International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 2, pp: 42-57 www.ijaem.net ISSN: 2395-5252

Implementing an Agent-Based Automatic Code Generator for Multi-Agent Based Process Control System Using Java Programming

Uju Mokwe V.[1], and Nwokolo Uchechukwu E.[2]
[1] *Department of Electronic and Computer Engineering, Nnamdi Azikiwe University Awka, Anambra State, Nigeria*
[2]*Department of Electrical/Electronic Engineering Technology, Auchi polytechnic Auchi, Edo State Nigeria*

Date of Submission: 07-07-2020 Date of Acceptance: 21-07-2020

ABSTRACT: Five agent classes are recommended for multi agent based design, namely classes 0 through 3 and a process agent control class, for use in the implementation of any process control which can be represented as an Algorithm State Machine (ASM) chart for control systems. The methodologies adopted are waterfall model and the multi agent software engineering methodology. The automatic code generator is developed using Java programming language. A typical control system was used to show how the automatic code generator works. The ASM chart representing the control system is converted into State Transition Table (STT). The STT is converted into a completely expanded STT. The state agents on the automatic code generator relevant to the present ASM chart are initialized with output code(s) taken from the fully expanded STT derived from the ASM chart. The generator will generate the source code when the generate source code button is clicked. Then the source code realized (in C language) was compiled using C compiler and a hex code was gotten. A prototype of the control system specified in the ASM chart used in this work was designed using simulation software named Proteus. The prototype comprises the Passive Infra Red (PIR) sensor, crystal oscillator, Peripheral Interface Controller (PIC) microcontroller (Pic16F877A), Light Emitting Diode (LED) (representing the light) and motor (representing the Air conditioner (AC)). The hex code is fitted into the PIC microcontroller. When the simulation is run, the PIR sensor accepts two inputs, 0 and 1. If the input is 0, the motor and LED will be off, but if it is 1, the LED and motor will be on. This shows that the hex code of the source code generated by the automatic code generator is correct.

KEYWORDS: Automatic programming, Multi-agent system, Code generator, Algorithmic State Machine, State transition table, PIC microcontroller.

I. INTRODUCTION

The concept automatic code generation (ACG) envelopes a number of different techniques intended at simplifying the task of writing a code. Apart from specific implementation details these techniques differ in the level of abstraction exposed to the developer: a very low level of abstraction is given by template-based techniques such as code completion or code insertion. These allow for the generation of code structures with a low complexity (e.g., getters/setters), which are inserted into the code by the user on an explicit (calling an editor function) or implicit (the editor recognizes the beginning of a construct and completes it) basis. This is a very general approach that can be applied in every programming language and in any kind of desired application. Code transformation represents a higher level of abstraction, where a piece of code is translated from a source language into a target language [1]. Code generators exist for various types of applications in computer science, for example, parser generators, database generators, or unified modeling language (UML) tools, rapidly generating production code and saving development costs. Apart from saving time by generating code which would otherwise have to be implemented manually, correct generators deliver correct code; additionally, all developmental iterations (with alterations to the specification) can be handled in the source language, again saving time. While these principles and methods are largely applied in the area of software development, hardware developers are supported by very few specific tools like Internet Protocol (IP)-Core Generators or C-to-hardware compilers, covering only a very small area of what could be done with ACGs (Automatic Code Generators). Each of these tools utilizes code generators for some hardware description language in their specific area, however, this functionality is not exposed to a developer wishing to implement own code generator [1]

In an earlier study by [2] identified five classes of intelligent agents namely one class of process control agent and four classes of state control agents as being sufficient for use in the implementation of any process control system which can be represented as an Algorithmic State Machine (ASM) chart. [2] also stated that these five intelligent agents form the basis for automated code generation for process control because their codes are object-oriented and reusable and when both the process control software and the process monitoring software can be automatically generated, the platform that offers this facility becomes unique to any process control system developer interested in automatic code generation. Based on these developments, this work used agent-based approach to develop and implement an automatic code generator that generates appropriate software code for any agent-based control system specified in an Algorithm State Machine chart.

The role and responsibility of agent-based control systems is ever increasing. Associated with this increase, is the need for a robust and reusable code which would automate and reduce software design for agent based control systems, presents a big challenge to the software developing industry. To resolve this difficulty, the need to create a software that can easily automate the generation of this reusable codes is significant. The aim of this work is to implement an Agent-Based Automatic Code Generator for Multi-Agent Based Process Control System Using Java Programming.

II. ALGORITHM STATE MACHINES (ASM) CHART FOR CONTROL SYSTEMS

Algorithm State Machines (ASM) is an algorithm that consists of a few steps, which is used to simplify a sequential digital system. An ASM chart resembles a conventional flow chart but the difference is, a conventional flow chart does not have timing relationships but the ASM takes timing relationship into account. An ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the other. It is employed to design a sequential circuit having a large number of external inputs because with a large number of external inputs it becomes very difficult to use state tables for designing the circuit. ASM Chart Notations: The different blocks used in the ASM chart are:

- The state box
- The decision box
- The conditional box [3]

2.1 Reusable Codes for Agent Based Control Systems

Agent technology has been the subject of extensive discussion and investigation within the scientific community for several years, but it is perhaps only recently that it has seen any significant degree of exploitation in commercial applications. Multi-agent systems are being used in an increasingly wide variety of applications, ranging from comparatively small systems for personal assistance to open, complex, mission-critical systems for industrial applications [4]. Examples of industrial domains where multi-agent systems have been fruitfully employed include process control, system diagnostics, manufacturing, transportation logistics and network management. When adopting an agent-oriented approach to solving a problem, there are a number of domain independent issues that must always be solved, such as how to allow agents to communicate. Rather than expecting developers to develop this core infrastructure themselves, it is convenient to build multi-agent systems on top of an agent-oriented middleware that provides the domain-independent infrastructure, allowing the developers to focus on the production of the key business logic [4]. The framework designed in this work facilitates the development of complete agent-based applications by means of a run-time environment implementing the life-cycle support features required by agents, the core logic of agents themselves, and a rich suite of graphical tools. As it is written completely in Java, it benefits from the huge set of language features and third-party libraries on offer, and thus offers a rich set of programming abstractions allowing developers to construct multi-agent systems with relatively minimal expertise in agent theory [4].

2.2 What Is An Agent?

An agent is essentially a special software component that has autonomy that provides an interoperable interface to an arbitrary system and/or behaves like a human agent, working for some clients in pursuit of its own agenda. Even if an agent system can be based on a solitary agent working within an environment and if necessary interacting with its users, usually they consist of multiple agents [4]. These multi-agent systems (MAS) can model complex systems and introduce the possibility of agents having common or conflicting goals. These agents may interact with each other both indirectly (by acting on the environment) or directly (via communication and negotiation). Agents may decide to cooperate for mutual benefit or may compete to serve their own interests.

An agent is autonomous, because it operates without the direct intervention of humans or others and has control over its actions and internal state. An agent is social, because it cooperates with humans or other agents in order to achieve its tasks. An agent is reactive, because it perceives its environment and responds in a timely fashion to changes that occur in the environment.

2.2.1 Agent Types and Classification

According to [5], five agent classes are recommended for multi agent based design, namely classes 0 through 3 and a process agent control class, for use in the implementation of any process control which can be represented as an Algorithm State Machine (ASM) chart. The agents are discussed here.

a. **Agent Class 0**: A class 0 agent makes a transition from its present state to another state without considering any qualifiers as shown in figure 5. This happens typically where two state boxes in an ASM chart are in sequence without any qualifier in between them. As shown in fig. 1, if the control system is in the state ST_X it must unconditionally transit to state ST_Y when a clock pulse occurs.

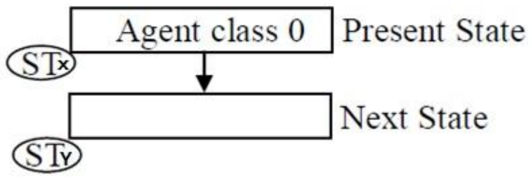


Fig. 1: Agent Class 0 [2]

b. **Agent Class 1:** State agent class 1 is handles transition from one state (ST_X say) to one of two alternative states (ST_Y and ST_Z) depending on the value of the qualifier, for instance Q. This is shown in Fig. 2. If the control system is in ST_X and the qualifier Q=0, control is transferred to the agent state for ST_Y. However, if qualifier Q=1, control is transferred to the agent for state ST_Z.

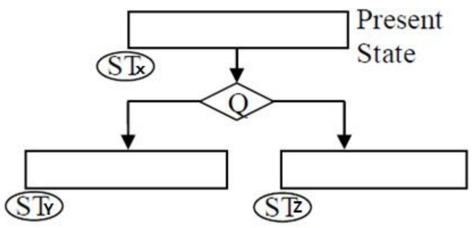


Fig. 2: Agent Class 1 [2]

c. **Agent Class 2:** The State 2 agent has two qualifiers in cascade and the agent in the present state has four alternative link paths that determine the next state agent to handover to. Each of the link paths may be selected depending on the values of the qualifiers, taken to be Q1 or Q2. For example, if the present state is named ST_0, the state agent for ST_0 would hand over to state ST_1 if Q1=0. It would still hand over to the state agent for ST_1 whether qualifier Q2 is 0 or 1. Therefore qualifier Q2 is said to be a don't-care. If however, Q1=1 then control would be transferred to state agent ST_2 if Q2=0 and to state agent ST_3 if Q2=1.

