAVA-BSF – AUTOMATIC PROGRAM GENERATION

Running SMAIS-GAP, using the "ACD GRAMMAR (to BSF)" and giving the XML file as input program, we get the generated "bsfProgram.java", see Figure 15.

```
------------------ ( 1 - Generation ) ---------------------------
Starting S+ (ACD->BSF generation):
     C:\Documents and Settings\LSD\SMAIS_ROOT\ACD.xml ->
C:\Documents and Settings\LSD\SMAIS_ROOT\bsfPack\bsfProgram.java

C:\Documents and Settings\LSD\SMAIS_ROOT\ACD.xml -> Read 36
Graphical Symbols. In [341ms]
.........................................................................................
....................................
C:\Documents and Settings\LSD\SMAIS_ROOT\bsfPack\bsfProgram.java <-
Generated Program with 335 lines. [9654ms]

----------------------- ( 2 - Compilation ) ----------------------
C:\jdk1.5.0\bin\javac "bsfPack\BsfProgram.java"
```

Figure 15: Screenshot of Script running, calling SMAISGAP (Generation) and compiling it

The generated program is a low-level simulation program written in a general purpose programming language including an event-driven executive that uses a java simulation library – BSF (Basic Simulation Facility) (Rodrigues 1987).

In this Example, SMAIS-GAP read the 36 graphical objects from the file 'ACD.xml' and generated a fully commented program with 335 lines in about 10 seconds. An extract of the automatically generated program from the Bartender ACD, including all comments, can be found below in Figure 16.

```
//||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| #####    #
//||PROGRAM AUTOMATICALLY GENERATED BY AIMS(JavaBSF ver. 1.9) #        #
//||         Authors: lsd & agr @dps.uminho.pt                 #####  #####
//||       URL: www.dps.uminho.pt/oio/simulation                   #    #
//||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| #####    #

// Program 'BARMAN Problem' Generated by S+ : Universidade do Minho,

//Queues Declaration and constants definition:
  static int  FEvent = 1;  //File to hold futur events
  static int  CLEAN = 2;  //queue to entity:GLASS
  static int  FULL = 3;  //queue to entity:GLASS
  static int  DIRTY = 4;  //queue to entity:GLASS
  static int  DOOR_FREE = 5;  //queue to entity:DOOR
  static int  READY = 6;  //queue to entity:customer
  static int  WAIT = 7;  //queue to entity:customer
  static int  OUTSIDE = 8;  //queue to entity:customer
  static int  IDLE = 9;  //queue to entity:BARMAN
  static int  ARRIVE_DOOR = 10;  //entity: DOOR in activity: ARRIVE
  static int  ARRIVE_customer = 11;  //entity: customer in activity: ARRIVE
  static int  POUR_customer = 12;  //entity: customer in activity: POUR
  static int  DRINK_customer = 13;  //entity: customer in activity: DRINK
  static int  POUR_GLASS = 14;  //entity: GLASS in activity: POUR
  static int  POUR_BARMAN = 15;  //entity: BARMAN in activity: POUR
  static int  DRINK_GLASS = 16;  //entity: GLASS in activity: DRINK
  static int  WASH_BARMAN = 17;  //entity: BARMAN in activity: WASH
  static int  WASH_GLASS = 18;  //entity: GLASS in activity: WASH

//Events List Declaration and constants definition
  static int  Code_End_simulation = 1; //Code to the last event: "end of
simulation"
  static int  Code_End_ARRIVE = 2;  //Code to the event: "end of ARRIVE"
  static int  Code_End_WASH = 3;  //Code to the event: "end of WASH"
  static int  Code_End_DRINK = 4;  //Code to the event: "end of DRINK"
  static int  Code_End_POUR = 5;  //Code to the event: "end of POUR"

  static int total_number_of_events=5;

  static int  code_warm_up=0;
  static int  code_report=99;
       o o o
//============= Program Main Loop =============

 while (event_code != Code_End_simulation){
  r=Bsf.remove(FEvent,Clock);
    event_code=r.at2;
    Clock=r.time;
  if (event_code==Code_End_ARRIVE) end_of_ARRIVE();
  if (event_code==Code_End_WASH) end_of_WASH();
  if (event_code==Code_End_DRINK) end_of_DRINK();
  if (event_code==Code_End_POUR) end_of_POUR();
  if (event_code==code_warm_up) Bsf.reset_statistic(Clock);
 } //& MainLoop
System.out.println(Bsf.report(Clock));
} //& main

//@@@@@@@@@@  Activities BEGIN  @@@@@@@@@@@
public void begin_of_WASH(){
 while(true){
  if( Bsf.number_in_queue(IDLE)<1) return;  //BARMAN available?
  if( Bsf.number_in_queue(DIRTY)<3) return;  //GLASS available?
  Bsf.set_distribution(WASH);
  int duration= (int)5;
 //Schedulle end of WASH activity:
  Bsf.insert(FEvent, Clock, Clock+duration, Code_End_WASH);
  r=Bsf.remove(IDLE,Clock);   //remove one BARMAN from queue IDLE
(returns r.time,r.at1,r.at2)
  Bsf.insert(WASH_BARMAN,Clock,Clock+duration,r.at2);   //puts one
BARMAN in the activity WASH
 for(int i=3;i>0;i--){ // 3 GLASS(s) necessary for this activity:WASH
  r=Bsf.remove(DIRTY,Clock);   //remove one GLASS from queue DIRTY
(returns r.time,r.at1,r.at2)
  Bsf.insert(WASH_GLASS,Clock,Clock+duration,r.at2);  //puts one GLASS in
the activity WASH
  }
 }
} //& begin_of_WASH
```

```
        o o o
//@@@@@@@@@@    Activities END    @@@@@@@@@@@@
public void end_of_WASH(){
  r=Bsf.remove(WASH_BARMAN,Clock); //(returns r.time,r.at1,r.at2)
  Bsf.insert(IDLE,Clock,Clock,r.at2);
 for ( int i=3; i>0; i--){
  r=Bsf.remove(WASH_GLASS,Clock); //(returns r.time,r.at1,r.at2)
  Bsf.insert(CLEAN,Clock,Clock,r.at2);
 }
  //Attempts to start subsequent activities that may be viabilized
  begin_of_POUR();  //priority=1
  begin_of_WASH();
} //& end_of_WASH
        o o o
//End of Program 'BARMAN Problem'
```

