Journal of Computing and Information Technology - CIT 22, 2014, 3, 171–189
doi:10.2498/cit.1002348

Using of DEVS and MAS Tools for Modeling and Simulation of an Industrial Steam Generator

Noureddine Seddari, Mohammed Redjimi and Sofiane Boukelkoul

University 20 August 1955, Faculty of Sciences, Department of Computer Science, Skikda, Algeria

Complex systems are made of many elementary components in interaction. To model such systems, it is generally more convenient to decompose them into subsystems that are simpler to handle. This new division is to be made in a methodical way, by identification and complete definition of the various structures, actions and interactions of those sub-systems. In this work, the decomposition of the overall system into sub-systems is based primarily on the use of the Discrete EVent System Specification (DEVS) formalism. The obtained atomic and coupled models are formally verified and validated. Then, we use the Multi-agent Development KIT (MAD-KIT) Multi Agent Systems (MAS) operational tools to implement an industrial simulator. This simulator is used by beginner operators in the petroleum field to ameliorate the process of training and learning without stopping the real processes. The advantage of this approach is its adaptability as well as its possibilities of extension (addition of new functionalities). Moreover, the decomposition into sub-systems reduces significantly the complexity of the elements being implemented and therefore, allows a great modularity and a better legibility to the system. Our work is realised in collaboration with the production department of natural gas liquefaction (GL1/K Skikda), one of the principal hydrocarbons poles from SONATRACH complex in Algeria.

Keywords: modeling and simulation, complex systems, industrial systems, discrete event system specification (DEVS), atomic and coupled models, multi-agent systems (MAS), MAD-KIT, AALAADIN

1. Introduction

A complex system is a system made of a great number of entities into interaction where the overall behaviour emerges from the actions and interactions of the components which make it up. The modelling and simulation [1-9] of those systems can be used for several purposes, for example, in order to handle the modelled system without risks, observe and analyse the phenomena emerging when we change the system inputs, improve understanding and prevent eventual damages of the system or while the system conception steps. The application domains of modelling are various and diversified. The formalisms used must be adequate in order to fully represent the structure and the behaviour of the studied system. Thereby, we need specific formalisms for modelling each category of systems. Today, modelled phenomena reach high degrees of complexities and smoothness, which require usage of performant and flexible Information Technology (IT) tools.

The IT tools are very widespread for systems modelling. A good prospection and choice of appropriate IT elements may ensure good solutions.

According to its importance, an industrial system can contain thousands of elements, even more. This is the case, for example, of the complexes of petroleum processing. The present work falls within this context.

Our work is realized in collaboration with the production department of natural gas liquefaction (GL1/K Skikda), one of the principal hydrocarbons poles from SONATRACH complex in Algeria and it aims to build an operational simulator of an industrial process (Steam Generator) used in this complex.

Thus, it may be convenient to define this work in these terms: what are the steps to be followed and the tools to be used for modelling this industrial process?

Today, there exists a large and significant panoply of methods, tools, products, platforms and development environments intended for modelling and simulation of processes.

The main difficulty when modelling a complex industrial system is the need to effectively manage its parameters. This is the field of systems integration, that is to say, the ability to produce a new industrial system from several existing systems.

The modelling of such systems requires a rigorous approach consisting, principally, of:

- In-depth knowledge of the real system.
- A perfect knowledge of the laws governing the area in which the system is supposed to evolve.
- Determination of the parts of the system to be modelled.
- Determination of the levels of abstraction that will determine the atomic and coupled models which compose the whole system.
- Determination for each model of the sequences of states, connections, communications, interactions and interfaces with other related models.
- Application of formal and operational tools to use for each model separately, testing its dynamic evolution and correcting eventual errors.
- When all the atomic and coupled models are errorless, one may connect the whole system components, then test them and make eventual corrections.

In the adopted approach, we use the Discrete Event System Specification (DEVS) [10, 11] for representing formally atomic and coupled models. Particularly, functioning of all the models is mathematically described and their interactions are graphically showed. On the other hand, complex industrial systems are generally classified as discrete event systems and according to B.P. Zeigler [11], we can prove that this formalism is adequate to represent the whole of this type of systems. Then, we use a Multi-Agent System (MAS) platform to implement this application.

We have chosen the MAD-KIT platform [12], particularly for the concepts of AGR (Agent, Group and Role). This platform is based on the organizational AALAADIN model [13- 15] which ensures hierarchical partitioning into elementary entities and gives many conception tools for static and dynamic representations of the system to be simulated. The result of these steps is an operational simulator of an industrial process in the hydrocarbons field.

The remaining part of this paper is organized as follows. Section 2 presents some related works. Section 3 concerns a general overview of the DEVS and MAS formalisms. The 4th section is devoted to the simulated industrial system. In Section 5, we present the system implementation and, finally, Section 6 concludes this paper.

2. Related Works and Motivation

Several approaches combine the DEVS and MAS formalisms. The method we propose in this paper goes in this way and is inspired by the following models.

The JAMES (Java-based Agent Modelling Environment for Simulation) and JAMES II (Java-based multipurpose Environment for Modelling and Simulation) platforms [16-18] allow dynamic simulations of multi-agents systems. An agent is characterized by a set of DEVS atomic model parameters, whereas a group of agents is represented by a coupled DEVS model parameters and the communication between each model and the environment is determined by the inputs and outputs events.

The agent environment plays an important role and it is perceived as a shared framework for the agent's interactions [19-22].

The GALATEA platform (GLIDER with Autonomous Logic-based Agents, Temporal reasoning and Abduction) [23-24] provides high level simulation concepts for specification and design of multi-agents systems such as Believe, Desire and Intention (BDI). Thus, one can model roles and action plans and establish communication, negotiation and dialogue between agents, allowing the emergence of efficient distributed solutions. The GALATEA platform uses DEVS formalism.

GALATEA combines two research lines: The simulation languages based on the Zeigler theory of simulation and the logical reasoning to select possible actions to perform in a knowledge environment based on logical agents. This platform uses a set of logical programming oriented languages and allows modelling by using several formalisms. The major disadvantage of this platform is its difficulty of implementation [25].

The Virtual Laboratory Environment (VLE) [26-27] is a high level simulation framework based on DEVS. This platform allows the simulation of the multi-agent systems by using the DEVS formalism. An agent is represented by a DEVS atomic model, an external stimulus by a message and the environment is characterized by CELL-DEVS [28-29].

The DEVSimPy platform [25] considers the advantages and disadvantages of these latter platforms and provides a framework for agents modelling and simulation by using DEVS formalism.

Thus, both the agents and the environments are represented by DEVS atomic models. A group of agents is represented by a DEVS coupled model.

Finally, one can conclude that, in these approaches, the agents, environments, interactions between the agents and groups of agents are represented by atomic and coupled DEVS models.

After having studied profoundly all the approaches enumerated above, we propose a new approach which respects the following constraints:

- Specific to the field of application, in this case, modelling of industrial processes (in particular the domain of hydrocarbons).
- Simple and fast to implement.
- Flexible and extensible.
- Very reliable: reliability is very important for industrial systems.
- Ensuring effective human-machine interfaces adapted to the application field (being able to represent graphically evolutions of the real time processes through various forms of display: synoptic, curved plans...).
- Allowing several modes of simulation (such as step by step or according to chosen clocks).

In order to answer to the aforementioned arguments, we propose a DEVS based software process simulation modelling technique combined with a multi-agent system (MAS). With this approach, we can clearly specify and verify the simulation model.

The proposed approach is based on the following principles:

- Partition of the global system into sub-systems.
- Using the DEVS formalism for the determination of the atomic and coupled models according to the levels of abstractions of the obtained components (Discrete event simulation has a fast execution).
- Checking and validating of these models can be made easy thanks to DEVS specifications.
- Implementing of the system, thanks to the MAD-KIT multi-agent platform, by using the AGR methodology. This means writing an extension of MAD-KIT to ensure the necessary transformations between the DEVS models and MAD-KIT on one hand and the modelling of interactions between these models on the other, and, lastly, to ensure coordination between the various components of the system.
- The choice of the AGR model can be justified by its simplicity of implementation and its adaptation to the specified domain. MAD-KIT is a MAS platform written in JAVA, which offers highly successful tools for the management of the agents (creation, scheduling, interactions with messages passing...).

2.1. Contribution

In our approach, instead of representing the MAS components by using atomic and coupled models, we proceed in a different way. At first, the atomic and coupled models which compose the system to be modelled are defined and then these entities are implemented by using MAS components.

Implementation of the DEVS models by the use of the formalism AGR facilitates enormously the task of programming: the atomic models are easily implemented thanks to the agents and the coupled models by the notion of group of agents, the interactions between agents and groups of agents are materialized by message passing tools.

In order to give a general overview of this implementation let us consider that [25]: 1) A DEVS model can be considered as an independent entity or as a model of a larger system, 2) For each atomic or coupled DEVS model it is possible to build an equivalent DEVS atomic model. That is to say that DEVS formalism is closed under coupling. Thus, one can consider only atomic DEVS models specifications (equation 1), for example, in order to simplify the implementation of DEVS models by using MAS.