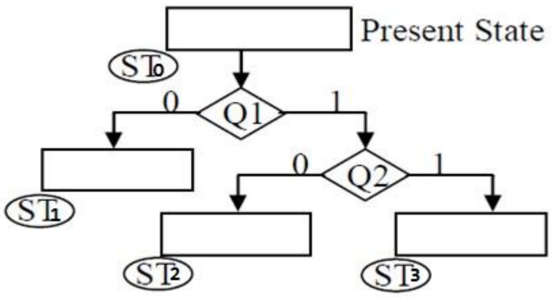


Fig. 3: Agent Class 2 [2]

d. **Agent Class 3:** Agent class 3 is required for the condition where the present state is separated from the alternative states by up to 3 qualifiers in cascade (Q1, Q2, Q3) Fig. 4. The transitions from the present state ST_0 to each of the alternative states is determined by the three qualifiers Q1, Q2, Q3. Note that when Q1 is a zero, transition must be to agent for state ST_1 no matter what Q2 and Q3 are. Similarly, when qualifier Q1=1 and Q2=0, transition must be to state ST_2 irrespective of the value of ST_3. The reason the fully expanded table is used is to facilitate the use of a look up table by the state agents which allows the indexing of the tables using qualifiers [2]. These transitions for agent class 0-3 are shown in table 1 to table 2 respectively.

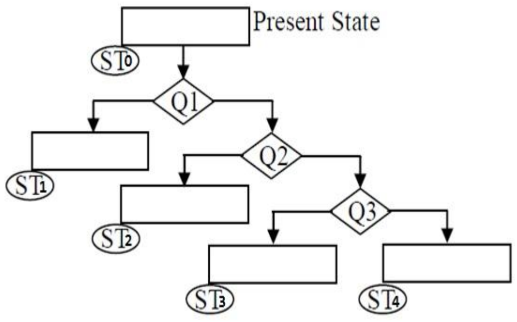


Fig. 4: Agent Class 3 [2]

Table 1: Class 0 agent transition [2]

Present State	Next State
STx	Sty

Table 2: Class 1 agent transition [2]

Present State is STx	
Qualifier	Next State
0	STy
1	STz

Table 3: Class 2 agent transition [2]

Q1	Q2	State Agent that takes Over
0	0	State agent ST_1
0	1	State agent ST_1
1	0	State agent ST_2
1	1	State agent ST_3

Table 4: Class 3 agent transition [2]

Q1	Q2	Q3	Next State Alternative
0	0	0	ST_1
0	0	1	ST_1
0	1	0	ST_1
0	1	1	ST_1
1	0	0	ST_2
1	0	1	ST_2
1	1	0	ST_3
1	1	1	ST_4

When looking at an ASM chart, the state agents are made to be an instantiation of one of the agent classes depending on the number of qualifiers n, between it and alternative transition link paths. When n= 0, $2^n = 2^0 = 1$, only one next state exists. When n=1, $2^n = 2^1 = 2$, two possible next states exist. The logic value of the qualifier determines the output. The next state value is concatenated with the state output bits to constitute the HEX output.

For example, if two qualifiers are represented as q1 q2, then, when q1q2 = 00, V_0(say) is output, when q1 q2 = 01, V_1 (say) is output, when q1 q2= 10, V_2(say) is output and q1 q2 =11 causes V_3(say) to be output (Inyiama, Obiora-Dimson and Okezie, 2015). A similar selection and output process is followed when there are 3 qualifiers q1 q2 q3 in between one state and another, in the case of type 3 state agents. Thus q1 q2 q3 would have $2^3 = 8$ possible binary combinations namely 000, 001, 010, 011, 100, 101, 110 and 111 and would lead to the output of V_0 or V_1 or V_2 or V_3 up to V_7 depending on the subscripts which Vs corresponds to [2].

2.2.2 Process Agent

Agents are independent by nature as discussed earlier, the number of agents necessary to implement an ASM chart is equal to the number of states in that ASM chart. To allow that number of agents to have full autonomy may cause a loss of control in the system especially if something goes wrong with one or more of the agents, hence the need for co-ordination[2] [6]. The concept of process agents is used to solve this problem. The process agent co-ordinates the activities of all the state agents in the same ASM chart as depicted in fig. 5.

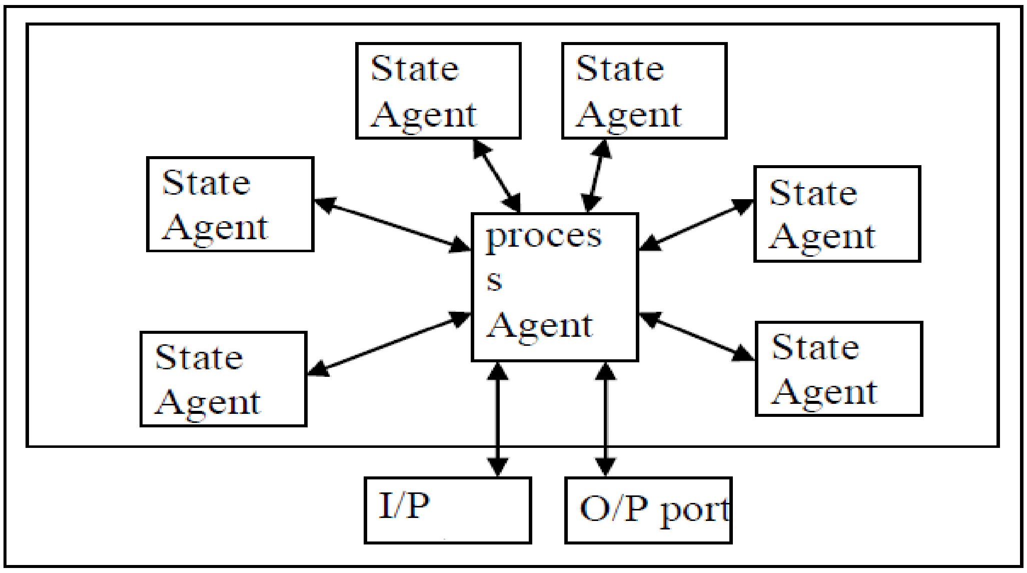


Fig. 5: Process Agent [2]

For each input data into the system, the process agent performs the following:

i. It detects the inputs
ii. Extracts the present state code
iii. Activates state agent that corresponds to the state code
iv. It supplies the values of the qualifier
v. The state agent that is activated produces an output pattern which is a combination of the next state code and the outputs.
vi. After executing the above operation, the agent is deactivated until it is called up again

The above process is repeated for any given input and this happens in a coordinated manner. Only the process agent has access to the output port of the microcontroller, thus it is the responsibility of the process agent to activate the needed state agents one at a time. [2]

III. MULTI-AGENT SYSTEM

A multi-agent system (M.A.S.) is a computerized system composed of multiple interacting intelligent agents within an environment. Multi-agent systems can be used to solve problems that are difficult or impossible for an individual agent or a monolithic system to solve. Intelligence may include some methodic, functional, procedural approach, algorithmic search or reinforcement learning. Although there is considerable overlap, a multi-agent system is not always the same as an agent-based model (ABM). The goal of an ABM is to search for explanatory insight into the collective behavior of agents (which don't necessarily need to be "intelligent") obeying simple rules, typically in natural systems, rather than in solving specific practical or engineering problems. The terminology of ABM tends to be used more often in the sciences and MAS in engineering and technology [7]. Topics where multi-agent systems research may deliver an appropriate approach include online trading, disaster response, and modeling social structures.

3.1 Concept of Multi-Agent systems

Multi-agent systems consist of agents and their environment. Typically multi-agent systems research refers to software agents. However, the agents in a multi-agent system could equally well be robots [8], humans or human teams. A multi-agent system may contain combined human-agent teams.

Agents can be divided into different types ranging from simple to complex. Some categories suggested to define these types include:

- Passive agents [9] or agent without goals (like obstacle, apple or key in any simple simulation)
- Active agents [9] with simple goals (like birds in flocking, or wolf–sheep in prey-predator model)
- Cognitive agents (which contain complex calculations)

Agent environments can be divided into:

- Virtual Environment
- Discrete Environment
- Continuous Environment

Agent environments can also be organized according to various properties like: accessibility (depending on if it is possible to gather complete information about the environment), determinism (if an action performed in the environment causes a definite effect), dynamics (how many entities influence the environment in the moment), discreteness (whether the number of possible actions in the environment is finite), episodicity (whether agent actions in certain time periods influence other periods), and dimensionality (whether spatial characteristics are important factors of the environment and the agent considers space in its decision making) [10]. Agent actions in the environment are typically mediated via an appropriate middleware. This middleware offers a first-class design abstraction for multi-agent systems, providing means to govern resource access and agent coordination [11].

