J. Basic. Appl. Sci. Res., 2(8)8272-8284, 2012

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

Visualized and Abstract Formal Modeling towards the Multi-Agent Systems

Ghulam Ali[1], Sher Afzal Khan[1], Farooq Ahmad[1] and Nazir Ahmad Zafar[2]

[1]Faculty of Information Technology, University of Central Punjab
Lahore, Pakistan
[2]Department of Computer Science, College of Computer Science and Information Technology
King Faisal University, Al Hassa, Saudi Arabia

ABSTRACT

Multi-agent systems technology has generated lots of incitements, because of it used for distributed, intelligent and safety critical system. The failure of some system may cause the loss of human life, severe injuries, loss of money and environmental damages. To get the robust, safe and reliable multi-agent system it requires formal modeling and step by step refinement for its construction and development. Some of the formal specification-based languages, such as, VDM, Object-Z and RAISE have been used for its modeling using abstract modeling approach. However, due to the distributed environment of agents and their communications, both graphical and abstract modeling is required to capture the agent behavior and their communication. In this research we present a formal modeling technique based on communicating stream X-machine and Z notation to formally specify the multi-agent systems. Initially, we decompose the complex system into standalone agents and model it using X-machine. Further, the model is specified using Z-notation. For communication between agents, we specify communication mechanism using communicating X-machine. Moreover, the model is transform to Z-notation to prove its syntax by using Z/EVES tool. The main theme of the paper is to see the visual development of agents using X and stream X-machine and then to transform it to the abstract model for its proof and syntax verification.

KEYWORDS: X-machine; Z-notation; multi-agent systems; communicating stream X-machine

1. INTRODUCTION

The real world is a dynamic place where things change in an unexpected way. The software must be able to adapt these changes to work efficiently in the real world. The modeling, design, implementation, verification and maintenance of complex distributed, intelligent and safety critical system require to be decomposed into various independent small entities. These entities can be viewed as an agent, which provides a simple way to understand such complex entities.

In recent years the multi-agent systems (MAS) technology has generated lots of incitements, because of its promise as a new paradigm for conceptualizing, designing, and developing complex software systems [1]. In MAS modeling, a complex system is viewed as a large number of autonomous communicating entities. For such modeling the main focus is to identify the components and their interactions from their local to global behaviors. The global behavior emerges from the local behaviors of the individual components as described in [2]. MAS approaches have used to model the large number of multi-behavior distributed applications in different areas of real life such as supply chain management, workforce management, distributed computing, business process management, e-health care, knowledge discovery, knowledge sharing and railway interlocking system [3][15][16][17].

A number of methodologies for the development of MAS have been developed such as AUML, AML, MaSE, GAIA, X-Machine, and Stream X-Machine. In [5], a MAS model is developed using a colored Petri nets, further it is verified by using Petri net verification. In [6] an attempt to verify whether properties of agent model is correct, the work has been done on model checking of MAS. In [7], a framework for describing agents, tasks and environments is developed using Z specification language. In [8], it is described that how the formal agent framework can be refined and used to support the development of multi-agent systems.

The above discussion reveals that the use of formal methods has been used for the development of MAS. However, it captures some of the features, but fail to describe the system completely. It is because; it provides a little or no reference at all to the internal data which affects each operation in the state transitions of the system. For example, the Statecharts captures the requirements of dynamic behavior and modeling of data but informal with respect to clarity and semantics [7].

***Corresponding Author**: Sher Afzal Khan, Faculty of Information Technology, University of Central Punjab, Lahore, Pakistan. ghulamali,sherafzal, dr.farooq@ucp.edu.pk

In this research the agent based model is developed using X-machine and the communication among agents is modeled by using stream X-machine and then both the models are transformed to Z notation. The main objective of this research is to refine the specification and to demonstrate the expressiveness and suitability of the formal methods used for modeling agent-based systems.

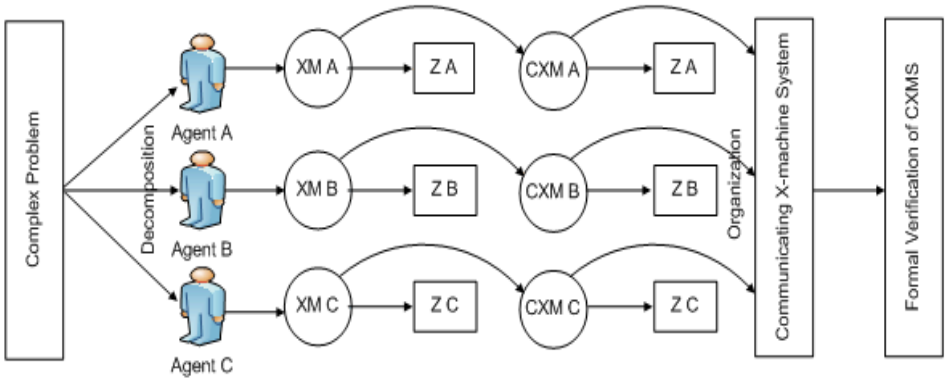

Figure 1: An X-machine Based Formal Model of Multi-agent Systems

Our propose approach to develop a formal model of multi-agent system is illustrated in Figure 1. The system is decomposed into small entities called agents which are modeled using X-machine. The X-machine is then specified using Z-notation which is denoted by ZA, ZB and ZC. Further, the communication mechanism between agents is added to the X-machine models which become a communicating X-machine, denoted by CXM A, CXM B and CXM C in the Figure. Moreover, the communicating X-machine model is formally specified using Z-notation as denoted by ZA, ZB and ZA in the sixth column of the Figure. These communicating X-machine models are organized, as a whole system in communicating stream X-machine system, where all agents can communicate. This communicating X-machine system is further specified using Z-notation.

This paper is organized as follows. In section 2, a survey on related work is presented. In section 3, the computational models of X-machine are introduced. In section 4, biological inspired reactive agent is given. Formal description of the system is given in section 5. The model is analyzed in section 6. Finally, conclusion and future work are discussed in section 7.

2. Computational Models of X-Machine

The X-machine (XM) model was originally introduced by Samuel Eilenberg in 1974 in his study of theory of computation as described in [4]. The XM is a more abstract model of computation than finite state machine, finite automat, pushdown automata and Turing-machine. The X-machine is equivalent to the Turing machine with respect to the power of carrying out computations. Both of these machines solve the same class of problems and are able to accept recursively enumerated languages. They have the same expressive power but X-machine provides a more intuitive, efficient, effective and elegant way of modeling the complex systems.

2.1. X-machine

The X-machine is a general computational model similar to other finite state machines with two differences [9]: (i) there is a memory attached with the machine and (ii) transitions are not only labeled with simple inputs but also with the functions that operate on inputs in data structure which make the machine more expressive than finite sate machines. Finite automata, pushdown automata and Turing machine are special cases of X-machine which can be obtained by varying definition of X-machine [10]. X-machines are capable to model both the data and behavior of a system.

