Agent-Based Automatic Code Generator for Control Systems Using the Algorithmic State Machine Chart Approach

Uju Mokwe V.[1], Stephen U. Ufoaroh[2], Obiora-Dimson Ifeyinwa[3], Kebiru Abu [4]
[1,2,3,4] *Department of Electronic and Computer Engineering, Nnamdi Azikiwe University Awka, Anambra State, Nigeria*

Abstract: **This work involves developing an agent based automatic code generator that can generate automatically, control software that can run in a microcontroller and perform the control action specified in a given Algorithmic 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 fully 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 fan). 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: **Code generator, Automatic programming, Multi-agent, Microcontroller, Algorithmic state machine, State transition table, PIC microcontroller.**

I. INTRODUCTION

Automatic code generator originated from the term automatic programming. Generally automatic programming identifies a type of computer programming in which some mechanism generates a computer program to allow human programmers to write the code at a higher abstraction level [1]. Furthermore, Automatic Code Generation (ACG) allows software engineers to create more concise, maintainable, and reusable solutions, ultimately improving their productivity. ACG has significant benefits and profound economic impact in the software development field and thus is everywhere. Designers and developers of Automatic Code Generators designed and developed automatic code generators for a variety of applications.

In an earlier study, [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 and monitoring 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 solve this problem, the need to create a software that can easily automate the generation of this reusable codes is important. The aim of this work is to develop an agent-based automatic code generator for control systems using the ASM chart approach. The rest of the paper is structured as follows. Section II presents the agent based technology, Automatic code generation and Algorithm state machines (ASM) chart for control systems. Section III describes the methodology adopted. Data presentation and results are presented in Section IV. We conclude the paper in section V by highlighting the main findings of the paper.

II. LITERATURE REVIEW

2.1 Agent-based Technology

An Agent-based model (ABM) is one of a class of computational models for simulating the actions and interactions of autonomous agents (both individual and

collective entities such as organizations or groups) with a view to assessing their effects on the system as a whole. It combines elements of game theory, complex systems, emergence, computational sociology, multi-agent systems, and evolutionary programming. Monte Carlo methods are used to introduce randomness. Particularly within ecology, ABMs are also called individual-based models (IBMs), and individuals within IBMs may be simpler than fully autonomous agents within ABMs. A review of recent literature on individual-based models, agent-based models, and multi-agent systems shows that ABMs are used on non-computing related scientific domains including biology, ecology and social science [3]. Agent-based modeling is related to, but distinct from, the concept of multi-agent systems or multi-agent simulation in that the goal of ABM is to search for explanatory insight into the collective behavior of agents obeying simple rules, typically in natural systems, rather than in designing agents or solving specific practical or engineering problems [3].

Agent-based models are a kind of micro-scale model [4] that simulates the simultaneous operations and interactions of multiple agents in an attempt to re-create and predict the appearance of complex phenomena. The process is one of emergence from the lower (micro) level of systems to a higher (macro) level. As such, a key notion is that simple behavioral rules generate complex behavior. This principle, known as K.I.S.S. ("Keep it simple, stupid"), is extensively adopted in the modeling community. Another central tenet is that the whole is greater than the sum of the parts. Individual agents are typically characterized as boundedly rational, presumed to be acting in what they perceive as their own interests, such as reproduction, economic benefit, or social status, using heuristics or simple decision-making rules. ABM agents may experience "learning", adaptation, and reproduction. Most agent-based models are composed of: (1) numerous agents specified at various scales (typically referred to as agent-granularity); (2) decision-making heuristics; (3) learning rules or adaptive processes; (4) an interaction topology; and (5) an environment. ABMs are typically implemented as computer simulations, either as custom software, or via ABM toolkits, and this software can be then used to test how changes in individual behaviors will affect the system's emerging overall behavior. The idea of agent-based modeling was developed as a relatively simple concept in the late 1940s. Since it requires computation-intensive procedures, it did not become widespread until the 1990s. [4]

2.2 Automatic Code Generation

The notion automatic code generation (ACG) envelopes a number of different techniques aimed 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 [5].

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 [5].

