Modelling of the multi-agent systems through specific DEDS methods and the Erlang functional language

EUGEN DIACONESCU[1], LUIGE VLADAREANU[2]
[1]Electronics and Computers Faculty, University of Pitesti
[2]Automatics Department, IMSAR Bucharest
ROMANIA
eugen.diaconescu@upit.ro, luige.vladareanu@arexim.ro

Abstract: - The paper contains an evaluation of the modeling multi-agent systems methods in the context of discrete event dynamic systems in technical and industrial settings. Also the ease of implementing solutions of common types of dynamic discrete models in Erlang language is considered and is showing that programming in Erlang / OTP is more suitable for multi-agent systems design compared to other platforms. Arguments for using Erlang for programming of conversational communication systems in multi-agent systems are presented.

Key-Words: - agent, multi-agent system, Petri–nets, finit-state-machine, functional language, Erlang

1 Introduction

Industrial production systems are usually seen as two main abstractions: continuous systems and discrete systems, both of which are most frequently encountered as dynamic versions. Since Multi-Agent Systems (MAS) are generally considered discrete systems, further work will be dealt with only discrete event dynamic systems (DEDS) or DEDs for short.

Considering $D = \{d_1, d_2, ... d_n\}$ the set of components of the system and $\Sigma = \{e_1, e_2, ..., e_n\}$ the set of events in the system, and assuming that $e_{ik} \in \Sigma$ is realized at time τ_{ik} where $\tau i_1 < \tau i_2 < ... < \tau i_k$, the sequence $\sigma = e_{i1}, e_{i2} ... , e_{in}$, is called discrete process [1].

Examples of discrete processes are the most common type of processes of manufacturing, where the d_i components are machinery, conveyors, local or global computer networks, and the e_i events are status changes and operations performed by some of these components.

In the study of DEDS systems, there is interest in the properties of their internal events: competition, parallelism, synchronization of the events, conflicts, deadlocks, viability of the system and event planning. To study these properties one uses mathematical tools or (graphical) software: boolean algebra, finite-state diagrams, Petri nets, statecharts, Grafcet, max-plus algebra and models of formal languages, languages of real time.

The best known description used for DEDs is the basic model of transitions [2]. System dynamics are given by the behavior of d_i system components. System modeling is necessary to design a control system to bring it in the desired state.

If the DEDs are also distributed, control is generally a difficult problem due of the complex nature and known nonlinearities of distributed systems. The lack of precision and efficiency of the modeling techniques for large systems, distributed spatially, is unacceptable and leads to search for new solutions for the control systems. In addition, the control structure of the system itself tends to be distributed in the form of an array of sensors, actuators, and controllers [3][4] [5]. In this context emerged as a possible solution the use of control systems based on multi-agents. The main advantages of agent-based solutions in the industry are:

- strong distributed robustness of solutions;
- ability of replanning and rescheduling operations on the fly;
- the ability to reconfigure the hardware and the equipment manufacturing, in real time, if a local failure or sudden change occurs in the environment;
- simple methods of development or expansion of the hardware and software (*plug and operate*) to reduce the time of employment;
- reusing the software;
- easy maintenance of software. After [6] the technology deployments of agents in the industry are particularly found in the fields:
- manufacturing control; diagnostic; integration.
- the applications of the robots: avoiding collisions and trajectory planning;
- reconfigurability (especially of the equipment);

ISBN: 978-960-474-305-6

- production planning and scheduling, assistance and brokerage services;
- preventing, detecting and removing foreign elements or intruders;
- driving an unmanned vehicle (AGV - Automated Guided Vehicle);
- simulation and networking (services and security);
- virtual enterprise (including supply chain integration).

The best methods for modeling DEDS control structures are Petri-nets and its main derivatives: finite-state machines FSM and Grafcet. Also UML and AUML for modeling agents are common design solutions for industrial applications using agents.

