THE ANNALS OF "DUNĂREA DE JOS" UNIVERSITY OF GALAŢI
FASCICLE V, TECHNOLOGIES IN MACHINE BUILDING,
ISSN 1221- 4566, 2017

APPLYING RELATIONAL DATABASE TO SIMULATE THE RECONFIGURABLE MANUFACTURING SYSTEM

Florin TEODOR[1], Vasile MARINESCU[2],
Cătălina MAIER[2], Viorel PĂUNOIU[2]

[1] Faculty of Naval Arhitecture, "Dunărea de Jos" University of Galaţi
[2] "Dunărea de Jos" University of Galaţi, Manufacturing Engineering Department
theo.florin@gmail.com

ABSTRACT

The operation of reconfigurable manufacturing systems (RMS) involves frequent changes in the structure and properties of different sales items, which implies the need to store and use a large amount of data. In this paper, we present an original solution to manage this problem. Storing the data of a RPD3D Petri network that models a RMS (and concerning its components, the relations between them, commands concerning different products and related operations, the information about RMS resources) can be done through a MySQL database. The transfer of the mentioned data and of the schedules concerning the expert numerical system entitled SODRMS (System for Optimal Driving the RMS) and having the architecture of a web portal implemented in Java with Java3Dgraphical elements, can be also realized by using the same MySQL database. SODRMS addresses to managers, which have to evaluate the received orders in terms of performance and to control the entire manufacturing process, from client order up to products delivery. The data regarding each order (product) can be recorded from the database in a SQL-type file and reloaded when needed (e.g. if intending to replay the simulation of fulfilling that order, with modified input parameters). Another application for such a database refers to the use of the compacting algorithm applicable to RPD3D Petri networks. This is necessary because they deal with high amounts of data and it enables to reach a compact visualization of these networks having large dimensions, by giving a new perspective on the simulation of a RMS functioning. The use of MySQL database for optimal controlling RMS-s has also the advantage of enabling the diversification of characteristics concerning the constitutive elements of the RPD3D Petri networks, by specific means.

KEYWORDS: reconfigurable manufacturing systems, resources allocation, relational database, Petri network

1. INTRODUCTION

The operation of reconfigurable manufacturing systems (RMS) implies frequent changes in the structure and properties of its various elements, which implies the need to store and use a large amount of data.

A comparison between relational databases and data warehouses is able to provide a coherent picture of their role in supporting manufacturing activities as well as relationships with other types of information systems. Both databases and data warehouses contain large amounts of structured data that can be quickly accessed through optimized access structures and is based, in most cases, on relational technologies. However, they have not been designed from the same objectives and are distinguished by many aspects.

Database management systems are appropriate to current management applications and serve to create and maintain operational database systems. These systems, known as OLTP (On-Line Transaction Processing) systems, have as objective the on-line execution of transactions and query processes. They incorporate all day-to-day operations in an organization such as: supplies, stocks, production, settlements, payments, accounting. Data warehouse systems, on the other hand, serve users or data analysis and decision-makers. Operational systems databases contain current, detailed data that need to be updated and quickly queried under full security, providing support for Transaction Processing Information Systems (TPS).

Data warehouses are specifically designed to support decision-making. They aim at aggregating data, aggregating and synthesizing them, organizing and coordinating information from different sources, integrating and storing them to give decision makers the right image to allow effective information retrieval and analysis. Common queries in a data repository are more complex and varied than those in database management systems. They apply to very large volumes of data and involve complex calculations (trend analysis, averages, dispersions, etc.) that often require aggregations (group by clauses).

An operational database is designed and adapted from known tasks and activities such as indexing, using primary keys, searching for specific records, optimizing queries.

The separation between operational BDs and data repositories is based on structure, content, users and different data. Decision making requires historical data, whereas operational databases do not usually contain historical data. In this context, operational data, though abundant, are usually far from complete for decision-making. Assisting the decision requires data to be consolidated (aggregations and aggregations) from different sources, resulting in high-quality, clean and integrated data.

Concluding, for storing and managing RMS data (data about its elements, data links, data about RMS commands - with related products and operations, data about RMS resources - equipment that competes in the production of ordered products , data related to the RMS time evolution - simulation of its manufacturing), including reconfiguration data, we chose to use a relational database, one of the simplest and best known, which has proven over time its stability and reliability, ie a MySQL database.

The transfer of the above data as well as the work programs for the expert computer system, called S.O.D.R.M.S. (System to Optimal Driving for RMS), with web portal architecture implemented in Java language with Java3D graphics elements for managers to evaluate incoming orders in terms of performance and control the entire production process from customer demand to when delivering products, it is also done through the same MySQL database (writing, reading, updating, saving and restoring data); The data for each command (product) can be saved from the database in an SQL file and reloaded into the database, then the resumption of the production simulation of that command (with modified data) is desired.

Another use of the database is given by the use of the RPD3D Petri Network Compaction algorithm, which generates a large amount of data and which is useful for compact viewing of these large-scale networks, and to provide novel insights into simulation of the operation of an RMS.

The MySQL server is offered free of charge for Unix, Windows and MacOS under the GNU General Public License (GPL) by the Swedish company MySQL AB, the most popular open-source SGBD at present, the open-source character being important through prism further developments.

The use of a MySQL database in optimal RMS management also allows for the diversification of features associated with the three-dimensional Petri net type RPD3D that are used to model the operation of reconfigurable manufacturing systems by:

- Entering the scale for RPD3D elements - a larger dimension of the element suggesting a greater importance of that element within the Petri network;
- Insertion of the visibility feature of the constituents of RPD3D, used for the level zoom function;
- Draw the links between the network elements according to the value of the weight attribute - the necessary attribute for the network reconfiguration (the weight 3 is allocated to the links between the elements of the RPD3D constructing a module, the weight of the 2 links between the modules located on the same hierarchical level and the weight 1 of the links between the modules on different hierarchical levels) - the higher the weight, the longer the binding becomes worse.
- Inserting the TimeToBuild attribute (TTB) - the length of materialization or the unlinking of the link in the reconfiguration process - directly proportional to the weight of the link.
- Inserting the TimeToFinish attribute (TTF) - contained in the PRD table of the database, its value gives the manufacturing time in seconds of a copy of the product;
- Inserting the TimeToExclude attribute (TTE - time to exclude a resource from RMS)
- Inserting the TimeToIntegrate attribute (TTE - time to include a resource in RMS);
- Inserting the TimeStamp attribute (in each database table) to mark the time of creation of that item. The value is useful for reconfiguration,

comparing the current time with TSTP to find out how old that element is. In the OPT table, you can set the TSTP_mode variable so that the oldest / new element / module is removed from the system when reconfiguring the RMS.