2.2. Stream X-machine

An interesting class of X-machine is the stream X-machine (SXM) in which the input and output are streams of symbols. The power of SXM is considerable and it is able to model many practical computing situations [11][14]. Stream X-machine provides a facility to model the non-trivial data structure of a system as a typed memory tuple. Stream X-machine employs a diagrammatic approach to model the behavior of a system by enhancing the expressive power of finite state machine. Transitions between states are executed by applying the functions which are written in a formal notation. The Figure 2 shows the input and output streams based on the alphabet $\sum$ and Γ respectively. The symbols $\varphi 1$, $\varphi 2$, $\varphi 3$ and $\varphi 4$ are partial functions which operate on memory which has two states m and m'. The

functions accept input and memory values, and produce output by manipulating the memory values. The transitions are based on the current state and memory values. Each transition uses an input stream and determines the next state, new memory state and the output symbol which is a part of the output stream.

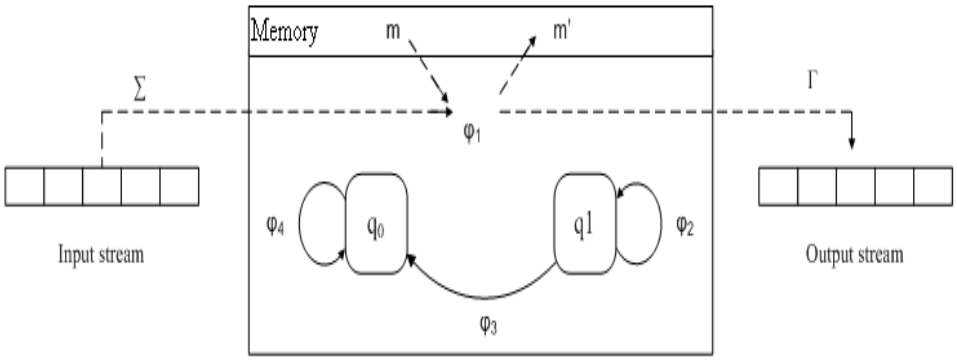

Figure2: Abstract stream X-machine

2.3. Communicating Stream X-machine

Communication of agents is a fundamental requirement for modeling the multi-agent systems. A system requires common communication mechanism for cooperation with the other agents. Communicating stream X-machine (CSXM) provides an effective communication mechanism for cooperation between the agents of a system. The CSXM is a computational model which can be applied to model the complex and distributed systems [12][13]. The Figure 3 shows four different types of streams in which the input stream C_j and output stream C_k are the communicating streams. The symbols φ1, φ2, φ3 and φ4 are partial functions which operate on the Memory. The function φ1 reads an input from standard input stream and writes an output on the standard output stream. The function φ2 and φ3 write on communicating output stream C_k and read from input stream C_j respectively. The function φ4 reads from input stream C_j and writes on output stream C_k. The symbols m and m' are two Memory states. The symbols C_j indicates that this function reads an input from a communicating input stream and C_k indicates that it writes an output alphabet on the communicating output stream. In CSXM the communication, data and control are modeled separately which provide the ease of decomposition of a complex system. This approach provides the facility of reusability of the X-machine models. The CSXMs models are being used to resolve the communication issues in various distributed application such as deadlock detection testing and cooperating distributed grammar systems [14][18][19].

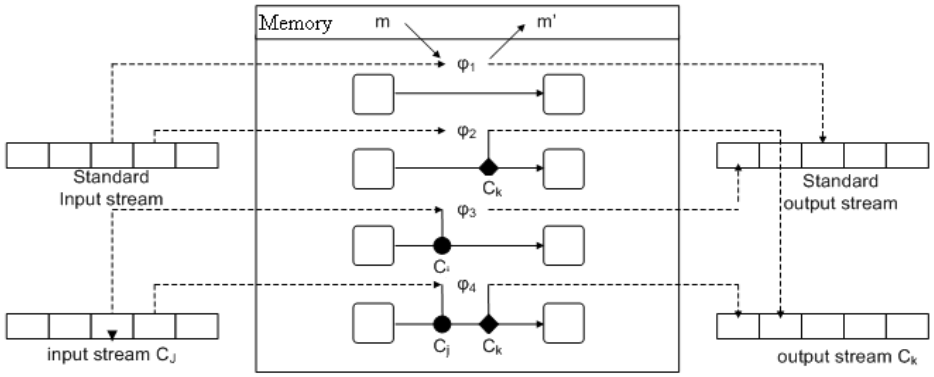

Figure 3: An abstract communicating stream X-machine

2.4. Communicating Stream X-machine System

Communicating stream X-machine system (CSXMS) provides a mechanism in which a number of communicating components can communicate. In CSXMS the communication between the communicating stream X-machines is confirmed by means of multiple streams. The CSXMS provides a common platform on which a number of communicating stream X-machines can communicate. Figure 4 shows how two stream X-machines can communicate with each other.

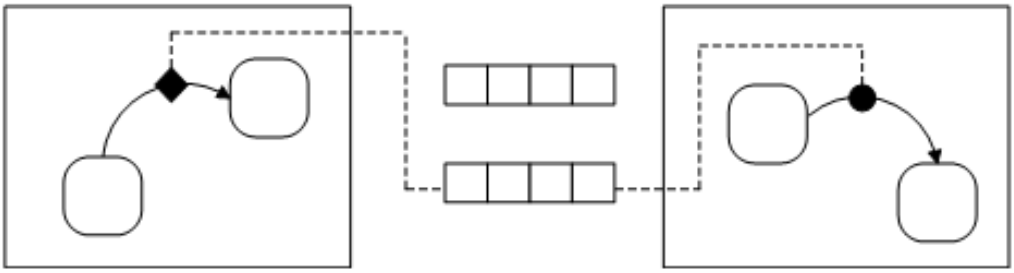

Figure 4: An example of communication between two X-machine functions

2.4.1. Attachment Operator

This operator is used to establish the communication between a set of CSXM components and existing communicating X-machine system (set of communicating X-machine components), i. e., ATT: $C \times Z \rightarrow Z'$.

C is the set of communicating X-machine components and Z is the set of communicating X-machine systems. Z' is the new communicating X-machine system which established different communication channels between communicating components C and communicating systems Z. The newly attached components remain the same except the function φ which can read from other components input streams and can write on other components output streams. Similarly the communication function of other components Z becomes related to the streams of newly attached components C. Through this way the whole system Z' becomes collection of cooperating components, which can send and receive messages from other components.

2.5. Detachment Operator

This operator is used to remove the communication channels between a set of communicating X-machine components from a communicating X-machine system with which it currently communicates, i. e., DET: $C \times Z \rightarrow Z'$, where C is a set of communicating X-machine components, Z is a communicating X-machine system and C is a subset of Z. Z' is the new communicating X-machine components system without C. All the communication channels and relationships between C and its input, output streams with other components Z' and their streams are removed.

2.6. Generation Operator

This operator is used to create and introduce a new communicating component into the system. If other components request to communicate with newly created component then communication channels are established, i. e., GEN: $C \times Z \rightarrow Z'$, where C is the newly created component and Z is the set of existing communicating X-machine systems.