Figure 16: Extract of Java generated simulation program

The Figure 17, below, contains a screenshot of the bartender program execution, with configurable initial entities allocation, progressive running bar and final report.

```
---------------------------- ( 3 - Execution ) ------------------
INIT:  12GLASS>DIRTY  1DOOR>DOOR_FREE  10customer>OUTSIDE
1BARMAN>IDLE
...10%...20%...30%...40%...50%...60%...70%...80%...90%..100%  [in 40ms]

Relatorio em t = 1000 (Warm-up period 0-120)
Fila                      In    Out   Now    Av-stay    Av-len
 1 FEvent                267    264     3      9.189     2.614
 2 CLEAN                 101     94     7     75.511     7.669
 3 FULL                   94     94     0      0.000     0.000
 4 DIRTY                  96     93     3     22.129     2.365
 5 DOOR_FREE              45     45     0      0.000     0.000
 6 READY                  94     94     0      0.000     0.000
 7 WAIT                   94     94     0      2.138     0.228
 8 OUTSIDE                52     45     7    147.667     7.334
 9 IDLE                  125    125     0      1.328     0.189
10 ARRIVE_DOOR            46     45     1     19.844     1.000
11 ARRIVE_customer        46     45     1     19.844     1.000
12 POUR_customer          95     94     1      5.947     0.635
13 DRINK_customer         94     93     1      7.516     0.802
14 POUR_GLASS             95     94     1      5.947     0.635
15 POUR_BARMAN            95     94     1      5.947     0.635
16 DRINK_GLASS            94     93     1      7.516     0.802
17 WASH_BARMAN            31     31     0      5.000     0.176
18 WASH_GLASS             93     93     0      5.000     0.528
```

Figure 17: Screenshot of program execution and final statistical report

This task (automatic translation) is quite complex since it implies different abstraction levels, from an high level (activity world view) to a much lower level (event world view) using a generic programming language.