2. DATABASE TABLES

The first thing to do is create the MySQL database using the Create Database command, a database that will store and manage all data related to the definition and operation of the assisted reconfigurable manufacturing system called rmsdb.

Once the database is created, it follows the actual creation of component tables by running an initialization file with the .SQL extension, which will contain commands such as Create table table_name.

The rmsdb database tables are shown in Figure 1, their grouping (indicated by the common color) being based on the data contained and their role in the RMS operation. The entire manufacturing process can be modulated by associating generic RPD3Ds to each RMS resource and to each technological process flow that produces a product on that RMS, these generic RPD3Ds differ only through work parameters and status. They can be predefined, can be edited and saved in the form of modules, the values of the working parameters that individualize the products, respectively the technological process of their manufacture, being saved in the database.

The first set of tables (the red ones in Figure 1) are required for building and storing RPD3D modules, editable modules that will be used to simulate resource operation or technological operations.

The POS, TRZ, LEG, and CON tables are used to create a RPD3D module, and its data is subsequently saved in two other tables: MODULES (where we find all the created modules and the moment of creation) and MOD_xxx (constructive of the module, concatenated from the four original tables).

According to the MySQL-based database organization, the data is stored in tables, each table being associated with a matrix that has a finite number of descriptors (matrix columns) and an unspecified number of records (matrix rows).

The second group of tables (the orange ones in Figure 1) - CMD, PRD, OT, FT and OT-RES, are required for storing data related to the manufacturing process - commands, products, operations and technological flows, and their interdependence with RMS resources.

The third group of tables (the blue ones in Figure 1) are required for storing the data related to the resources of the reconfigurable manufacturing system (RES, AGV), Order Ordinance (POR), and keeping data on the evolution over time and influencing this evolution of manufacturing made on RMS (OPT and PRODUCTION or FAB).

Another table related to the RMS reconfiguration evolution is the LOG table.

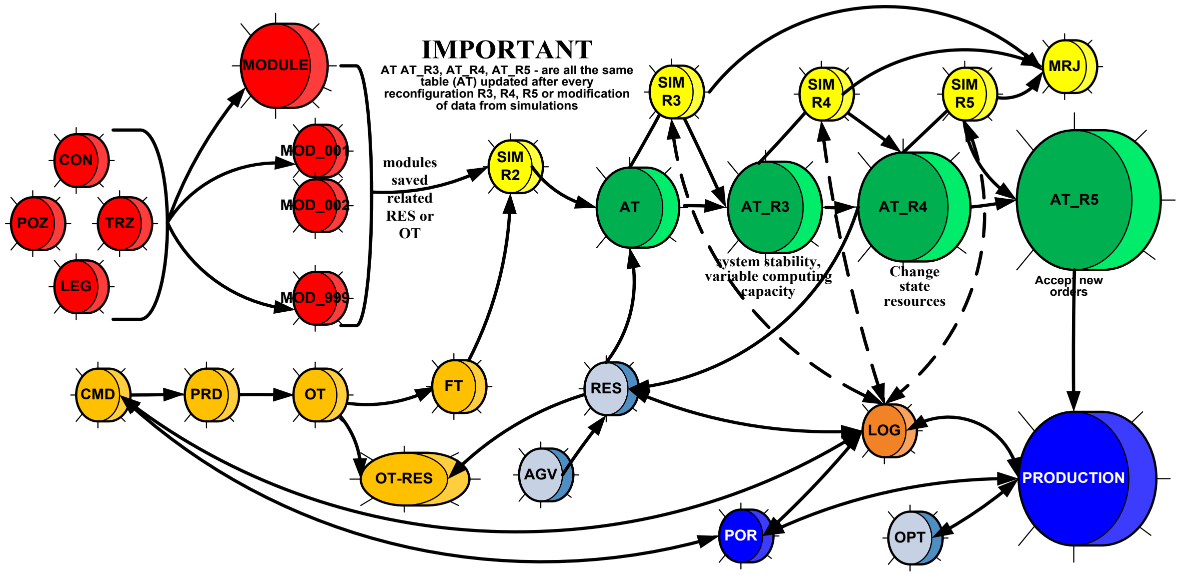

IMPORTANT
AT AT_R3, AT_R4, AT_R5 - are all the same table (AT) updated after every reconfiguration R3, R4, R5 or modification of data from simulations
MODULE
MOD_001
MOD_002
MOD_999
CON
POZ
TRZ
LEG
modules saved related RES or OT
SIM R2
SIM R3
SIM R4
SIM R5
MRJ
AT
AT_R3
AT_R4
AT_R5
system stability, variable computing capacity
Change state resources
Accept new orders
CMD
PRD
OT
FT
OT-RES
AGV
RES
LOG
POR
OPT
PRODUCTION

Fig. 1: Relationships between database tables associated with RMS

The last set of tables (the green and yellow ones in Figure 1) are required to store the data for the reconfiguration process to allow both the return of the manufacturing system to a previous state and simulation of the system evolution to a later state if the simulation results are favorable to evolution towards that state.

The AT, AT_R3, AT_R4, and AT_R5 tables contain the evolution data of the same production workshop (AT) reconfigured in one of the reconfiguration options R2, R3, R4 or R5.

The SIM R2 R2, SIM R3, SIM R4 and SIM R5 tables contain factory production simulation data (AT) reconfigured in one of the reconfiguration options R2, R3, R4 or R5.

The MRJ table contains the evolution data for the three-dimensional Petri network that simulates the operation of the reconfigurable manufacturing system.

Next, the tables of the first group will be presented, specifying the descriptors and the usefulness of storing those data.

Like any Petri network, the RPD3D modules are made up of two main types of elements: positions and transitions linked together by segments oriented by different weight (thickness).

A position can model operations (processing, transport, manipulation, assembly, etc.), fixed resources (processing centers, AGVs, robots), variable resources (pallets, parts, buffers), or various operating conditions are called O-positions).

POZ
01. **ID_POZ** (int 10) / notnull / autoincr / primary key
02. **DEN_POZ** (vchar 10) / notnull / def ' '
03. **DESCR_POZ** (vchar 40) / def ' '
04. **MARCAJ** (int 10) / notnull / def 0
05. **X_POZ** (double) / notnull / def 0.0
06. **Y_POZ** (double) / notnull / def 0.0
07. **Z_POZ** (double) / notnull / def 0.0
08. **SCALE_POZ** (double) / notnull / def 1
09. **TIMP_EXEC_POZ** (int 10) / notnull / def 0
10. **VIZ_POZ** (int 1) / notnull
11. **TSTP_POZ** (Bigint 20) / notnull

POZ table - stores data of position elements from a RPD3D module.