IV. DESIGN METHODOLOGY

The water fall model was used to guide the development of the multi agent based system. Since the software designed in this work is for multi-agent based system, the methodology used is the multi-agent software engineering methodology. This methodology is made for agent design and consists of both the analysis phase and design phase [12]. The analysis phase captures user requirements/roles and presents the sequence of events with charts such as the Algorithm State Machine (ASM) chart. Once this is accomplished, the design phase transforms the defined roles into agent types and implements the complete system configuration.

4.1 System Design

Multi-agent systems can be realized using any kind of programming language. In particular, object-oriented languages are considered a suitable means because the concept of *agent* is not too distant from the concept of *object* [4]. In fact, agents share many properties with objects such as encapsulation, inheritance and message passing. However, agents also differ from objects in several key ways; they are autonomous (i.e. they decide for themselves whether or not to perform an action on request from another agent); they are capable of a flexible behavior; and each agent of a system has its own thread of control. Agent-oriented programming languages are a new class of programming languages that focus on taking into account the main characteristics of multi-agent systems [4]. Minimally, an agent-oriented programming language must include some structure corresponding to an agent, but many also provide mechanisms for supporting additional attributes of agency such as beliefs, goals, plans, roles and norms.

Object orientation is thus very useful in that it leads to a high number of software codes that can be re-used in different unrelated projects featuring agent-based design. One of the key components of multi-agent systems is communication. In fact, agents need to be able to communicate with users, with system resources, and with each other if they need to cooperate, collaborate, and negotiate and so on. In particular, agents interact with each other by using some special communication languages, called agent communication languages. In this work Java programming language is used.

4.2 Software Architecture

The diagram in Fig. 6 shows the main architectural elements of the platform. The platform is composed of agent containers that can be distributed over the network. Agents live in containers which are the Java processes that provide the run-time and all the services needed for hosting and executing agents [13]. There is a special container, called the main container (i.e the process agent container), which represents the bootstrap point of a platform: it is the first container to be launched and all other containers must join to a main container by registering with it.

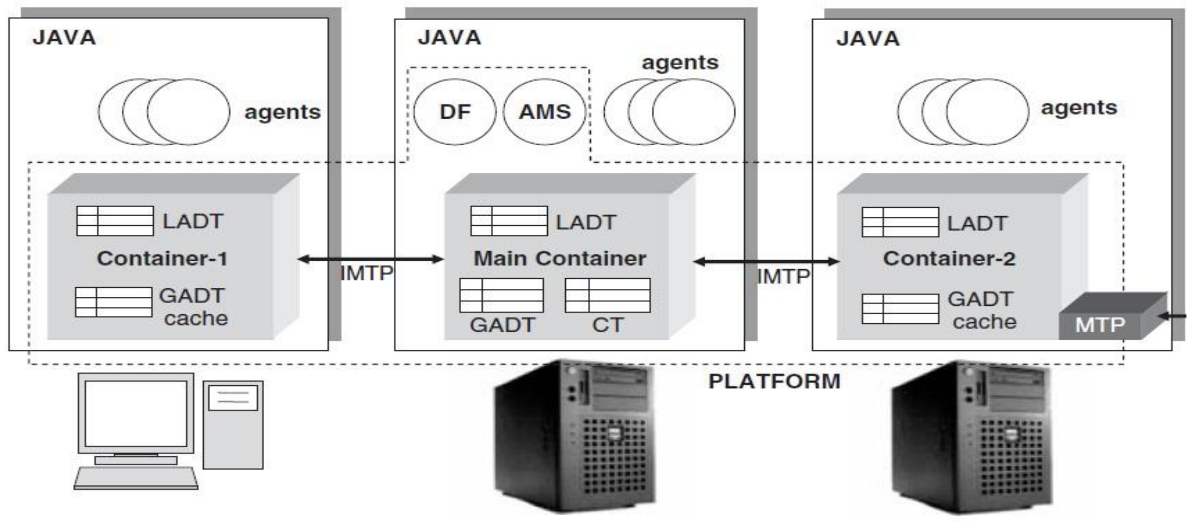


Fig. 6: Architectural elements of the multi agent based platform [4]

The programmer identifies containers by simply using a logical name; by default the main container is named 'Main Container' while the others are named 'Container-1', 'Container-2', etc. Command-line options are available to override default names. As a bootstrap point, the main container has the following special responsibilities [14]:

- Managing the container table (CT), which is the registry of the object references and transport addresses of all container nodes composing the platform
- Managing the Global Agent Descriptor Table (GADT), which is the registry of all agents present in the platform, including their current status and location;
- Hosting the Agent Management System (AMS) and the Directory Facilitator (DF), the two special agents that provide the agent management and white page service, and the default yellow page service of the platform, respectively.

When the main-container is launched, two special agents are automatically instantiated and started by the software, whose roles are defined by the Agent Management System:

1. The Agent Management System (AMS) is the agent that supervises the entire platform. It is the contact point for all agents that need to interact in order to access the white pages of the platform as well as to manage their life cycle. Every agent is required to register with the AMS (automatically carried out by agent start-up) in order to obtain a valid Agent Identity.

2. The Directory Facilitator (DF) is the agent that implements the yellow pages service, used by any agent wishing to register its services or search for other available services. The DF also accepts subscriptions from agents that wish to be notified whenever a service registration or modification is made that match some specified criteria. Multiple DFs can be started concurrently in order to distribute the yellow pages service across several domains. These DFs can be federated, if required, by establishing cross-registrations with one another which allow the propagation of agent requests across the entire federation.

The agent addresses are transport addresses inherited by the platform, where each platform address corresponds to an MTP (Message Transport Protocol) end point where compliant messages can be sent and received.

The IMTP (Internal Message Transport Protocol) is exclusively used for exchanging messages between agents living in different containers of the same platform. It is considerably different from inter-platform MTPs.

The overall design process involved in the automatic code generator is as shown below:

- Adapt the State Transition Table (STT) to Fully Expanded State Transition Table
- Carryout assignment of State and Process agent
- Develop control logic for agents initialization
- Develop reusable codes for multi agent control
- Output code for compilation to be fitted into microcontroller

4.3 Developing Algorithm

The first step in program design is implies listing the steps involved in developing the software from beginning to the end. In this work, the algorithm is as stated below:

- ➢ Beginning of program
- ➢ Definition of process agent and state agent classes and methods
- ➢ Initialization of process agent and state agents
- ➢ Activation of process control agent
- ➢ Process control agent reads process control inputs
- ➢ Process control agents activates the state agents specified in the inputs
- ➢ Active state agent releases the next output pattern to process agent
- ➢ End

V. RESULTS AND DISCUSSION

5.1 Implementing the system

With automatic code generation, a control system can be automated to perform its function by simply applying the codes that shall be developed here to this system. By supplying the relevant input codes that would initialize this automatic code generator, the software code with the information to handle the function is automatically generated and when executed, will make the system function in its capacity. This automatic code generator is aimed at reducing software design effort from scratch when the need to design a new control system arises.

Before the automatic code generator software is used, the Engineer automating the control system is required to have an ASM chart. This ASM chart is converted to State Transition Table (STT). The STT is converted to Fully Expanded State Transition Table (FESTT). Then it is from this FESTT that the state agents are gotten, and these are the inputs to the automatic code generator. An ASM chart of a system that lights up a hotel room and turns the Air conditioner (AC) on when there is an occupant and turns off the light and Air conditioner when there is no occupant is depicted in fig. 2

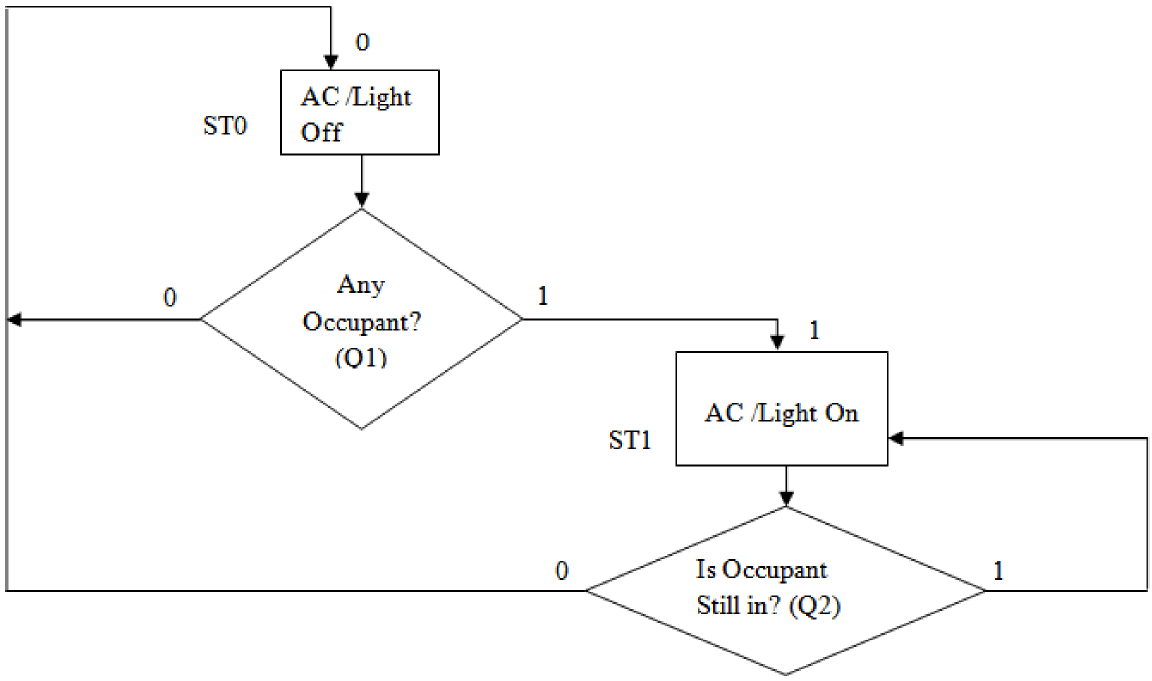


Fig. 7: Diagram Depicting the ASM Chart

The working standard of the ASM chart is represented in fig. 7:

i. At state ST0, Air conditioner (AC) and light are off, the control system checks to see if there is any occupant (Q1) in the room. If there is no occupant in the room, it goes back to ST0. If there is an occupant, it moves to state ST1.

ii. At state ST1, Air conditioner (AC) and light are on, the control system checks to see if the occupant in the room is still there (Q2). If yes it goes back to state ST1, if no it goes to state ST0 and the entire cycle is repeated.