The models (the agents) we have developed are implemented by using java code and the interactions between agents (inputs and outputs events) are performed by MAS message passing tools. Some algorithms are presented in Section 5.

3. General Overview

3.1. The DEVS Formalism

The DEVS formalism was introduced by B.P. Ziegler in the seventies [10, 11]. It is an approach to modelling based on the overall theory of the systems. This approach was adopted and developed by an international community of researchers [30-38]. The DEVS formalism has been improved and adapted to a significant number of applications [39-42].

There are two kinds of models in DEVS: atomic and coupled models. The atomic models are based on continuous time inputs, outputs, states and functions. The coupled models are constructed by connecting several atomic models.

3.1.1. DEVS atomic models

A DEVS atomic model is described with the following equation:

$$AtomicDEVS:\ \langle \mathrm{X};\mathrm{Y};\mathrm{S};\delta_{int};\delta_{ext};t_a;\lambda\rangle \quad (1)$$

X is the set of input events.
Y is the set of output events.
S is the set of states.
δ_{int}: S→S: is the internal transition function which makes the system evolve from one state to another in an autonomous way. It depends on the time elapsed in the current state.
δ_{ext}: Q×X→S: is the external transition function which occurs when the model receives an external event. It returns the new state of the system based on the current state.
Q= $\{(s,e)|s \in \mathrm{S},\ 0 \le e \le ta(s)\}$ is the total state set, e is the time elapsed since last transition.
λ: S→Y: is the output function of the model. It is activated when the elapsed time in a given state is equal to its life.
$t_a(s)$ shows the life of a state "s" of the system. It is the time during which the model will remain in this state if no external event occurs.

3.1.2. DEVS coupled models

The equation (2) describes a coupled DEVS model:

$$CoupledDevs:\langle \mathrm{X}_{self};\mathrm{Y}_{self};\mathrm{D};\{Md/d\in \mathrm{D}\}; EIC;EOC;IC\rangle \quad (2)$$

Self: is the model itself.
X_{self} is the set of inputs of coupled model.
Y_{self} is the set of outputs of coupled model.
D is the set of names associated with the components of the model, self is not in D.
$\{Md/d \in \mathrm{D}\}$ is the set of components of coupled model.
EIC, EOC and IC define the coupling structure in coupled model.
EIC: The set of input links. They connect the coupled model to its components.
EOC: The output links. They connect the components to the coupled model.
IC defines internal coupling. It connects the outputs of components with entries from other components in the same coupled model. However, no direct feedback loops are allowed. This means that an output port of a component (model) can't be connected to an input port of the same component.

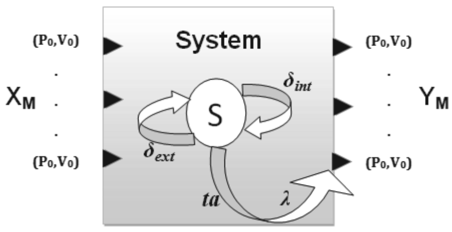

Figure 1. Graphical representation of an atomic model.

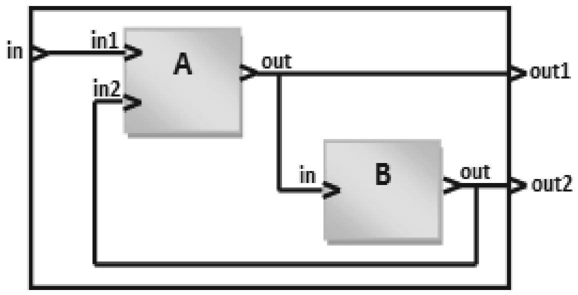

Figure 2. Graphical representation of a coupled model consisting of two atomic models.

3.1.3. The DEVS modelling

The multitude of modelling tools and the desire to reuse existing models have stimulated researchers to guide their work to the standardization of the tools to be used. The DEVS formalism seems to be well suited for this mission.

The strength of DEVS is summed up in its ability to express, thanks to the concept of abstraction applied to each level, ranging from atomic models to a collaboration of a set of models where each model interacts with the others. Although independent of the implementation, DEVS provides a modular and hierarchical vision of dynamic models. Events generated by a model can take values in various fields and can be stimuli for other models. Thus, according to B. P. Zeigler [10-11], we can prove that there is a DEVS model for all discrete events systems. But we can go further. In fact, DEVS can be 'universal' [43], allowing the coupling of models and formalisms described as heterogeneous paradigms. The main idea is that the models are considered as black boxes that have no link with the outside world except through input and output ports to exchange events and values. With this feature abstraction, several models can be coupled while enjoying the reuse of existing models. It is also possible to perform formal verification of DEVS models, which is a valuable aid in the design of systems [44]. Coupling models based on DEVS is a typical task. However, non-DEVS models require extra effort to be coupled. Two methods exist to incorporate a non-DEVS model in the DEVS: co-simulation and transformation [46]. Transformation of non-DEVS models to the DEVS comes in the way to specify models in a uniform language. Vangheluwe [9] represents the various possible transformations by using formalism transformation graph "FTG". In the case of co-simulation, which is standardized, one considers the communication between simulators and not the model specifications. There are several works such as High Level Architecture (HLA) [47] which come into this framework.

3.2. The Multi-Agent Systems (MAS)

The multi-agent approach is at the root of the connection of several specific domains of artificial intelligence, distributed computing systems and software engineering. It is a discipline that focuses on collective behavior produced by the interactions of several autonomous and flexible entities called agents. These interactions turn around the cooperation, competition or coexistence between these agents. A multi-agent system (MAS) is a distributed system consisting of a set of interacting agents. It is [13], [48-49] generally characterized by: each agent has limited information and problem-solving abilities and each agent has an individual space (its environment) and a partial view of a distributed space (sharing environment);

- There is no overall control of multi-agent system.
- Data are decentralized.
- Calculations are asynchronous.
- The agents can interact by transmitting messages or by producing and consuming data via shared memories (blackboards).

J. Ferber [13] defines components of MAS as follows:

- An identified environment provided with a system, that is to say, a space delivered, generally, with a metric.

- A set of passive objects that can be perceived, created, modified or destroyed by agents.
- A set of active agents.
- A set of relations that link objects between them.
- A set of operations that provide the opportunity for the agents to perceive, produce, consume, transform and manipulate objects.
- A set of universal laws which are operators responsible for representing the application of agent actions on the world and the world's reaction to these actions.

3.2.1. AALAADIN model

With AALAADIN [14, 15], one can describe multi agent systems with different forms of organizations such as market-like or hierarchical organizations and therefore it could be useful for designing MAS. The organization in AALAADIN is a framework for the activity and interaction through the definition of groups, roles and their relationships. Figures 3 and 4 represent the diagrams of this model. AALAADIN uses the concept of AGR (Agent, Group and Role):

Agent: An agent is only specified as an active communicating entity which plays roles within groups. The model places no constraints on the internal architecture of the agents. Figure 3 represents the Agent model.

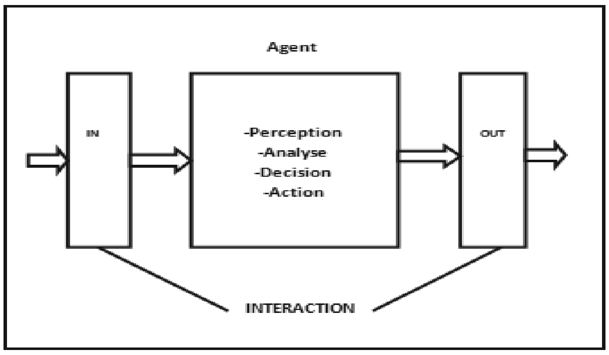

Figure 3. Basic architecture of an agent.

Group: Groups are defined as atomic sets of agent aggregation representing any usual multi-agent system. Each agent is part of one or more groups. In AALAADIN, groups can freely overlap.

Role: The role is an abstract representation of an agent function, service or identification within a group.

Each agent can handle multiple roles, and each role handled by an agent is local to a group. Figure 4 represents the diagram of AALAADIN model.

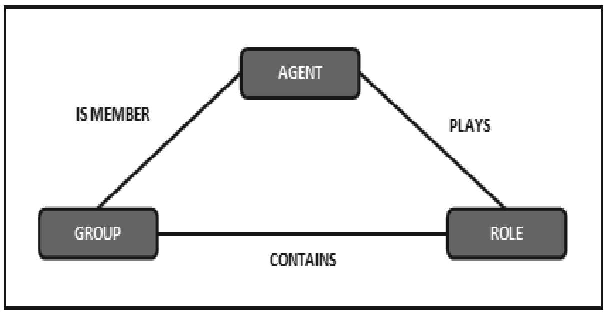

Figure 4. Agent/Group/Role model.

4. Simulated Industrial System

Our implementation concerns the simulation of an industrial regulation system (Steam Generator) used in industrial petroleum process [50-51]. In this section, we will show this process.