1 - position ID;

2 - position name (position identifier);

The positions of the resources the number of which is fixed in the design planning, such as robots, machines and conveyors are called R-positions, and positions corresponding to variable resources (blades, pieces or tasks for be processed) with role in resource sharing are called V-positions. The marking of variable resources must be determined so that the system can not become blocked or run empty (sub-capacities).

There are also intermediate positions called I-positions that can model operations involving a variable number of resources, such as buffering or warehouse operations, or to facilitate the maintenance of some behavioral properties of modeling systems. At the same time, in order to allow the ordering of the activities of the different subsystems, the positions that shape the control information required for the ordination, called C-positions, are maintained.

3 - position description (max 40 characters);

4 - the initial position mark;

5, 6, 7 - the coordinate X, Y, Z of the position center (real number);

8 - position scale (real number) - the size at which the position is represented (0.5X = 0.5, 1X = 1.0, 2X = 2.0 and 3X = 3.0), a larger element size suggesting greater importance of that element within the Petri network (VOC type V which models the availability of a robot - RI Ready - have a 2X scale, and the IO type that models intermediate states of the system are 0.5X scale);

9 - the maximum allowable time for the modeled mode operation (τ_{max}) - in seconds.

10 - Position visibility - 1 (ON) and 0 (OFF) - used for level zooming;

11 - Time stamp (Position Time Stamp) (integer no more than 19 digits) - time of insertion of the position in the form of a 10-digit numeric sequence;

When inserting, the TimeStamp field value is calculated automatically and can be used to reconfigure RMS (compare current time to the value of TSTP and find out which link or 3D object is more recent or older.

TRZ
01. **ID_TRZ** (int 10) / notnull / autoincr / primary key
02. **DEN_TRZ** (vchar 10) / notnull / def ' '
03. **DESCR_TRZ** (vchar 40) / def ' '
04. **X_TRZ** (double) / notnull / def 0.0
05. **Y_TRZ** (double) / notnull / def 0.0
06. **Z_TRZ** (double) / notnull / def 0.0
07. **ERR_TRZ** (int 5) / notnull / def 0
08. **SCALE_TRZ** (double) / notnull / def 1
09. **VIZ_TRZ** (int 1) / notnull
10. **TSTP_TRZ** (Bigint 20) / notnull

Ex: 1256953732 = Sat Jul 23 02: 16:57 2005

TRZ table - stores data of transition type elements from a RPD3D module.

1 - Transition ID (integer no. Max. 9 digits);

2 - transition name (transition identifier);

The transitions that represent the beginning or end of an operation are called O-transitions (TOnnn), and transitions that trigger sub-networks for error handling are called E-transitions (TEnnn).

3 - Transition description (max 40 characters;

4, 5, 6 - the X, Y, Z coordinate of the transition center (real number);

7 - initial transition error code (full integer with max 4 digits);

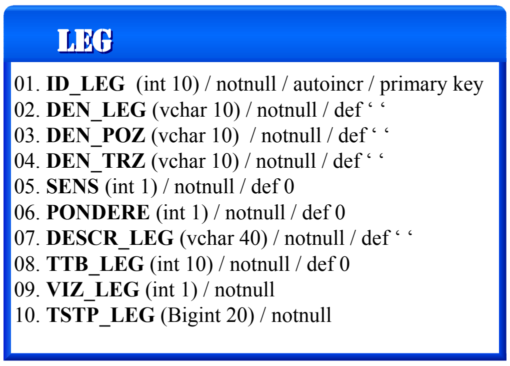
Thus, a maximum of 105 error codes, sufficient for any RPD3D module, can be defined, each error code having a description and an error-solving mode, even though for most of these error codes, solving is the passing of the resource into the state D (Damaged) and its exclusion from the RMS.

8 - the transition scale (real number) - the size at which the transition is represented (0,5X,1X,2X,3X);

9 - Transition visibility - 1 (ON) and 0 (OFF) - used for level zooming;

10 - Time stamp - Transition Time Stamp - transition time;

The links between the elements (arcs) are oriented (meaningful) and carry information about the reconfiguration process of the manufacturing system.



LEG
01. **ID_LEG** (int 10) / notnull / autoincr / primary key
02. **DEN_LEG** (vchar 10) / notnull / def ‘ ‘
03. **DEN_POZ** (vchar 10) / notnull / def ‘ ‘
04. **DEN_TRZ** (vchar 10) / notnull / def ‘ ‘
05. **SENS** (int 1) / notnull / def 0
06. **PONDERE** (int 1) / notnull / def 0
07. **DESCR_LEG** (vchar 40) / notnull / def ‘ ‘
08. **TTB_LEG** (int 10) / notnull / def 0
09. **VIZ_LEG** (int 1) / notnull
10. **TSTP_LEG** (Bigint 20) / notnull

LEG table - stores the data of link elements in a RPD3D module, which can be associated with a production workshop resource or a technological operation from a product manufacturing stream.

1 - link ID;

2 - link name (link identifier);

3 - position name (position identifier);

4 - transition name (transition identifier);

5 - the meaning of the link - (1 direct, -1 inverse);

6 - the weight of the link (probability of disappearance of that link to a possible reconfiguration);

The value 3 is assigned to the arcs between the RPD3D constituent elements of the RPD3D modules, the weight 2 is given to the arcs between the RPD3D modules located on the same level and the weight 1 is allocated to the arcs between the RPD3D modules located on different hierarchical levels. Thus, the thickness of the link indicates the strength of the link, a weaker link indicating a greater probability of disappearance of that link to a possible reconfiguration of the system, the strongest being 3 and the weakest 1.

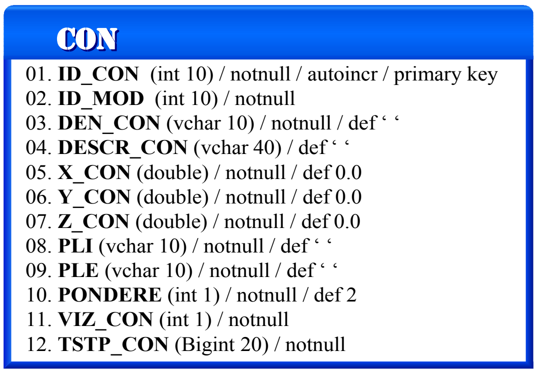
7 - Description of the link - especially for links with weights 1 and 2;

8 - TTB_LEG - TimeToBuild_LEG - the information, in seconds, which represents the duration of materialization or unlinking in the reconfiguration process, directly proportional to the weight of the link.

