

Using X-machines to model and test discrete event simulation programs

KEHRIS E., ELEFThERAKIS G., KEFALAS P.

Computer Science Department

CITY LIBERAL STUDIES - Affiliated College of the University of Sheffield

13 Tsimiski st., 54624 Thessaloniki

GREECE

{kehris, eleftherakis, kefalas}@city.academic.gr <http://www.city.academic.gr>

Abstract: - Simulation is a powerful technique for the study of real-life complex problems. Simulation requires the development of a program, which mimics the problem under consideration. The simulation program development follows closely the software engineering process. Although, many techniques for modelling problems have been proposed, the testing phase of the developed programs has been under-discussed. In this paper, we introduce an alternative framework for modelling and simulation, which will facilitate the phases of the conceptual modelling and the verification of the software. X-machines is a formal technique which extends the Finite State Machines by introducing memory attached to the machine and functions that operate on input symbols and memory values. This is shown to be a powerful model, subsuming other well-known models, since not only the specification of a problem is done in an intuitive manner but also because the method is used to prove that the implementation is correct with respect to the specification. A simple example is thoroughly investigated in terms of modelling and testing by employing the X-machine formal methodology. A complete set of test-cases is generated and applied in the code which is written in a conventional programming language. It is demonstrated that faults are found in the implementation by comparing the output sequences to those intended by the specification. Finally, a discussion is made in order to propose further course of research in this area, which would add the necessary characteristics in the X-machine theory in order to serve modelling and simulation development to its full extent.

Key-Words: - Formal specification, Testing, Simulation, Verification

1 Introduction

A simulation project evolves around three basic building blocks [1]: the real-life problem, the conceptual model and the computer model, where each basic block is derived from another (Fig.1). The conceptual model is the deliverable of the analysis and modelling of the real-world problem. Based on the conceptual model, programming and implementation produces a computer program. Finally, conclusions and/or suggestions about the real-life problem are derived through the experimentation with the computer simulation program.

The correctness of the derivations is established through the appropriate validation and verification processes (solid lines in Fig. 1). Software verification is the "substantiation that the implementation of the computer program of a model is correct and performs as intended" [2]. The process of verification aims to assure that the implementation matches the specification, which in this case is the conceptual model. According to Balci [3], the validation and verification techniques

are categorised as informal, static, dynamic, symbolic, constraint and formal.

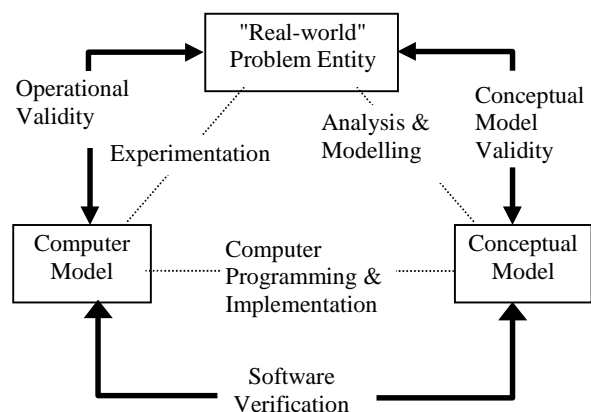


Fig. 1: The Sargent Framework

Dynamic techniques require the execution of the simulation program and most of them impose the insertion of additional code (probes) into the executable model for collecting information about model behaviour during execution [3]. Information

produced during the execution of a simulation program is called a *trace*. The trace is one of the most powerful techniques [4] that can be used to debug a discrete-event simulation program [5]. Issues that need to be considered when the trace is used for testing the correctness of a simulation program include the determination of the total simulated time and the values of the system parameters.

We propose a different approach to program simulation modelling in order to address the above-mentioned issues. By using a formal method, namely the X-machine, we argue that the description of the conceptual model as well as the test-case generation can be achieved through the same model. X-machine specification is an intuitive method to describe simulation programs through the use of states that possess memory and transitions between those states, which are activated by functions. Furthermore, it has been proved that under some conditions the X-machine method provides a complete test-suite, i.e. it generates all the necessary and sufficient test-cases in order to establish the correctness of the program.

2 The Conceptual Model

A number of modelling techniques have been proposed for the development of conceptual models. Some of them are semiformal, e.g. the Activity Cycle Diagram, while others are characterised as formal, e.g. Petri-Nets. An evaluation of these techniques may be found in [6]. However, these models provide limited support during the verification stage. To overcome this problem the X-machine specification formalism has been adopted in this work.