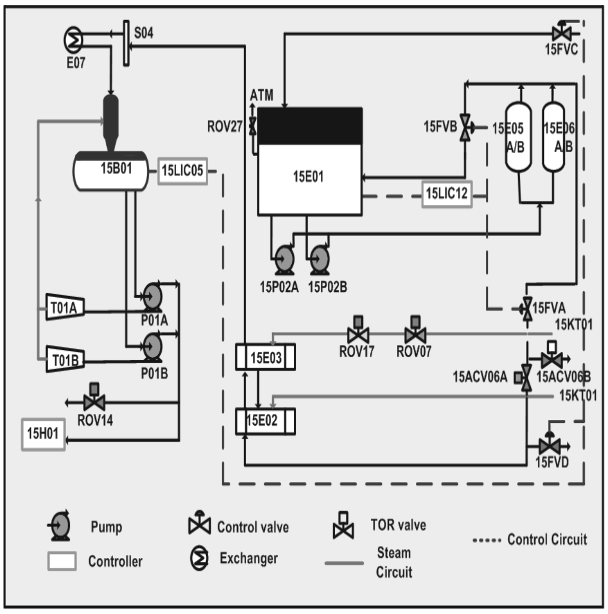

Figure 5. Technical drawing of the water station.

Two major components compose the process: the feeding water which provides appropriate quantities of water as requested by the second entity: the boiler, which generates the steam as required for other industrial processes (turbo-compressors, turbines...). Figures 5 and 6 illustrate this system.

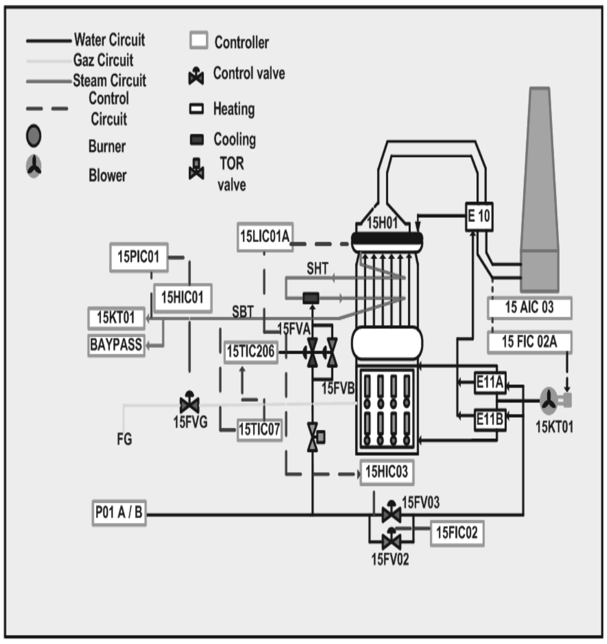

Figure 6. Technical drawing of the steam generator.

4.1. The Feed Water Station

This station supplies the steam generator with feeding water, it includes:

- The condenser (15E01).
- The degasser (15B02).
- The feed water tank (15B01).

4.1.1. Circuit description

The extracting water comes from the condenser (15E01) by the bottom, it gets through a valve and a mesh filter before its inhalation by the pumps (15 PM 02 A/B). This water is inhibited with 12 bars towards the maintaining ejectors (15 E 05 A/B) and it will feed the heaters while passing by the valve (15 FVA) towards the degasser (15B02) and the valve (15FVB) turns back towards the condenser.

Those two valves functions are in split range and controlled by the regulator (15 LIC12) (level 15 E01). Then, the water crosses the valves (15 ACV 06 A/B): A towards the heaters and B towards the channel).

The valve (15 FVD) goes towards U50 and the valve (15 FVA) supplement of Unit 50 towards the condenser (15E 01) and they are controlled in split range by the regulator (15 LIC 05) in order to maintain a constant level in the feed water tank.

The extracting water is then conducted towards the controllers (15 E 02 and 03) in contact with steam racking 2 and 3 of the steam circuit (15 KT 01). The two heaters are provided with By-pass, the extracting water goes then to feed the degasser while crossing through a chain valve with auto clean filter.

The level of water in the condenser (15E01) is regulated to 65% by a loop of regulation 15 LIC 12. The level of water in the feeding tank (15B01) is regulated to 65% by a loop of regulation 15 LIC05.

4.2. The Steam Generator

A boiler is a steam generator; it aims to raise the temperature of water until changing its status, that is to say, to become steam, and then to bring it to proper defined pressure and temperature.

Regarding boiler's construction, we distinguish:

- Boiler making: tubes return box, screens, atomizer, super-heaters and drums
- The combustion chamber: burners, air circuits and auxiliaries.

4.2.1. Principle of operation

The water is brought to the boiler by a feeding-water pump. The power of the boiler is featured by intensity of steam expressed in kilograms produced per hour.

The air is provided by two blower's fans, the first one is conducted by turning turbine and the second one operates through an engine (these two fans do not operate simultaneously), the air is then heated by the air super-heater before entering the burners by using the heat of evacuated gases. The air flow is regulated according to the fuel, to have a complete combustion.

The feeding of water is done starting from a pump controlled by two valves mounted in parallels, the first one is used to fill the tank and the other one for regulation purpose (body of adjustment/regulation); from where it comes out to go then by an economizer in order to be heated by the calories of the evacuated smoke and returns finally to the higher tank to produce the steam.

The saturated steam comes out from the higher tank, goes by the super heater (in the hearth), its temperature increases while its pressure varies slightly. At the end, the final temperature of the produced steam will be regulated. This dry steam is also called load of the boiler.

4.2.2. Regulations which ensure the generation of steam

Five conditions are required to ensure a proper production of steam:

1. Regulation of heating (a correct air-gas factor).
2. Control of combustible flow (15FVG).
3. Control of air flow (15KT01).
4. Regulation of balloon level (15H01) to 50%.
5. Control of superheated steam temperature (495°C).

4.3. DEVS Model of the Industrial Simulator

To model our system, we have used primarily the DEVS formalism; therefore, we have split up our process into four models. Each coupled model is composed of a set of atomic models. Figure 7 represents the overall structure of the simulator based on the suggested DEVS model.

The release of disturbances is done by the operator which controls the system by getting instructions from the load (i_l)[1] of the simulator.

4.3.1. The feeding coupled model

Decomposition on atomic DEVS models of the feeding coupled ensures the operation of water feeding:

- Model (15LIC05): Shows the regulator (15LIC05). Its role consists of the control of the tank. The regulator (15LIC05) sends the orders of opening and closing (o_fw) towards the 15FVC valve until the measured level is equal to the desired level (65%).
- Model (15B01): Shows the tank (15B01). Its role consists of checking the level of water (t_l) in the tank.
- Model (15FVC): Shows the valve (15FVC). This valve controls water flow (c_w_f) entering the condenser.

4.3.2. The condensation coupled model

Decomposition of the condensation coupled model into a set of atomic DEVS models to carry out the condensation operation of water.

- Model (15LIC12): Shows the regulator (15LIC12). Its role consists of the control of the condenser. The regulator (15LIC12) sends the orders of opening and closing system (o_cn) towards 15FVA valve until this measured level is equal to the desired level (65%).
- Model (15E01): Shows the condenser (15E01). Its role consists of checking the variation of water level (c_l) in the condenser.
- Model (15FVA): Shows the valve (15FVA). Its role is to control the water flow (t_w_f) coming out towards the tank.

4.3.3. The combustion coupled model

This model is formed by a set of atomic DEVS models making it possible to carry out the combustion operation:

- Model (15HIC01): Shows the regulator (15HIC01). Its role is to control the combustion and to adjust the steam flow (v_f).

[1] Appendix 1 and Appendix 2 explain all the symbols and functions used in the following text.

In fact, if this flow is not consistent to the load of the boiler, it sends orders of opening and closing (o_cm) towards the Turbo-fans (15KT01) and to the valve 15FVG in order to reach the desired sizes.

- Model (15KT01): Shows the Turbo-fan (15KT01). Its role is to control the air flow (a_f).
- Model (15FVG): Shows the valve (15FVG). Its role is to control the gas flow (g_f).

4.3.4. The Hydraulic coupled model

- Model (15LIC01A): Shows the regulator (15LIC01A). Its role is to regulate the level of the tank. It sends the commands for opening and closing (o_hy) towards the valve 15FV03 until this measured level equals the desired level (50%).
- Model (15FV03): Shows the valve (15FV03). Its role is to control the feeding water flow (w_f) coming from the water station.
- Model (15H01): Shows the tank (15H01). Its role is to calculate the level of the balloon (b_l) by using the values provided by the regulator (15HIC01) and the valve 15FV03.
- In addition to these models, we consider two others: the APM model which provides internal system cooperation and monitoring and the graphical user interface GUS which provides the data inputs, outputs and controls. These DEVS models constitute the perturbation coupled model.