9 - Link visibility - 1 (ON) and 0 (OFF);

10 - Time stamp - Time stamp of the link - holds the moment of inserting the link in the form of a 10-digit numeric sequence, and is useful for comparing the links to each other according to their age.

In the RPD3D model, to easily make connections between different RMS modules placed on the same level or at different levels, a third type of element, called a connector, was adopted, represented by a cube with the side equal to the radius of the sphere represents a position (the color is red if the value is "OFF" or green if the value is "ON"), the connectors being arranged on a sphere circumscribing RPD3D in compact form (obtained after the compaction operation).
The base connectors 6 are placed at the points where the coordinate axes intersect the sphere comprising compact RPD3D and are called M1 (y +), M2 (x +), M3 (z +), M4 (x-), M5 z-), M6 (γ-). If these 6 connectors are not sufficient for the module links, then there are 12 additional connectors, located



CON
01. **ID_CON** (int 10) / notnull / autoincr / primary key
02. **ID_MOD** (int 10) / notnull
03. **DEN_CON** (vchar 10) / notnull / def ‘ ‘
04. **DESCR_CON** (vchar 40) / def ‘ ‘
05. **X_CON** (double) / notnull / def 0.0
06. **Y_CON** (double) / notnull / def 0.0
07. **Z_CON** (double) / notnull / def 0.0
08. **PLI** (vchar 10) / notnull / def ‘ ‘
09. **PLE** (vchar 10) / notnull / def ‘ ‘
10. **PONDERE** (int 1) / notnull / def 2
11. **VIZ_CON** (int 1) / notnull
12. **TSTP_CON** (Bigint 20) / notnull

between the 6 base units, called M12, M13, M14, M15, M23, M25, M26, M34, M36, M45, M46 , M56. Outside the module, the six connectors are connected by weight arrays 1 or 2 by other connectors, and inside the module they are connected by weight arrays 3 to control positions to which they transmit the various orders given from the higher hierarchical levels: start or end of an OT, transfer of the number of completed finished pieces, change of the state of a RES).

The connectors do not influence the RPD3D mode of operation or analysis to which they are associated but only have the role of transmitting to and from the RMS higher levels the commands that will result from the reconfiguration process or the RMS operation.

The **CON table** - stores the data of the connector type elements in a RPD3D module, the way that can be associated with a production workshop resource or a technological operation from a product manufacturing stream.

1 - Connector ID;

2 - RPD3D module ID;

3 - connector name; Ex: M1-MOD23 (M1 connector associated with module 23 of RMS);

4 - description of the connector;

5, 6, 7 - The fields X_CON, Y_CON and Z_CON are the relative coordinates of the connector to the circumference of the sphere;

8 - PLI - Internal link position (inside the module) - Ex: "C2"

9 - PLE - External Link Position (Module Connector) - Ex: "M1_MOD_23"

10 - Weight - value 1 for links between RMS modules on different levels and value 2 for inter-module linkages;

11 - Connector visibility - 1 (ON) and 0 (OFF);

12 - Time stamp - the time of insertion of the connector in the form of a 10-digit numeric sequence;

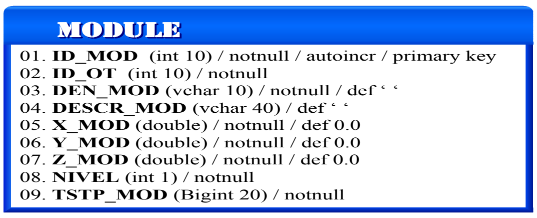
Identifying the activities and resources needed to process a product begins by defining a product processing order on the RMS.

After creating a RPD3D module, its data is saved in two tables: MODULE and MOD_xxx.

Upon editing a module, the data is saved in the database, and then the editable graphics module is passed into a compact graphical form, this being an operation by which the Petri elements are concentrated inside a sphere of the smallest diameter.

After compaction, the circumscribed sphere of the module is generated, the bookmarks are generated and the data is re-stored in the database, in a SQL table called MOD_nnn, where nnn is a self-enhanced number.

The **MODULE table** - where we find all the modules created, the technological operation they describe and the moment of creating each module. Also, in the MODULE table, there is a level field that allows a multi-level hierarchy of the RMS.



MODULE

01. **ID_MOD** (int 10) / notnull / autoincr / primary key
02. **ID_OT** (int 10) / notnull
03. **DEN_MOD** (vchar 10) / notnull / def ‘ ‘
04. **DESCR_MOD** (vchar 40) / def ‘ ‘
05. **X_MOD** (double) / notnull / def 0.0
06. **Y_MOD** (double) / notnull / def 0.0
07. **Z_MOD** (double) / notnull / def 0.0
08. **NIVEL** (int 1) / notnull
09. **TSTP_MOD** (Bigint 20) / notnull

1 - module ID;

2 - The ID of the technological operation it describes (eg: O29);

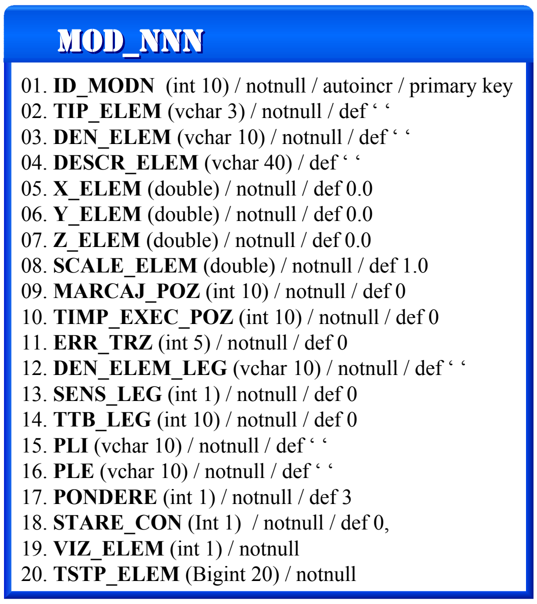
3 - module name;

4 - description of the module;

5, 6, 7 - the coordinates of the sphere center that circumscribe all the elements of the created/edited module;

8 - the level on which the RPD3D network module describes the RMS operation;

9 - Stores in the database when the module was created or terminated;