Table 5: The State Transition Table (STT)

Link path	Present state		Qualifiers		Next state		State Output	
	Name	Code	Q1	Q2	Name	Code	AC	Light
L1	ST0	0	0	-	ST0	0	0	0
L2	ST0	0	1	-	ST1	1	0	0
L3	ST1	1	-	1	ST1	1	1	1
L4	ST1	1	-	0	ST0	0	1	1

Table 6: Fully Expanded State Transition Table

Link Path	Present State		Qualifiers		Next State		State Output		Hex Output	State agent
	Name	Code	Q1	Q2	Name	Code	AC	Light		
L1	ST0	0	0	0	ST0	0	0	0	0	State agent Zero
L1	ST0	0	0	1	ST0	0	0	0	0	
L2	ST0	0	1	0	ST1	1	0	0	4	
L2	ST0	0	1	1	ST1	1	0	0	4	
L3	ST1	1	0	1	ST1	1	1	1	7	State agent One
L3	ST1	1	1	1	ST1	1	1	1	7	
L4	ST1	1	0	0	ST0	0	1	1	3	
L4	ST1	1	1	0	ST0	0	1	1	3	

From table 6, two state agents are derived; state agent zero and state agent one. These are the inputs to the automatic code generator.
The automatic code generator software is developed and executed. The screen snap shots of different stages of the execution were shown and described in figures 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12. These snap shots showed how the automatic code generator generated the source code that will automate the control system specified in the ASM chart in fig. 7.

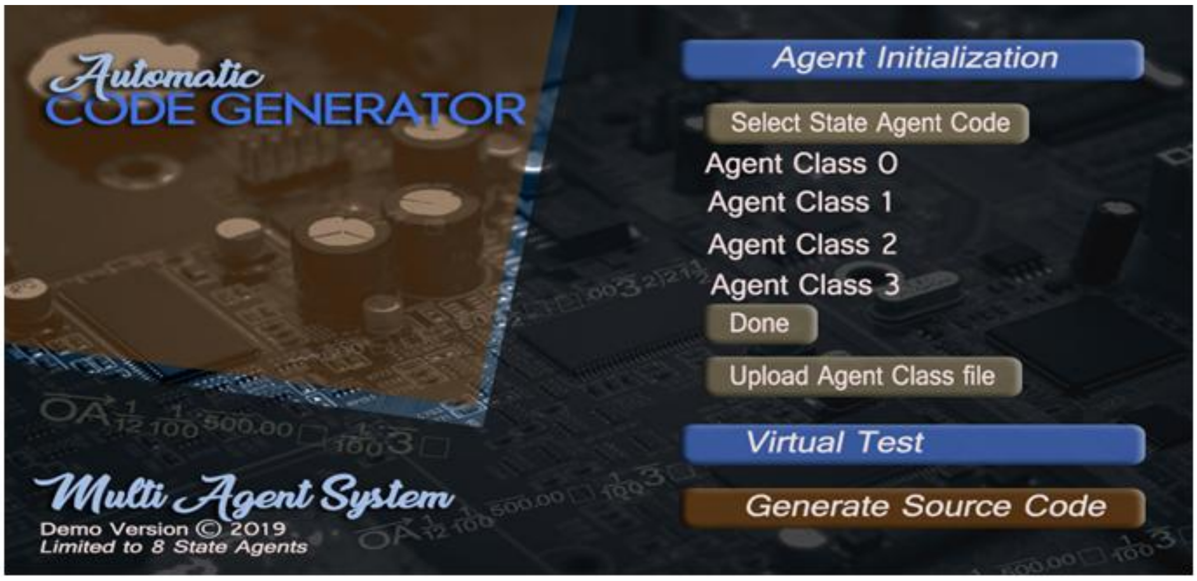


Fig. 8: User Interface of the Automatic Code Generator Software

From the fig. 8, the select state Agent code button enables the initialization of the state Agents from state agent 0 to state agent 7. This software is limited to Multi Agent Systems with only 8 state agents. For each state agent, the next state code, state output and conditional output (if any) is inputted from the left, in fig. 6. An alternative to this is to upload a text file containing the concatenation of the state agent, the next state code, state output and conditional output (if any) separated by commas. The upload Agent class file button is used for this purpose. The software incorporates a virtual test button to allow the user to view the generated fully expanded STT. The values for a particular link path can also be inputted and tested to see if the output generated, matches with what is contained in the fully expanded STT. The generate source code button enables the generation of the source code which when compiled would be burnt into a microcontroller for the execution of the programme instructions.

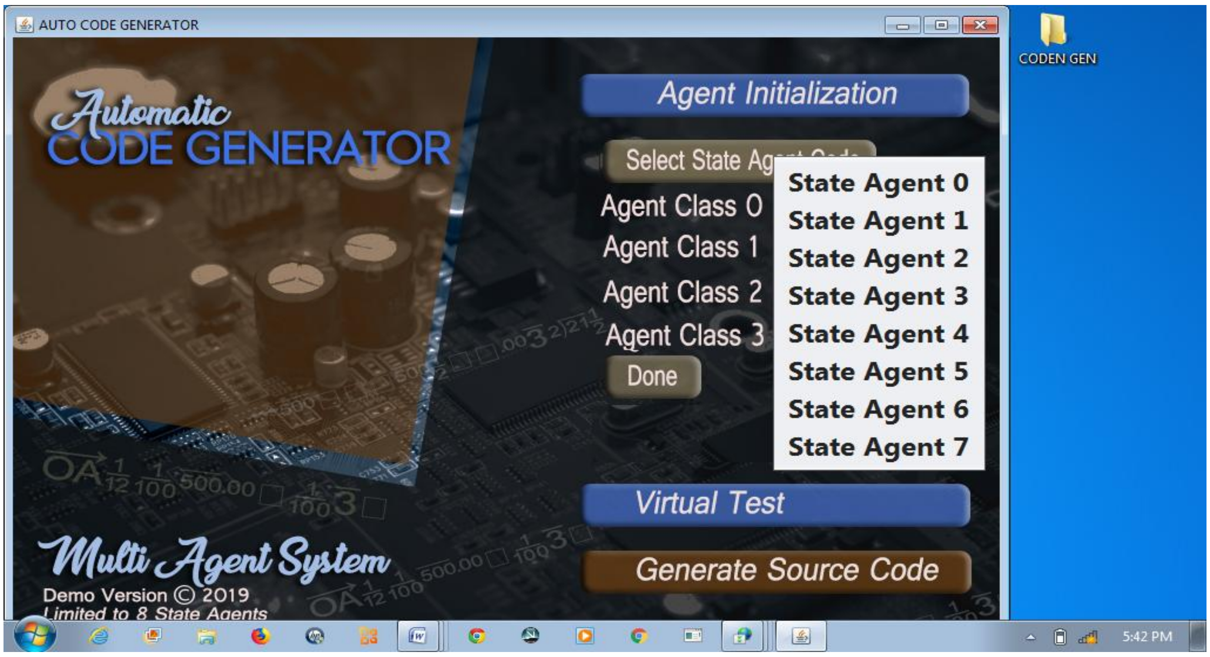


Fig. 9: Screen showing when Select Agent Code Button is Clicked

When the Select State Agent Code button is right clicked as depicted in fig. 9, the list of the state agents are displayed. From table 2, two state agents are derived; state agent zero and state agent one. To fill in the state agents, the one to be filled is highlighted and clicked. In fig. 10, the State Agent Zero is highlighted and clicked. In fig. 11, the values of state agent zero are inputted. The information filled in is gotten from the FESTT in table 2. These are 000, 000, 100 and 100. The allocation for these inputs has eight digits maximum, but our inputs have three digits. The inputs are concatenation of the next state and the state output for this particular ASM chart in fig. 12. So in filling in the input, the first five digits are filled with zeroes, and then followed by the three digits of the inputs. In fig. 12, the state agent one is highlighted and clicked. In fig. 13, the values are inputted. It is filled in with these information, 111, 111, 011 and 011 as done for state agent zero