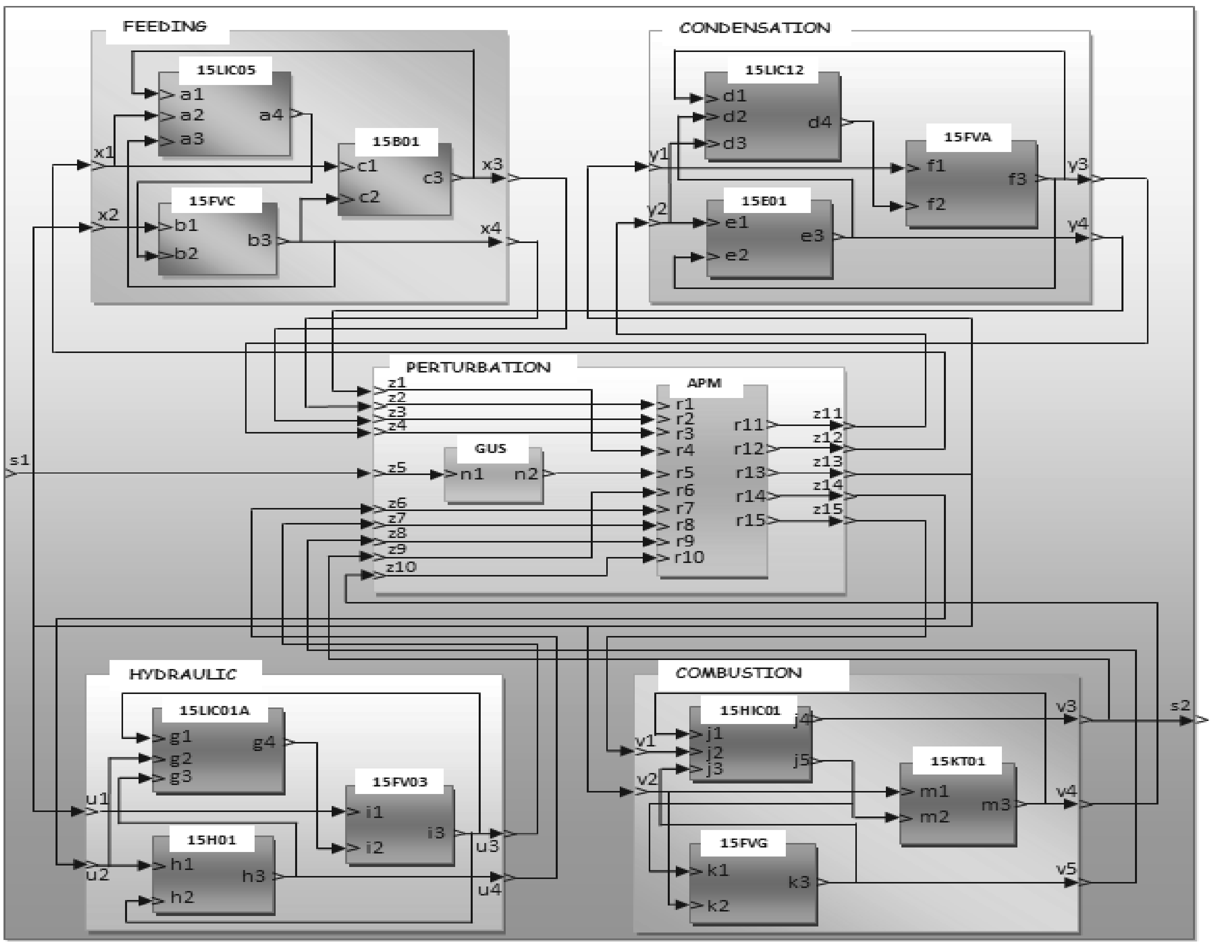

Figure 7. DEVS models of the simulator.

4.4. An Atomic Model Behaviour and DEVS Specification Example

In order to give a practical example of the behavior of the modeled system, let us consider the regulator 15LIC12 illustrated in Figure 7. Initially, this component is in phase "passive", when a news t_w_f, c_l and c_w_f respectively arrive on the inputs ports "d1, d2, d3" it goes to regulate the condensation. This is achieved by the sentence "hold-in *active condensation time*", which sets the phase to *active* and sigma to *condensation time*. Such handling of incoming values is represented in the external transition function, when a value arrives while the regulator is in "active" phase, it simply ignores it. This is achieved by the "continue" sentence which updates sigma to reflect the passage of elapsed time (t_a), but otherwise leaves the state unchanged. When the regulator has finished regulation, it places the orders (o_cn) on port "d4" and returns to the "passive" phase. Sending of the orders is done by the output function which is called just before the state transition function. The latter contains the sentence "passive" which returns the model to the idle state in which the phase is "passive" and sigma is "infinity". All the other models of our system follow the same way of representation. Thus, the DEVS specification of the 15LIC12 regulator is represented in Table 1.

15LIC12	
InPorts={"d1","d2","d3"}	
X= {(d1,t_w_f), (d2,c_l), (d3,c_w_f)}	
OutPorts={"d4"}	
Y={*(d4, o_cn)*}	
δ*ext (phase,*σ*,e,X)=*	
("active", condensation time)	*if phase ="passive"*
(phase,σ*-e)*	*otherwise*
δ*int (phase,*σ*)=*	
("passive",∞*)*	*if 15LIC12.InPorts.Values=*$\emptyset$
("active", condensation time)	*otherwise*
λ*("active")=o_cn*	

Table 1. The DEVS specification of the 15LIC12 regulator.

The DEVS formal specifications of all the atomic and coupled models of the simulator are summed up in Appendix 1 and Appendix 2, respectively.

5. Using Multi-agent System

This section is devoted to the system implementation. The models, previously obtained and formally verified and validated in DEVS, are developed by using the AALAADIN organizational model [14], [15].

AALAADIN focuses on the analysis, design and formalization of multi-agent systems in an organizational perspective. It is based on the conceptual model (AGR) by using the concepts of agents, groups and roles [12].

In the present multi-agent design, we have retained the same decomposition of the system into atomic and coupled models. Thus, coupled DEVS models are represented by groups of agents and atomic models by agents.

The decomposition of agents, groups and roles played by agents between the groups is obtained in conformity with the DEVS decomposition based itself on the elementary components of the system which formed the atomic models. DEVS specification gives a rigorous and mathematical scheme of the formal structure and dynamical behaviour of each of these components. The second step, shown here, concerns the system implementation.

We use MAD-KIT platform as a programming environment in order to have executable codes and data structures concerning our system. Thus, to have an operational simulator, we take advantage of the flexibility as well as of the well adapted tools provided by the MAD-KIT environment.

Table 2 presents these groups and agents. The roles played by the different agents are defined previously, by using DEVS specification, particularly the transitions between the states when specific events occur.

Practically, the atomic models are represented by agents, the coupled models by groups, the roles represent the states evolutions of the model, the inputs and outputs are materialized by interactions between agents or groups.

It is important to note here that we don't preconise this decomposition as a general shape (which would be formally proved). We adopted this approach to resolve only the application considered here. We are working actually to generalize this approach for other industrial systems and particularly the petroleum processes.

Groups	Agents
Perturbation group	APM, GUS
Feeding group	15FVC, 15B01, 15L1C05
Condensation group	15FVA, 15E01, 15LIC12
Combustion group	15FV0, 15KT01, 15HIC01
Hydraulic group	15H01, 15FV03, 15LIC01

Table 2. Groups and agents.

Figure 8 presents the organizational AALAADIN design of the system to be modeled: agents, groups and interactions between agents and groups are clearly expressed.

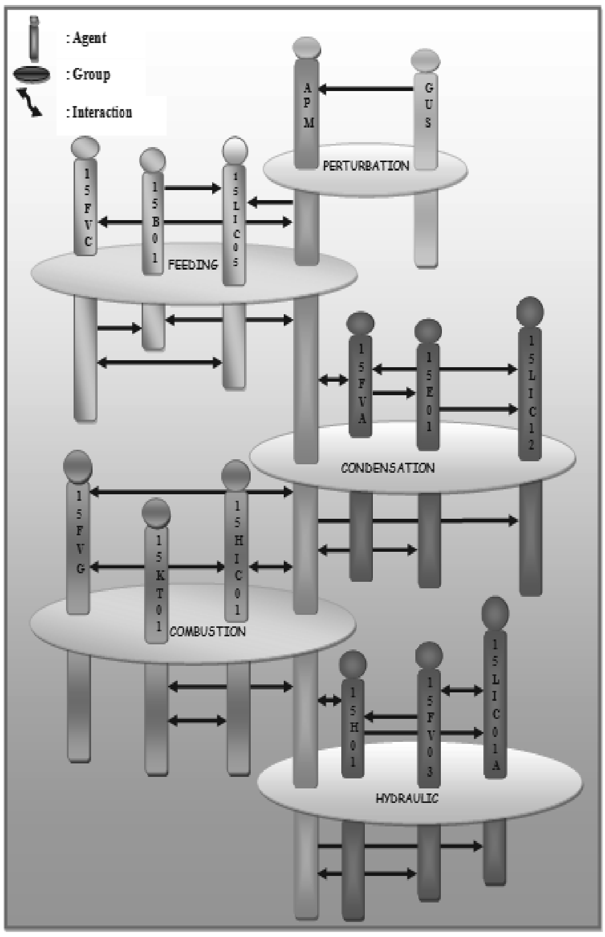

Figure 8. Organizational system.

