22

Manufacturing System Simulation Model Synthesis: Towards Application of Inductive Inference

Goran D. Putnik, João Almeida das Rosas
University of Minho, Production Systems Engineering Centre
Azurem, 4800 Guimaraes, Portugal, fax.:+351-53-510268
e-mail: {putnikgd, mpic_jar}@eng.uminho.pt

Abstract

A hypothesis of an inductive inference approach as a paradigm for automation of a simulation model synthesis process is presented. It differs from a standard approach in the sense that it does not refer the simulation model specification by some simulation programming language. The inductive inference algorithm/system infers a probably good approximation to a target system simulation model. It is expecting that the inductive inference approach could contribute to a productivity, correctness and independence of a human expert's experience in the process of the simulation model synthesis. An example, of the "standard" approach and of the inductive inference approach applied to a simple hypothetical production system, is given. Elements of the software tool developed, for inductive inference, are presented.

Keywords

Simulation, simulation model, simulation model synthesis, learning, inductive inference

1 INTRODUCTION

In this paper we are concerned with the simulation system model building issue. A hypothesis of the **inductive inference** approach **as a paradigm for automation of a simulation model synthesis process** is presented.

In the first part of the paper we present the concept of automation of the simulation concept. The objective of this part is to locate the problem we elaborate. In the second part of the paper we present a standard (conventional?) approach to the simulation model specification, based on an intuitive model of the simulation model synthesis process, applied to one hypothetical production system. In the third part of the paper we present the *automated inductive inference application* for a simulation model synthesis and elements of the software tool developed (for inductive inference).

L. M. Camarinha-Matos (ed.), *Re-engineering for Sustainable Industrial Production*

2 THE CONCEPT OF SIMULATION AND ITS AUTOMATION

Today a simulation is probably the most used tool for manufacturing systems design. Different authors specify a concept of simulation in different ways, for example (Guimarães Rodrigues, 1988), (Tempelmeier, Kuhn, 1993). Generally, we could say that the simulation, as a concept, is defined by processes performed on two levels:

1) The level of a model building, i.e. **simulation model synthesis process** and
2) The level of experimentation over the model, i.e. **the simulation as a process** (the simulation itself).

To be implemented, these processes need to have:

1) The **simulation concept logical (abstract, mathematical) definition**, specifying its 2 levels structure (referred above), and
2) The corresponded **physical implementation**.

The physical implementation of the simulation concept is a critical issue. It depends of the language type we use. Of the greatest importance is use of some formal language. It provides us with a possibility for **algorithmisation** and, further, for **automation** of simulation processes, or of the simulation model synthesis process itself. Automation of these processes implies their physical implementation in a computerised environment, defining specific software and hardware resources. This type of implementation we will call **machine implementation**.

Otherwise, it means that we have only an intuitive model, which is usually known, and performed, by a human. In this case, processes, within the concept of the simulation, are *not implementable in a computer environment*. We do not have a knowledge representation of the simulation processes and, thus, necessarily, they must be performed by a human.

Logical definition and corresponded implementations graphical symbols, we will use in the further text, are given in Figure 1.

Figure 1. Graphical symbols used for the simulation processes representation.

Two level simulation concept logical structure definition, is given in Figure 2. **The simulation model synthesis process outputs the simulation model**.

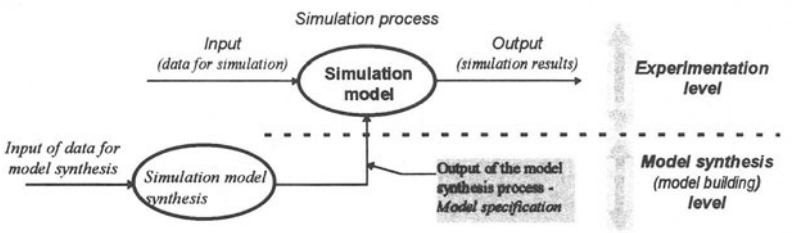

Figure 2. The simulation concept logical definition.

Our objective is to achieve a certain level of automation of all processes within the simulation concept which, is represented by the *implementation model 2*, Figure 3.

Unfortunately, due to enormous difficulties in a general design theory developments, application of the formal specification language was limited to the experimentation level, i.e. to the simulation model specification. It means that the model of the simulation model synthesis process is based on an intuitive model, *performed by a human*. This (intuitive) model **outputs a simulation model specification** in some *formal language* which is *implementable in a computer environment*, in some specific computer programming language, for example ECSL - Extended Control and Simulation Language.