2.3 Algorithm State Machines (ASM) Chart for Control Systems

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 [6]

2.4 Review of Related literatures

Xiang-Hu et al [7] proposed an algorithm which can generate code automatically by analyzing the characteristics of flow-chart and they put forward a structure identification algorithm for structured flowchart, after then taking the flowchart identified in previous step as input. They verified the

correctness and effectiveness of algorithms proposed using enumeration iteration strategy. The work did not use agent-based approach.

McBurney and McMillan [8] developed a documentation generator which is a programming tool that creates documentation for software by analyzing the statements and comments in the software's source code. In this paper, they proposed a technique that includes this context by analyzing how the Java methods are invoked. Also they hypothesize that existing documentation generators would be more effective if they included information from the context of the methods, in addition to the data from within the methods. This work is strictly on generating automatic code generator for documentation not for agent based control systems.

Sadiq et al [9] proposes an approach based on model driven architectures, multi-agent systems (MAS) for DSS (Decision Support System) development. They are particularly interested in this work in developing a multi-agent based extraction, transformation and loading process (ETL), retrieving, extracting and integrating external data into Data Warehouses (DW) and generating automatically the DW code. This work did not go into designing an automatic code generator.

Inyiama et al [2] also considered the use of multi-agents which has facilitated the level of complexity encountered in industrial automation. In this paper, agent types and classification with respect to this design method were discussed and an industrial control and monitoring examples were used to showcase the code generator multi-agent based design method envisaged in this paper. This work did not implement the automatic code generator.

Adenuga et al [10] developed an automated agent-based control system methodology (ACSME) and was proposed for Reconfigurable Bending Press Machine (RBPM) application due to an ongoing research. The proposed methodology will help manufacturer of RBPM to address the need for more flexible control systems and to demonstrate their industrial flexibility in several reconfigurable machines applications. Consequently, to facilitate the use of agent technology, there must be design methodology that allows a non-expert in agent technology to design an agent-based control system given the specification of the control system. This work did not go into designing an automatic code generator.

Pardeed and Rajesh [11], using Class Diagram, Use cases and Activity Diagram stated that in Regression testing, test case generation is a process of generating test cases from the existing test suite to ensure that modifications made in the system have not affected its existing functionality. In this research, they have used the combination of UML class diagram, use cases and activity diagram to identify changes at both syntax and semantics level. They compared UML class diagrams, use cases and activity diagrams of old and modified code to identify these changes. The work did not use agent-based approach.

Palau, et al [12] presents a methodology to assess the prognostics in modern fleets of assets. They investigated the cost, reliability and implications of using different multi-agent systems architectures for collaborative failure prediction and maintenance optimization in large fleets of industrial assets. The work did not go into designing an automatic code generator.

III. METHODOLOGY

In this work, 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 [13]. 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.

3.1 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* [14]. 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 [14]. 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.

3.2 Software Architecture

The diagram in Fig. 1 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 [15]. 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.