5.1. Implementation of the System

The MAD-KIT platform is a software environment which offers advanced tools for agents managing the use of the concepts of AGR. The MAD-KIT MAS platform is based on the AALAADIN organizational model.

MAD-KIT is a set of packages of Java classes that implements the agent kernel, several libraries of messages, probes and agents. Tools for graphical development environment and standard agent models are included in MAD-KIT. This environment allows us to implement all the DEVS models by using MAS modelling. Interactions and communication between the agents are realised by message passing.

Algorithms 1, 2 and 3 below give some examples of java codes.

Algorithm 1: Implementation of the 15LIC05 regulator

```
public class 15LIC05 extends atomic{
protected float w_f, c_w_f;
protectedint o_wf, i_l, t_l;
protected double regulation_fee_time;
protected String input;
public 15LIC05(String name, double
Regulation_fee_time)}
super(name);
addInport(“a1”);
addInport(“a2”);
addInport(“a3”);
addInport(“a4”);
addOutport(“a5”);
phases.add(“active”);
regulation_fee_time=Regulation_fee_time;
}
public void initialize(){
phase=“passive”;
sigma=INFINITY;
o_fw=0 // closing
input=new String(“in”);
super.initialize();
}
```

5.2. Sending Messages

Basically, the aim of the agents is to exchange messages. This exchange of information is done through the class *Message*. This class defines only things like transmitter, receiver and the date of issuance.

The method *sendMessage*(*AgentAddress*, *message*) is the most basic and lowest level. It allows sending the message "*message*" to the agent at "*AgentAddress*" which may be obtained by another agent by requesting either group or the role. This method accepts two parameters:

- The first one is *AgentAddress* type that allows a unique identification of the address of the receiving agent; we get back thanks to the method: *getAgentWithrole* (group, role).
- The second parameter shows the message to be sent; the *ObjectMessage* class, for example, allows sending any object which gives great freedom to the programmer to compose the message.

Example:

Algorithm 2 shows how the 15FVA agent sends the new water flow (t_w_f) coming out towards the tank to the agent regulator 15LIC12:

Algorithm 2: Sending message

```
voidsend measurement ()
{
valmes=new Val ();
mes.setv1(t_w_f);
mes.settype("(t_w_f");
ObjectMessageobj=
newObjectMessage((Object)mes);
AgentAddressadr=
getAgentWithRole("condensation","agent_15LIC12");
sendMessage(adr,obj);
    ...
    }
```

5.3. Receiving Messages

Each agent has a private mailbox in which the kernel posts messages, at the lowest level; there are two methods of receiving messages:

- *is Message Box Empty ():* Checks if the agent has not read his mailbox messages, which returns true if it is empty and if there are unread messages it returns false.
- *next Message ():* Method of recovering the first unread, which is then removed from the message box.

Example:

The portion of the following code shows the use of such methods by the 15L1C05 agent:

Algorithm 3: Receiving message

```
public voidt_msg()
{while(!isMessageBoxEmpty())
    }

msg_15L1C05=(ObjectMessage)nextMessage();
traiting_message(msg_15L1C05);
      }
    }
```

5.4. Execution Policy

Within multi-agent systems, it is necessary to manage a wide number of granularity agents enough thin and to control their scheduling. It is almost impossible if we based on a process mechanism due to induced overload. In this case, we use "synchronous" agents via external agents that will define their synchronous execution policy. For the purpose of execution policy implementation, we set two levels: scheduler and activator.

The scheduling under MAD-KIT type is delegated to the agents inheriting the scheduler class. A scheduler agent aims to coordinate the execution of agents through generic objects tools called activators; the latter are the means for the scheduler to identify a set of agents.

The code of an activator has all the methods of other agents that will be set forth while it is activated by the scheduler agent. It should be noted here, that the methods set forth by activators are those of the agents which inherit the *AbstractAgent* and implement the *ReferenceableAgent* interface, and not the agents which inherit from the Agent class.

Indeed, the use of the schedulers and the activators reduces the rate of processor's use by system agents; moreover, their use provides more flexibility.

As shown in Figure 9, we have used three scheduling agents (the water station scheduler, the perturbation scheduler and the boiler scheduler) which use a set of activators to execute the agents of our system:

- An activator which executes the code of APM agent.
- An activator which executes the code of GUS agent.
- Two activators, one for processing purpose and another for graphic display, which executes water station agents.
- Two activators, one for processing purpose and another for graphical display which execute boiler agents.

Figures 9, 10, 11 and 12 show the interfaces obtained for user-system interaction.

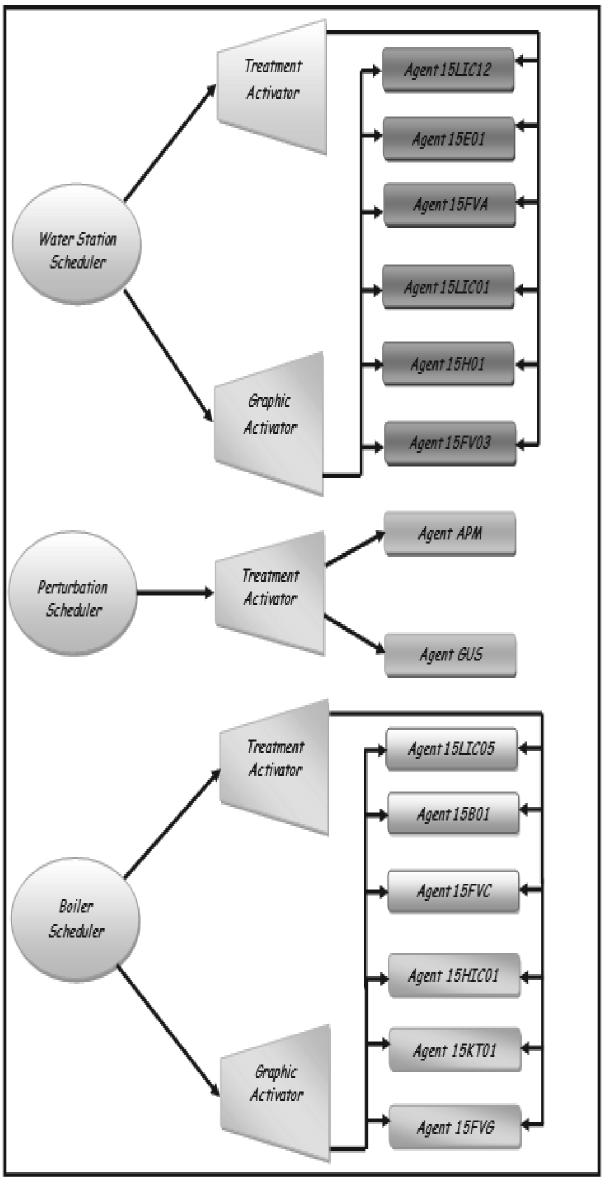

Figure 9. Structure of the simulator scheduler.

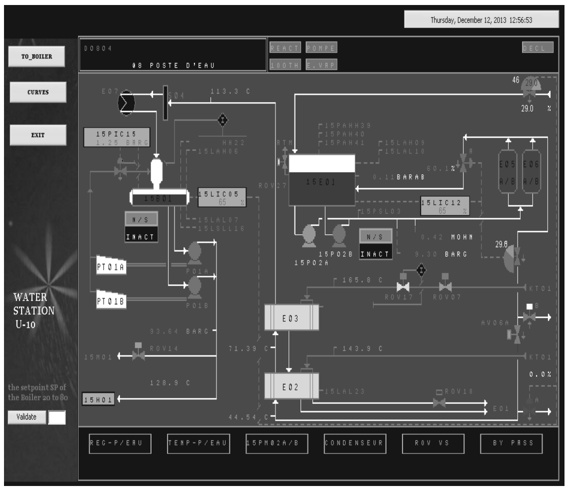

Figure 10. Interface of water station section of the simulator.

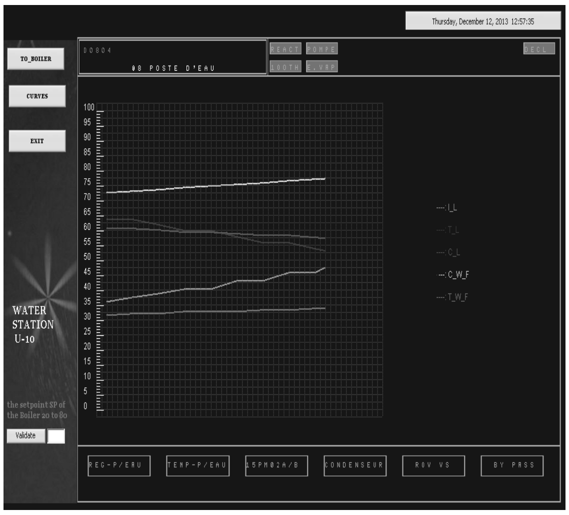

Figure 11. Interface of the curves of the water station simulator.