The structure described above is represented by the *implementation model 1*, Figure 3. The simulation process, i.e. the simulation model specification, is implemented in a computer environment. This is *the output of an intuitive simulation model synthesis process on the model building level. **The model synthesis process itself is not specified !***

We will call this approach the **standard (conventional?) approach**.

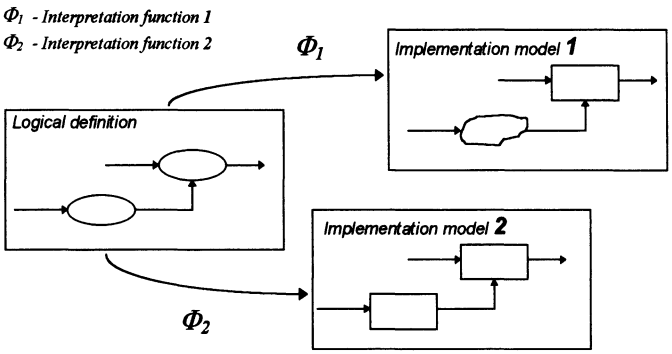

Figure 3. The simulation concept logical definition and its implementations.

Today, for the simulation model specification and implementation, a number of programming languages, and corresponded interfaces for data input and output, are developed. For example, GPSS, SIMULA, SIMSCRIPT, GASP, SLAM, ECSL, etc., ***but still not for the model synthesis process !***

An example of a higher level interface language for the simulation model specification is Activity Cycle Diagram (ACD), which is a kind of specific graph language. A specific software tool, in this case it is CAPS-ECSL (CAPS - Computer Aided Programming of Simulation) simulation software tool, compiles a computer readable format of an ACD description and outputs a simulation model specification in the simulation model specification "original", "lower" level, language, in this case in ECSL language. Improved "*implementation model 1*" of the simulation model specification is represented in Figure 4.

The problem is that the **simulation model synthesis process** itself is still performed intuitively by the human, which is very a serious limitation from some points of view.

Thus, we came to the problem of the **simulation model synthesis process automation**. Implementation of the simulation concept with the **automated simulation model synthesis process** is shown in Figure 8, chapter 5.2.

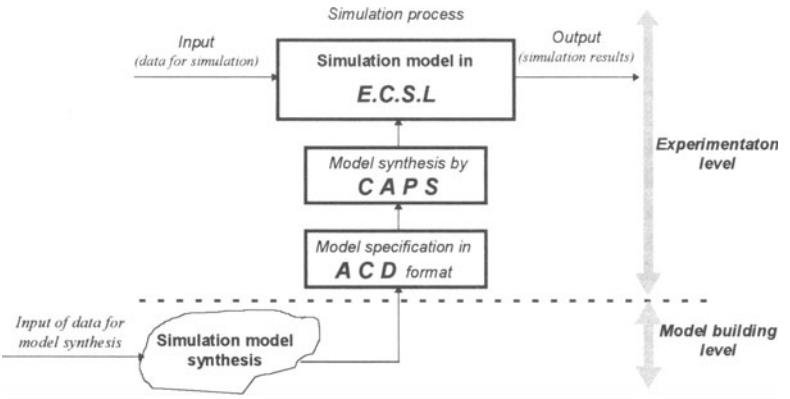

Figure 4. Standard approach to a simulation model synthesis.

As this approach requires automation of the process usually considered as a creative one, or, as an intelligent one, we will call this approach the **"intelligent" approach**.

3 PRODUCTION SYSTEM EXAMPLE - PROBLEM DEFINITION

A production system to be analysed represents an assembly line for car-radios, Figure 5. The line has six workstations: 3 workstation for chassis, boards and mechanisms assembly, 1 workstation for wiring, 1 workstations for quality control and 1 workstations for packaging.

The production process is characterised by a high level rejection rate of finished products caused by defects on chassis, boards, mechanisms and wiring. These defects are identified by the quality control performed on the corresponded workstation. Car-radios with defects are sent to reparation and after the reparation car-radios are introduced again on the assembly line to be refined. Defects originated by the wiring are repaired by the new wiring.

The objective is to design and to simulate the system. The system simulation should determine the impact of the strategies applied for resources management with the objective to optimise the system performances, for example, to obtain higher efficiency, higher production rate, optimisation of human resources, etc.

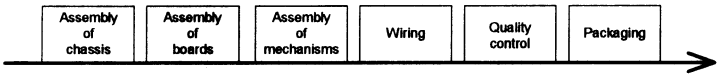

Figure 5. Car-radios assembly line.