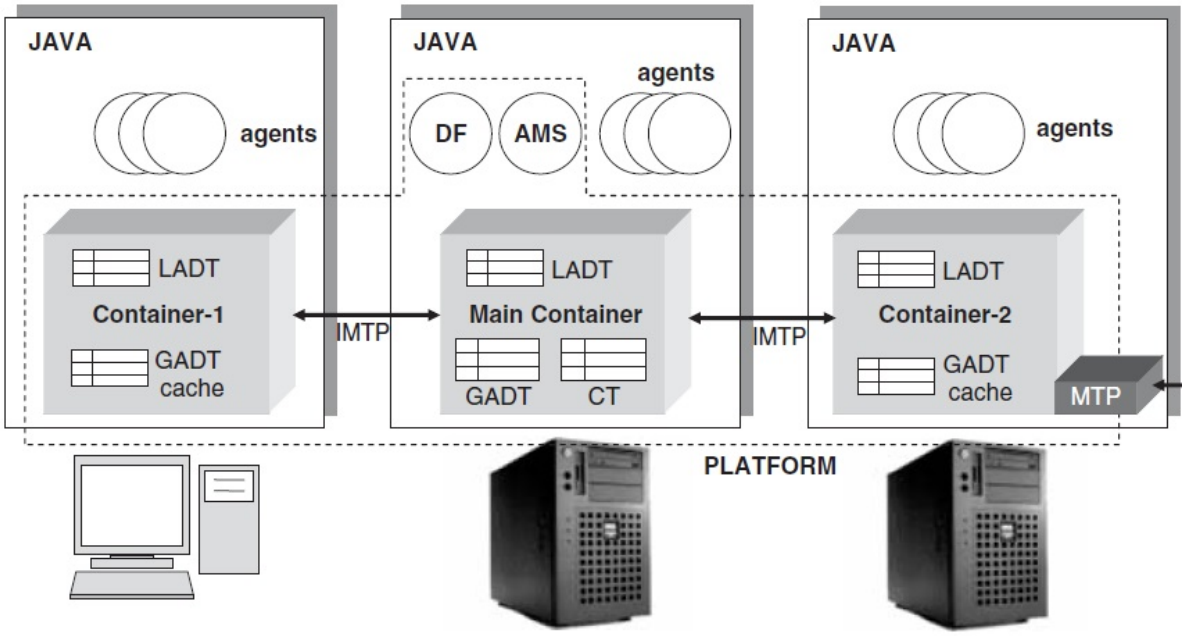

Fig. 1: Architectural elements of the multi agent based platform [14]

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 [16]:

- 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

3.3 Developing Algorithm

This is the first step in program design. It 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

IV. RESULTS AND DISCUSSION

4.1 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 room and turns the fan on when there is an occupant and turns off the light and fan when there is no occupant is depicted in fig. 2

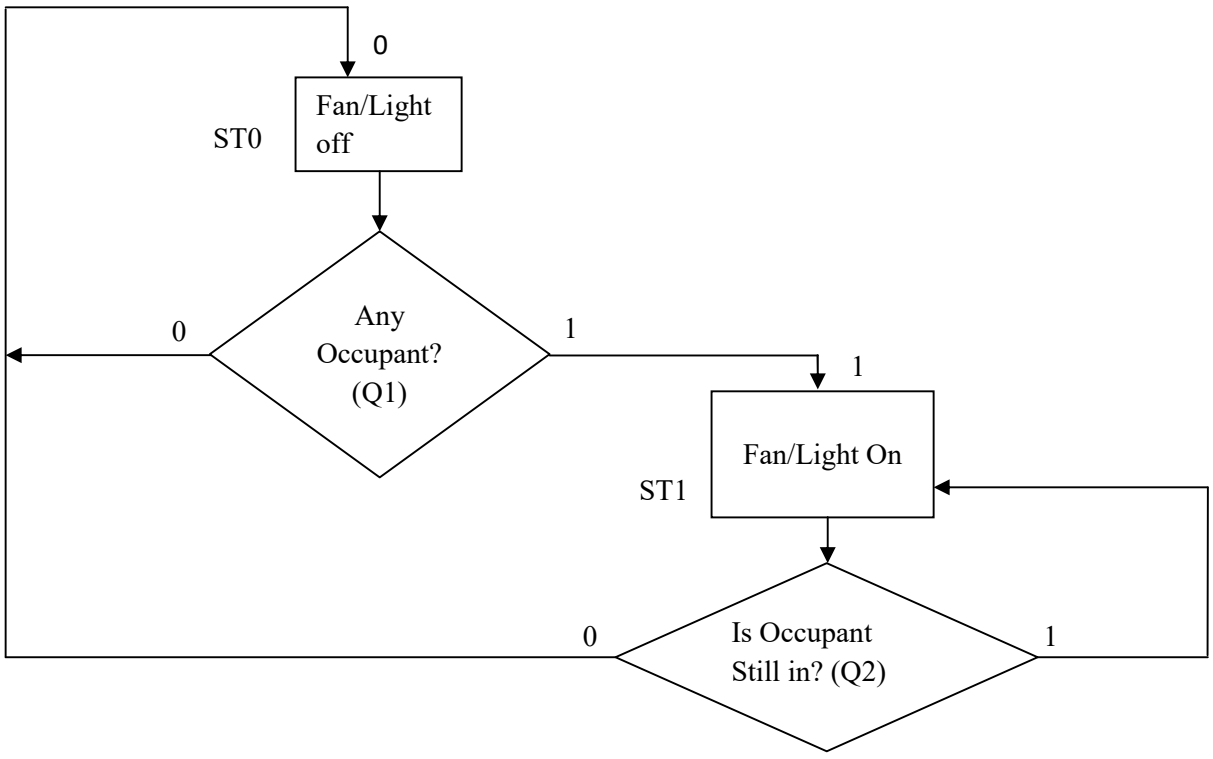

Fig. 2: Diagram Depicting the ASM Chart

The working principle of the ASM chart is represented in fig. 2:

i. At state ST0, fan 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, fan 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 1: The State Transition Table (STT)