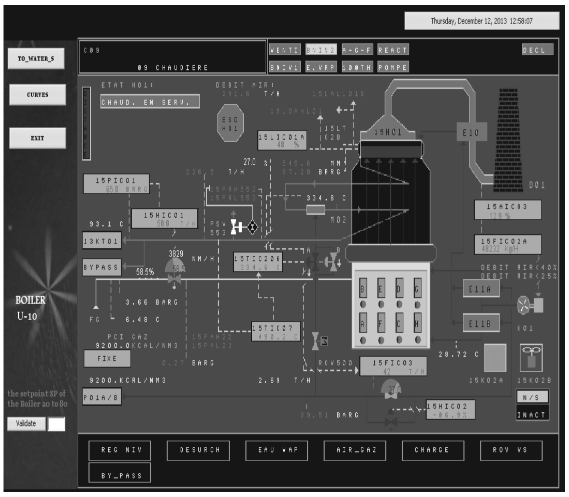

Figure 12. Interface of steam generator section of the simulator.

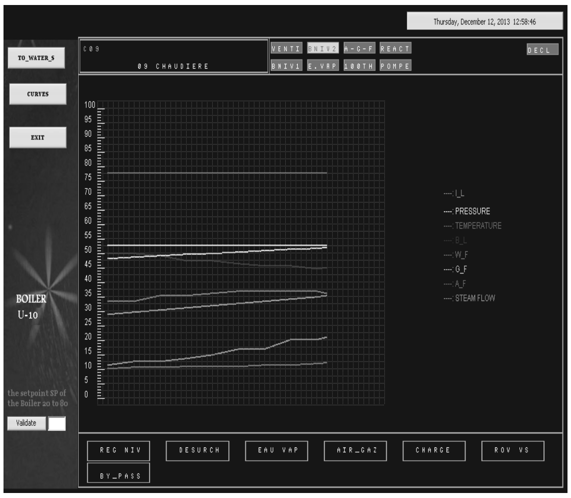

Figure 13. Interface of the curves of the steam generator simulation.

5.5. The Use of the System

The user has a set of control and data inputs to initialize the system in several modes, the system runs according to these data. The states of the system are displayed on the synopsys in real time.

The user can launch the simulation by pressing the buttons TO_WATER_S and TO_BOILER. Therefore these buttons will give him the possibilities of rocking the interfaces of the water station section and the boiler section. It can seize the load of the system in the text field. The results of simulation can be presented by curves with CURVES Button. A real time simulation is executed, as shown in Figures 10 and 12 which represent the water station and the boiler (see the technical diagram, in Figures 5 and 6) Thus, the user can pilot the operation of steam regeneration and (water) feeding of the boiler with water demineralized by publishing different data of regulation(c_w_f, t_l, c_l, g_f, t_w_f,...) and can follow the system evolutions.

These data are presented graphically by curves (see Figures 11 and 13).

6. Conclusion

The objective of our simulation is to use rigorous formalisms and tools for a good representation of a complex industrial system process.

In this work, we have built a simulator of an industrial process. Use of this system is dedicated principally for training beginner operators, workers and new recruits, enabling them to understand and to simulate the system without acting on the real system and without stopping processes. This simulator saves time and money and prevents potential damages to the real system through manipulation. For the design and implementation of the system, we opted for a hybrid method that combines DEVS and multiagent platform MAD-KIT.

Actually, we are working on the generalization of the idea developed in this paper and we aim to propose a general method for building simulators able to operate on many classes of industrial systems.

References

[1] L. V. BERTALANFFY, *Théorie générale des systèmes*. Dunod, 1968.

[2] H. A. SIMON, *The sciences of the artificial*. Cambridge (MA), MIT Press, 1969.

[3] P. A. FISHWICK, *Simulation Model Design and Execution. Building Digital Worlds*. Prentice Hall, 1995.

[4] M. SONNESSA, Modelling and simulation of complex systems. Doctoral dissertation, PhD Thesis in "Cultura e impresa", University of Torino, Italy, 2004.

[5] R. E. SHANNON, Simulation modelling and methodology. *Proceedings of the 76th Bicentennial Conference on Winter Simulation*, (1976), pp. 9–15.

[6] R. G. INGALLS, Introduction to simulation. *Proceedings of the 33rd Conference on Winter Simulation, IEEE Computer Society*, (2001), pp. 7–16.

[7] R. E. SHANNON, Introduction to the art and science of simulation. *Proceedings of the 30th Conference on Winter Simulation, IEEE Computer Society Press*, (1998), pp. 7–14.

[8] P. A. FISHWICK, Computer simulation: growth through extension. *Transactions of the Society for Computer Simulation International*, **14**(1) (1997), 13–23.

[9] H. VANGHELUWE, Foundations of modelling and simulation of complex systems. *Electronic Communication of the EASST, 10: Graph Transformation and Visual Modeling Techniques*, (2008). `http://eceasst.cs.tuBerlin.de./index.php/eceasst/issue/view/19`

[10] B. P. ZEIGLER, *Theory of Modeling and Simulation*. Academic Press, 1976.

[11] B. P. Zeigler, T. G. Kim, H. Praehofer, *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA, 2000.

[12] O. Gutknecht, J. Ferber, The MadKit agent platform architecture. *Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier*, (2000).

[13] J. Ferber, Les systèmes multi-agents: vers une intelligence collective. *Informatique, Intelligence Artificielle, Intereditions, Paris*, (1995).

[14] J. Ferber, O. Gutknecht, Aalaadin: a meta-model for the analysis and design of organizations in multi-agent systems. *ICMAS (International Conference on Multi-Agent Systems)* (Y. Demazeau, Ed.), (1998) pp. 128–135. IEEE Press, Paris.

[15] F. Michel, J. Ferber, O. Gutknecht, Generic Simulation Tools Based on MAS Organization. *LIRMM Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier*, (2001).

[16] A. M. Uhrmacher, B. Shattenberg, Agents in Discrete Event Simulation. *in Proceedings of ESS98*, (1998).

[17] J. Himmelspach, M. Röhl, A. M. Uhrmacher, Component based modelling and simulation for valid multi-agent system simulations. *International Journal for Applied Artificial Intelligence*, **24**(5) (2010), 414–442.

[18] J. Himmelspach, JAMES II: Extending, Using, and Expeiments. In *Proceedings of the 5th Simutools Conference ICST*, (2012), pp. 208–210.

[19] D. Weyns, H. Van Dyke Panurak, F. Michel, T. Holvoet, J. Ferber, Environments for Multi-agent Systems State of the Art and Research Challenges. In *Environments for Multi-agent Systems* (D. Weyns, H. Dyke Panurak, F. Michel, Eds.), vol. 3374 of Lecture Notes in Computer Science, pp. 1–47, (2005) Berlin, Heidelberg: Springer Berlin Heidelberg.

[20] D. Weyns, M. Schumacher, A. Ricci, M. Viroli, T. Holvoet, Environments in Multi-agent Systems. *The Knowledge Engineering Review*, **20**(2) (2005).

[21] M. Wooldridge, *An introduction to Multi-agent Systems*. 2nd ed., John Wiley & Sons, Chistester, UK, 2009.

[22] A. Steiniger, F. Krüger, A. M. Uhrmacher, Modeling Agents and Their Environment. In *Multi-level-DEVS, Proceedings of the Winter Simulation Conference*, (2012).

[23] J. Davilla, M. Uzcategui, GALATEA: A Multi-agent simulation platform. In *International Conference on Modelling, Simulation and Neural Networks MSNN'2000*, (2000) Mérida, Venezuela.

[24] J. Davilla, E. Gomez, K. Lafaille, K. Tucci, M. Uzcategui, MultiAgent Distributed Simulation with GALATEA. *Proceedings of the 2005 Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'05)*, (2005).

[25] S. Mattei, P. A. Bisgambiglia, M. Delhom, E. Vittori, Towards Discrete Event Multi Agent Platform Specification, Computation Tools. *The Third International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking*, (2012), pp. 14–21.

[26] E. Ramat, P. Preux, Virtual Laboratory Environment (VLE): a software environment oriented agent and object for modeling and simulation of complex systems. *Simulation and modeling practice and theory*, **11**(1) (2003), 45–55.

[27] G. Quesnel, R. Duboz, E. Ramat, The Virtual Laboratory Environment – An operational framework for multi-modelling and analysis of complex dynamical systems. *Simulation and modeling practice and theory*, **17**(4) (2009), 641–653.

[28] A. Ilachinski, *Cellular Automata, a Discrete Universe*. World Scientific Publishing Co, 2001.