2.1 X-machines for Specification

There are several formal techniques (Z, VDM, Finite State Machines etc) that may be used for the specification of computer software, each one possessing advantages in describing either the static or the dynamic part of a system [7]. Thus, the majority of formal specification languages facilitate the modelling of either the data processing, or the control of the system. X-machines is a specification formalism introduced by Eilenberg [8], which is capable to model both the data and the control by integrating specification methods which describe each of these aspects in the most appropriate way. X-machines employ a diagrammatic approach to model the control by extending the expressive power

of the Finite State Automata (FSA). Transitions between states are no more performed through simple input symbols but through the application of functions. These functions are written in a formal notation and model the processing of the data. Data is held in memory, which is attached to the X-machine. Functions receive input symbols and memory values, and produce output while modifying the memory values.

Although initially introduced in 1974, X-machines did not receive interest until 1988 when Holcombe proposed this model as a basis for a possible specification language [9]. In 1992, stream X-machines were defined as X-machines with input and output sets of streams of symbols. In short, the idea was that the machine has infinite internal memory and depending on the current state of control and the current state of the memory, an input symbol from the input stream determines the next state, the new memory state and the output symbol which will be part of the output stream. The formal definition of a deterministic stream X-machine [10] is given in table 1.

A stream X-machine is a 8-tuple:

$M = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:

- Σ, Γ is the input and output finite alphabet respectively,
- Q is the finite set of states,
- M is the (possibly) infinite set called memory,
- Φ is the type of the machine M , a finite set of partial functions ϕ that map a memory state and an input to a new memory state and an output, $\phi: M \times \Sigma \rightarrow \Gamma \times M$
- F is the next state partial function that given a state and a function from the type Φ , denotes the next state. F is often described as a transition state diagram.
$$F: Q \times \Phi \rightarrow Q$$
- q_0 and m_0 are the initial state and memory respectively.

Table 1: Definition of a Stream X-machine

The associated automaton $A=(\Phi, Q, F, q_0)$ of an X-machine $M = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ is defined as the conversion of the X-machine M to a FSA by treating the elements of Φ as abstract input symbols. It has been shown that Turing machines, pushdown automata and finite state machines are all special cases of the X-machine model [11].

2.2 X-Machines for Testing

Ipate and Holcombe [10] presented a testing method, which is a generalisation of Chow's W-method [12] for finite state machine testing. It is

proved that this testing method finds all faults in the implementation [13]. The method works based on the following assumptions:

- the specification and the implementation of the system can be represented as X-machines,
- the machine corresponding to the specification and the machine corresponding to the implementation have the same type Φ .

Assuming the above, the method also requires that:

- the specification satisfies the design for test conditions, and
- its associated automaton is minimal.

The design for test conditions state that the type Φ of the two machines is both complete with respect to M (table 2) and output distinguishable (table 3).

A processing function $\varphi \in \Phi$ is called complete w.r.t. M if:

$$\forall m \in M, \exists \sigma \in \Sigma \text{ such that } (m, \sigma) \in \text{dom } \varphi$$

A type Φ is called complete w.r.t. M if any basic function will be able to process all memory values, that is if:

$$\forall \varphi \in \Phi, \varphi \text{ is complete w.r.t. } M$$

Table 2: Type Φ complete w.r.t. M

A type Φ is called output distinguishable if any two different processing functions will produce different outputs on each memory/input pair, that is if

$$\begin{aligned} \forall \varphi_1, \varphi_2 \in \Phi \text{ if } \exists m \in M, \sigma \in \Sigma \text{ such that for some} \\ m_1', m_2' \in M, \gamma \in \Gamma \\ \varphi_1(m, \sigma) = (\gamma, m_1') \text{ and } \varphi_2(m, \sigma) = (\gamma, m_2'), \\ \text{then } \varphi_1 \neq \varphi_2. \end{aligned}$$

Table 3: Output-distinguishable Φ

When these requirements are met, the W-method may be employed to produce the k-test set X of the associated automaton, where k is the difference of the number of states of the two associated FSAs. The test-set X consists of sequences of inputs for the associated automaton. The fundamental test function is defined recursively (table 4), and converts these sequences into sequences of inputs of the X-machine. The produced test-set is proved to find all faults in the implementation. The testing process can be therefore performed automatically by checking whether the output sequences produced by the implementation are identical with the ones expected from the specification.

The methodology described above is applied to a simple problem adopted from [14], in order to demonstrate its suitability for specifying and testing a simulation program. We argue that X-machines provide a convenient and intuitive way to specify the conceptual model and then test the implementation against it.