Link path	Present state		Qualifiers		Next state		State Output	
	Name	Code	Q1	Q2	Name	Code	Fan	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 2: Fully Expanded State Transition Table

Link Path	Present State		Qualifiers		Next State		State Output		Hex Output	State agent
	Name	Code	Q1	Q2	Name	Code	Fan	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 2, 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. 1.

Fig. 3: User Interface of the Automatic Code Generator Software

From the fig. 3, 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.

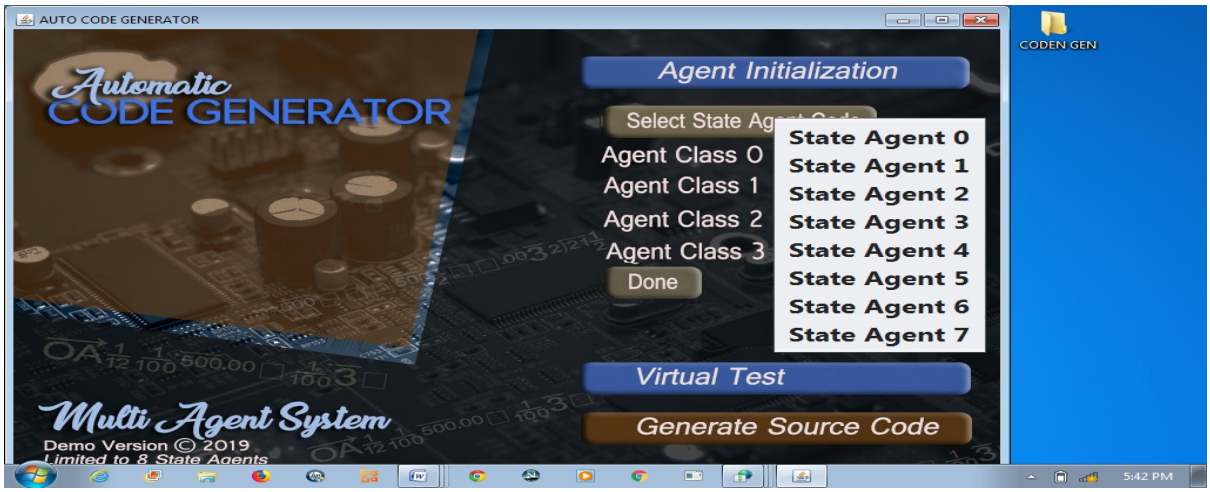

Fig. 4: Screen showing when Select Agent Code Button is Clicked

When the Select State Agent Code button is right clicked as depicted in fig. 4, 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. 5, the State Agent Zero is highlighted and clicked. In fig. 6, 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. 1. 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. 7, the state agent one is highlighted and clicked. In fig. 8, the values are inputted. It is filled in with these information, 111, 111, 011 and 011 as done for state agent zero.