4 STANDARD APPROACH TO A SIMULATION MODEL SYNTHESIS

The model synthesis is performed through the phases (Figure 4.):

1. Preparation of a complete detailed Activity Cycle Diagram (ACD). Preparation of the simulation model attributes: the maximum number of each type of entity, the queue discipline, the length of simulation, activity priorities, etc.

2. Describing the Activity Cycle Diagram in a format readable by the computer - Input and editing of cycles and their attributes in CAPS;
3. CAPS generates a file containing the ECSL program for simulation.

Simulation model concept specification - Activity Cycle Diagram

The specification of the model concept by the ACD diagram is conceived as follows. A cycle shows the sequence of states through which the typical entity (product, task) will pass. There are two types of states: activities and queues.

Three main conventions for an ACD diagram drawing, which we will use, are:

1. For any particular type of entity there will be always an alternation of activities and queues;
2. Activities are represented by rectangles and queues by circles inside which is written the name of the state;
3. The states are connected by arrows to show the valid sequences to show the valid sequence in which states can occur;

In our experiment, activities (processes) and queues, on the assembly line, are named as:

ACTIVITIES		*QUEUES*	
CHRD	- radios arrival (fictive activity - the *task* is entered the line)	EXT	- external world
		RDESP	- task radio is waiting for a chassis
CHMONT	- assembly of chassis	ESPPL	- task radio is waiting for a board
PLMONT	- assembly of boards	MESPM	- task radio is waiting for a mechanism
MMONTA	- assembly of mechanisms	AFIESP	- task radio is waiting for a wiring
TEAFIN	- wiring	RDECQ	- task radio is waiting for a quality control
RDCQ	- quality control	ERM	- task radio is waiting for a chassis reparat.
RDRMEC	- mechanism reparation	ERP	- task radio is waiting for a board reparation
RDRPL	- boards reparation	ERC	- task radio is waiting for a mechanism reparation
RDRCH	- chassis reparation		
REEMB	- packaging	EMBRD	- task radio is waiting for packaging

Corresponded ACD diagram of the system given is represented in Figure 6.

The second phase of the simulation model synthesis by the CAPS software tool is input and editing of the simulation model ACD diagram defined above. Description of the ACD diagram, for the system given, in the CAPS readable format, for the entity RADIO is:

RADIO,600,QEXT,ACHRD,QRDESP,ACHMONT,QESPPL,APLMONT,QMESPM,AMMONTA,QAFIESP,
ATEAFIN,QRDECQ,ARDCQ,QERM,RSA,40,QERP,RSA100,QERC,RSA,10,QAFIESP,RSA,50,QEMBRD,
AREEMB,QEXT,+,QERM,ARDRMEC,QAFIESP,+,QERP,ARDRPL,QAFIESP,+,QERC,ARDRCH,QAFIESP

For example, the cycle presented in Figure 7a, is represented in the input file as (in bold):

RADIO,*600*,**QEXT,ACHRD,QRDESP,ACHMONT,QESPPL,APLMONT,QMESPM,AMMONTA,**
QAFIESP,ATEAFIN,QRDECQ,ARDCQ,*QERM,RSA,40,QERP,RSA100,QERC,RSA,10,QAFIESP,RSA,50,*
QEMB
RD,AREEMB,QEXT,*+,QERM,ARDRMEC,QAFIESP,+,QERP,ARDRPL,QAFIESP,+,QERC,ARDRCH,QAFIESP*

and the cycle presented in Figure 7b, is represented in the input file as (in bold):

RADIO,*600,QEXT,ACHRD,QRDESP,ACHMONT,QESPPL,APLMONT,QMESPM,AMMONTA,*
QAFIESP,TEAFIN,Q
RDECQ,ARDCQ,*QERM,RSA,40,*
QERP,RSA100,*QERC,RSA,10,QAFIESP,RSA,50,QEMBRD,REEMB,QEXT,+,*
QERM,ARDRMEC,QAFIESP,+,**QERP,ARDRPL,QAFIESP**,*+,QERC,ARDRCH,QAFIESP*

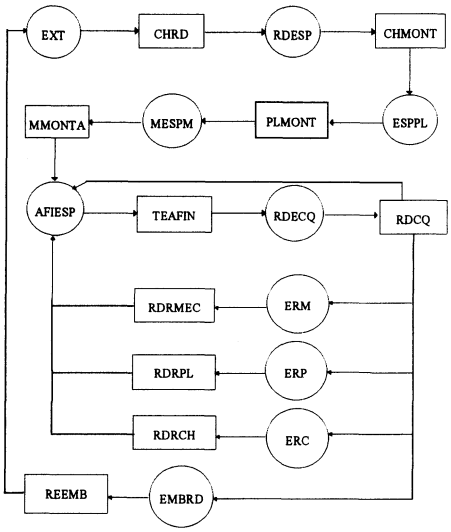

Figure 6. Activity Cycle Diagram for the manufacturing system analysed.