2. Agent model

As defined by the creators of the theory of agents, an agent is a computing unit or a computer program that has several characteristics: autonomy, reactivity, pro-activity, social skills and more. Of these, the most important is autonomy. The agent exists in an interactive environment, being able to evolve so as to achieve its own goals [7].

A definition and graphical representation suggestion was given in [8]: an agent is a system that presents the main features of Figure 1. The greater the area covered by the perimeter, the more the entity acts as an agent. All 6 components (assumed orthogonal): autonomy, collaboration, adaptability, cognoscibility, persistence and mobility combine and work together to make the system more flexible and robust to change. Features are supported by infrastructure mechanisms and interfaces, models and policies configured in each agent or group of agents.

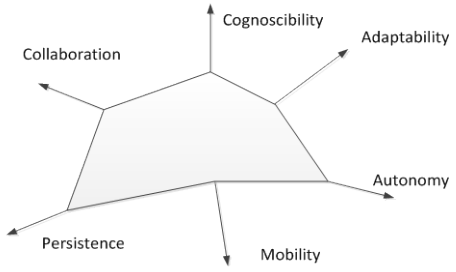

Figure 1 Proprieties of intelligent agent software [Mart01]

- Autonomy is the degree to which an agent is responsible for their own actions, independent of the messages sent by other agents.
- Collaboration is the degree to which agents communicate and work cooperatively with other agents in the MAS for a common goal.
- Adaptability is the extent to which the behavior of an agent can be changed after it has been deployed.
- Cognoscibility is the degree to which the agent can reason about their goals and knowledge.
- The mobility is the ability to move from one context to another.
- Persistence is the degree to which valid infrastructure agencies retain knowledge and status for an extended period of time, including robustness against possible failures at run-time.

Modeling agents and multi-agent systems have been created in almost all languages and development environments. By analysis, it is found that the preferred programming option is JAVA language, a procedural language, which most applications support. To provide more intelligence to agents, several tools have been added to JAVA, leading to the achievement of real multi-agent systems platforms: JADE, JADEX, Jason, Jack, etc. [9],[10],[11]. This suggests that despite the fact that JAVA is widely used for programming agents, some native qualities still lack for agent engineering [12].

3. Functional language Erlang for agent programming

Functional languages are characterized by the fact that by definition they allow a large modularity of programs. The existence of modularity implies automatically also great opportunities for component code reuse. Another reason for using the functional language for programming complex technical systems is the need to simplify the program and increase their robustness. It was easier for Erricson to write and maintain a program with over a million lines in the source language Erlang [13]. An essential quality of a simple programming language for industrial applications is distribuity. Language Erlang has been designed from the beginning with the aim to obtain reliable and robust software code by implementing the concept of redundancy in concurrent and distributed systems. The solutions using Erlang are more suitable than traditional procedural languages C, C + + or Java for the development of distributed applications. By its characteristics, Erlang is very close to the requirements of the standard FIPA-ACL for conversation and communication of multi-agent systems [14].

At the base of the distributive property, Erlang puts concept of node. The Erlang nodes are connected and communicate transparently in the distributed systems and asynchronously together, and simultaneously permit the building of redundant

ISBN: 978-960-474-305-6

structures by implementing the synchronization trees and other techniques.

4. Modeling and analysing MAS with specific DEDS methods

4.1 Problem formulation

For clarification will use a simple example, taken from manufacturing: a production line consisting of two floors, each floor containing a single CNC. All operations of one CNC are modeled as a single activity, called "processing", Figure 2. The two CNC machines, although different, evolve through three states p1, p2, p3, or p4, p5 and p6, changeable by the t1-t5 events in Table 1.