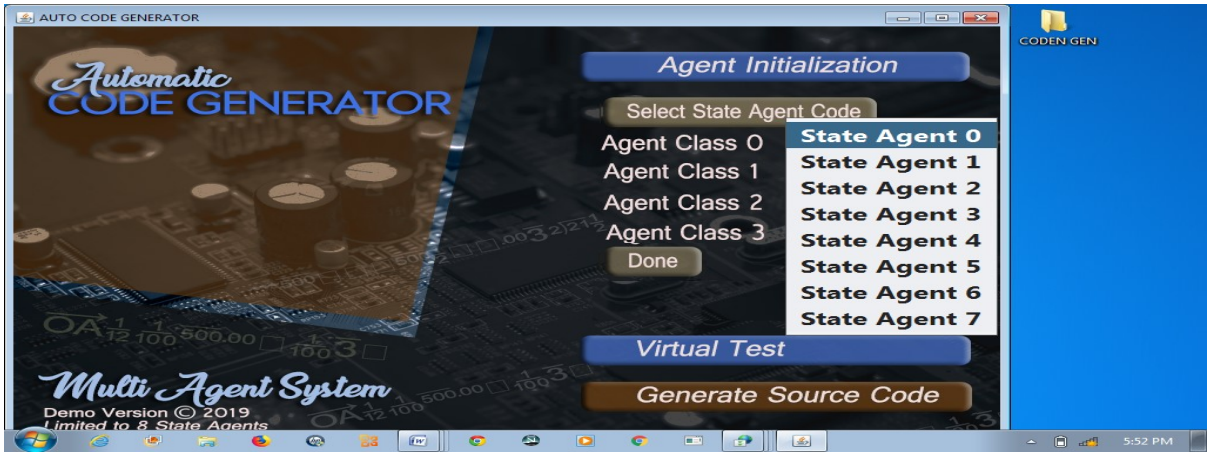

Fig. 5: Screen showing when State Agent Zero is highlighted.

Fig. 6: Screen showing when the values of State Agent Zero are inputted.

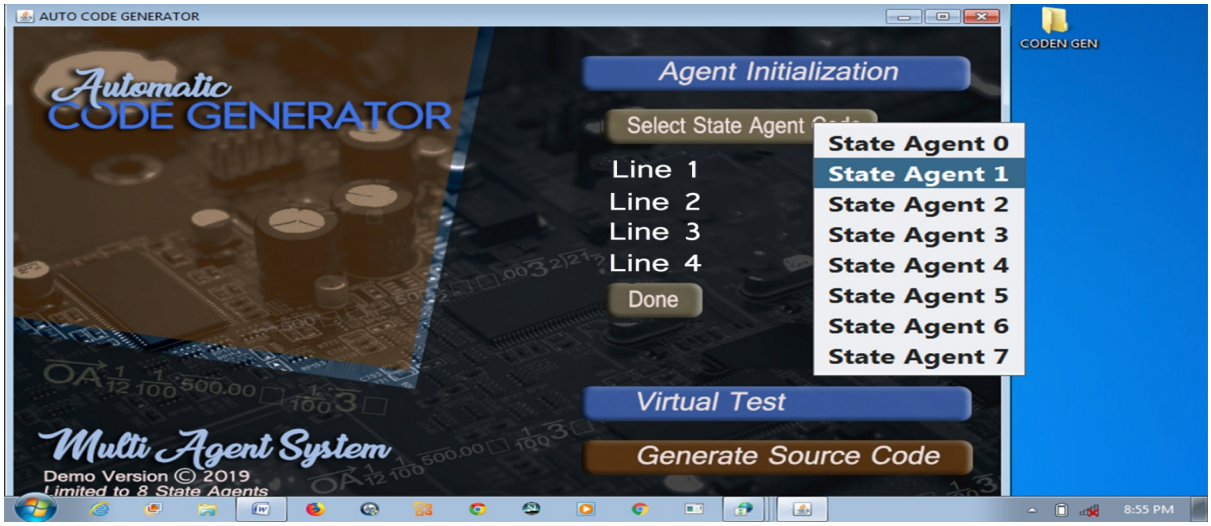

Fig. 7: Screen showing when State Agent One is highlighted.

Fig. 8: 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. 9.

Fig. 9: Screen showing when Done Button is clicked.