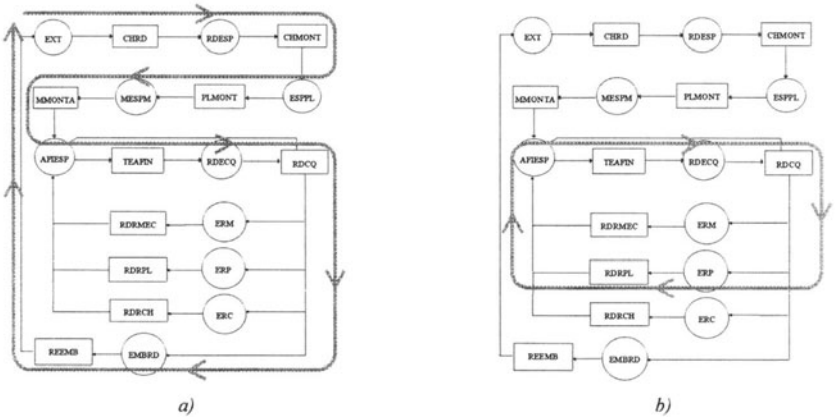

Figure 7. Activity cycles examples.

5 INDUCTIVE INFERENCE APPROACH TO AUTOMATED SYNTHESIS OF THE SIMULATION MODEL

5.1 Inductive inference

The inductive inference as a learning paradigm means that a learning algorithm takes, in an iterative process, as input, a set of particular examples (one by one) by taking into account all

given examples and by adding some new rules to build some hypothesis about correct general rule. Inductive inference can be defined on many ways. For example (Angluin, Smith, 1983):

"Inductive inference is a system which try to infer a general rules from examples", or

"Inductive inference is a process of hypothesising a general rule from examples"

The set of examples given, denoted by I, is the set of sentences which belong to the language we want to learn. Thus, the inductive inference is attempt to find such grammar that $I \subset L(G)$. The methodological definition is (Miclet, 1990):

Definition 1: *"Inductive inference is automatic learning of formal grammars from finite subset of the languages they generate"*

Learning is provided by the **separated** learning/design algorithm for which the object transformation system - a general rule or a machine - is output (it means, we learn about the rule or we design the machine - both are *transformation systems*), accordingly to the given set of examples. Learning/design algorithm is the **meta-transformation system** for the object transformation system.

Besides the learning algorithm itself, the most important problems is quantification of the learning process attributes and of the learned concept attributes.

The paradigm of Probably Approximately Correct (PAC) learning algorithm provides us with a quantitative measure for evaluation of the learning process and of the concept learned. This paradigm is introduced by L. Valiant (Valiant, 1984).

The PAC learning algorithm can be described as follows. Let the correct output concept, or the target concept, be denoted by f, and the hypothesis, or learned concept, be denoted by g. Both belong to a corresponded class of concepts F. The learning algorithm will output, with probability at least $(1-\delta)$, good approximation g of the target concept f, where by "good approximation" is meant that the probability that f and g differ on a randomly chosen string (example, instance) is at most ε , i.e., $P(f \Delta g) \leq \varepsilon$. Parameters δ and ε are called "confidence parameter" and "error of learning" respectively, and they can be controlled by the number of examples and by the number of hypothesis changing. Thus, "the output of the learning algorithm is *probably* a good *approximation* to the target concept" (Natarajan, 1991).

Time complexity of the learning algorithm is directly connected with a representation class by which we represent a problem, or, in other words, with a knowledge representation system.

5.2 Synthesis of the simulation model by inductive inference

The inductive inference approach refers **the simulation model synthesis, not the simulation model description compilation**, as it is by the standard approach (compilation of the system specification from one language to another language).

The inductive inference algorithm infers the model **based on some incomplete information about the objective system (which, information, is different than the system description itself**. For example, this information could be examples of the system behaviour.