MOD_NNN

01. **ID_MODN** (int 10) / notnull / autoincr / primary key
02. **TIP_ELEM** (vchar 3) / notnull / def ‘ ‘
03. **DEN_ELEM** (vchar 10) / notnull / def ‘ ‘
04. **DESCR_ELEM** (vchar 40) / def ‘ ‘
05. **X_ELEM** (double) / notnull / def 0.0
06. **Y_ELEM** (double) / notnull / def 0.0
07. **Z_ELEM** (double) / notnull / def 0.0
08. **SCALE_ELEM** (double) / notnull / def 1.0
09. **MARCAJ_POZ** (int 10) / notnull / def 0
10. **TIMP_EXEC_POZ** (int 10) / notnull / def 0
11. **ERR_TRZ** (int 5) / notnull / def 0
12. **DEN_ELEM_LEG** (vchar 10) / notnull / def ‘ ‘
13. **SENS_LEG** (int 1) / notnull / def 0
14. **TTB_LEG** (int 10) / notnull / def 0
15. **PLI** (vchar 10) / notnull / def ‘ ‘
16. **PLE** (vchar 10) / notnull / def ‘ ‘
17. **PONDERE** (int 1) / notnull / def 3
18. **STARE_CON** (Int 1) / notnull / def 0,
19. **VIZ_ELEM** (int 1) / notnull
20. **TSTP_ELEM** (Bigint 20) / notnull

The **MOD_xxx table** - which contains the data of all construction elements of module no. xxx, concatenated from the four initial tables - POZ, TRZ, LEG, and CON.

1 - module ID;

2 - type of Petri network element (eg: POZ);

3 - name of the element;

4 - description of the element;

5, 6, 7 - the relative coordinates of the center of the element towards the center of the sphere circumscribing all the elements of the module;

8 - the graphic size of the element (1X - normal, 0.5X, 2X or 3X), resulting from the importance of the element in the module;

9, 10 - are data taken from the POS table;

11 - data are taken from the TRZ table;

12, 13, 14 - data taken from the LEG table;

15,16,17,18 - data taken from the CON table;

19 - Stores in the database when the module is created or terminated, data identical to those in field 9 of the MODULE table;

The processing command comes from the outside of the system that we simulate, and it contains the initial production specifications (RMS initial data).

Obviously, the order data is stored and processed in the same MySQL database that serves the entire reconfigurable manufacturing system.

CMD
01. **ID_CMD** (int 10) / notnull / autoincr / primary key
02. **ID_PRD** (int 10) / notnull
03. **DEN_CMD** (vchar 6) / notnull / def ‘ ‘
04. **DESCR_CMD** (vchar 40) / def ‘ ‘
05. **NR_EX** (int 10) / notnull
06. **TTF** (int 10) / notnull
07. **DD** (Bigint 20) / notnull
08. **EP** (double) / notnull
09. **STARE_CMD** (vchar 1) / notnull
10. **TSTP_CMD** (Bigint 20) / notnull

CMD table - stores the data of the processing orders and their statuses.

1 - order ID;

2 - product ID - Specifies the type of product (family of products or individual product);

3 - command name - type CMDnnn;

4 - description of the order;

5 - the number of copies to be made from the specified product;

6 - TTF - TimeToFinish - Total manufacturing time of all copies in the product;

7 - DD (Due Date) - calculated in a time-like manner;

8 - EP (Earning Power) - The actual value is obtained from a specific calculation algorithm and is used to decide on the subsequent order of the order - if it is the most profitable, the order is launched in the manufacture, if less profitable, the order can be stopped in the Order POR portfolio to be launched later or the order can be rejected being considered unprofitable;

9 - Status of the command that can have the values: L - Launched, P - Paused, R - Resumed, F - Finish, J - Rejected, D - Deferred, N - New;

10 - Time stamp - command TimeStamp - holds the moment when the command is released - when the command is in the L state (command launched).

After creating the RPD3D modules, the initial workshop is created based on available resources.

The RES table - one of the largest tables of the MySQL database, stores the data for different resources (machine tools, robots, buffers, parts or tools warehouses, AGVs, etc.) from which the virtual workshop will be assembled manufactures the product / products ordered.

1 - resource ID;

RES
01. **ID_RES** (int 10) / notnull / autoincr / primary key RES001
02. **DEN_RES** (vchar 10) / notnull / def ‘ ‘
03. **DESCR_RES** (vchar 40) / def ‘ ‘
04. **TIP_RES** (vchar 5) / notnull / def ‘ ‘
05. **SCALE_RES** (double) / notnull / def 1.0
06. **X_RES** (double) / notnull / def 0.0
07. **Y_RES** (double) / notnull / def 0.0
08. **Z_RES** (double) / notnull / def 0.0
09. **STARE_RES** (vchar 1) / notnull / def ‘ ‘
10. **TTST** (int 4) / notnull / def 0
11. **TTSP** (int 4) / notnull / def 0
12. **TTDR** (int 4) / notnull / def 0
13. **TTE** (int 4) / notnull / def 0
14. **TTI** (int 4) / notnull / def 0
15. **TTLT** (int 4) / notnull / def 0
16. **TTLP** (int 4) / notnull / def 0
17. **TTM** (int 4) / notnull / def 0
18. **TTDP** (int 4) / notnull / def 0
19. **TTDT** (int 4) / notnull / def 0
20. **TTA** (int 4) / notnull / def 0
21. **TTD** (int 4) / notnull / def 0
22. **TTRD** (int 4) / notnull / def 0
23. **TTRM** (int 4) / notnull / def 0
24. **CAR1** (vchar 50) / notnull / def ‘ ‘
25. **VAL1** (vchar 20) / notnull / def ‘ ‘
26. **CAR2** (vchar 50) / notnull / def ‘ ‘
27. **VAL2** (vchar 20) / notnull / def ‘ ‘
28. **CAR3** (vchar 50) / notnull / def ‘ ‘
29. **VAL3** (vchar 20) / notnull / def ‘ ‘
30. **VIZ_RES** (int 1) / notnull
31. **TSTP_RES** (Bigint 20) / notnull

2 - resource name;

3 - description of the resource;

4 - resource type – Ex: RMTFU;

5 - resource scale - 0.5, 1.0, 2.0, 3.0;

6, 7, 8 - X_RES, Y_RES and Z_RES are the absolute coordinates of the resource;

9 - resource state - R (Ready), W (Work), D (Damaged), P (Pause), I (IntoRMS) and E (Error correction);

10 - TTST (TimeToStart) resource start time;

11 - TTSP (TimeToStop) resource stop time;