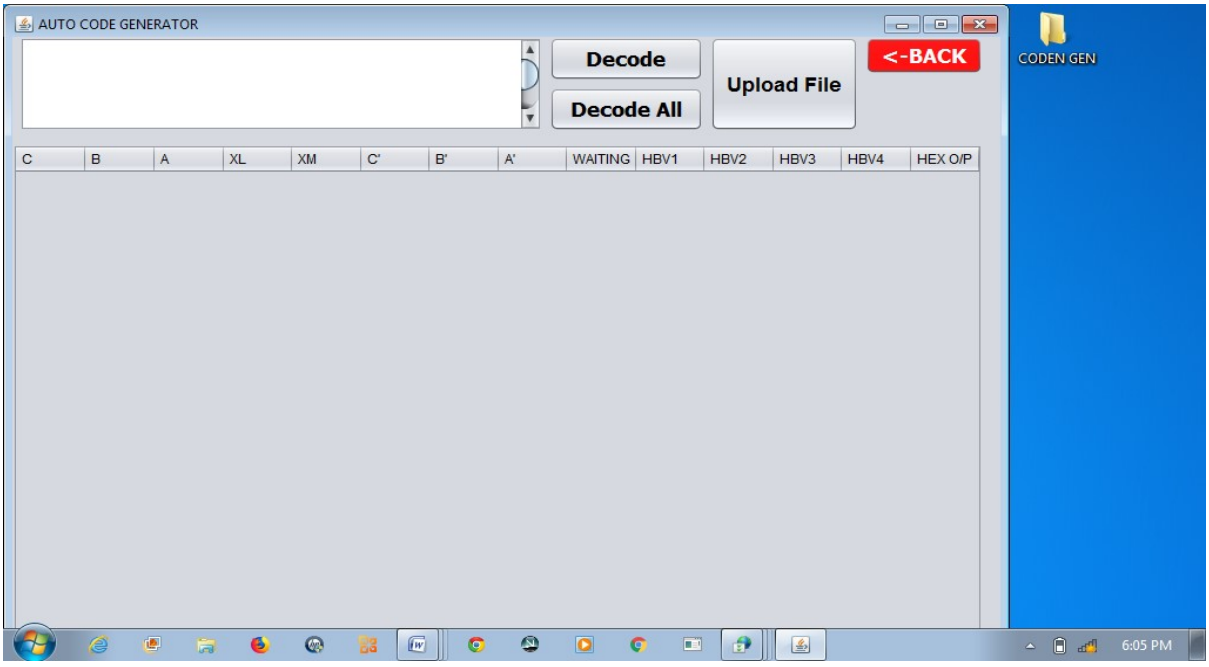

Fig. 10: Screen showing when Virtual Test Button is Clicked.

In fig. 11, 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.

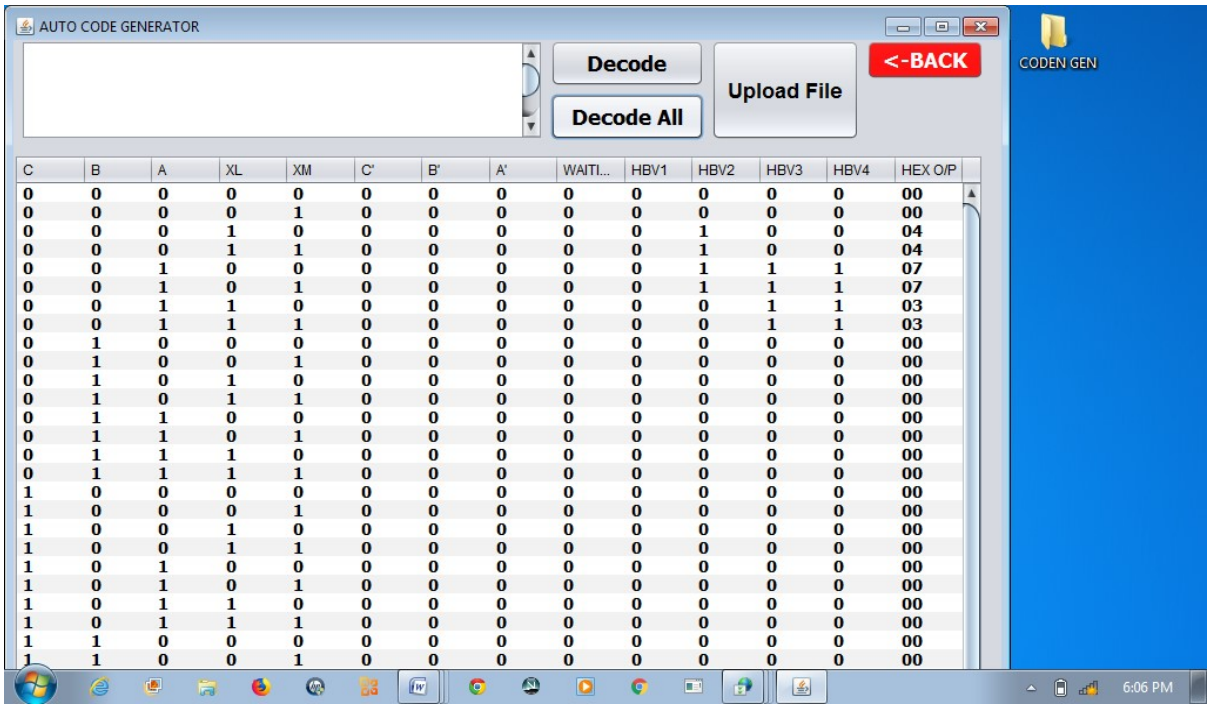

Fig. 11: Screen showing when Decode All Button is clicked.