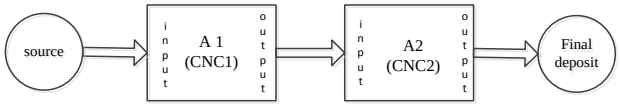

Figure 2 Pipeline with 2 agents

Table 1 Places, transitions and FSM codes

Places/ Trans.	**FSM code**	**Description**
*p*1	A1(1,0,0)	waiting for external input
*t*1		external input for CNC1
*p*2	A1(0,1,0)	CNC1 processing
*t*2		CNC1 processing finished
*p*3	A1(0,0,1)	waiting CNC1 output
*t*3		internal transfer (input to CNC2)
*p*4	A2(0,1,0)	CN2 processing
t4		CNC2 processing finished
*p*5	A2(0,0,1)	waiting CNC2 output
*t*5		CNC2 External output
*p*6	A2(1,0,0)	waiting input CNC2

Each one CNC corresponds to a software agent Ai (input, processing, output), with parameter the bits of the status code.

It would analyze the SMA consisting of two agents to obtain information on the existence of deadlock and other properties, first by finite state machine FSM modeling and then Petri-nets.

4.2 Modeling MAS with FSM

Finite state machine - FSM-is an abstract mathematical model of a system that can be in a finite number of states. The FSM may only be in one state at any point in time, known as the current state. The transition from one state to another is called state transition and is caused by an occurrence or the fulfillment of a condition. In general, an FSM is defined by specifying the list of states and the list of conditions for transition from one state to another.

Examples of real-world systems that can function as a state machines are vending machines, elevators, traffic lights, communication protocols, etc. FSM's have limitations compared to other models of computing machines, such as Turing machines that can do things that FSM sites cannot do. The limitations of FSM are due to the limited memory (limited number of states). Figure 4 shows the graph of the FSM with nine states for the example shown in this paper, the arcs representing the possible events that allow passage from one state to another.

FSM models are easily transferred from the abstract representation in Erlang program and the load on the processor is not very high [15].

Code sequence of Figure 3 shows how to program in Erlang such state machine for event control. Processing of new parts in a CNC cannot begin if the previous output is not done. The sequence contains the code necessary to strictly control the representation of states and transitions between them. The states are each represented by receiver functions, and events are represented by messages like this: *t*1=ms69, ms35; *t*2=ms97, ms24; *t*3= ms48, ms13, ms25, ms41; *t*4=ms36, ms59, ms87; *t*5= ms74, ms61.

```
-module(fms2ag).
-export([init/0]).
init() ->
register(pid1, spawn(fms2ag, s1, [])),
register(pid2, spawn(fms2ag, s2, [])),
. . . . . .
register(pid9, spawn(fms2ag, s9, [])).

s1() ->      %% A1&A2(100100)
   receive
       active -> true
   end,
   activity_in_node_s1(),
   receive
       t1  ->
              displ("t1->s2"),
              pid2 ! active;
       t3  ->
              displ("t3->s3"),
              pid3 ! active
   end,
   s1().
s2() ->      %% A1A2(010100)
   receive
       active -> true
   end,
   activity_in_node_s2(),
   receive
       t2   ->
              displ("t2->s4"),
              pid4 ! active;
       t3  ->
              displ("t3->s5"),
              pid5 ! active
   end,
       s2().
```

```
s3() ->          %% A1&A2(100010)
    receive
          active -> true
    end,
    activity_in_node_s3(),
    receive
          t1   ->
                displ("t1->s5"),
                pid5 ! active;
          t4  ->
                displ("t4->s6"),
                pid6 ! active
    end,
    s3().
. . . . . .
```

```
s9() ->          %% A1&A2(010001)
    receive
          active -> true
    end,
    activity_in_node_s9(),
    receive
          t2   ->
                displ("t2->s7"),
                pid7 ! active;
          t5   ->
                displ("t5->s2"),
                pid2 ! active
    end,
    s9().
.....
```