2.7. Destruction Operator

This operator is used to remove the communicating X-machine component from the system along with the channels that are used for communication with other components of the system, i. e., DES: $C \times Z \rightarrow Z'$, where C and Z are same as defined previously. Z' is the set of communicating X-machine components without C

3. X-Machine Model of an Ant

Here we take the biological inspired intelligent agent as case study. The case study is taken from [19]. The stream X-machine model of ant agent is illustrated in Figure 5. The stream X-machine model of ant agent consists of five states, which the ant can be in: *Inactive, Dead, Taking, Giving,* and *Hungry*. The state *Inactive* shows that the agent holding enough food quantity. The state *Hungry* shows that the ant is hungry and it holding the food quantity below its hunger threshold. When a non hungry ant meets a hungry ant it shares the food with it, and the hungry ant that meet the non hungry ant receives food from it. The ant is in *Dead* state when food quantity dropped to zero. The memory of the agent holds its current location, the amount of food it holds, threshold level to indicate below that level the agent becomes hungry, food consumption rate that an ant consumed in a time unit, and the amount of food quantity to share with other agent that is hungry.

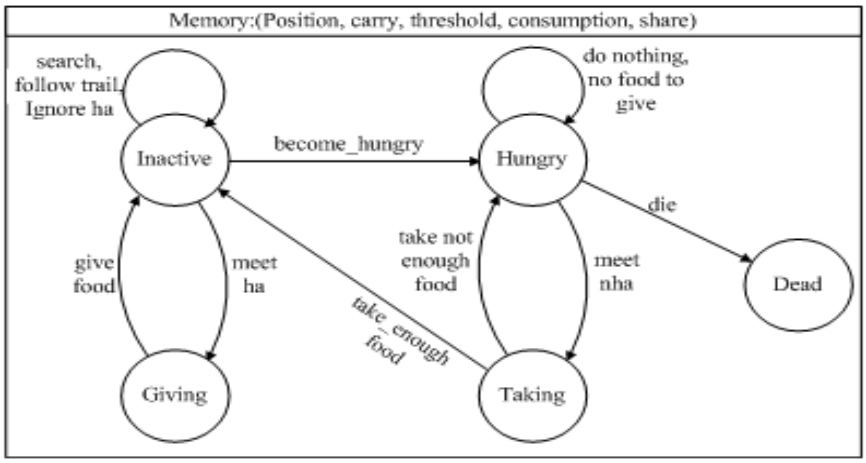

Figure 5: Stream X-machine model of an ant

These stream X-machine models can communicate by directing the output of one X-machine function as input to another X-machine. In this scenario the X-machine models of an ant can communicate while sharing food. The communicating stream X-machine model of an ant is same except these functions, *give_food, take_enoughfood* and *take_less_food*. The function *give_food* shows that it gives a specified amount of food to a hungry ant while writing the output to communicating stream. Similarly the functions *take_enoughfood* and *take_less_food* indicates that it receives a certain amount of food quantity from another ant by reading the input from a communicating stream. In Figure 6 the symbols ♦ and ● indicates an output and input message respectively.

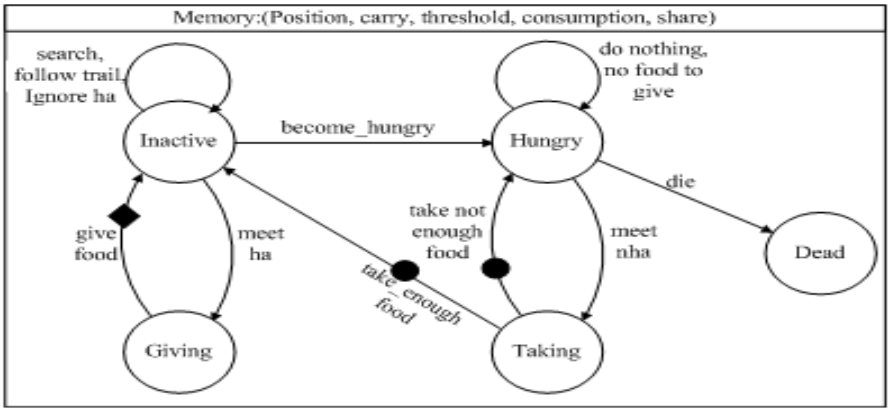

Figure 6: Communicating Stream X-machine model of an ant

4. Formal Modeling Using Z

In this section formal specification of X-machine models of an ant in Z notation is given. The agent models are: (a) stand-alone agent, (b) communicating agent, and (c) communicating multi-agent system. The formal specification of abstract X-machine models SXM and CSXM, which are used to specify these agent models are presented in our previous work [20][21].

4.3. Design of an Ant

To formally specify the X-machine model of an Ant agent we reused the formal specification of abstract stream X-machine SXM which is described in [21]. The SXM imposed all the general constraints that an Ant should possess. Therefore, we have only defined the specific functions and some other required data types along with operations which an agent can perform. It demonstrates that how abstract X-machine models can be used to model the stand-alone agents. Figure5 shows the stream X-machine model of an ant as: $(\sum, \Gamma, Q, M, \Phi, F, q0, m0)$ where, input alphabet $\sum$ is defined as *(Pisition, stimuli, food)*. The set of outputs Γ is defined as set of messages *{got_hungry, become_inactive, giving_food, getting_food, …}*, Q is the set of states, memory of the agent is *(Position, carry, threshold, consumption, cangive)*, initial memory m_0 is defined as *((0, 0), 80, 20, 5, 10)*, start state q0 is "Inactive", (0,0) is assumed the position

of the inactive ant. The type Φ is a set of functions of the form: function_name (input_tuple, memory_tuple) → (output, memory_tuple').

$Q ::= Inactive \mid Hungry \mid Giving \mid Taking \mid Dead$

$Position: \mathbb{P}(\mathbb{Z} \times \mathbb{Z})$

$\forall a, b: \mathbb{Z} \mid 0 \leqslant a \wedge 0 \leqslant b \bullet (a, b) \in Position$

$MsgOut ::= got\_hungry \mid become\_inactive \mid giving\_food \mid getting\_food \mid ignoring \mid dying \mid looking\_for\_food \mid following\_pheromone \mid food\_taken \mid hungry\_ant \mid nha\_ant$

$SigmaOut: \mathbb{P}(MsgOut \times Position \times \mathbb{Z})$

$\forall msg: MsgOut; p: Position; f: \mathbb{Z} \mid f \geqslant 0 \bullet (msg, p, f) \in SigmaOut$

$Memory: \mathbb{P}(Position \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z})$

$\forall i, j, k, l: \mathbb{Z}; p: Position \mid 0 \leqslant i \wedge 0 \leqslant j \wedge 1 \leqslant k \wedge 1 \leqslant l \bullet (p, i, j, k, l) \in Memory$

$stimuli ::= space \mid pheromone \mid hungry \mid nh \mid ant \mid food$

$SigmaIn: \mathbb{P}(Position \times stimuli \times \mathbb{Z})$

$\forall p: Position; s: stimuli; f: \mathbb{Z} \mid f \geqslant 0 \bullet (p, s, f) \in SigmaIn$

Position is the current location of the agent; *MsgOut* is a set of possible output messages. *SigmaOut* contains the output message, current location of the agent and food quantity sharing with other agent. Similarly *SigmaIn* contains the possible new location, a *stimuli* that it perceived from the environment and food quantity received from a non-hungry ant. *Stimuli* is the possible set of perceptions that an agent perceived from the environment that may be space, pheromone trail, amount of sharing food quantity, a hungry or non-hungry ant. The formal specification using Z of an ant is as follow.