9 ARENA MODEL CREATOR

The hard part of this task was to create an activity-based executive over a process-oriented environment. The executive developed shows to be more efficient than a three-phase approach since we used a message passing mechanism that only tries to start an activity when entities have arrived to one of its predecessor queues.
Given that we generate a model into a high-level simulation environment it becomes possible to overcome limitations of the activity-based approach since the model may be completed in ARENA.

Figure 18 and Figure 19 illustrate the model as automatically created in Arena for the Bartender Problem and a Screenshot of animation phase. Figure 20 have details of two activities with predefined statistics. Figure 21 have a screenshot with part of the logic template (code) of the activity block.

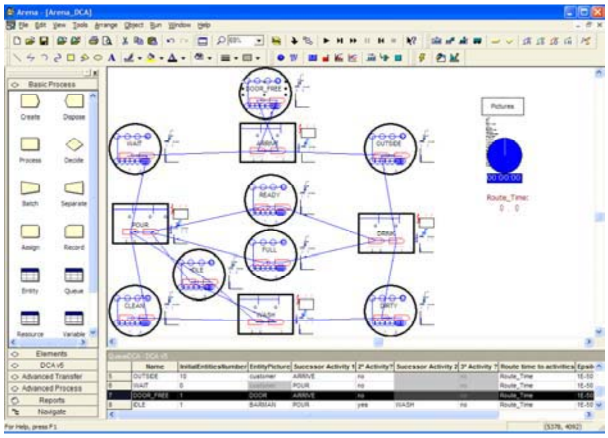


Figure 18: Screenshot of one automatically builded model in Arena (Bartender)

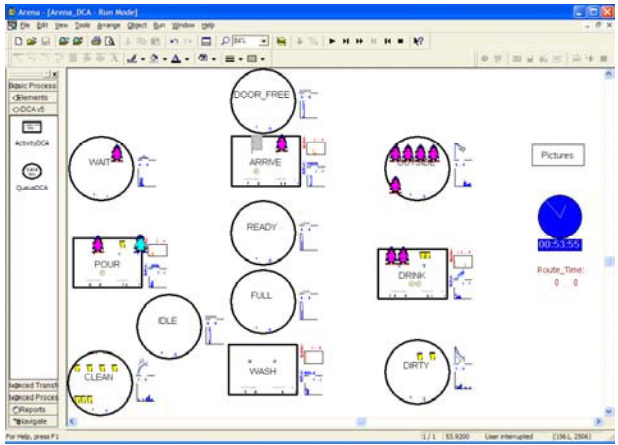


Figure 19: Screenshot of animations in Arena (Bartender)

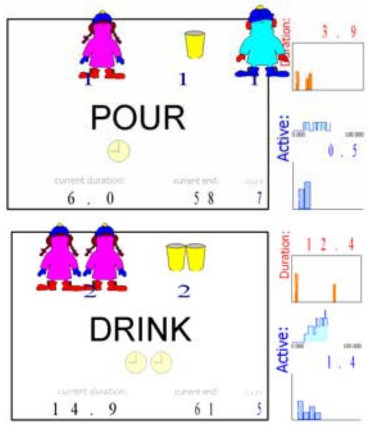


Figure 20: Screenshot of Arena animation details, with statistics (Bartender)

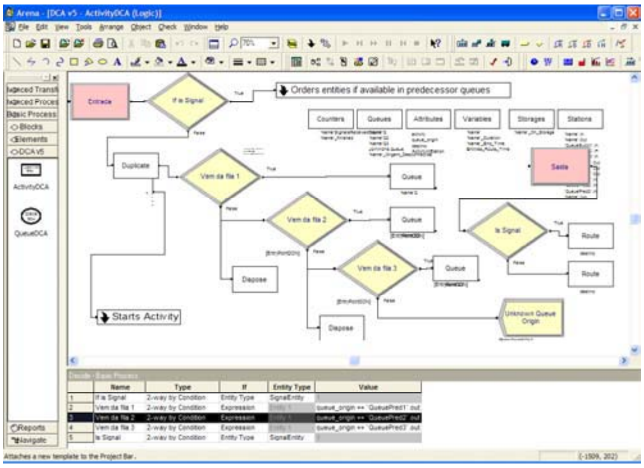


Figure 21: Activity definition - Part of Arena Logic Template.