Figure 3 Erlang code for FSM model

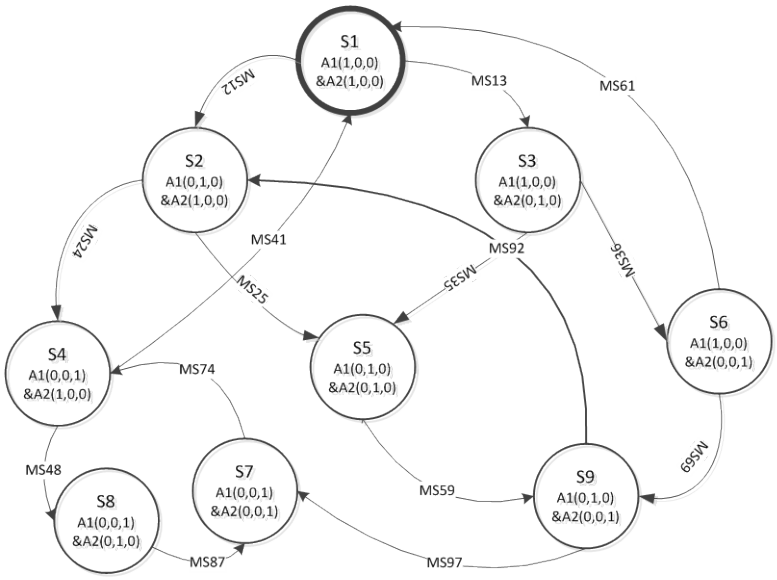

Figure 4 FSM graph for the MAS with 2 agents

For testing of the Petri-nets model, the events can be simulated sending the messages from a file containing messages, figure 5.

```
-module(fsm_cmd).
-export([start/0]).
   start() ->
      prg2ag_np_3:init(),
      pid1 ! active,
      pid1 ! t1,
      pid2 ! t2,
      pid4 ! t1,
      pid8 ! t4,
      pid7 ! t5,
      pid4 ! t3.
.....
```

Figure 5 Events as messages for FSM

The state functions (*receive*) wait in reception the occurrence of an event-message. When a message is received, the state machine jump to a new state. The FSM states is a finite space.

In each state, the data can be processed by adding functions. The actions on data must be performed before event-message reception, and other actions necessary to leave the state must be made in sequence receive after the message is received and before passing in the new state that follows.

Analyzing the result of MAS implementation with two agents, immediately it is deducted some problems of modeling the FSM:

- The number of states increases exponentially with the number of agents;
- the individual identity of agents is lost;
- Structural information on the system is confusing / obscure.
- Information on concurrency can not be represented; same about timing.

ISBN: 978-960-474-305-6

FSM models are simpler because they are old (although they have demonstrated derivability from Petri-nets) knowing several methods to achieve results; so, the design, implementation and execution of a specific FSM model is faster.

4.3 Modeling MAS with Petri-nets

Unlike the FSM, Petri-nets are powerful in construction of complex structures and expression of competition [17]. The great advantage of Petri-nets on FSM is the able to represent as actives simultaneously multiple states of system (FSM always has only one active state).

By analyzing MAS for the pipeline shown in figure 2 with the Petri-nets method, a graph as in figure 6 results. Studying this graph model with various tools, we can obtain information about network properties: boundedness, conservativeness, cyclicity (repetitiveness), and consistency. By combining these results we can deduce other properties like deadlock and performance indicators [1][16].

The assessment of the reachability tree (figure 7) shows that the set R(M0) is finite, the number of tokens in each place is limited and no dead transitions, so the network is bounded.

Analyzing the reachability graph (figure 8) with the start marking M0, active transitions are observed regardless of the state of network and all transitions T are included in the graph. From this we can conclude that the Petri model of corresponding MAS is bounded and viable, so no deadlock.

For the larger Petri nets it is not possible to build graphical construction and analysis by inspection of the reacheability tree and graph.