SXM

$states: \mathbb{P} Q$
$alphaIn: \mathbb{P} SigmaIn$
$alphaOut: \mathbb{P} SigmaOut$
$memory: \mathbb{P} Memory$
$function: SigmaIn \times Memory \rightarrow SigmaOut \times Memory$
$trans: Q \times (SigmaIn \times Memory \rightarrow SigmaOut \times Memory) \rightarrow Q$
$q0: Q; m0: Memory; T: \mathbb{P} Q$

$states \neq \{\} \wedge q0 \in states \wedge m0 \in memory \wedge T \subseteq Q$
$\forall q, q1: Q; d, d1: Memory; i: SigmaIn; o: SigmaOut \mid q \in states \; q1 \in states \wedge d \in memory \wedge d1 \in memory \wedge i \in alphaIn \wedge o \in alphaOut \bullet (i, d) \in \mathrm{dom}\ function \wedge (o, d) \in \mathrm{ran}\ function \wedge (q, function) \in \mathrm{dom}\ trans \wedge q1 \in \mathrm{ran}\ trans$

Ant

ΔSXM

search, follow_trail,become_hungry, ignore_ha, do_nothing, die, meet_nha, meet_ha: FUNCTION
take_enoughfood, give_food, take_less_food, no_foodto_give: FUNCTION

$q0 = Inactive \land\ m0 = ((0, 0), 80, 20, 5, 10)$
$function' = \{search, follow\_trail, become\_hungry, ignore\_ha, do\_nothing, die, meet\_nha, meet\_ha, no\_foodto\_give, take\_enoughfood, give\_food, take\_less\_food\}$
$\forall g: SigmaOut; s: SigmaIn; m, m1: Memory; x, y, f, carry, threshold, consumption, taken, cangive: \mathbb{Z}; p: Position; st: stimuli \mid s \in alphaIn \land g \in alphaOut \land m \in memory \land m1 \in memory' \land 0 \leqslant carry \land 1 \leqslant threshold \land 1 \leqslant consumption$
$\land\ 1 \leqslant cangive \land 0 \leqslant x \land 0 \leqslant y \land (x, y) = p \land (p, carry, threshold, consumption, cangive) \in Memory$
$\bullet\ (s = ((x, y), hungry, 0) \land threshold > carry - consumption$
$\Rightarrow m1 = ((x, y), carry - consumption, threshold, consumption, cangive) \land g = (got\_hungry, (x, y), 0) \land ((s, m), (g, m1)) \mid = become\_hungry) \land (s = ((x, y), space, 0) \land carry - consumption > 0 \land threshold > carry - consumption$
$\Rightarrow m1 = ((x, y), carry - consumption, threshold, consumption, cangive) \land g = (looking\_for\_food, (x, y), 0)$
$\land ((s, m), (g, m1)) = search) \land (s = ((x, y), pheromone, 0) \land carry - consumption > 0 \land threshold > carry - consumption$
$\Rightarrow m1 = ((x, y), carry - consumption, threshold, consumption, cangive) \land g = (following\_pheromone, (x, y), 0)$
$\land ((s, m), (g, m1)) = follow\_trail) \land (s = ((x, y), hungry, 0) \land carry - consumption \leqslant 0$
$\Rightarrow m1 = ((x, y), carry - consumption, threshold, consumption, cangive) \land g = (dying, (x, y), 0)$
$\land ((s, m), (g, m1)) = die) \land (s = ((0, 0), nonhungry, 0) \land carry - consumption > threshold$
$\Rightarrow m1 = ((0, 0), carry - consumption, threshold, consumption, cangive) \land g = (become\_inactive, (x, y), 0)$
$\land ((s, m), (g, m1)) = do\_nothing) \land (s = ((0, 0), nonhungry, 0) \land carry - consumption > threshold$
$\land carry - (consumption + cangive) < threshold \Rightarrow m1 = ((0, 0), carry - consumption, threshold, consumption, cangive) \land g = (become\_inactive, (x, y), 0) \land ((s, m), (g, m1)) = no\_foodto\_give) \land (s = ((x, y), ant, 0)$
$\land carry - (consumption + cangive) > threshold$
$\Rightarrow m1 = ((x, y), carry - consumption, threshold, consumption, cangive) \land g = (hungry\_ant, (x, y), 0)$
$\land ((s, m), (g, m1)) = meet\_ha) \land (s = ((x, y), space, 0) \land carry - (consumption + cangive) > threshold$
$\Rightarrow m1 = ((0, 0), carry - (consumption + cangive), threshold, consumption, cangive)$
$\land g = (giving\_food, (x, y), f) \land ((s, m), (g, m1)) = give\_food) \land (s = ((x, y), ant, 0) \land threshold > carry - consumption \mid \Rightarrow m1 = ((x, y), carry - consumption, threshold, consumption, cangive)$
$\land ((s, m), (g, m1)) = ignore\_ha) \land g = (ignoring, (x, y), 0) \land (s = ((x, y), ant, 0) \land threshold > carry - consumption$
$\Rightarrow m1 = ((x, y), carry - consumption, threshold, consumption, cangive) \land g = (nha\_ant, (x, y), 0)$
$\land ((s, m), (g, m1)) = meet\_nha) \land (s = ((x, y), food, f) \land threshold > carry - consumption$
$\Rightarrow m1 = ((x, y), carry + f - consumption, threshold, consumption, cangive) \land g = (food\_taken, (x, y), 0)$
$\land carry + f - consumption < threshold \land ((s, m), (g, m1)) = take\_less\_food) \land (s = ((x, y), food, f)$
$\land threshold > carry - consumption$
$\Rightarrow m1 = ((x, y), carry + f - consumption, threshold, consumption, cangive) \land g = (become\_inactive, (x, y), 0)$
$\land carry + f - consumption > threshold \land ((s, m), (g, m1)) = take\_enoughfood)$
$\forall q: Q; f: FUNCTION \mid q \in states' \land f \in function'$
$\bullet\ (q = Inactive \land f = do\_nothing \Rightarrow ((q, \{f\}), Inactive) \in trans)$
$\land (q = Inactive \land f = no\_foodto\_give \Rightarrow ((q, \{f\}), Inactive) \in trans)$
$\land (q = Inactive \land f = become\_hungry \Rightarrow ((q, \{f\}), Hungry) \in trans)$
$\land (q = Inactive \land f = meet\_ha \Rightarrow ((q, \{f\}), Giving) \in trans)$
$\land (q = Hungry \land f = ignore\_ha \Rightarrow ((q, \{f\}), Hungry) \in trans)$
$\land (q = Hungry \land f = search \Rightarrow ((q, \{f\}), Hungry) \in trans)$
$\land (q = Hungry \land f = follow\_trail \Rightarrow ((q, \{f\}), Hungry) \in trans)$
$\land (q = Hungry \land f = die \Rightarrow ((q, \{f\}), Dead) \in trans)$
$\land (q = Hungry \land f = meet\_nha \Rightarrow ((q, \{f\}), Taking) \in trans)$
$\land (q = Taking \land f = take\_less\_food \Rightarrow ((q, \{f\}), Hungry) \in trans)$
$\land (q = Taking \land f = take\_enoughfood \Rightarrow ((q, \{f\}), Inactive) \in trans)$
$\land (q = Giving \land f = give\_food \Rightarrow ((q, \{f\}), Inactive) \in trans)$