10 CONCLUSION

The work presented in this paper could constitute a major step towards the generalisation of the use of simulation. In fact, we suggest the use of a simple interface (Activities Cycle Diagrams) to model a real situation. Then we present a tool capable of generating a simulation program. Based on event scheduling simulation modelling philosophy, our tool automatically generates a program to use Basic Simulation Facility routines. Based on process flow simulation modelling philosophy, our tool automatically generates an ARENA program. Furthermore the mentioned automatic generation of simulation programs does not require expertise in simulation.

REFERENCES AND BIBLIOGRAPHY

Banks, Jerry. 1998. *Handbook of Simulation*. Industrial and Systems Eng. Georgia, Wiley.

Bekey, George A., ed. 2003. *Modeling and Simulation Theory and Practice (A Memorial Volume for Professor Walter J. Karplus)*. Kluwer Academic Publishers,

Bell, Brigham. 1992. "Using Programming Walkthroughs to Design a Visual Language". Ph.D. dissertation, Department. Of Computer Science, University of Colorado, Boulder, Technical Report CU-CS-581-92.

Bennett, Brian S. 1995. *Simulation Fundamentals*, Prentice Hall.

Campione, M.; K. Walrath; A. Huml. 1999. *The Java Tutorial Continued, The Rest of the JDK*, Addison Wesley.

Clementson, A.T. 1982. *Extend Control and Simulation Language, Users Manual*. (ECSL). CLE.COM Ltd, University of Birmingham.

Costagliola, Gennaro; Andrea de Lucia; Sergio Orefice and Genoveffa Tortora. March 1995. "Automatic Generation of Visual Programming Environments". *Computer IEEE, vol.8* nº3, pp. 56-66.

Dias, Luís S. 1997. "Linguagens Visuais de Programação – Paradigmas e Ambientes", MSc Thesis (*in Portuguese*), Computer Science Department, University of Minho, Braga, Portugal.

Dias, Luis S. and A. Guimarães Rodrigues. 2002. "Language independent modelling of discrete event simulations – AIMS", *AIS'2002 – AI, Simulation and planning in High Autonomy Systems*, pp.195-200. SCS Publication.

Dias, Luis S. and A. Guimarães Rodrigues. 2002. "Towards simplicity in modelling for simulation", *Operational Research Society - Simulation Study Group. First biennial Simulation Study Group Workshop*, Birmingham, UK.

Filho, Wladimir De Lara Araújo and Celso M. Hirata. 2004. "Translating activity cycle diagrams to Java simulation programs". *IEEE 37th Simulation Symposium*, 18-22 April 2004. Page(s):157 - 164

Fishwick, Paul A. 1994. *Simulation Model Design And Execution; Building Digital Worlds*. Prentice Hall.

Freimer, M., T. Roeder, C. Harmonosky, L. Schruben, C. Standridge, I. Ståhl, 2004. You Are Going To Teach Simulation – Now What? Tips and Strategies. In *Proceedings of the 2004 Winter Simulation Conference*, ed. R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters. 2057-2065. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Garrido, José M. 2001. *Object-Oriented Discrete-Event Simulation with Java - A Practical Introduction*, Kluwer Academic/Plenum Publishers.

Harrell, Charles R. et.al. 2003. *Simulation using PROMODEL*, McGraw-Hill, 2nd edition.

Henriques, Pedro Rangel. 1992. "Atributos e Modularidade na Especificação de Linguagens Formais", PhD thesis, University of Minho, Braga, Portugal.

Karayanakis, N. M. 1995. *Advanced System Modeling and Simulation W/ Block Diagram.* Springer.

Kelton, W. David; Randall P. Sadowski, and David T. Strurrock, 1998-2004. *Simulation With Arena* (third edition). McGraw-Hill.

Kheir, Naim A., (Ed.). 1996. *Systems Modeling and Computer Simulation*. 2nd Ed. Dekker.

Law, Averill M. and W. David. Kelton. 1991. *Simulation Modeling & Analysis*. McGraw-Hill.

Martins, F. Mário. 1998. "Programação Orientada aos Objectos usando Java"*(in Portuguese)*, Computer Science Department, University of Minho, Braga, Portugal.

Mooney, Christopher Z. 1997. *Monte Carlo Simulation*. Sage Publications.