12 - TTDR (TimeToDetect and Repair) time to detect and repair the resource

13 - TTE (TimeToExclude) resource exclusion time in RMS;

14 - TTI (TimeToIntegrate) resource integration time in RMS;

15 - TTLT (TimeToLoadTool) tool load time per resource;

16 - TTLP (TimeToLoadPiece) load time of the piece on the resource;

17 - TTM (TimeToManufacture) the manufacturing time of the technological operation on the resource;

18 - TTDP (TimeToDownloadPiece) the download time of the piece on the resource;

19 - TTDT (TimeToDownloadTool) Download time of the resource from the resource;

20 - TTA (TimeToAproach) approach time of RI of resource;

21 - TTD (TimeToDistance) distance of RI of the resource;

22 - TTRD (TimeToRotateDep) rotation time with a part store of the parts;

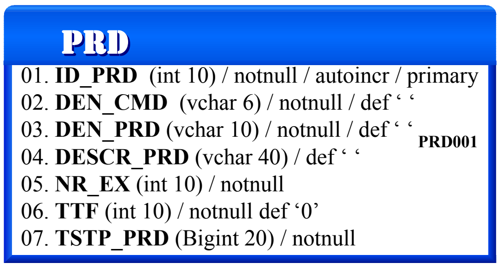
23 - TTRM (TimeToRotateMove) (time of rotation or movement of the piece between two successive machining operations with the same tool) OT1SPn; **10 ÷ 23** - are expressed in seconds;

24, 26, 28 - CAR1, CAR2 and CAR3 - are the main features that define the resource.

25, 27, 29 - VAL1, VAL2 and VAL3 - are the values of the main features that define the resource.

30 - Visibility of the resource;

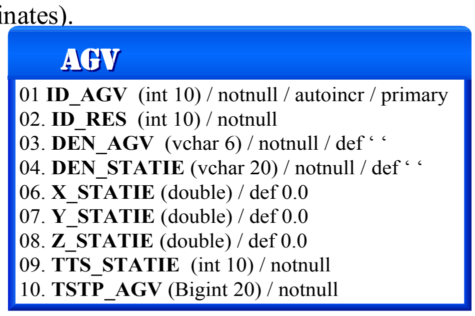
31 - Time stamp (Resource Time Stamp) - the moment of RES insertion;


PRD

01. **ID_PRD** (int 10) / notnull / autoincr / primary
02. **DEN_CMD** (vchar 6) / notnull / def ‘ ‘
03. **DEN_PRD** (vchar 10) / notnull / def ‘ ‘ **PRD001**
04. **DESCR_PRD** (vchar 40) / def ‘ ‘
05. **NR_EX** (int 10) / notnull
06. **TTF** (int 10) / notnull def ‘0’
07. **TSTP_PRD** (Bigint 20) / notnull

Initially, all resources are entered in the RES table, even if they are not available at the time of tab insertion, then the ones available (which have the R state) and depending on the technological operation to be performed and the correspondence between the CAR1, 2.3 of the resource and of the technological operation is made up the initial workshop.

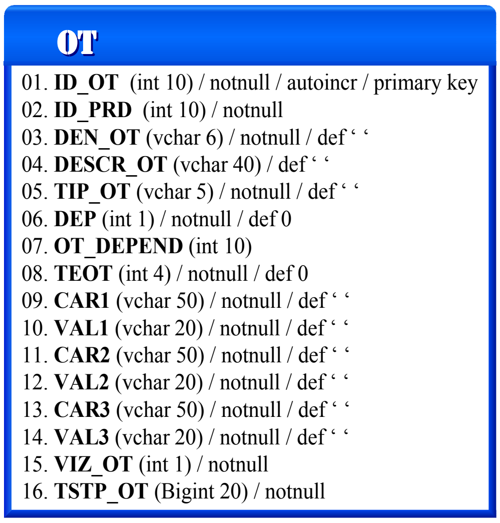
A more special resource, requiring a separate table in the database, due to more complex information to be stored in the database, is the autonomous guided vehicle (AGV) - a table containing the data on a palletized transport system based on self-governed vehicles that have a specific route based on stations (places with clearly specified coordinates where vehicles stop and transfer the contents of the blades to the resources at those coordinates).


AGV

01 **ID_AGV** (int 10) / notnull / autoincr / primary
02. **ID_RES** (int 10) / notnull
03. **DEN_AGV** (vchar 6) / notnull / def ‘ ‘
04. **DEN_STATIE** (vchar 20) / notnull / def ‘ ‘
06. **X_STATIE** (double) / def 0.0
07. **Y_STATIE** (double) / def 0.0
08. **Z_STATIE** (double) / def 0.0
09. **TTS_STATIE** (int 10) / notnull
10. **TSTP_AGV** (Bigint 20) / notnull

Stations are named so as to individualize the stopping place (Base, Station 1, Robot 1, Machine 2). Also, in the AGV table are given IDs of the serviced resources and the station time in each station.

As a rule, AGVs run on predetermined routes, but in the case of RMS, which involves a multitude of changes in the structure and location of resources, perhaps the best option is to move the AGV on the shortest route to the next station, avoiding obstacles encountered in the path.


OT

01. **ID_OT** (int 10) / notnull / autoincr / primary key
02. **ID_PRD** (int 10) / notnull
03. **DEN_OT** (vchar 6) / notnull / def ‘ ‘
04. **DESCR_OT** (vchar 40) / def ‘ ‘
05. **TIP_OT** (vchar 5) / notnull / def ‘ ‘
06. **DEP** (int 1) / notnull / def 0
07. **OT_DEPEND** (int 10)
08. **TEOT** (int 4) / notnull / def 0
09. **CAR1** (vchar 50) / notnull / def ‘ ‘
10. **VAL1** (vchar 20) / notnull / def ‘ ‘
11. **CAR2** (vchar 50) / notnull / def ‘ ‘
12. **VAL2** (vchar 20) / notnull / def ‘ ‘
13. **CAR3** (vchar 50) / notnull / def ‘ ‘
14. **VAL3** (vchar 20) / notnull / def ‘ ‘
15. **VIZ_OT** (int 1) / notnull
16. **TSTP_OT** (Bigint 20) / notnull

OT Table (Technological Operations) - Stores the data for the different technological operations that the ordered product / products need to manufacture.

1 - OT ID; **2** - product ID;

3 - OT designation; Ex: RES1;

4 - OT description - optional;

5 - OT type; Ex: GAN, DEB etc;

6 - OT dependence on other previous OTs (1-DA, 2-NU);

7 - The ID of the OT to which he is dependent;

8 - OT execution time;

9, 11, 13 - CAR1, CAR2 and CAR3 - are the main features that define OT.