Invariants: (a) *Search, follow_trail, become_hungry, ignore_ha, do_nothing, die, meet_nha, meet_ha, no_foodto-give, take_enoughfood, give_food*, and *take_less_food* are the functions of an Ant. All the functions have the same type as the function of *SXM*. Each function is an element of *function'*, (b) the memory element *m0* is an initial memory state which shows that agent currently carrying maximum food quantity, its current location is *(0, 0)* and it can share some amount of food, its food consumption rate and threshold level, (c) The state *q0* is an initial state which is *Inactive*, (d) all the possible operations an agent can perform are defined as functions in which it takes an input of type *SigmaIn* and a *Memoey* element and generates a new memory status by altering the current memory and returns a message, (e) for each input *s* of type *SigmaIn* in the set of alphaIn of *SXM* a memory element m in memory of *SXM*, m1 is new state of memory *memory'* and output alphabet *g* in set of *SigmaOut*, there exist a partial function of type *Function* in *function'* where *(s, m)* in set domain of function and *(g, m1)* is in set range of function, and (f) for each partial function *f* in function' where *s* is input alphabet, *g i*s output alphabet and *m, m1* of type *memory*, and *q* of type *Q* in set states of an Ant, there exists a transition function trans' *((q, f), q1)* where trans acts on q and partial function *f* and returns a new state.

4.4. Design of a Communicating Ant

To specify the communicating ant agent we reused the specification of *Ant* and *CSXM* schema. There is only need to define the communicating functions of the machine through which it can communicate with other agents of the system.

CSXM

ΔSXM
$SISO: SigmaIn \times Memory \rightarrow SigmaOut \times Memory$
$SIOS: SigmaIn \times Memory \rightarrow SigmaOut \times Memory$
$ISSO: SigmaIn \times Memory \rightarrow SigmaOut \times Memory$
$ISOS: SigmaIn \times Memory \rightarrow SigmaOut \times Memory$
$is:$ seq (seq $SigmaIn$); $os:$ seq (seq $SigmaOut$)

$function' = SISO \cup SIOS \cup ISSO \cup ISOS$
$\forall i: \mathbb{N}; isq:$ seq $SigmaIn \mid i \geqslant 1 \wedge (i, isq) \in is \bullet$
$\forall j: \mathbb{N} \mid 1 \leqslant j \wedge j \leqslant \# isq \bullet isq\ j \in alphaIn$
$\forall i: \mathbb{N}; osq:$ seq $SigmaOut \mid i \geqslant 1 \wedge (i, osq) \in os \bullet$
$\forall j: \mathbb{N} \mid 1 \leqslant j \wedge j \leqslant \# osq \bullet osq\ j \in alphaOut$
$\forall i, j, k: \mathbb{N} \mid 1 \leqslant i \wedge 1 \leqslant j \leqslant k \bullet (\exists m, m1: Memory; sq:$ seq $SigmaIn; gq:$ seq $SigmaOut \mid m \in memory$
$\wedge\ m1 \in memory' \wedge sq \in$ ran $is \wedge gq \in$ ran $os \wedge \# sq \geqslant i$
$\wedge\ \# gq \geqslant i \bullet (sq\ i, m) \in$ dom $SISO \wedge (gq\ i, m1) \in$ ran $SISO)$
$\vee (\exists m, m1: Memory; sq:$ seq $SigmaIn; gq:$ seq $SigmaOut \mid m \in memory \wedge m1 \in memory' \wedge sq \in$ ran is
$\wedge\ gq \in$ ran $os \wedge \# sq \geqslant I \wedge \# gq \geqslant k \bullet (sq\ i, m) \in$ dom $SIOS \wedge (gq\ k, m1) \in$ ran $SIOS)$
$\vee (\exists m, m1: Memory; sq:$ seq $SigmaIn; gq:$ seq $SigmaOut \mid m \in memory \wedge m1 \in memory' \wedge sq \in$ ran is
$\wedge\ gq \in$ ran $os \wedge \# sq \geqslant j \wedge \# gq \geqslant i \bullet (sq\ j, m) \in$ dom $ISSO \wedge (gq\ i, m1) \in$ ran $ISSO)$
$\vee (\exists m, m1: Memory; sq:$ seq $SigmaIn; gq:$ seq $SigmaOut \mid m \in memory \wedge m1 \in memory' \wedge sq \in$ ran is
$\wedge\ gq \in$ ran $os \wedge \# sq \geqslant j \wedge \# gq \geqslant k \bullet (sq\ j, m) \in$ dom $ISOS \wedge (gq\ k, m1) \in$ ran $ISOS)$

CSXM

ΔSXM
$SISO: SigmaIn \times Memory \rightarrow SigmaOut \times Memory$
$SIOS: SigmaIn \times Memory \rightarrow SigmaOut \times Memory$
$ISSO: SigmaIn \times Memory \rightarrow SigmaOut \times Memory$
$ISOS: SigmaIn \times Memory \rightarrow SigmaOut \times Memory$
$is:$ seq (seq $SigmaIn$); $os:$ seq (seq $SigmaOut$)