[29] G. A. Wainer, N. Giambiasi, Application of the Cell-DEVS Paradigm for Cell Spaces Modelling and Simulation. *Simulation*, **76**(1) (2001), 22–39.

[30] G. A. Wainer, P. Mosterman, *Discrete-Event Modeling and Simulation: Theory and Applications*. Taylor and Francis Group, 2011.

[31] S. H. Sarjoughian, B. P. Zeigler, The role of collaborative DEVS modeler in federation development. In *Proceedings of the 99th System Interoperability Workshop*, (1999).

[32] E. Kofman, N. Giambiasi, S. Junco, Fdevs: A general DEVS based formalism for fault modeling and simulation. *European Simulation Symposium*, vol. 1, (2000) Hamburg, pp. 77–82.

[33] D. R. Hild, Discrete Event System Specification (DEVS), Distributed Object Computing, Modeling and Simulation. PhD thesis, University of Arizona, 2000.

[34] A. Anglani, P. Caricato, A. Grieco, F. Nucci, A. Matta, G. Semeraro, T. Tolio, Evaluation of capacity expansion by means of fuzzy-devs. *14th European Simulation Multi Conference*, (2000) Gent, Belgium.
`citeseer.ist.psu.edu/499458.html`

[35] J. B. Filippi, F. Bernardi, M. Delhom, The JDEVS environmental modeling and simulation environment. *IEMSS, Integrated Assessment and Decision Support*, (2002) Lugano, Suisse, pp. 283–288.

[36] H. Vangheluwe, The Discrete EVent System specification DEVS Formalism. *Technical report*, (2001). `http://moncs.cs.mcgill.ca/`

[37] C. Jacques, G. A. Wainer, Using the cd++ DEVS tookit to develop petrinets. In *SCS, Proceedings of the SCS Conference*, (2002).

[38] A. Hamri, N. Giambiasi, C. Frydman, Min-Max DEVS modeling and simulation. *Simulation Modelling Practice and Theory (SIMPAT)*, **14**(7) (2006), 909–929.

[39] F. BARROS, Dynamic structure discrete event system specification: a new formalism for dynamic structure modelling and simulation. In *Proceedings of the Winter Simulation Conference*, (1995).

[40] A. M. UHRMARCHER, Dynamic Structures in Modeling and Simulation: A Reflective Approach. *ACM Transactions on Modeling and Computer Simulation*, **11** (2001), 206–232.

[41] L. NTAIMO, B. P. ZEIGLER, Expressing a forest cell model in parallel DEVS and timed cell-DEVS formalisms. *Proceedings of the 2004 Summer Computer Simulation Conference*, (2004).

[42] A. TROCCOLI, G. A. WAINER, Implementing parallel cell-DEVS. In *IEEE, Proceedings of the 36th Annual Simulation Symposium*, (2003).

[43] L. TOURAILLE, M. K. TRAORÉ, D. R. C. HILL, SimStudio: une Infrastructure pour la modélisation, la simulation et l'analyse de systèmes dynamiques complexes. *Research Report LIMOS/RR-10-13*, (2010).

[44] R. FREIGASSNER, H. PRAEHOFER, B. P. ZEIGLER, Systems approach to validation of simulation models. *Cybernetics and Systems*, (2000), 52–57.

[45] J. H. BYUN, C. B. CHOI, T. G. KIM, Verification of the DEVS model implementation using aspect embedded DEVS. In *Proceedings of the 2009 Spring Simulation Multi Conference*, (2009) San Diego, USA.

[46] D. C. SCHMIDT, Model-Driven Engineering Guest Editor's Introduction *IEEE Computer*, **39**(2) (2006), 25–31.

[47] Simulation Interoperability Standards Committee, IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules, *IEEE Std 1516*, (2000).

[48] J. FERBER, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co. Inc. Boston, MA, USA, 1999.

[49] E. V. KRISHNAMURTHY, V. K. MURTHY, Distributed agent paradigm for soft and hard computation. *Journal of Network and Computer Applications*, **29**(2) (2006), 124–146.

[50] N. SEDDARI, M. REDJIMI, Multi-Agent Modeling of a Complex System. In *IEEE 3rd International Conference on Information Technology and e_Services (ICITeS)*, (2013) Sousse, Tunisia.

[51] N. SEDDARI, M. REDJIMI, Operational approach to modelling and simulation of an industrial process. In *IEEE International Conference on Computer Application Technology (ICCAT)*, (2013) Sousse, Tunisia.

Received: January, 2014
Revised: July, 2014
Accepted: August, 2014

Contact addresses:

Noureddine Seddari
Department of Computer Science
Faculty of Sciences
University 20 August 1955
Algeria
e-mail: `seddarinoureddine@yahoo.fr`

Mohammed Redjimi
Department of Computer Science
Faculty of Sciences
University 20 August 1955
Algeria
e-mail: `redjimimed@yahoo.fr`

Sofiane Boukelkoul
Department of Computer Science
Faculty of Sciences
University 20 August 1955
Algeria
e-mail: `bouk.sofiane@yahoo.fr`

NOUREDDINE SEDDARI obtained his Master degree in computer science from Université 20 Aout 1955, Skikda, Algeria in 2011. He is a PhD student at the same university. His research interests include modelling and simulation of complex systems, MAS concepts and platforms and DEVS formalisms.

MOHAMMED REDJIMI is an associate professor of computer science at the Université 20 Aout 1955, Skikda, Algeria. He obtained his PhD in computer science from the Université des Sciences et des Techniques de Lille I, France in 1984. He was an associate teacher and researcher at the same university (1982 to 1986). He obtained the postdoctoral degree (*Habilitation Universitaire*) from Université Badji Mokhtar, Annaba, Algeria in 2007. He was Head of Computer Science Department and Dean of Engineering and Sciences Faculty at the Université 20 Aout 1955, Skikda, Algeria. M. Redjimi has several publications in international journals and conferences. He is reviewer in several international journals and conferences. M. Redjimi is currently Head of the team: modeling and simulation of complex systems at the Laboratoire d'Informatique et de Communication de l'Université de Skikda (LICUS) – Université 20 Aout 1955, Skikda, Algeria. His current research interests include image processing, complex systems, modelling and simulation, MAS concepts and platforms and DEVS formalisms.

SOFIANE BOUKELKOUL obtained his engineer in computer science degree from the Université 20 Aout 1955, Skikda, Algeria in 2008. He is a Master student of computer science at the same university. His research interests include modelling and simulation of complex systems, MAS concepts and platforms and DEVS formalisms.

Appendix 1: Atomic DEVS specifications of the system

S= {"active","passive"}$\times R_0^+$ and ta (phase, σ)= σ for all atomic models of the system.

APM	*GUS*
InPorts={"r1", "r2","r3","r4","r5", r6","r7","r8","r9", "r10"} X ={(r1, c_w_f), (r2,t_l), (r3, t_w_f),(r4,c_l), (r5,i_l),(r6,v_f), (r7,b_l), (r8,w_f), (r9,g_f), (r10,a_f)} OutPorts = {"r11", "r12","r13", "r14", "r15"} Y={(r11, c_w_f), (r12, w_f), (r13, o_s), (r14, v_f), (r15, i_l)} δext (phase, σ, e, X) = ("active",regulating time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ", ∞) if APM.InPorts.Values= Ø ("active", regulating time) otherwise λ (" active ") = c_w_f, w_f, v_f, i_l and o_s	*InPorts={"n1"} X = {(n1,i_l)} OutPorts ={"n2"} Y={(n2,i_l)} δext (phase, σ, e, X) = ("active", perturbation time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ",∞) if GUS.InPorts.Values= Ø ("active", perturbation time) otherwise λ (" active ") = i_l*
15LIC05	***15B01***
InPorts={"a1", "a2","a3"} X ={(a1,t_l), (a2, w_f), (a3,c_w_f)} OutPorts = {"a4"} Y={(a4, o_wf)} δext (phase, σ, e, X) = ("active", feeding time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ", ∞) if 15LIC05.InPorts.Values= Ø ("active", feeding time) otherwise λ (" active ") = o_fw	*InPorts={"c1", "c2"} X ={(c1,w_f), (c2,c_w_f) } OutPorts = {"c3"} Y={(c3, t_l)} δext (phase, σ, e, X) = ("active", t_l calculation time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ", ∞) if 15B01.InPorts.Values= Ø ("active", t_l calculation time time) otherwise λ (" active ") = t_l*
15FVC	***15LIC12***
InPorts={"b1", "b2"} X ={(b1,o_s), (b2,o_fw)} OutPorts = {"b3"} Y={(b3, c_w_f) } δext (phase, σ, e, X) = ("active", c_w_f calculation time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ",∞) if 15LIC12.InPorts.Values= Ø ("active", c_w_f calculation time) otherwise λ (" active ") = c_w_f	*InPorts={"d1", "d2","d3"} X ={(d1,t_w_f), (d2,c_l),(d3, c_w_f)} OutPorts = {"d4"} Y={(d4, o_cn)} δext (phase, σ, e, X) = ("active", condensation time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ",∞) if 15LIC12.InPorts.Values= Ø ("active", condensation time) otherwise λ (" active ") = o_cn*
15E01	***15FVA***
InPorts={"e1", "e2} X ={(e1,c_w_f), (e2,t_w_f)} OutPorts = {"e3"} Y={(e3, c_l)} δext (phase, σ, e, X) = ("active", c_l calculation time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ",∞) if 15E01.InPorts.Values= Ø ("active", c_l calculation time) otherwise λ (" active ") = c_l	*InPorts={" f1"," f2"} X ={(f1,o_s), (f2,o_cn)} OutPorts = {"f3"} Y={(f3, t_w_f)} δext (phase, σ, e, X) = ("active", t_w_fcalculation time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ",∞) if 15FVA.InPorts.Values= Ø ("active", t_w_fcalculation time) otherwise λ (" active ") = t_w_f*