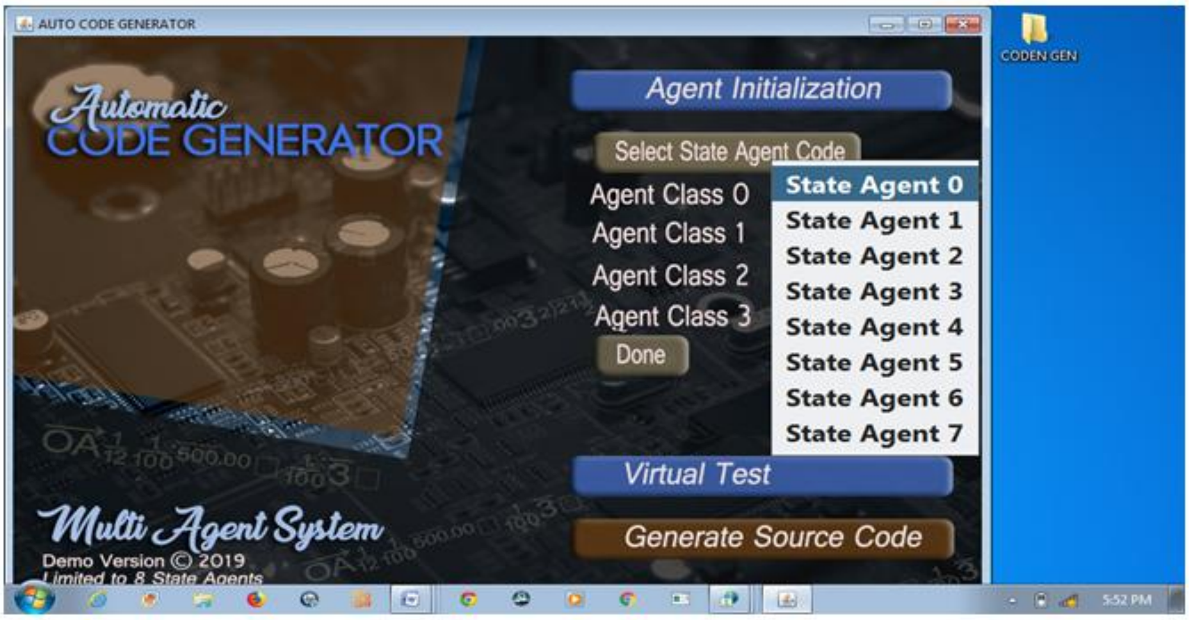
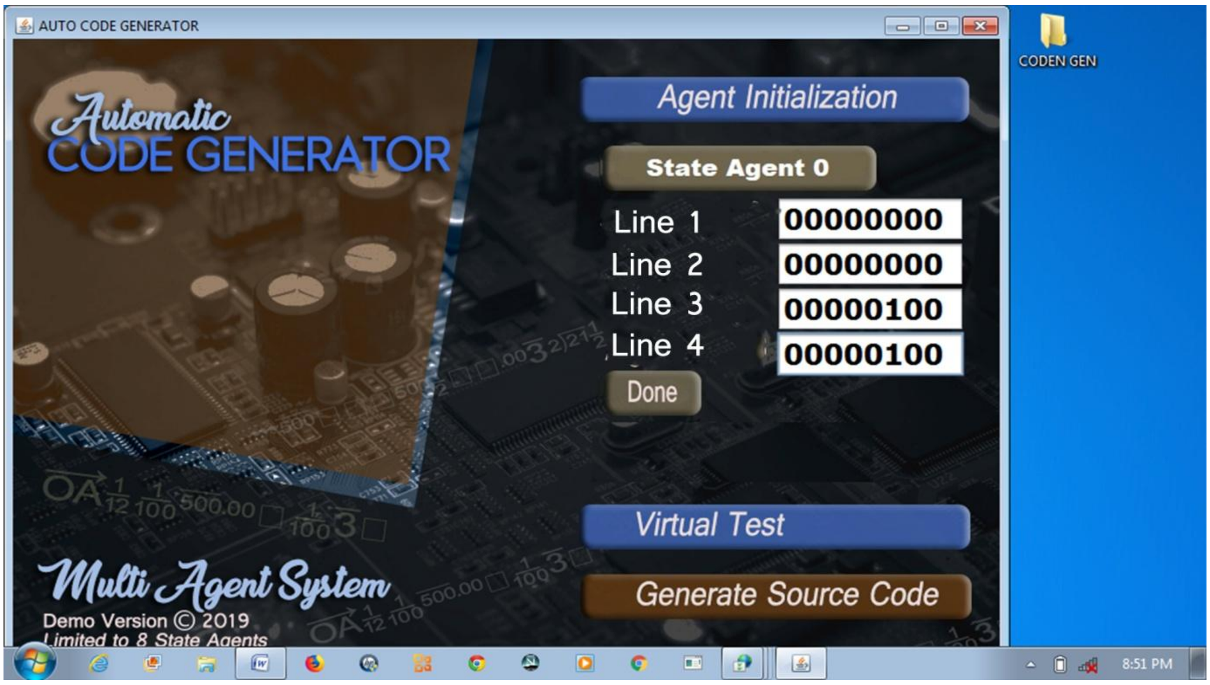


Fig. 10: Screen showing when State Agent Zero is highlighted.



Line 1	00000000
Line 2	00000000
Line 3	00000100
Line 4	00000100

Fig. 11: Screen showing when the values of State Agent Zero are inputted.

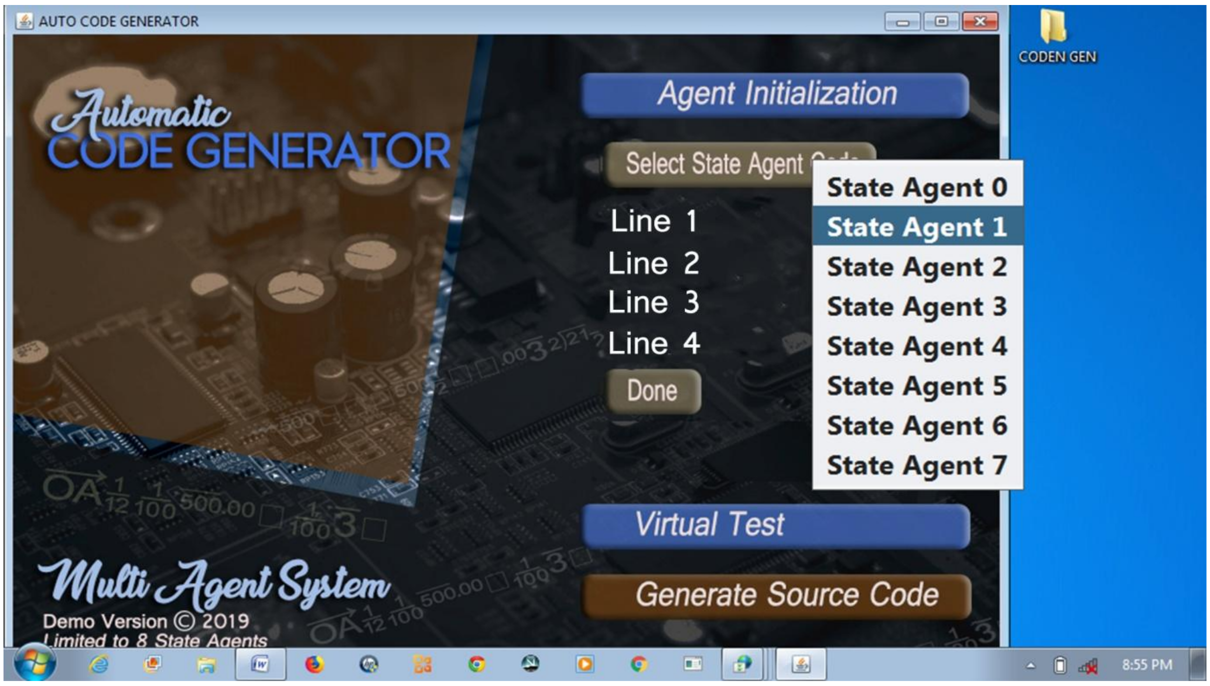
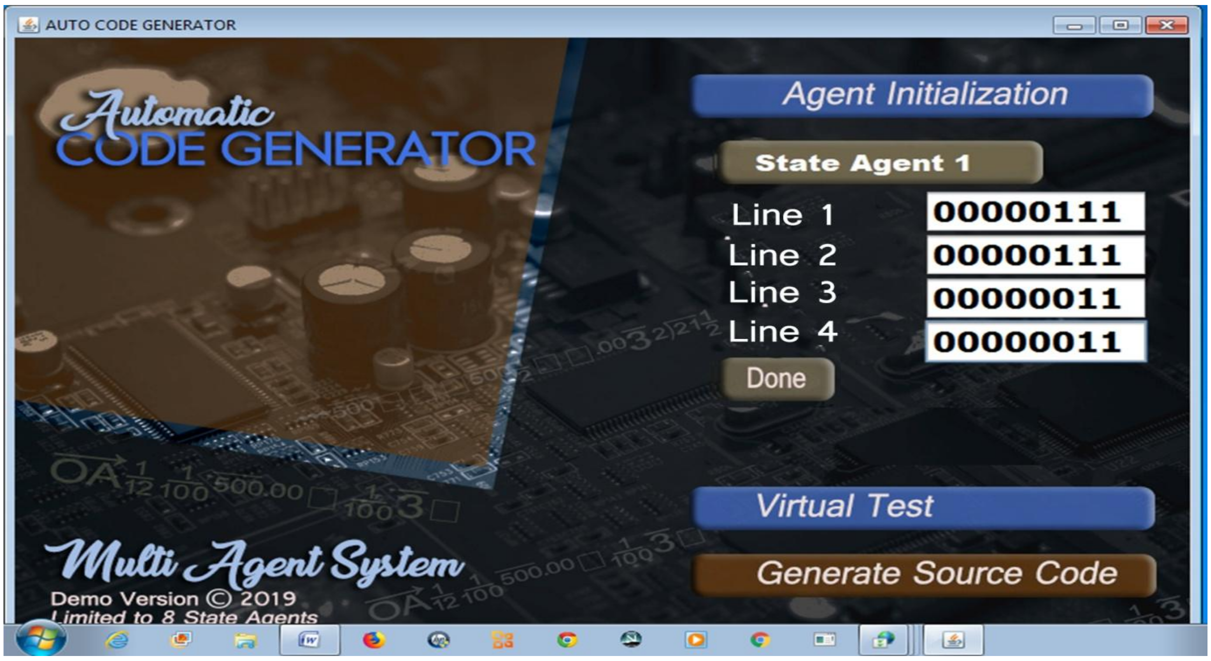