$function' = SISO \cup SIOS \cup ISSO \cup ISOS$
$\forall i: \mathbb{N}; isq:$ seq $SigmaIn \mid i \geqslant 1 \wedge (i, isq) \in is \bullet$
$\forall j: \mathbb{N} \mid 1 \leqslant j \wedge j \leqslant \# isq \bullet isq\ j \in alphaIn$
$\forall i: \mathbb{N}; osq:$ seq $SigmaOut \mid i \geqslant 1 \wedge (i, osq) \in os \bullet$
$\forall j: \mathbb{N} \mid 1 \leqslant j \wedge j \leqslant \# osq \bullet osq\ j \in alphaOut$
$\forall i, j, k: \mathbb{N} \mid 1 \leqslant i \wedge 1 \leqslant j \leqslant k \bullet (\exists m, m1: Memory; sq:$ seq $SigmaIn; gq:$ seq $SigmaOut \mid m \in memory$
$\wedge\ m1 \in memory' \wedge sq \in$ ran $is \wedge gq \in$ ran $os \wedge \# sq \geqslant i$
$\wedge\ \# gq \geqslant i \bullet (sq\ i, m) \in$ dom $SISO \wedge (gq\ i, m1) \in$ ran $SISO)$
$\vee (\exists m, m1: Memory; sq:$ seq $SigmaIn; gq:$ seq $SigmaOut \mid m \in memory \wedge m1 \in memory' \wedge sq \in$ ran is
$\wedge\ gq \in$ ran $os \wedge \# sq \geqslant I \wedge \# gq \geqslant k \bullet (sq\ i, m) \in$ dom $SIOS \wedge (gq\ k, m1) \in$ ran $SIOS)$
$\vee (\exists m, m1: Memory; sq:$ seq $SigmaIn; gq:$ seq $SigmaOut \mid m \in memory \wedge m1 \in memory' \wedge sq \in$ ran is
$\wedge\ gq \in$ ran $os \wedge \# sq \geqslant j \wedge \# gq \geqslant i \bullet (sq\ j, m) \in$ dom $ISSO \wedge (gq\ i, m1) \in$ ran $ISSO)$
$\vee (\exists m, m1: Memory; sq:$ seq $SigmaIn; gq:$ seq $SigmaOut \mid m \in memory \wedge m1 \in memory' \wedge sq \in$ ran is
$\wedge\ gq \in$ ran $os \wedge \# sq \geqslant j \wedge \# gq \geqslant k \bullet (sq\ j, m) \in$ dom $ISOS \wedge (gq\ k, m1) \in$ ran $ISOS)$

CAnt

$\Xi CAnt$
$\Delta CSXM$

$SISO' = \{search, follow\_trail, become\_hungry, ignore\_ha, do\_nothing, die, meet\_nha, meet\_ha, no\_foodto\_give\}$
$SIOS' = \{give\_food\}$
$ISSO' = \{take\_enoughfood, take\_less\_food\}$
$ISOS' = \{\} \land is' = is \land os' = os$

Invariants: (a) In communicating X-machine of ant, the definition of functions *follow_trail, become_hungry, ignore-ha, do_nothing, die, meet_nha, meet_ha* and *no_foodto- give* remain same. The set of function *SISO,* reads and writes on standard input and output streams. It becomes *SISO'* which contains these functions, (b) the definition of function *take_less_food* and *take_enoughfood* are changed as it reads from communication input stream and writes on standard output stream. The set of functions *ISSO'* contains two functions *take_less_food* and *take_enoughfood*, (c) the set of functions *SIOS'* contains one function *give_food,* (d) the set of functions *ISOS'* is empty , and (e) the sequence of input and output streams *is* and *os* remains unchanged as *is'* and *os'*.

4.5. Communicating Stream X-machine System

A communicating X-machine system consists of a number of X-machines that can exchange messages with each other. A CSXMS is defined as Z= $((C_i)_{i=1,\ldots,n}, CR)$ where: i) C_i is a i-th communicating X-machine component. Ii) CR is a relation which defines the communication between the communicating X-machine components, i. e., CR ⊆ C × C and C = $\{C_1,\ldots, C_n\}$. A tuple $(C_i, C_k) \in CR$ denotes that the X-machine component C_i can output a message to a corresponding input stream of the X-machine component C_k for any i, k ∈ {1, . . . , n}, i ⌊ k.

The formal specification of abstract communicating stream X-machine systems (CSXMS) is described in schema Z where we introduced a variable C of type power set of CSXM to define a set of communicating stream X-machines. CR is a relation of type power set of *(CSXM × CSXM),* where *c* and *c1* are communicating stream X-machines which can communicate with each other.

Z

$C: \mathbb{P}\ CSXM$
$CR: \mathbb{P}\ (CSXM \times CSXM)$

$\forall c, c1: CSXM \mid c \in C \land c1 \in C \land c \mapsto c1 \in CR \land c \neq c1 \bullet (\exists m, m1: c\ .\ memory; s: SigmaIn; g: SigmaOut \mid s \in \text{ran}\ c\ .\ ist \land g \in \text{ran}\ c\ .\ ost \bullet (s, m) \in \text{dom}\ c\ .\ SISO \land (g, m1) \in \text{ran}\ c\ .\ SISO) \land (\exists m, m1: c\ .\ memory; s: SigmaIn; g: SigmaOut \mid s \in \text{ran}\ c\ .\ ist \land g \in \text{ran}\ c1\ .\ ost \bullet ((s, m) \in \text{dom}\ c\ .\ SIOS \land (g, m1) \in \text{ran}\ c\ .\ SIOS)) \land (\exists m, m1: c\ .\ memory; s: SigmaIn; g: SigmaOut \mid s \in \text{ran}\ c1\ .\ ist \land g \in \text{ran}\ c\ .\ ost \bullet ((s, m) \in \text{dom}\ c\ .\ ISSO \land (g, m1) \in \text{ran}\ c\ .\ ISSO)) \land (\exists m, m1: c\ .\ memory; s: SigmaIn; g: SigmaOut \mid s \in \text{ran}\ c1\ .\ ist \land g \in \text{ran}\ c1\ .\ ost \bullet ((s, m) \in \text{dom}\ c\ .\ SIOS \land (g, m1) \in \text{ran}\ c\ .\ ISOS))$

MultiAgentSystem

$C: \mathbb{P}\ CAnt$
$CR: \mathbb{P}\ (CAnt \times CAnt)$

$\forall ant: CAnt; x, y, carry, thr, con, sh: \mathbb{Z} \mid ant \in C \land x \geqslant 0 \land y \geqslant 0 \land carry \geqslant 0 \land con \geqslant 1 \land thr \geqslant 1 \land sh \geqslant 1 \bullet ant\ .\ q0 = Inactive \land ant\ .\ m0 = ((x, y), carry, thr, con, sh)$
$\forall ant, ant1: CAnt \mid ant \in C \land ant1 \in C \land ant \mapsto ant1 \in CR \land ant \neq ant1 \bullet (\exists m, m1: ant\ .\ memory; iq: \text{seq}\ SigmaIn; oq: \text{seq}\ SigmaOut; s: ant\ .\ alphaIn; g: ant\ .\ alphaOut \mid iq \in \text{ran}\ ant\ .\ is \land oq \in \text{ran}\ ant\ .\ os \bullet s \in \text{ran}\ iq \land g \in \text{ran}\ oq \land (s, m) \in \text{dom}\ ant\ .\ SISO \land (g, m1) \in \text{ran}\ ant\ .\ SISO) \land (\exists m, m1: ant\ .\ memory; iq: \text{seq}\ SigmaIn; oq: \text{seq}\ SigmaOut; s: ant1\ .\ alphaIn; g: ant\ .\ alphaOut \mid iq \in \text{ran}\ ant1\ .\ is \land oq \in \text{ran}\ ant\ .\ os \bullet (s \in \text{ran}\ iq \land g \in \text{ran}\ oq \land (s, m) \in \text{dom}\ ant\ .\ ISSO \land (g, m1) \in \text{ran}\ ant\ .\ ISSO)) \land (\exists m, m1: ant\ .\ memory; iq: \text{seq}\ SigmaIn; oq: \text{seq}\ SigmaOut; s: ant\ .\ alphaIn; g: ant1\ .\ alphaOut \mid iq \in \text{ran}\ ant\ .\ is \land oq \in \text{ran}\ ant1\ .\ os \bullet (s \in \text{ran}\ iq \land g \in \text{ran}\ oq \land (s, m) \in \text{dom}\ ant\ .\ SIOS \land (g, m1) \in \text{ran}\ ant\ .\ SIOS))$