Evaluation of the structural properties of the Petri net of corresponding MAS is obtained by algebraic methods: calculation of the incidence matrix and the invariance theorems [18]. For the SMA with two agents taken as an example, the incidence matrix is:

$$A_{ij} = \begin{vmatrix} -1 & 1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{vmatrix} \quad (1)$$

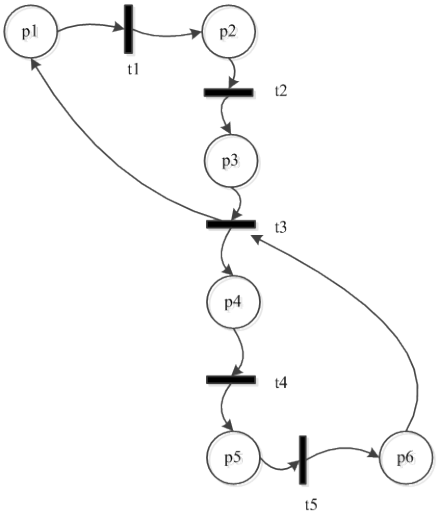

Figure 6 Petri-net model of the MAS with 2 agents

In addition to the incidence matrix, for the algebraic analysis of SMA are useful and necessary the P and T invariants. For this example, the invariant P is y_T=[1,1,1,1,1,1]^T, and the T invariants are x_{p1}=[1,1,1,0,0,0]^T i x_{p2}=[0,0,0,1,1,1]^T . The matrix a_{ij} and the invariants can provide the necessary informations on the characteristics of marginality and viability of the PN.

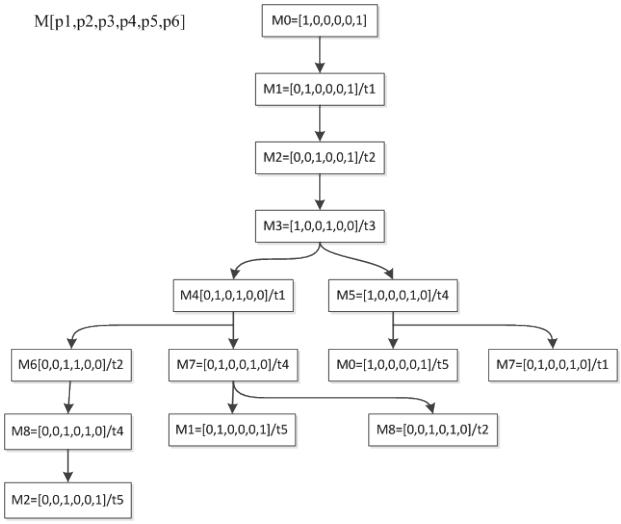

Figure 7 Reachability tree

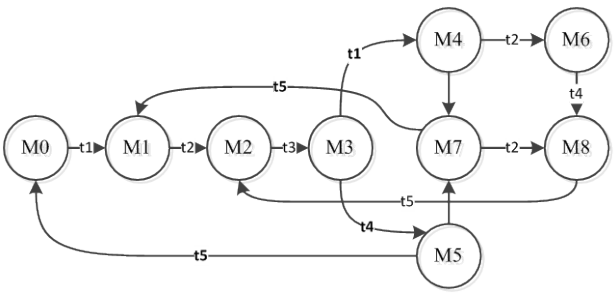

Figure 8 Reachability graph

6 Conclusion

In the design and analysis of MAS, both FSM and PN have advantages and disadvantages. The FSM is effective for describing MASs with predictable deterministic behavior, but sometimes the strictly predictive nature of FSM can be less desirable. For large systems, FSM models of MAS will be harder to maintain, due the phenomenon known as "spaghetti clutter" caused by state transitions.

The scope of the FSM is limited to cases where the MAS can be clearly decomposed in separate states with well-defined conditions for state transitions. This requires prior knowledge of the states, transitions and conditions to be defined. In addition, the conditions for transition are rigid.

When modeling complex MAS with Petri-nets, the distinction between a state and an action is important. A state may involve one or more actions. Design software requires for efficiency the decomposition of large functionalities in smaller activities or actions. The general solution can be made modular and easy to maintain, better by Petri-nets.