Fig. 12: Screen showing when State Agent One is highlighted.



Line 1	00000111
Line 2	00000111
Line 3	00000011
Line 4	00000011

Done

Fig. 13: Screen showing when the values of State Agent One are inputted.

Then after the filling in of the inputs, the done button is clicked and information showing success is displayed on the screen. This is shown in fig. 14.

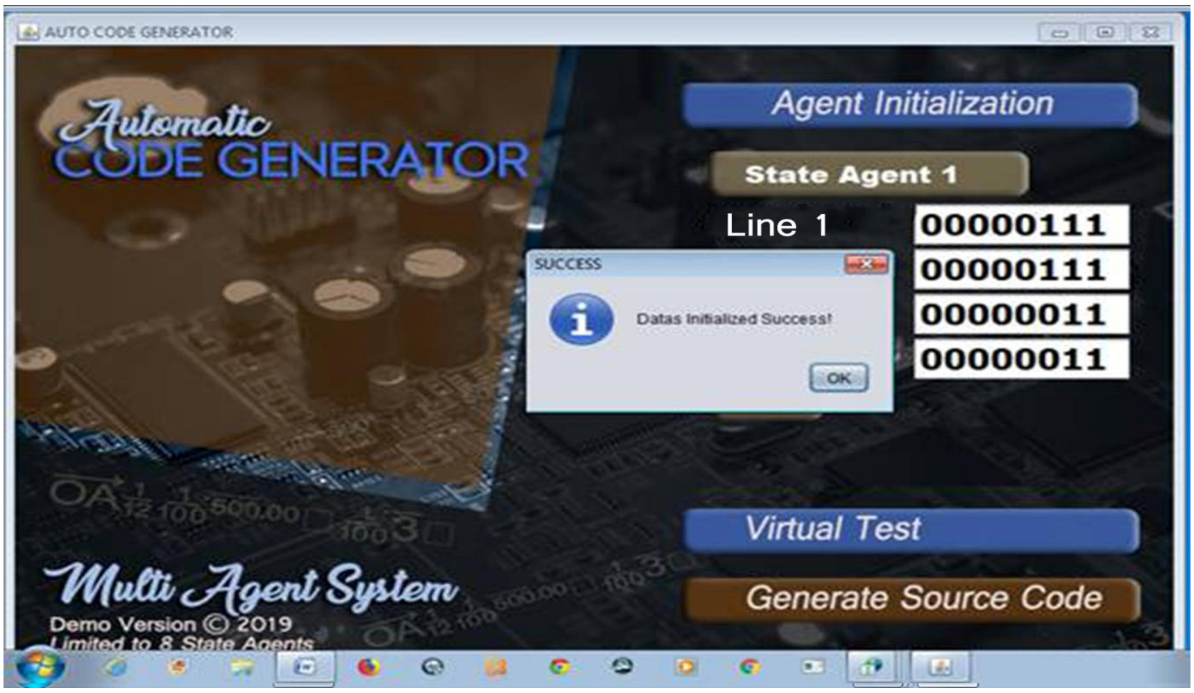
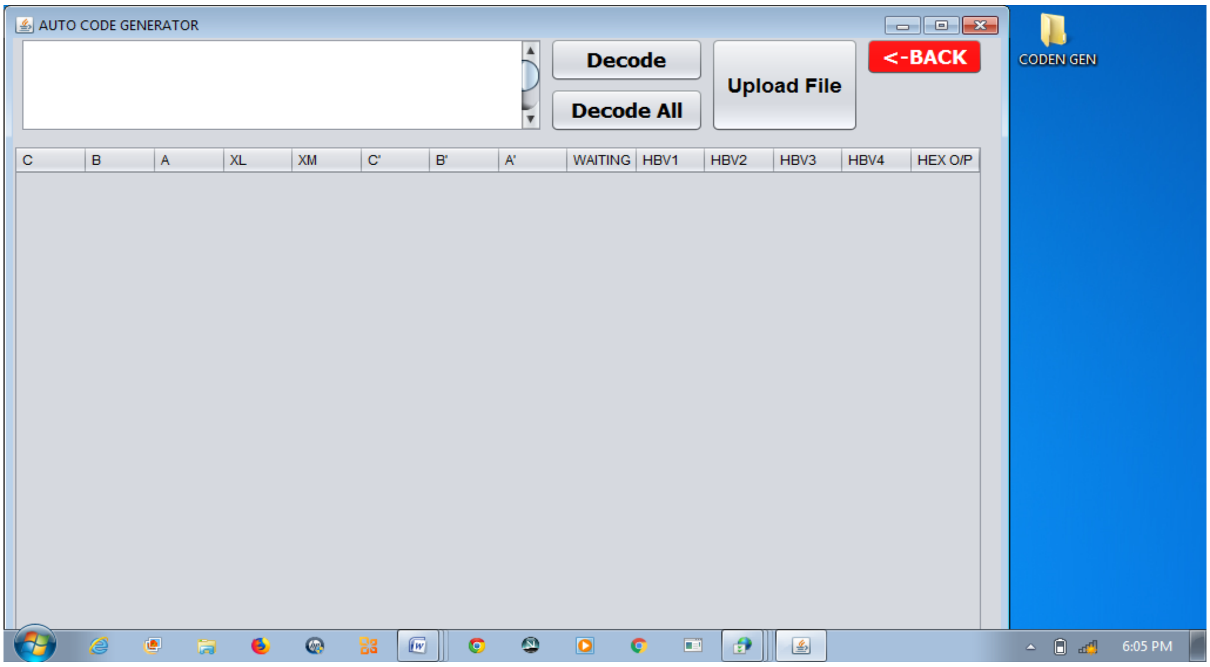


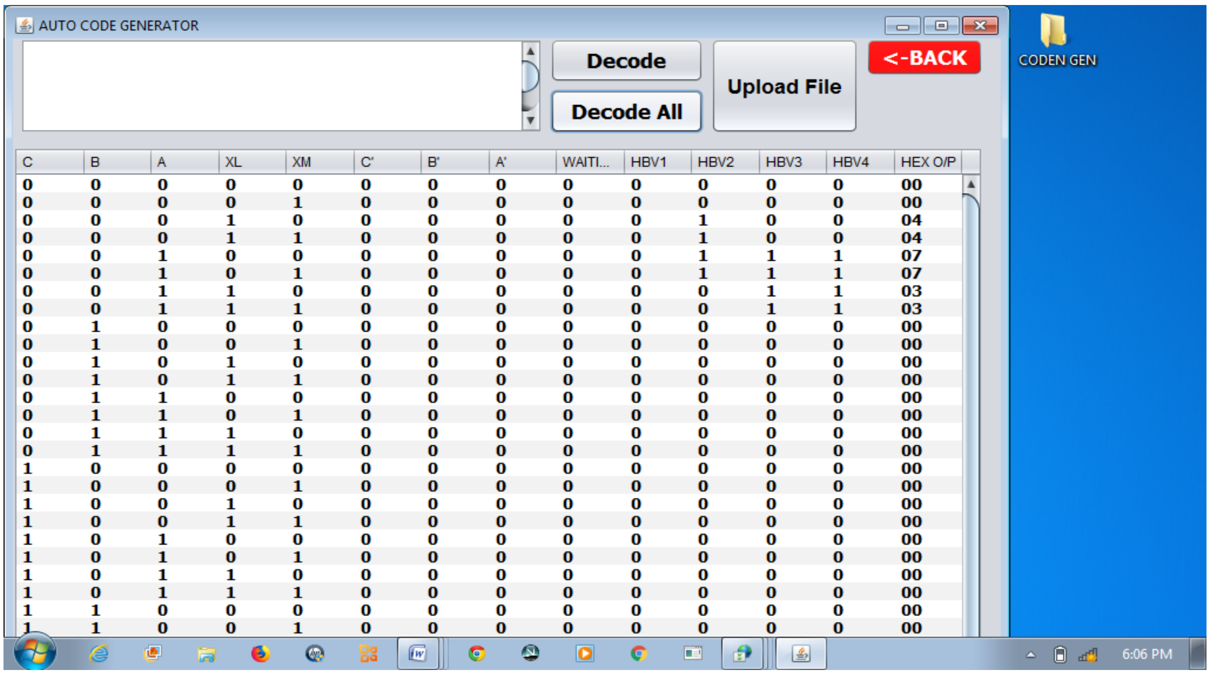
Fig. 14: Screen showing when Done Button is clicked.