Invariants: For each *c* and *c1* of type CSXM where *(c, c1)* is in relation CR, there exists *m, m1* of type memory of machine c. Input and output alphabets *s* and *g* are of type SigmaIn and SigmaOut respectively. If *s* belongs to the range of standard input stream *ist* of machine *c* and *g* belongs to the range of standard output streams *ost* of machine *c* then the partial function *(s, m), (g, m1)* belongs to function *SISO*. Further if *g* belongs to the output stream of machine *c1* then the partial function *(s, m), (g, m1)* belongs to *SIOS*. Similarly, if *s* belongs to the input stream of machine *c1* then the partial function *(s, m),(g, m1)* belongs to *ISSO*, otherwise the partial function *(s, m), (g, m1)* belongs to *ISOS*.

4.6. Design of Multi-agent System

A communicating multi-agent system consists of a number of communicating agents, which can exchange messages with each other. A communicating multi-agent system is defined as MultiAgentSystem= $((C_i)_{i=1,\ldots,n}, CR)$ where: (a) C_i is a i-th communicating agent, (b) *CR* is a relation which defines the communication between the communicating ant agents, i. e., $CR \subseteq C \times C$ and $C = \{C_1, \ldots, C_n\}$. A tuple $(C_i, C_k) \in CR$ denotes that the CAnt component C_i can output a message to a corresponding input stream of the CAnt component C_k for any i, $k \in \{1, \ldots, n\}$, $i \lfloor k$.

Invariants: (a) This specification shows that we are not defining all the agents from scratch, we just specify a single agent of one type and then we can create instances of these agents with different initial memory values and start state according to the requirement, (b) For each *ant* and *ant1* of type *CAnt* where *(ant, ant1)* is in relation CR, there exists *m, m1* of type memory of agent *ant*. Input and output alphabets *s* and *g* are of type *SigmaIn* and *SigmaOut* respectively. If *s* belongs to the range of standard input stream *is* of agent *ant* and *g* belongs to the range of standard output streams *os* of agent *ant1* then the partial function *(s, m), (g, m1)* belongs to function *SISO*. Further if *g* belongs to the output stream of agent *ant1* then the partial function *(s, m), (g, m1)* belongs to *SIOS*. Similarly, if *s* belongs to the input stream of agent *ant1* then the partial function *(s, m),(g, m1)* belongs to *ISSO*, otherwise the partial function *(s, m), (g, m1)* belongs to *ISOS*.

The formal specification of Multi-agent system is described in schema *MultiAgentSystem* where we introduced a variable C of type power set of *CAnt* to define a set of communicating ant agents. *CR* is a relation of type power set of *(CAnt× CAnt)*, where *ant* and *ant1* are communicating ant agents which can communicate with each other. In the specification of communicating multi-agent system we reused the predefined communicating ant agent models to design the complete system, therefore, there is no need to model the communicating agent model from scratch.

To control the dynamic behavior of the systems it is required to remember the current state, current memory status and last applied function of the agent. To fulfill this requirement we define the schema S which is a set of 3-tuples S={qc, mc, fc}, where qc is the state in which C_i is in, mc is the memory value of C_i, and fc is the last function that has been applied in C_i.

S

Ξ *CAnt*
qc: Q
mc: Memory
fc: FUNCTION

$qc \in states$
$mc \in memory$
$fc \in function$

SZ

$sz: \mathbb{P}\ S$

$\forall s: S \bullet s \in sz$

SZ is a state of a system of communicating agents which is defined as a set of S components which contains the three elements the current state, current memory and last applied function.

We specify the attachment operator in schema *Attachment* which establishes communication between two existing agents. It takes two communicating agents of type *Cant* as an input and establishes the communicating between these two agents.

Attachment

$\Delta CSXMS$
ci!: CAnt
cj!: CAnt

$ci! \in C$
$cj! \in C$
$CR' = \{(ci!, cj!)\} \cup CR$

Generation

$\Delta CSXMS$
c!: CAnt

$c! \notin C$
$C = \{c!\} \cup C$

This schema takes two agent *ci!* and *cj!* as input which belongs to C set of communicating agents and add to relation CR to establish the communication channels between them.
The scheme *Detachment* is defined to remove the communication channels of an existing agent from the system. it takes an agent of type *CAnt* as and input and verifies it that it must be an existing agent and removes its communicating channels from the system.

The *Generation* schema creates a new agent of type *CAnt* and added this agent into the system. The variable c! is defined to show that it is an input and it is not a part of the system. It does not belong to the set of communicating agents of the systems, which means that it is a newly created agent.

The destruction schema is defined to remove an excising CAnt agent from the system along with all the communication channels which allow it to communicating with the other agents of the system.
Prior to the removal of an agent from the systems, we have to remove its communicating channels from the system and then we remove the agent from the system.

Destruction

$\Delta CSXMS$
$c!: CAnt$

---

if $c! \in C$
then $CR' = \{c!\} \lhd CR \land CR' = CR \rhd \{c!\} \land C' = C \setminus \{c!\}$
else $C' = C \setminus \{c!\}$

5. RESULTS

Any specification written in a formal notation does not mean that it is correct, complete and meaningful. The remarkable feature of formal specification over all other traditional means of informal specification is that it can be checked and analyzed for the presence of typographical and syntactical errors. The Z/Eves tool provides various exploration techniques to prove the properties of a system. We focus on the mechanics of using Z/Eves and some model exploration techniques provided by Z/Eves tool to analyze the Z specification which we have used in this research. The Z/Eves Mathematical Toolkit includes the declaration of all the constants of the standard mathematical tools and provides a useful theorem proving facility about these constants as described in [22]. The Z/Eves is used to analyse the system's schema expansion, precondition calculation domain checking, syntax and type checking, and general theorem proving [23].

The syntax of Z language is quite complex and the mathematical toolkit contains a number of functions, symbols and key words. The syntax and type checking does not require to interact with theorem prover. Unlike many other tools, however, Z/Eves can be used incrementally such as each paragraph of a specification is written; it can be immediately checked and corrected if necessary [19]. Another kind of analysis domain checking is used to make sure that all the expressions appear in the specification are meaningful. For the domain checking, the Z/Eves examines each paragraph as it is entered and checks each function application and definition description for meaningfulness. To analyze the Z specification, the Z/Eves can also perform three types of reduction commands simplify, rewrite and reduce which can drastically change the goal predicate. Each of the command can apply a certain definitions or theorems to examine the formula and subformula of the goal, and may replace it by a logically equivalent formula which Z/Eves considers simpler. Prove by reduction is one of the analysis techniques used for analyzing the Z specification which provides the highest level of proof automation. This command repeatedly applies a commonly used combination to reduce the goal, until reduction has no effect.