Information, about the systems behaviour, given to the learning algorithm does not present all possible system's behaviours, only some of them. It means that the system does not infer the "ideal" model, but **the inductive inference algorithm/system infers *probably* a good *approximation* to the target model**, in our case, **to the system simulation model**.

The inductive inference approach to the system simulation model synthesis is conceived as a structure shown in Figure 8. The inductive inference algorithm will take as an input description of the system functionality in some "very high level" language. Output of the inductive inference algorithm will be the system simulation model specification in the "ACD language". This is from the very practical reasons: 1. the "pipeline" ACD-CAPS-ECSL (the sequence of the compilations) is already developed and 2. the ACD diagram can be represented in some "graph language", which, in general, belongs to the class of Context-Free Languages and in some special cases to the class of Regular Languages, for which we already have developed inductive inference algorithms.

In our experiment, we use a *Finite State Automata* (FSA) as a representation class.

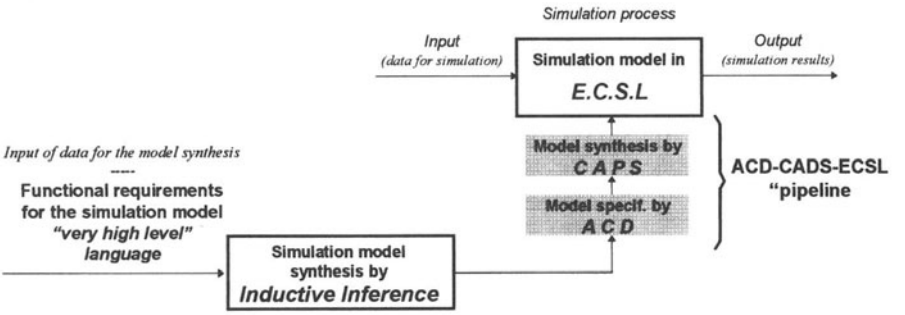

Figure 8. Synthesis of the simulation model - "intelligent" or inductive inference approach.

5.2.1 Problem representation by the functional requirements for the system : examples of production processes required as examples for learning

As it can be seen on the ACD diagram, of the system under consideration, each activity is preceded by a corresponded queue. Thus, we will attach to each pair "queue-activity" a particular name as bellow, and as it is presented in Figure 9.:

- EXT - CHRD - **o01**
- RDESP - CHMONT - **o02**
- ESPPL - PLMONT - **o03**
- MESPM - MMONTA - **o04**
- AFIESP - TEAFIN - **o05**
- RDECQ - RDCQ - **o06**
- ERM - RDRMEC - **o08**
- ERP - RDRPL - **o09**
- ERC - RDRCH - **o10**
- EMBRD - REEMB - **o07**

The **system functionality description is introduced by giving the process sequences**, which reflect the system functionality.

This type of the functionality representation represents a kind of *a "very high level" language for the system specification*.

We distinguish two types of the system functionality, i.e. examples of the system behaviour. These will be represented by a pair (x, y), where x will be a string, a sequence of process/operation symbols representing the system functionality, and the $y=f(x)$, where $y \in \{1,0\}$. If $y=f(x)=1$, then (x, y) is a ***positive*** example, representing the required system functionality, and if $y=f(x)=0$, then (x, y) is a ***negative*** example, representing the system functionality to be avoided.

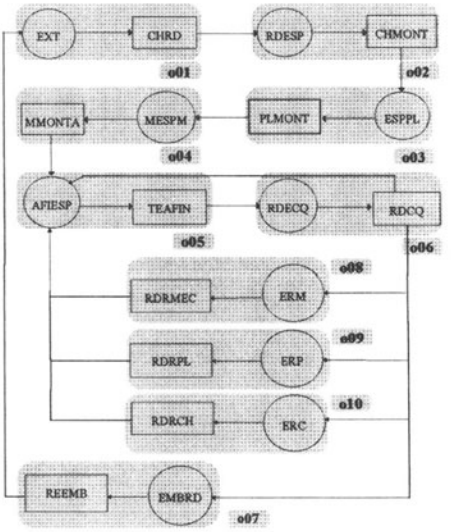

Figure 9. Renaming Activity Cycle Diagram for the manufacturing system analysed.