C	B	A	XL	XM	C'	B'	A'	WAITING	HBV1	HBV2	HBV3	HBV4	HEX O/P

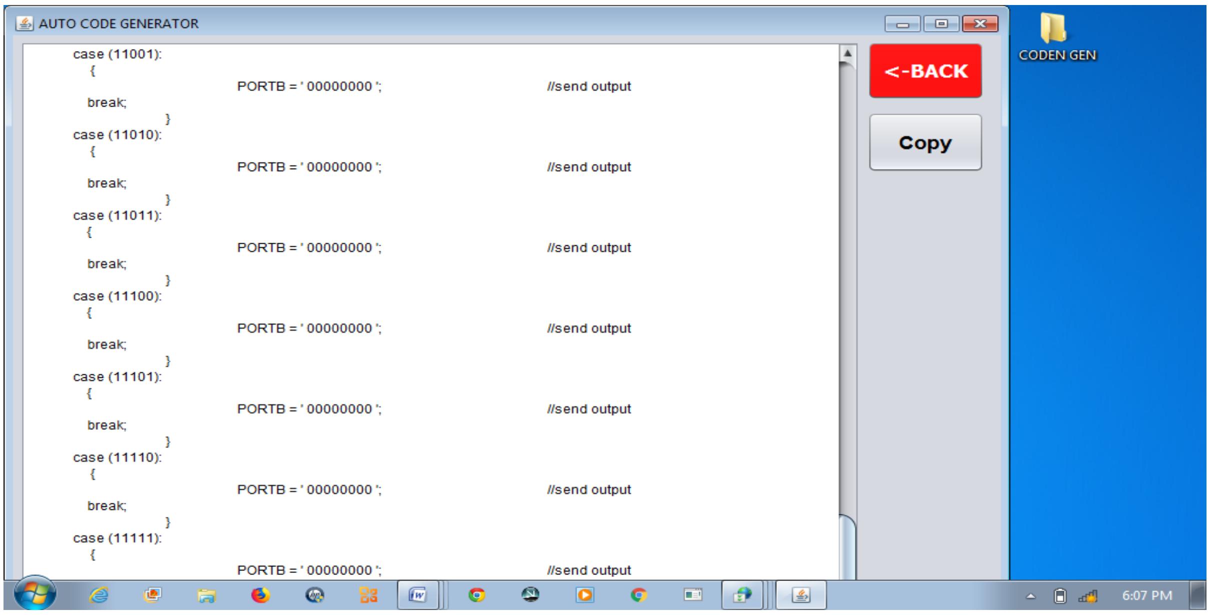
Fig. 15: Screen showing when Virtual Test Button is Clicked.

In fig. 16, FESTT would be generated when Decode All button is clicked. Decode Button generates the information for a particular link path, when clicked while Upload File button uploads the file where the information from the engineer's FESTT is stored, when clicked.



C	B	A	XL	XM	C'	B'	A'	WAITI...	HBV1	HBV2	HBV3	HBV4	HEX O/P
0	0	0	0	0	0	0	0	0	0	0	0	0	00
0	0	0	0	1	0	0	0	0	0	0	0	0	00
0	0	0	1	0	0	0	0	0	0	1	0	0	04
0	0	0	1	1	0	0	0	0	0	1	0	0	04
0	0	1	0	0	0	0	0	0	0	1	1	1	07
0	0	1	0	1	0	0	0	0	0	1	1	1	07
0	0	1	1	0	0	0	0	0	0	0	1	1	03
0	0	1	1	1	0	0	0	0	0	0	1	1	03
0	1	0	0	0	0	0	0	0	0	0	0	0	00
0	1	0	0	1	0	0	0	0	0	0	0	0	00
0	1	0	1	0	0	0	0	0	0	0	0	0	00
0	1	0	1	1	0	0	0	0	0	0	0	0	00
0	1	1	0	0	0	0	0	0	0	0	0	0	00
0	1	1	0	1	0	0	0	0	0	0	0	0	00
0	1	1	1	0	0	0	0	0	0	0	0	0	00
0	1	1	1	1	0	0	0	0	0	0	0	0	00
1	0	0	0	0	0	0	0	0	0	0	0	0	00
1	0	0	0	1	0	0	0	0	0	0	0	0	00
1	0	0	1	0	0	0	0	0	0	0	0	0	00
1	0	0	1	1	0	0	0	0	0	0	0	0	00
1	0	1	0	0	0	0	0	0	0	0	0	0	00
1	0	1	0	1	0	0	0	0	0	0	0	0	00
1	0	1	1	0	0	0	0	0	0	0	0	0	00
1	0	1	1	1	0	0	0	0	0	0	0	0	00
1	1	0	0	0	0	0	0	0	0	0	0	0	00
1	1	0	0	1	0	0	0	0	0	0	0	0	00

Fig. 16: Screen showing when Decode All Button is clicked.



```
case (11001):
 {
                    PORTB = ' 00000000 ';                //send output
  break;
          }
case (11010):
 {
                    PORTB = ' 00000000 ';                //send output
  break;
          }
case (11011):
 {
                    PORTB = ' 00000000 ';                //send output
  break;
          }
case (11100):
 {
                    PORTB = ' 00000000 ';                //send output
  break;
          }
case (11101):
 {
                    PORTB = ' 00000000 ';                //send output
  break;
          }
case (11110):
 {
                    PORTB = ' 00000000 ';                //send output
  break;
          }
case (11111):
 {
                    PORTB = ' 00000000 ';                //send output
```

Fig. 17: Screen Showing when Generate Source Code Button is Clicked.

Fig. 17 depicts the source code or the program code automatically generated by the automatic code generator. The programming language used for the generated source code is C programming language. The source code generated by the automatic code generator will then be compiled with C compiler known as MiKro C Pro and this hex code will be burnt into a microcontroller. The microcontroller chosen for the test execution and simulation is a PIC microcontroller. The memory architecture of the source code is specially designed for PIC microcontrollers, thus subsequent improvement should enable the selection of other microcontrollers. The simulation environment is Proteus.