The main display of the Z/Eves tool is the specification window which shows the formal part of a specification. In Figure7 the first left most columns shows the status of syntax correctness and in the second left most columns shows correctness of proof. The symbol "Y" shows that the paragraph is syntactically correct and has correct proof.

6. Conclusion

In this research the use of agent technology improved our ability to design and model the complex and distributed systems. To build the complete and accurate agent-based application we have integrated two modeling techniques X-machine and Z notation. This dual modeling approach supported the behavioral modeling, data modeling and property

analysis of the multi-agent systems. It can provide a common communication mechanism which allows the agents of the system to communicate with each other. The behavior, data and communication are modeled using stream X-machine and communicating stream X-machine respectively. To model a complete multi-agent system the communicating stream X-machine system's model is used. These X-machine models are specified and verified using Z notation. The developed formal agent models based on X-machines are also verified and analyzed using computer tool Z/EVES.

For the complete and accurate modeling of multi-agent system first we specified the abstract models of X-machine, stream X-machine, communicating stream X-machine and communicating stream X-machine system in Z notation. We have used the Z notation in a modular fashion in such a way that to specify the communicating stream X-machine we have reused the predefined specification of stream X-machine. In communicating stream X-machine, communication and data are considered as two separate distinct activities which provide the benefit of reusability of stream X-machine models. The communicating stream X-machine system facilitates the designer to reused the predefined communicating stream X-machine models to design the complete system, therefore, there is no need to model the communicating stream X-machine models from scratch. These X-machine models are also reused to specify the stand-alone agent, communicating agent and a complete multi-agent system consisting on communicating agents. The rebuilding of any multi-agent system requires only modeling the communication part of the system. The formal specification of agent models is analyzed for the correctness of type, syntax, proof and completeness using Z/EVES. The type and syntax is checked for typographical and syntactical errors respectively. The domain checking is performed for the completeness of the specification. The syntax and type checking was performed automatically but to perform domain checking, in some cases we interacted with theorem prover.

This relationship reduced the implementation issues of X-machine models. It is believed that the use of this formal technique for multi-agent system modeling and verification will increase the confidence that the developed system has the required characteristics. These models will be used for formal testing of the multi-agent systems to prove the correctness of the model. Further, it will increase the confidence in correctness and completeness for the construction of multi-agent systems. This integration will also be useful for modeling the various applications such as modeling intelligent agents, object oriented testing, parallel processing, multi-agent systems, reverse engineering, agent oriented testing and modeling the complex distributed systems.

7. REFERENCES

[1] Isaksen, U., Bowen, J. P. and Nissanke, N. 1996. Systems and Software Safety in Critical Systems, Department of Computer Science, the University of Reading.

[2] Williams, L. G. 1992. Formal Methods in the Development of Safety Critical Software Systems, Technical Report SERM-014-91, Software Engineering Research, Boulder, CO.

[3] Mitkas, P. 2005. Knowledge discovery for training intelligent agents: methodology. tools and applications, autonomous intelligent systems: Agents and Data Mining, pp. 2-18.

[4] Wooldridge, M. N., R. Jennings, and D. Kinny. 2000. The gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems, Vol. 3, pp. 285-312.

[5] Trencansky, I., and R. Cervenka. 2005. Agent modeling language (AML): a comprehensive approach to modeling MAS. Informatics 29, pp. 391–400.

[6] Stamatopoulou, I., M. Gheorghe, and P. Kefalas. 2005. Modelling of dynamic organization of biology-inspired multi-agent systems with communicating X-machines and P systems. 5th Workshop in Membrane Computing, Vol. 3365, 389-403, Springer-Verlag, Berlin, Milan Italy.

[7] Ipate, F. and Gheorghe, M. 2009. Testing Non-deterministic Stream X-machine Models and P systems. Electronic Notes in Theoretical Computer Science, Vol. 227, pp: 113-126, Elsevier Science Ltd.

[8] Ipate, F. and Gheorghe, M. 2008. Testing data processing-oriented systems from stream X-machine models. Theoretical Computer Science, Vol. 403, pp: 176-191, Elsevier Science Ltd.

[9] Moussavi. M. 1992. A computer simulation model for urban transportation planning process. Advances in Engineering Software archive, Vol. 14, No. 3, pp: 213-218, Elsevier Science Ltd.

[10] Gowri, A., Venkatesan, K. and Sivanandan, R. 2009. Object-oriented methodology for intersection simulation model under heterogeneous traffic conditions. Advances in Engineering Software, Vol. 40, No. 10, pp: 1000-1010, Elsevier Science Ltd.

[11] Kefalas, P., Holcombe, M., Eleftherakis, G. and Gheorghe, M. 2003. A formal method for the development of agent-based systems. Intelligent Agent Software Engineering, Chapter 4, Idea Group Publishing, pp. 68-98.

[12] France, R., Evans, A., Lano, K. and Rumpe, B. 1998. The UML as a formal modeling notation. Lecture Notes In Computer Science; Vol. 1618, pp: 336 - 348, Springer-Verlag.
[13] Eilenberg, S. 1974. Automata, machines and languages. Vol. A, Academic Press.
[14] Holcombe, M. 1988. X-Machines as a basis for dynamic system specification. Software Engineering Journal, Vol. 3, No. 2, pp. 69-76, Springer-Verlag.
[15] Khan SA, Zafar NA (2007). Promotion of local to global operation in train control system. Journal of Digital Information Management, 5(4):231-236.
[16] Khan SA, Zafar NA (2011). Improving moving block interlocking system using fuzzy multi agent specification language. International Journal of Innovative Computing, Information and Control, 7(7B): 4517-4534
[17] Khan SA, Zafar NA, Ahmad F (2011). Petri net modeling of railway crossing system using fuzzy brakes. International Journal of Physical Sciences , 6(14): 3389–3397.
[18] Kyu, J. L., and Y. C. Sik. 2006. A framework of optimization agent for supply chain management. Multiagent Based Supply Chain Management.
[19] Chiu, D. K.W., S. C. Cheung, and H. Leung. 2005. A multi-agent infrastructure for mobile workforce management in a service oriented enterprise. Proceedings of the 38th Hawaii International Conference on System Sciences, pp. 85-87.
[20] Decker, K., K. Sycara, and D. Zeng. 1996. Designing a multi-agent portfolio management system. Proceedings of the Intelligent Adaptive Agents.
[21] Varghese, B., and G. T. McKee. 2009. Applying autonomic computing concepts to parallel computing using intelligent agents. Proceeding of World Academy of Science, Engineering and Technology, pp. 366-370.
[22] Saaltink, M. 1999. The Z/Eves 2.0 mathematical toolkit. TR-99-5493-05b, ORA Canada.
[23] Saaltink, M. 2006. The Z/Eves system. The Z Formal Specification Notation, Springer Berlin, pp. 72-85.