For example, one positive example of the system functionality can be represented as:

```
"o01o02o03o04o05o06o07"          1
```

One negative example, of the system functionality, to be avoided, can be represented as:

```
"o02o01o02o03o04o05o06o07"       0
```

5.2.2 Synthesis of the model in the FSA representation class, or, *Learning Finite State Automata*

Learning Finite State Automata (FSA) in general, is a process which belongs to the NP class. So, it is necessary to apply nondeterministic (ND) algorithm for learning with polynomial time complexity, in order to be efficient. Realisation of the ND learning algorithms is performed by introducing an additional source of information, carried on by a routine called ORACLE. In a real system, the oracle can be a human expert, database, deduction system, which will implement corresponded expertise, heuristics, helping in searching for the target concept. Of course, we are interested in outside help, during the learning process, to be minimal.

The second condition for the learning process efficiency is related to the sample size (number of the examples for learning), which is necessary to be the polynomial-sample size.

Algorithm defined by Angluin D. (Angluin, 1987), presented bellow, *satisfies* above mentioned conditions for efficiency (for the proof see (Angluin, 1987) or (Natarajan, 1991)).

The algorithm constructs a hypothesis about FSA for the target concept and then tests the hypothesis over examples. Algorithm halts if it is found the hypothesis that is valid with high probability. Otherwise it uses counterexample for construction a new and improved hypothesis.

To algorithm is permitted to make checking of membership of each sentence to the target concept. This is realised by a subroutine called MEMBER (performing ORACLE function).

A nondeterministic algorithm for FSA learning [1] :

input : ε, δ, n .
begin
set S = { λ } and E = { λ } ;
set i = 1 ;
call MEMBER on λ and each a ∈ Σ ;
build the initial observation table (S, E, T) ;
repeat forever
while (S, E, T) is not closed or not self-consistent **do**
if (S, E, T) is not self-consistent **then**
find x, y ∈ S , a ∈ Σ , and z ∈ E such that
row(x) = row(y) but T(xaz) ≠ T(yaz) ;
E = E ∪ {az} ;
extend T to (S ∪ S·Σ)·E by calling MEMBER ;
end
if (S, E, T) is not closed **then**
find x ∈ S , a ∈ Σ such that
for all y ∈ S , row(y) ≠ row(xa) ;
S = S ∪ {xa} ;
extend T to (S ∪ S·Σ)·E by calling MEMBER ;
end
end
let M = M(S, E, T) ;
make $m = \frac{1}{\varepsilon}\left[2\cdot\ln(i+1) + \ln\left(\frac{1}{\delta}\right)\right]$ calls of EXAMPLE ;
let Z be the set of examples obtained ;
if M is consistent with Z **then**
output M and halt;
else
let (x,y) be an example in Z with which M is
not consistent ;
add x and all prefixes of x to S ;
extend T to (S ∪ S·Σ)·E by calling MEMBER ;
end
i = i + 1
end
end

[1] For detailed explanation of the algorithm. see references. Also, for the purpose of our experiment, it is not considered the control of parameters ε and δ.

5.2.3 Synthesis of the model

Synthesis of the simulation model is performed through the input of the system functional requirements - *examples for learning*, and running the inductive inference algorithm which outputs the simulation model in FSA representation class. During the algorithm run, the designer communicates with the MEMBER routine defining *implicitly* the heuristic which helps the model synthesis.

The first part of the input is input of the alphabet used:

```
..define "o01"
  define "o02"
  define "o03"
  define "o04"
  define "o05"
  define "o06"
  define "o07"
  define "o08"
  define "o09"
  define "o10"
```

In the second part of input, examples (for learning/synthesis) are given.

In *the first experiment* it is given the description of the system functionality by the positive example of a process sequence (which reflects the system functionality):

```
"o01o02o03o04o05o06o07"        1
```

The system, generated over this example of the system functionality was not satisfactory.

In *the second experiment*, to the first (positive) example, it is added the second example, the negative one, giving the information about the system functionality to be avoided:

```
"o01o02o03o04o05o06o07"        1
"o02o01o02o03o04o05o06o07"     0
```

The output, i.e. the system generated over these examples of the system functionality, in the FSA representation class, is given:

```
Recognizer = RADIOS

Initial_state = {0}
Final_states = {7}
Alphabet = {o01,o02,o03,o04,o05,o06,o07,o08,o09,o10}
Productions
{
   S0 -> o01S1          0
   S1 -> o02S2          0
   S2 -> o03S3          0
   S3 -> o04S4          0
   S4 -> o05S5          0
   S5 -> o06S6          0
   S6 -> o07S7          0
   S6 -> o08S4          0
   S6 -> o09S4          0
   S6 -> o10S4          0
}
```

The system generated can be evaluated as satisfactory. The graphical representation of the output generated is given in Figure 10. It is evident the similarity with the ACD representation of the system simulation model defined in the chapter 5.2.1, Figure 9.