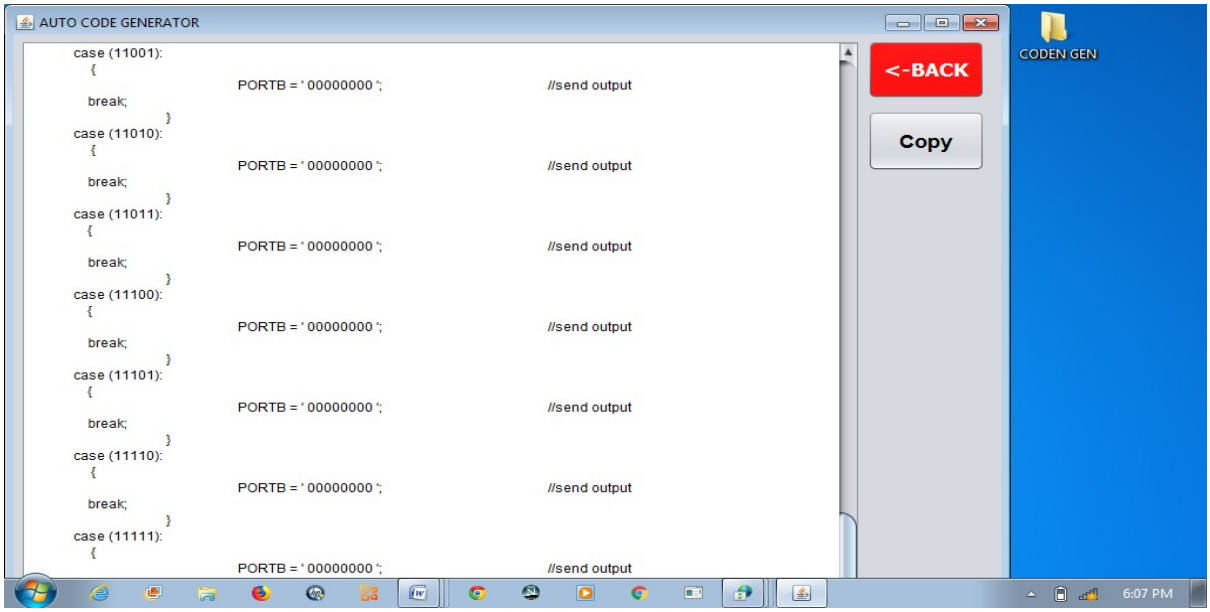

Fig. 12: Screen Showing when Generate Source Code Button is Clicked.

Fig. 12 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.