MASs can be analyzed in terms of structural Petri-nets networks, deciding on properties such as viability and boundedness. Therefore one can study the phenomenon of deadlock in the system, and the solutions can be found to avoid it.

Erlang is specialized for the field of distributive systems. It easily allows generation of thousands, even millions of concurrent processes in program, not in the operating system. Erlang can create new processes on remote nodes as easily as on the local node. It is therefore ideal for implementing DEDS applications based on models of FSM or Petri-nets type, either for simulation or for real-time functioning.

References:

[1] B. Hruz, M.C. Zhou, *Modelling and control of Discrete-event Dynamic Systems, with Petri Nets and Other TOOL*, Springer, 2007

[2] Z. Manna, A. Pnueli, *The temporal logic of reactive and concurrent systems, Springer Verlag, New York*, 1992

[3] A. B. Soglo, Zang Xianhui, Networked Control System Simulation Design and Its Application, *Tsinghua Science and Technology*, ISSN 1007+0214 05/16 pp287+294, Volume 11, Number 3, June 2006

[4] I. Timm, T. Scholz, O. Herzog, *Capability-based emerging organization of autonomous agents for flexible production control*, *Advanced Engineering Informatics 20*, 2006, 247-259

[5] Vladareanu L., Capitanu L., *„Hybrid Force-Position Systems with Vibration Control for Improvment of Hip Implant Stability*" Journal of Biomechanics, vol. 45, S279, Elsevier, 2012

[6] Michal Pechoucek, Vladimir Marik, Industrial deployment of multi-agent technologies: review and selected case studies, *Autonomous Agent and Multi-Agent Systems*, 2008, vol.17, no.3, pag. 397-431

[7] Michael Wooldridge, *An introduction to Multiagent Systems*, John Wiley & Sons, 2002

[8] Martin L. Griss, chapter 36 in Component-based Software Engineering: *Putting Pieces Together*, edited by George T. Heineman & Wiliam Council, May 2001, Addison Wesley

[9] F.Bellifemine, G.Caire, A.Poggi, G.Rimassa, *A software framework for developing multi-agent applications*. *Inf. and Soft. Tech.*(2008),doi:10.1016/j.infsof. 2007.10. 008

[10] R. Evertsz, M. Fletcher, R. Jones, J. Jarvis, J. Brusey, S. Dance, Implementing Industrial Multi-Agent System using JACK, Programming Multi-Agents, *Lectures Notes in Computer Science*, Vol. 3067, 2004, pp. 18-48

[11] L. Braubach, A. Pokahr, W. Lamersdorf, Jadex: A BDI-Agent System Combining Middleware and Reasoning, Software Agent-Based Applications, Platforms and Development Kits, *Whitestein Series in Software Agent Technologies* 2005, pp 143-168

[12] A. D. Stefano, C. Santorro, Using the Erlang for Multi-Agent System Implementation, *Proc. of Int. Conf. on Intelligent Agent Technology*, IEEE/WIC/ACM, 19-22 sept. 2005, 679 – 685, doi: 10.1109/IAT.2005.141

[13] Joe *Armstrong, Programming Erlang*, The Pragmatic bookshelf, 2007

[14] Foundation for Intelligent Physical Agents. www.fipa.org, 2013

[15] Ericson AB, *Erlang/OTP System Documentation* 5.9.1, april 2012

[16] O. Pastravanu, M. Matcovski, C. Mahulea, *Applications of the Petri-nets in the study of the Discrete Event Systems*, Pub. House Gh. Asachi, 2002

[17] L.Vladareanu, G. Tont, Hongnian Yu, D.Bucur, *The Petri Nets and Markov Chains Approach for the Walking Robots Dynamical Stability Control*, Proc. of the 2011 Int. Conf. on Advansed Mechatronic Systems, IEEE , Zhengzhou, China, 2011

[18] J. R. Celaya, A. Desrochers, R. J. Graves, *Modeling and Analysis of Multi-agent Systems using Petri Nets*, J. of Computers, Vol. 4, No. 10, 2009