10, 12, 14 - VAL1, VAL2 and VAL3 - are the values of the main defining characteristics. OT.

15 - Visibility of OT - 1 (ON) and 0 (OFF);

16 - Time stamp (OT TimeStamp);

An automated synchronization can be performed by the database using the data in the CAR1, 2 and 3, VAL1, 2 and 3 fields of the OT and RES tables as follows: o OT with certain features can only be performed on resources of a particular type whose technological capacity is sufficient.

The sync factor (FACT_SINC) can be of the following type:

➢ - E - Exclusive type synchronization (the selected RES fits perfectly with dedicated OT - RES),

- - U - Universal type synchronization (the selected RES is the universal type and it can also perform the respective OT),
- - L - Synchronization at the technological limit of RES;

The synchronization mode (MOD_SINC) can be: A - Automatic, S - Semiautomatic (program selection with user confirmation) or M - Manual. If we have for, OT - CAR1 = vit. rotation, VAL1 = 1500 rpm, and we have 3 resources available:

RES1 - CAR1 = vit. rotation,
VAL1 = 2000-3000 rpm,
RES2 - CAR1 = vit. rotation,
VAL1 = 800-1500 rpm,
RES3 - CAR1 = vit. rotation,
VAL1 = 1000-2500 rpm

It is obvious that we can not choose RES1, RES2 has VAL1 at the technological limit (it is a risky choice) so the best choice is RES3.

OT_RES
01. **ID_OT_RES** (int 10) / notnull / autoincr / primary key
02. **ID_OT** (int 10) / notnull
03. **ID_RES** (int 10) / Notnull
04. **MOD_SINC** (vchar 1) / notnull
05. **FACT_SINC** (vchar 1) / notnull
06. **TSTP_OT_RES** (Bigint 20) / notnull

The results of automatic synchronization are deposited in the OT_RES table of the database.

1 - OT_RES ID; **2** - OT ID; **3** - RES ID;

4 - Mod_sinc-OT synchronization mode with RES;

5 - Synchronization factor:

6 - TimeTamp OT_RES;

FT
01. **ID_FT** (int 10) / notnull / autoincr / primary key
02. **DEN_FT** (vchar 6) / notnull / def ‘ ‘
03. **TTE_FT** (int 10) / def 0
04. **NOT_FT** (int 10) / def 0
05. **TSTP_FT** (Bigint 20) / notnull

By analyzing technological operations and their interdependencies, technological flows (FT) are created.

MRJ
01. **ID_MRJ** (int 10) / notnull / autoincr / primary key
02. **DEN_MRJ** (vchar 6) / notnull / def ‘ ‘
03. **VAL_MRJ** (vchar n) / def ‘ ‘ , n – nr. de POZ (999)
04. **TSTP_MRJ** (Bigint 20) / notnull

1 - Technology Flow Id;

2 - Name of the technological flow;

3 - TTE_FT (9-digit integer) **-** TimeToExecute - Total execution time of the technological operation included in the technological flow;

4 - NOT_FT- NumberOfOT - the total number of technological operations included in the technological flow;

SIM_T
01. **ID_SIM_T** (int 10) / notnull / autoincr / primary key
02. **MRJ_I** (vchar 10) / notnull / def ‘ ‘
03. **TRZE** (vchar 10) / def ‘ ‘
04. **TEX** (Int 10) / def ‘0‘
05. **MRJ_F** (vchar 10) / def ‘ ‘
06. **TSTP_SIM_T** (Bigint 20) / notnull

Creating the streams is done automatically by combining different machine tool set variants, parts of the tool and tool transport system, parts of the tool and tool storage system, industrial robots, etc., resulting in a large number of FT possible.

AT	
01. **ID_AT** (int 10) / notnull / autoincr / primary key	
02. **ID_OT** (int 10) / notnull	
03. **DEN_MOD** (vchar 10) / notnull / def ‘ ‘	**MOD_023**
04. **DESCR_MOD** (vchar 40) / def ‘ ‘	
05. **X_MOD** (double) / notnull / def 0.0	
06. **Y_MOD** (double) / notnull / def 0.0	
07. **Z_MOD** (double) / notnull / def 0.0	
08. **NIVEL** (int 1) / notnull	
09. **TSTP_MOD** (Bigint 20) / notnull	
10. **TIP_ELEM** (vchar 3) / notnull / def ‘ ‘	
11. **DEN_ELEM** (vchar 10) / notnull / def ‘ ‘	
12. **DESCR_ELEM** (vchar 40) / def ‘ ‘	
13. **X_ELEM** (double) / notnull / def 0.0	
14. **Y_ELEM** (double) / notnull / def 0.0	
15. **Z_ELEM** (double) / notnull / def 0.0	
16. **SCALE_ELEM** (double) / notnull / def 10	
17. **MARCAJ_POZ** (int 10) / notnull / def 0	
18. **TIMP_EXEC_POZ** (int 10) / notnull / def 0	
19. **ERR_TRZ** (int 5) / notnull / def 0	
20. **DEN_ELEM_LEG** (vchar 10) / notnull / def ‘ ‘	
21. **SENS_LEG** (int 1) / notnull / def 0	
22. **TTB_LEG** (int 10) / notnull / def 0	
23. **PLI** (vchar 10) / notnull / def ‘ ‘	
24. **PLE** (vchar 10) / notnull / def ‘ ‘	
25. **PONDERE** (int 1) / notnull / def 3	
26. **STARE_CON** (Int 1) / notnull / def 0	
27. **VIZ_ELEM** (int 1) / notnull / def 1	
28. **TSTP_ELEM** (Bigint 20) / notnull	

Involved in the simulation process are the POS, TRZ, LEG tables listed above as well as **the MRJ** table (containing the list of RPD3D during simulation) and **SIM_T** (containing the simulation data - what transitions are executed, when and what markup is generated).

The **AT table** contains all assembled production workshop data from the three-dimensional Petri dish modules, and consists of concatenating the data in the MODULE table and all MODnnn tables related to the modules that made up the workshop.