Morgan, B.J.T. 1984-1995. *Elements of Simulation*. Chapman&Hall.

Odum, Howard T. and Elisabeth C. Odum. 2000. *Modeling for all Scales: An Introduction to System Simulation*, Academic Press.

Overstreet, C. M. and Richard E. Nance. 2004. Characterizations And Relationships Of World Views. In *Proceedings* *of the 2004 Winter Simulation Conference*, ed. R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters. 279-287. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Pidd, Michael. 1984-1990. *Computer Simulation in Management Science*, Wiley.

Pidd, Michael. 2004. Simulation Worldviews - So What?. In *Proceedings of the 2004 Winter Simulation Conference*, ed. R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters. 288-292. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Repenning, Alex and T. Sumner. March 1995. "Agentsheets: A medium for Creating Domain-Oriented Visual Languages", *Computer IEEE, vol.8* nº3, pp.17-25.

Rocha, Jorge Gustavo. 1995. "Especificação de Linguagens Visuais de Programação", MSc Thesis (*in Portuguese*), Computer Science Department, University of Minho, Braga, Portugal.

Rodrigues, A. Guimarães. 1987-98. *Simulação*, Engineering School, University of Minho, Braga, Portugal.

Rodrigues, A. Guimarães. 1995. "Modelação em simulação discreta, (lição de síntese)", University of Minho, Braga, Portugal.

Ross, Sheldon. 2002. *Simulation*. Elsevier Academic Press.

Rubinstein, Reuven Y., and Benjamin Melamed. 1998. *Modern Simulation and Modeling*, Wiley.

Sargent, Robert G. 2004. Some recent advances in the process world view. In *Proceedings of the 2004 Winter Simulation Conference*, ed. R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters. 293-299. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Sarjoughian, Hessam S.; Francois E. Cellier, and Bernard P. Zeigler, (Editors). 2001. *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies: A Tribute to the 60th Birthday of Bernard P. Zeigler*. Springer Verlag.

Swain, James J. 1991-2003. "Simulation Software Survey". *OR/MS Today magazine* from Institute for Operations Research and the Management Sciences (INFORMS). Lionheart Publishing.

Tocher, K.D. 1963. *The Art of Simulation*. UNIBOOKS – English Universities Press.

Tumay, H. Harrington Kerim. 2000. *Simulation Modeling Methods To Reduce Risks and Increase Performance*, McGraw-Hill.

Varanda M. J., Henriques P. e Rocha J. 1997. "Concepção, especificação e implementação de Processadores de Linguagens Visuais". *II Simpósio Brasileiro de Linguagens de Programação*.

Zeigler, Bernard P.; Herbert Praehofer, and Tag Gon Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd ed. Academic Press.

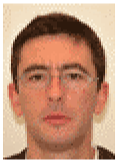
AUTHOR BIOGRAPHIES

LUÍS MIGUEL DA SILVA DIAS is a Lecturer at the Department of Production and Systems Engineering at the University of Minho. He has earned a degree in Informatics and Systems Engineering, and a Master in Informatics from the University of Minho. His interests are in modelling & simulation, operational research and visual languages.

<lsd@dps.uminho.pt>
<http://www.dps.uminho.pt>

ANTÓNIO JOSÉ M GUIMARÃES RODRIGUES is a full Professor in the Department of Production and Systems Engineering at the University of Minho. He is, since 2002, the Rector of the University. He has earned a Master's Degree and Ph.D. from the University of Birmingham in Production Engineering. Prof. Guimarães Rodrigues interests are in simulation, operational research and mathematical programming. He is Associate Editor of Investigação Operacional, the Portuguese Journal of OR.

<agr@reitoria.uminho.pt>



GUILHERME AUGUSTO BORGES PEREIRA is a Professor in the Department of Production and Systems Engineering at the University of Minho. He is the director of the industrial engineering and management *licenciatura* (first degree). He has earned a Master's Degree in Operational Research. and Ph.D. in Manufacturing and Mechanical Engineering from the University of Birmingham. His interests are in simulation and operational research.

<gui@dps.uminho.pt>

University of Minho was founded in 1973, is located in Braga and Guimarães, in the north of Portugal. Has more than 15000 students, and around 500 Ph.D. Teachers. <www.uminho.pt>