15HIC01	15KT01
InPorts={" j1", " j2", " j3"} X ={(j1,a_f), (j2,i_l), (j3, g_f)} OutPorts = {" j4 ", " j5"} Y={(j4,v_f), (j5,o_cm)} δext (phase, σ, e, X) = ("active", combustion time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ",∞) if 15HIC01.InPorts.Values= Ø ("active", combustion time) otherwise λ (" active ") = v_fand o_cm	InPorts={" m1", " m2"} X ={(m1,o_s), (m2,o_cm)} OutPorts = {"m3"} Y={(m3, a_f) } δext (phase, σ, e, X) = ("active", a_f calculation time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ", ∞) if 15KT01.InPorts.Values= Ø ("active", a_f calculation time) otherwise λ (" active ") = a_f
15FVG	**15LIC01A**
InPorts={" k1", " k2"} X ={(k1,o_cm), (k2,o_s)} OutPorts = {"k3"} Y={(k3, g_f)} δext (phase, σ, e, X) = ("active", g_f calculation time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ",∞) if 15FVG.InPorts.Values= Ø ("active", g_f calculation time) otherwise λ (" active ") = g_f	InPorts={"g1", "g2","g3"} X ={(g1,w_f), (g2, v_f), (g3, b_l)} OutPorts = {" g4"} Y={(g4, o_hy)} δext (phase, σ, e, X) = ("active", hydraulic time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ", ∞)if 15LIC01A.InPorts.Values= Ø ("active", hydraulic time) otherwise λ (" active ") = o_hy
15FV03	**15FV03**
InPorts={" i1", " i2"} X ={(i1,o_s), (i2,o_hy)} OutPorts = {"i3 "} Y={(i3, w_f) } δext (phase, σ, e, X) = ("active", w_f calculation time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ",∞) if 15FV03.InPorts.Values= Ø ("active", w_f calculation time) otherwise λ (" active ") = w_f	InPorts={"h1", "h2"} X ={(h1,v_f), (h2,w_f) } OutPorts = {"h3"} Y={(h3,b_l)} δext (phase, σ, e, X) = ("active", b_l calculation time) if phase = "passive" (phase, σ-e) otherwise δint (phase, σ) = (" passive ",∞) if 15FV03.InPorts.Values= Ø ("active", b_l calculation time) otherwise λ (" active ") = b_l

Appendix 2: Coupled DEVS specifications of the system

SIMULATOR(SIM)	PERTURBATION(PER)
InPorts={"s1"} X ={(s1, i_l)} OutPorts= {"s2"} Y={(s2, v_f)} D ={FEE, PER, CON, COM, HYD} MFEE =FEDDING MPER =PERTURBAT- ION MCON =CONDENSAT-ION MCOM =COMBUSTION MHYD =HYDRAULIC EIC ={(SIM , s1),(PER, z5)} EOC ={(COM, v3), (SIM , s2)} IC={((FEE,x3),(PER,z3)),((FEE,x4),(PER,z2)), ((CON,y3),(PER,z4)),((CON,y4),(PER,z1)), ((HYD,u3),(PER,z7)),((HYD,u4),(PER,z6)), ((COM,v3),(PER,z9)),((COM,v4),(PER,z10)), ((COM,v5),(PER,z8)),((PER,z11),(CON,y2)),((PER,z12), (FEE,x1)),((PER,z13),(CON,y1)),((PER,z13), (COM,v2)),((PER,z13),(HYD,u1)),((PER,z13),(FEE,x2)), ((PER,z14), (HYD,u2)),((PER,z15),(COM,v1))}	InPorts = {"z1", "z2","z3", "z4", "z5","z6", "z7", "z8", "z9", "z10"} X ={(z1, c_l),(z2, c_w_f), (z3, t_l), (z4,t_w_f), (z5,i_l), (z6,b_l),(z7,w_f),(z8,g_f), (z9,v_f), (z10,a_f)} OutPorts={"z11", "z12", "z13", "z14", "z15"} Y={(z11, c_w_f), (z12,w_f), (z13,(o_s),(z14,v_f), (z15,i_l)} D= {GUS, APM } MGUS= DISTURBING MAPM= CONTROLLER EIC= {((PER, z1), (APM,r4)), ((PER,z2), (APM,r1)), ((PER,z3), (APM,r2)),((PER,z4), (APM,r3)),((PER,z5), (GUS,n1)),((PER,z6), (APM,r7)),((PER,z7), (APM,r8)), ((PER,z8), (APM,r9)), ((PER,z9), (APM,r6)), ((PER, z10), (APM, r10)) } EOC= {((APM, r11),(PER, z11)), ((APM, r12), (PER, z12)), ((APM, r13), (PER, z13)), ((APM, r14), (PER, z14)), ((APM, r15),(PER,z15)) } IC ={(GUS,n2), (APM,r5) }
FEDDING(FEE)	**CONDENSATION (CON)**
InPorts={"x1", "x2"} X ={(x1, i_l), (x2, o_s)} OutPorts= {"x3","x4"} Y= {(x3, t_l), (x4, c_w_f)} D= {15LIC05,15B01, 15FVC } M15LIC05=Regulator M15B01=Tank M15FVC=Water valve EIC = {((FEE , x1), (15LIC05,a2)),((FEE, x2), (15FVC, b1)), ((FEE, x1),(15B01,c1))} EOC ={((15FVC, b3), (FEE,x4)), ((15B01, c3), (FEE , x3))} IC ={((15LIC05,a4), (15FVC,b2)), ((15B01,c3), (15LIC05,a1)),((15FVC,b3),(15B01,c2)), ((15FVC,b3), (15LIC05,a3)) }	InPorts= {"y1", "y2"} X = {(y1, o_s), (y2, c_w_f)} OutPorts= {"y3", "y4"} Y= {(y3, t_w_f), (y4, c_l)} D={15LIC12,15E01,15FVA } M15LIC12=Regulator M15E01= Condenser M15FVA = Water valve EIC = {((CON, y2), (15LIC12, d3)), ((CON, y2), (15E01, e1)), ((CON, y1), (15FVA, f1))} EOC = {((15E01, e3), (CON, y4)), ((15FVA, f3), (CON, y3))} IC = {((15LIC12, d4), (15FVA, f2)), ((15FVA, f3), (15LIC12, d1)), ((15FVA, f3), (15E01, e2)), ((15E01, e3), (15LIC12, d2))}
COMBUSTION(COM)	**HYDRAULIC(HYD)**
InPorts= {"v1", "v2"} X ={(v1, i_l), (v2, o_s)} OutPorts={"v3", "v4","v5"} Y={(v3, v_f), (v4,a_f), (v5, g_f)} D={15HIC01,15KT01,15FVG } M15HIC01=Regulator M15KT01=Turbo-fan M15FVG =Gas valve EIC = {((COM , v1), (15HIC01, j2), ((COM, v2),(15KT01,m1)), ((COM, v2),(15FVG,k2))} EOC ={((15HIC01, j4), (COM , v3)), ((15KT01,m3), (COM , v4)), ((15FVG, k3), (COM , v5))} IC ={((15HIC01, j5), (15KT01,m2)), ((15HIC01, j5), (15FVG,k1)),((15KT01,m3),(15HIC01,j1)), ((15FVG,k3), (15HIC01,j3)) }	InPorts= {"u1", "u2"} X ={(u1, o_s), (u2, v_f)} OutPorts={"u3", "u4"} Y={(u3, w_f), (u4, b_l)} D={15LIC01A,15FV03, 15H01 } M15LIC01A =Regulator M15FV03=Water valve M15H01 =Balloon EIC ={((HYD , u2), (15LIC01A, g3)), ((HYD, u2),(15H01,h1)),((HYD, u1),(15FV03,i1))} EOC = {((15FV03, i3), (HYD, u3)), ((15H01, h3), (HYD , u4))} IC ={((15LIC01A, g4), (15FV03,i2)),((15FV03,i3), (15LIC01A,g1)),((15FV03,i3), (15H01,h2)),((15H01,h3), (15LIC01A,g3))}