Rules for completing the AT table:

- ➢ The field DEN_MOD is taken from the MOD table;
- ➢ The fields X_MOD, Y_MOD and Z_MOD are the coordinates of the module center and are taken from the MOD table;
- ➢ The Level field is the RMS level on which the DEN_MOD module is;
- ➢ The field TIP_ELEM has the values: POZ, TRZ, LEG, CON;
- ➢ The fields X_ELEM, Y_ELEM and Z_ELEM are the relative coordinates of the element RPD3D to the center of the sphere circumscribed to the module;
- ➢ SCALE fields have values: 1X = 10, 1 / 2X = 5, 2X = 20, 3X = 30;
- ➢ Fields 17-24 are taken from the MOD_nnn tables
- ➢ The SENS_LEG field has a value of 2 for the POS, TRZ and CON elements, 0 for LEG in the direct sense, and 1 for Indirect LEG;
- ➢ The TTB_LEG field has a value of 0 for the elements POZ, TRZ and CON, values between 1-3 seconds for the LEG between the V, C and I type POS with TRZ, values between 4-6 seconds for the LEG between the POS of the R type with TRZ and values between 7-9 seconds for LEG between POS of type O with TRZ;
- ➢ The PLI field is automatically filled with the POS to which the connector is connected, and the PLE is completed when the modules are connected to each other.
- ➢ The PONDER field has a value of 1 or 2 for the CON type element, 0 for POS and TRZ, and 3 for Leg elements.
- ➢ The STARE_CON field has a value of 0 or 1 only for the CON type element, for the other elements its value is 0.

POR (PORTOFOLIU COMENZI)
01. **ID_POR** (int 10) / notnull / autoincr / primary key
02. **NR_PER** (int 10) / notnull def ‘1’
03. **VAL_PER** (int 10) / notnull def ‘1’
04. **DEN_CMD** (vchar 6) / notnull / def ‘ ‘
05. **TSTP_POR** (Bigint 20) / notnull

The table required for ordering orders is the **POR (Command Portfolio) table**.

1 - Table ID;

2 - The dosing period number (fifth);

3 - Value of the dosing period (fifth);

4 - Name of the order (from the CMD table);

5 - The moment of the order update in the portfolio;

FAB (FABRICATIE)
01. **ID_FAB** (int 10) / notnull / autoincr / primary key
02. **PERIODOZ** (int 10) / notnull / def ‘0‘
03. **EP_RMS** (int 10) / def ‘0‘
04. **TSTP_FAB** (Bigint 20) / notnull

The tables required to keep data related to the time evolution of the production system and to influence this manufacturing evolution on the RMS are OPT and PRODUCTION or FAB.

2 - Amount of the dosing period of the simulation;

3 - The total value of the EP reconfigurable manufacturing system;

4 - Moment of updating the data in each record;

The **OPT table** contains the set options that affect the entire RMS:

OPT
01. **ID_OPT** (int 10) / notnull / autoincr / primary key
02. **OPTIUNE** (vchar 20) / notnull / def ‘ ‘
03. **VALOARE** (vchar 20) / def ‘ ‘

2 - Option name; **3** - Value of the option;

Ex: The TSTP_mod option can have values: "oldest" or "newest", the zoom_niv option can have values: 0.053 or 0.2 or 35.1;

Another table related to the RMS reconfiguration evolution is the LOG table.

2 - The type of log action (writing, reading, updating, deleting registration, inserting or removing a resource from RMS);

3 - Action data (SQL sequence, alphanumeric

LOG
01. **ID_LOG** (int 10) / notnull / autoincr / primary key
02. **ACTIUNE** (vchar 20) / notnull / def ‘ ‘
03. **VAL_LOG** (vchar 50) / notnull / def ‘ ‘
04. **TSTP_LOG** (Bigint 20) / notnull

values, simple or complex characters);

4 - Moment of updating the data in each record (timestamp);

After creating the RMS and RPD3D modeling it, by analyzing the OT and their dependencies, creating technological fluxes and creating the initial workshop on the basis of the available RES, it is done using the SIM R2 table, analyzing the possibilities of manufacturing the products (it might (for a certain OT there is no available resource at the time of that OT), simulating the production of the ordered products with the initial data (resources, relationships, technological flows, number of parts, delivery date and specific profit rate " winning power "- EP).

SIM R2

01. **ID_SIM_R2** (int 10) / notnull / autoincr / primary key
02. **MRJ_I** (vchar 10) / notnull / def ' '
03. **TRZE** (vchar 10) / def ' '
04. **TEX** (Int 10) / def '0'
05. **MRJ_F** (vchar 10) / def ' '
06. **TSTP_SIM_R2** (Bigint 20) / notnull

1 - Table ID;
2 - Initial Mark;
3 - Transition executed;
4 - Execution time of the executed transition;
5 - The final mark;
6 - Moment of execution of the transaction;

The SIM R3, SIM R4 and SIM R5 tables are associated with the AT_R3, AT_R4 and AT_R5 tables and have the same structure as the SIM R2 and AT tables, and are supplemented with modified RMS data after reconfigurations of type :

R3 - Reconfiguration of RMS to ensure system stability and viability;

R4 - Reconfiguration of RMS due to change in status of RES;

R5 - RMS reconfiguration due to changing production specifications to get the best value for the EP index;

3. CONCLUSIONS

The use of a database in the optimal management of RMS is essential because it allows both the storage and processing, but also the transfer of a large amount of data between different modules, which can simulate the optimal operation of a reconfigurable manufacturing system.

After running RMS manufacturing simulations of various ordered products, you can aggregate existing data in the database to display results in the form of reports like:

- Functions I, O, C,
- Grace of accessible bookmarks
- Tree and cover graph
- Triggering frequencies for transitions
- Own and maximum operating speed
- Graph of technological reconfigurable flow
- Placement of different reconfigurations in time
- Time evolution of performance indices
- Conclusions drawn from the simulation to optimize actual production
- Synopsis of RMS evolution during simulation

Also, among the advantages of using a database in the optimal management of RMSs can be the possibility of remote multiplication of activities by synchronizing data with other RMSs via Internet networks, thus assembling virtual factories integrates a variety of software products, modeling tools and methodologies to solve a wide range of manufacturing issues. In these virtual factories it will be possible to detail the designs of the systems, to test the various configurations, to briefly scroll through all the features of the system that influence the performance and efficiency of these operations without making important changes to the actual manufacturing system. Virtual manufacturing can help make economic decisions based on collected statistical data. In this way, it is possible to change the way of decision-making, establishing virtual manufacturing alternatives associated with lower costs.

REFERENCES

[1] **Marinescu,V.,** *Research on the flexible management of metalworking processes cold*, PhD Thesis, University "Lower Danube", Galati, 2000;
[2] **Teodor, F.,** *Optimal management of reconfigurable manufacturing systems*, PhD Thesis, University "Lower Danube", Galati, 2016.