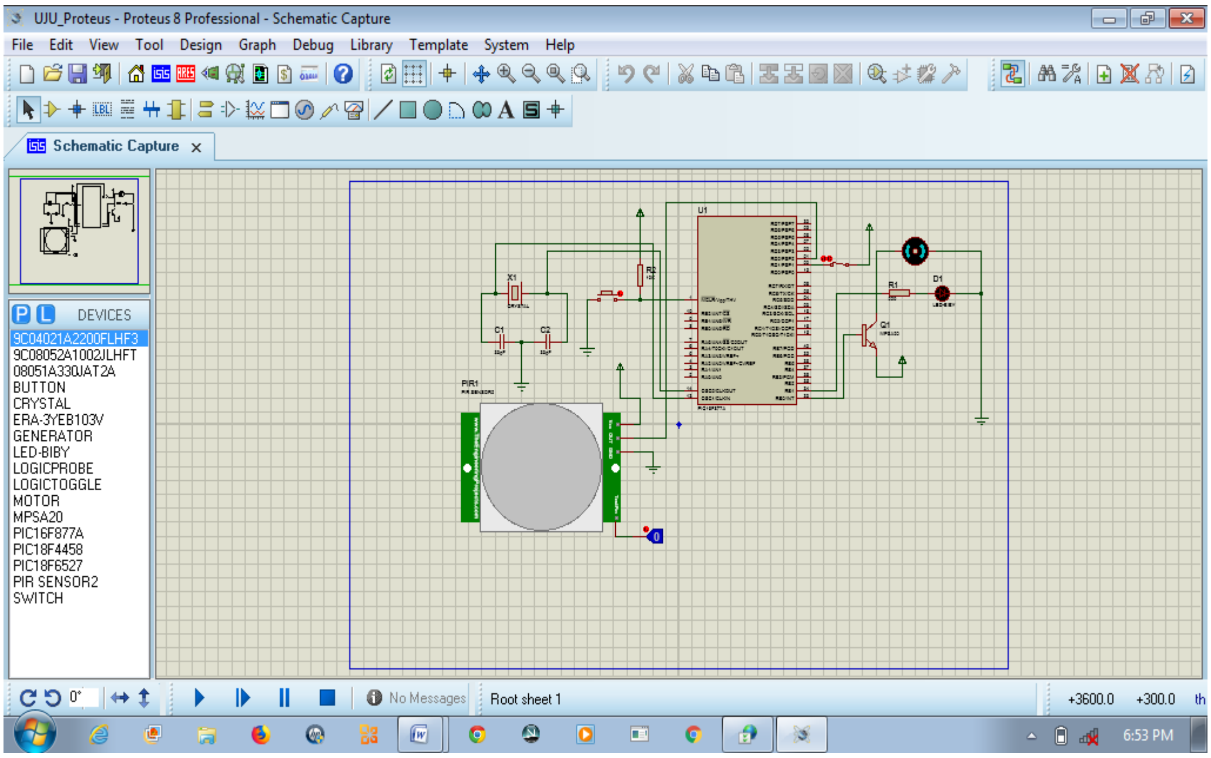


Fig. 18: Proteus Representation of the System with ASM chart of Fig. 7

In the fig. 18, the microcontroller used is the Pic16F877A, which is clocked at 32MHz. A PIR (Passive Infra Red) sensor is used to sense human presence in the room. The PIR sensor used in the simulation has 4 pins namely: VCC, OUT, GND and Test pin. The VCC pin is connected to 5V source, the OUT pin is connected to pin RD1 of the PIC microcontroller. The GND pin is connected to ground. The RD1 pin of the microcontroller is used as the input to sense when there is human presence, but because this is a simulation, the PIR sensor module for Proteus has a “Test pin” which is used to simulate human presence. When a HIGH(Logic 1) is sent to this pin using the logic probe connected to it, the RD1 pin goes high which means there is human presence and the LED turns ON and the motor (representing the fan) rotates. When a LOW (Logic 0) is sent, the RD1 pin goes low which means there is no human presence, the LED turns off and the motor stops rotating.

VI. CONCLUSION

Automated Process control using multi-agent has become popular in recent time. Automatic code generator makes it less tasking and time consuming to generate process control codes. Thus a researcher with any automation design problem that can be tailored to an ASM chart can benefit from the automated code generator design example showcased in this paper.

The method discussed here is generic and is not limited to monitoring agent-based process control systems. Other process control systems designed using any other method can be monitored using this method, except that the ASM chart and the modified STT must be provided to aid the design of the agent monitoring system. Automatic Code Generator (ACG) allows software engineers to create more concise, maintainable and reusable solutions ultimately improving their productivity.

REFERENCES

[1]. Pohl, C., Paiz, C., and Porrmann, M. (2009). *vMAGIC—Automatic Code Generation for VHDL.International Journal of Reconfigurable Computing*.http://dx.doi.org/10.1155/2009/205149. Volume 2009 (2009), Article ID 205149, 10 pages. Heinz Nixdorf Institute, University of Paderborn, Fürstenallee 11, D – 33102 Paderborn, Germany.

[2]. Inyiama, H.C., Obiora-Dimson, C.I. and Okezie, C.C. (2015). “*Designing an automated code generator for multi-agent based process control and monitoring*”,RexCOMMPAN©2015 www.rex.commpan.com Page 1.

International Journal of Advanced Multidisciplinary Research Reports. Volume I Number 1. Maiden Edition.

[3]. Hill, J. (2004) *Brief Introduction to ASM Charts.*

[4]. Fabio Bellifemine, G. C. (2007). *Developing Multi-Agent Systems with JADE*. John Wiley & Sons Ltd.

[5]. Inyiama, H. C., Okezie C, C., and Okafor, I. C. (2012). "*Agent Based Process Control System Design"*. Proceedings of the peer reviewed 2012 National conference on infrastructural development and maintenance in the Nigerian environment.

[6]. Terán, J. A. (2014). "*Collective Learning in Multi-Agent Systems Based on Cultural Algorithms*". Clei Electronic Journal .

[7]. Niazi, M. and Hussain, A. (2011). *Agent-based Computing from Multi-agent Systems to Agent-Based Models*: A Visual Survey (PDF). Scientometrics. Springer. 89 (2): 479–499. doi:10.1007/s11192-011-0468-9.

[8]. Ghasemlou, S., Ali, M ., Taher, A. S., and Mohammadreza, T. (2014). *Homecoming: A multi-robot exploration method for conjunct environments with a systematic return procedure*. In European Conference on Multi-Agent Systems, pp. 111-127. Springer International Publishing,

[9]. Kubera, Y., Mathieu, P. and Picault, S. (2010). *Everything can be Agent!* (PDF), Proceedings of the ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2010), Toronto, Canada: 1547–1548

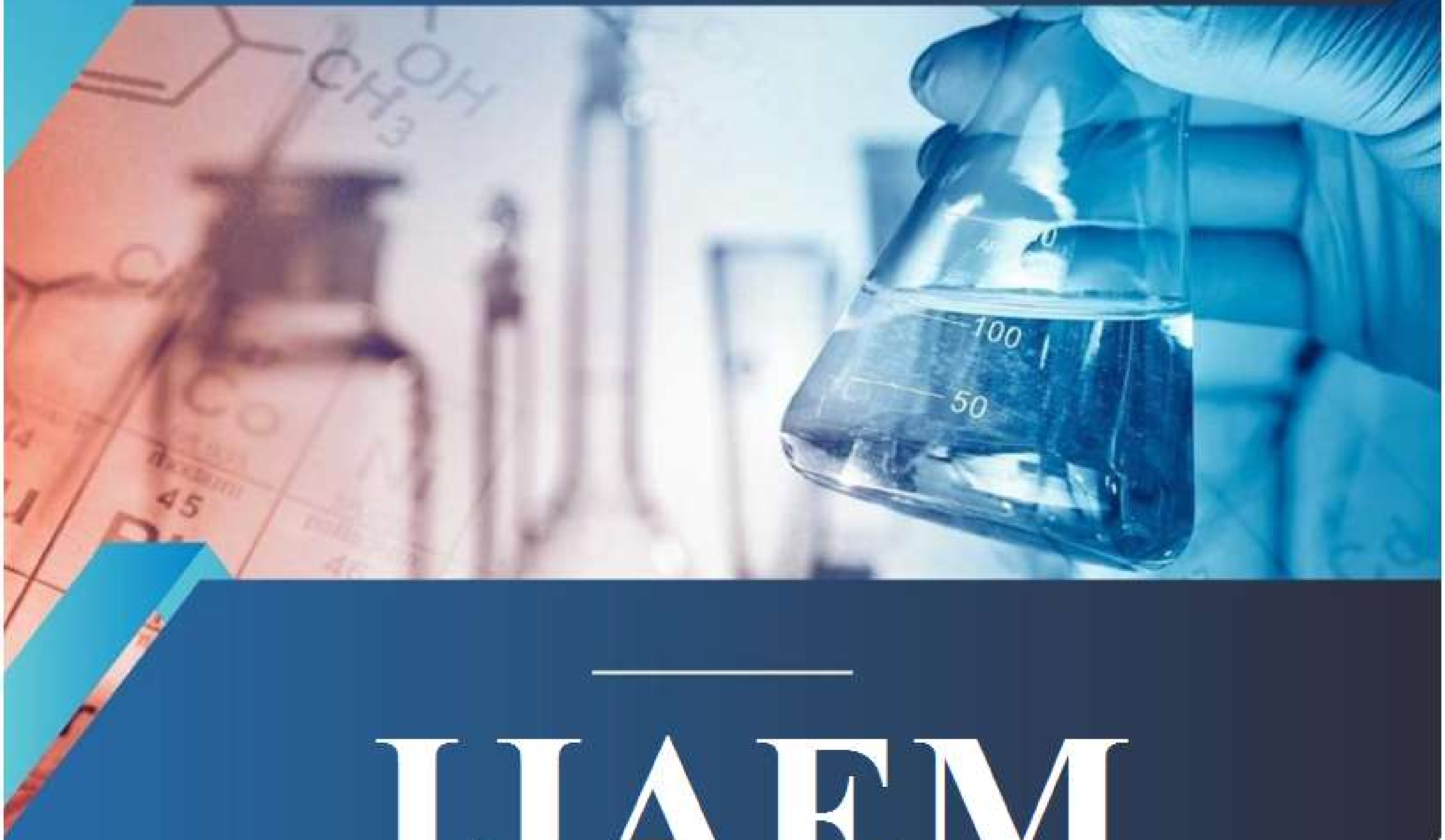
[10]. Salamon, T. (2011). *Design of Agent-Based Models*. Repin: Bruckner Publishing. p. 22. ISBN 978-80-904661-1-1.

[11]. Weyns, D., Omicini, A. and Odell, J. (2007). *Environment as a first-class abstraction in multiagent systems* (PDF). Autonomous Agents and Multi-Agent Systems.

[12]. Obiora-Dimson I. and Inyiama H. C. (2017). Re-Engineering Complex Process Control Systems Using Sub-Process Agents. Journal of Engineering Research and Application , pp 53-61.

[13]. Hayzelden, A. A. (2001). *Agent Technology for Communication Infrastructures*. John Wiley & Sons.

[14]. Uju Mokwe V., Stephen U. Ufoaroh, Obiora-Dimson Ifeyinwa and Kebiru Abu (2020). "*Agent-Based Automatic Code Generator for Control Systems Using the Algorithmic State Machine Chart Approach*" International Journal of Latest Technology in Engineering, Management & Applied Science (IJLTEMAS) Volume IX, Issue VI. PP. 1-12.



Volume: 02 Issue: 01 DOI: 10.35629/5252