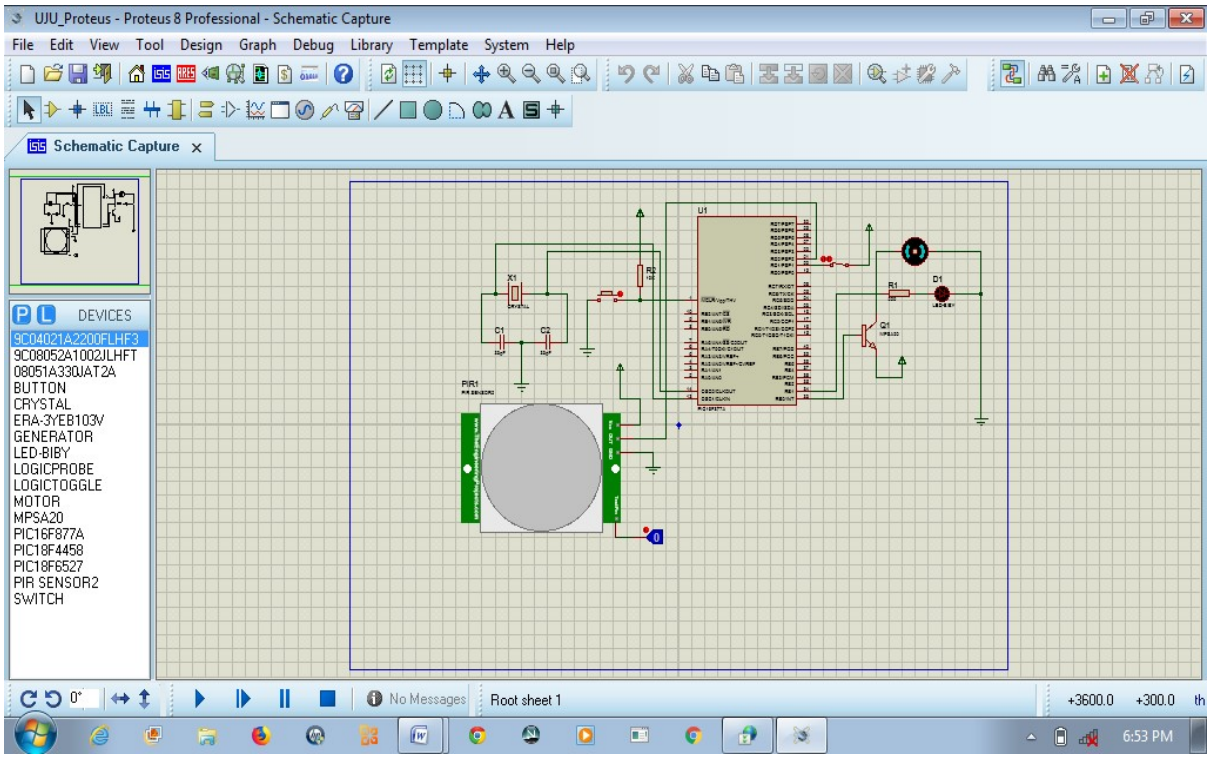

Fig. 13: Proteus Representation of the System with ASM chart of Fig. 1

In the fig. 13, 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.

V. CONCLUSION

Automated Process control and monitoring 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] Mur, R.A. (2006) *Automatic Inductive Programming*, ICML 2006 Tutorial.

[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", International *Journal of Advanced Multidisciplinary Research Reports*. Volume I No. 1. Maiden Edition.

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

[4] Gustafsson, L. and Mikael, S. (2010). *Consistent micro, macro, and state-based population modeling*. Mathematical Biosciences.225(2): pp 94–107. doi:10.1016/j.mbs.2010.02.003.

[5] 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.

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

[7] Xiang-Hu Wu, Ming-Cheng Qu, Zhi-Qiang Liu and Jian-Zhong Li, (2011) Research and Application of Code Automatic Generation Algorithm Based on Structured Flowchart. *Journal of*

Software Engineering and Applications 4, pp 534-545. doi:10.4236/jsea.2011.49062.

[8] McBurney, P.W. and McMillan, C. (2014) "Automatic Documentation Generation via Source Code Summarization of Method Context". ACM 978-1-4503-2879-1/14/05 $15.00.

[9] Sadiq, A., El Fazziki, A. and Sadgal, M. (2014). "An Agent Based Etl System: Towards an Automatic Code Generation". DOI: 10.5829/idosi.wasj.2014.31.05.268.. World Applied Sciences Journal 31 (5): pp 979-987.

[10] Adenuga, O.T., Mpofua, K. and Kanisurua, A.M. (2016). Agent-based control system methodology for Reconfigurable Bending Press Machine, ScienceDirect. *49th CIRP Conference on Manufacturing Systems (CIRP-CMS 2016). Procedia CIRP 57*, pp 362 – 367. doi: 10.1016/j.procir.2016.11.063

[11] Pardeep Kumar Arora and Rajesh Bhatia, (December, 2017) "Agent-Based Regression Test Case Generation using Class Diagram, Use cases and Activity Diagram", *6th international conference on smart computing and communications,* ICSCC 2017, 7-8 kurukshetra, India.

[12] Palau, A. S., Parlikad, A. K. and Dhada, M. (2019). "Multi-Agent System Architectures for Collaborative Prognostics". *Journal of Intelligent Manufacturing*. DOI: 10.1007/s10845-019-01478-9.

[13] 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.

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

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

[16] Genesereth, M. A. (1994.). *Software Agents*. Communications of the ACM, 37(7):, pp. 48–53.