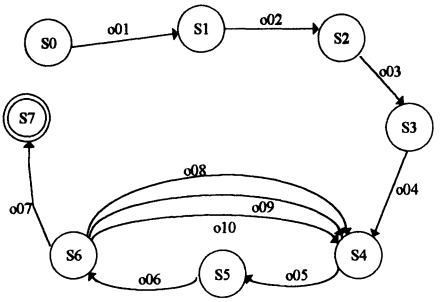

Figure 10. System simulation model generated in the FSA representation class.

6 SOFTWARE TOOL DEVELOPED

The inductive inference algorithm presented is implemented in the C language in the PC environment. In Figures 11a,b,c,d. are given, respectively: the screen of the introductory menu, the screen of the data input, the screen of the dialogue with the MEMBER (oracle) routine and the screen of the output FSA.

7 CONCLUSION

The *inductive inference* approach presented in the paper, represents *a paradigm for automation of the simulation model synthesis process*, and *does not refer the simulation model description compilation*, as it is by the standard approach. Inference algorithm takes as input an incomplete information about the system functionality (examples for learning), and outputs the system simulation model, which is the hypothesis about the targeted "ideal" manufacturing system/enterprise model. The *inductive inference algorithm/system infers probably a good approximation to the target model, in our case, to the system simulation model*. We call this approach **the "intelligent" approach**.

The simulation model generated is a **hypothetical model** about the target manufacturing system/enterprise model. This is the real case we have in manufacturing system/enterprise re-engineering problems. The first step should be to define the target model. The best practise is if we can *test different hypothesis, i.e. different hypothetical manufacturing system/enterprise models*, candidates to satisfy our objectives.

The approach, we have proposed, is believed to be highly applicable and useful tool for an **manufacturing system/enterprise organisation re-engineering**. This expectation is based on very high intelligence built in the concept of inductive inference, together with implemented capabilities for quantitative measuring of learning/synthesis process **enabling the best engineering approach**. We propose the inductive inference approach expecting that it could contribute to a manufacturing system/enterprise re-engineering by:

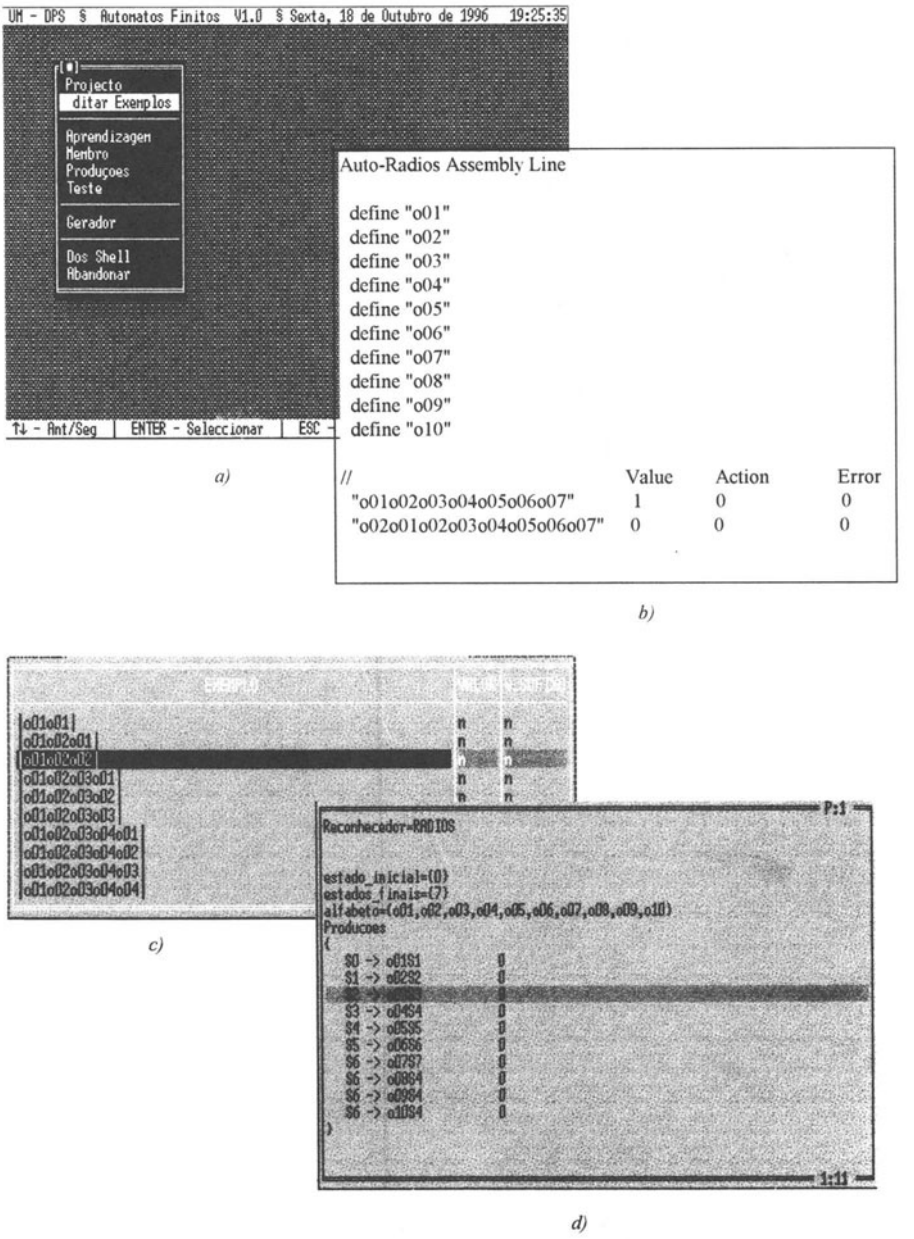

Figure 11. a) Introductory menu, b) Data input, c) Dialogue with the MEMBER routine, d) Output FSA.

1. **productivity** in building enterprise simulation models to simulate hypothetical organisational models, which is *of the greatest importance in re-engineering processes*;
2. **correctness** of the simulation model synthesis process and its *controllability*;
3. **independence** of a high skilled expert's experience, which enables company to use own resources in the best way.

The experiment, we have presented in the paper, is performed over one simple hypothetical production system in order *to strengthen the initial hypothesis of the inductive inference applicability to the simulation model synthesis process automation*. The model generated by the inductive inference algorithm differs from the reference model in few details (cycle AFIESP-TEAFIN-RDECQ-RDCQ-AFIESP, figure 6. and in the arc which closes the main cycle REEMB-EXT) but we think this is not a serious obstacle.

The future research direction identified are: 1) implementing inductive inference algorithm for more powerful representation classes, for example, for the class of Context-Free Languages; 2) development of a tool for a hypothesis generation control strategies, that is, heuristics on hypothesis generation process management, 3) implementing attribute concepts, in order to permit real simulation model automated synthesis, 4) developing post-processors for chosen simulation programming languages.

8 REFERENCES

Angluin D. (1987) Learning Regular Sets from Queries and Counterexamples, *Information and Control*, Vol. 75, pp 87-106.

Angluin D., Smith C.H. (1983) Inductive Inference: Theory and Methods, *ACM Computing Surveys*, Vol. **15**, No. 3, pp 237-269.

E.C.S.L. System - *User's Manual*

Guimarães Rodrigues A. (1988) *Simulation*, University of Minho (on Portuguese).

Miclet L. (1990) Grammatical Inference, in *Syntactic and Structural Pattern Recognition Theory and Applications* (ed. Bunke H., Sanfeliu A), World Scientific Publ. Co.

Natarajan B. K. (1991) *Machine Learning: A Theoretical Approach*, Morgan Kaufmann.

Tempelmeier H., Kuhn H. (1993) *Flexible Manufacturing Systems - Decision Support for Design and Operation*, John Wley & Sons Inc.

Valiant L.G. (1984) A Theory of the Learnable, *Communication of the ACM*, Vol. **27**., Number 11.

9 BIOGRAPHIES

Dr. Goran D. Putnik received his MSc and Ph.D. from Belgrade University, both in domain of Intelligent Manufacturing Systems. Dr. Putnik's current position is assistant professor in the Department for Production and Systems Engineering, University of Minho, Portugal, for subjects CAD/CAPP, Intelligent Manufacturing Systems and Design Theory. His interests are machine learning and manufacturing system design theory and implementations.

João Almeida das Rosas received his Eng. Lic. diploma from the University of Minho, in the domain of industrial electronic engineering. Eng. Rosas' current position is lecturer on Escola Superior de Tecnologia e Gestão. He is also a MSc student at the University of Minho, Course on Computer Integrated Manufacturing. His interests are CIM systems, design theory and process control and automation.