Let $M = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic stream X-machine with Φ complete w.r.t. M and let $q \in Q, m \in M$. A function $t_{q,m}: \Phi^* \rightarrow \Sigma^*$ will be defined recursively as:

$$t_{q,m}(\varepsilon) = \varepsilon \text{ (the empty input symbol)}$$

or

$$t_{q,m}(\varphi_1 \dots \varphi_n \varphi_{n+1}) = \begin{cases} t_{q,m}(\varphi_1 \varphi_2 \dots \varphi_n) \sigma_{n+1}, \\ \text{if } \exists \text{ a path } q, q_1, \dots, q_{n-1}, q_n, \\ \text{in } M \text{ starting from } q, \\ \text{where } \sigma_{n+1} \text{ is such that} \\ (m_n, \sigma_{n+1}) \in \text{dom } \varphi_{n+1} \\ \text{and } m_n \text{ is the final value} \\ \text{computed by the} \\ \text{machine along the above} \\ \text{path on the input} \\ \text{sequence } t_{q,m}(\varphi_1 \varphi_2 \dots \varphi_n) \\ t_{q,m}(\varphi_1 \varphi_2 \dots \varphi_n), \text{ otherwise} \end{cases}$$

and this function $t_{q,m}$ will be called a *test function of M w.r.t. q and m* .

Table 4: Fundamental test function of a X-machine

3 Simulation of a Bank Cashier

The scenario of the problem is as follows: A bank employs a cashier that has two responsibilities: (a) to serve the customers that arrive in the bank and (b) to answer to incoming phone calls. It is assumed that the customers are waiting in front of the cashier in a queue and that there is at most one phone to be answered. If the cashier is busy (either serving a customer or answering a phone call) and a customer arrives, she joins a queue and remains until she is served. If the cashier is serving a customer and the phone rings, the cashier completes the service of the customer and then he answers the phone, independently of the customer queue cardinality.

3.1 Formal description

The state transition diagram of the stream X-machine corresponding to the system described above is shown in figure 2.

The X-machine's input set is:

$$\Sigma = \{\text{customer, bell_sound, leave, done, click, answer}\}$$

where

- `customer` models the arrival of a new customer
- `bell_sound` models a new incoming phone
- `leave` models the departure of a customer while there are other customers waiting to be serviced and the phone is not ringing
- `done` models the completion of a service and no further service is requested from the cashier

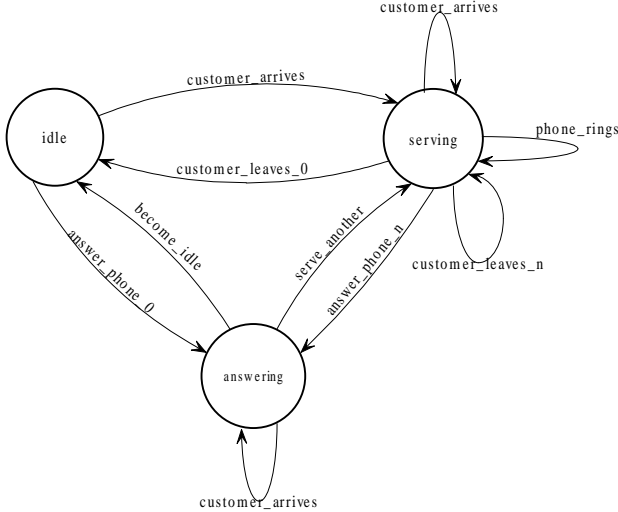


Fig.2: The state transition diagram of the stream X-machine specification of a bank cashier

- click models the completion of a phone answer when there are waiting customers.
- answer models the completion of a customer service and there is a phone ring to be answered by the cashier.

The output of the system consists of a set of messages that are displayed on the screen

$$\Gamma = \{m1, m2, m3, m4, m5, m6, m7, m8\}$$

where:

- m1=CustomerArrives
- m2=PhoneRingsCashierIsBusy
- m3=CustomerServedCashierisBusyNoPhone
- m4=CustomerServedCashierAnsweringPhone
- m5=ClosePhoneServeAnotherCustomer
- m6=ClosePhoneCashierBecomesIdle
- m7=AnsweringPhone
- m8=CustomerServedCashierIsIdleNoPhone.

The set of states is:

$$Q = \{\text{idle}, \text{serving}, \text{answering}\}$$

The X-machine's memory M is:

$$M = \text{QUEUE} \times \text{BELL}$$

where:

- QUEUE: seq customer is the queue of customers waiting to be served,
- BELL is a boolean variable notifying whether the phone rings while the cashier is serving a customer.

Initially, the cashier is idle, no customers are waiting in the queue and no phone is ringing:

$$q_0 = \text{idle}, m_0 = (\langle \rangle, \text{false})$$

The next state function $F: Q \times \Phi \rightarrow Q$ is shown diagrammatically in Fig.2.

Finally, the functions of the X-machine need to be defined. The X-machine functions get as input an input event and the current state of the memory, and they produce an output and a new memory:

$$\varphi : \Sigma \times M \rightarrow \Gamma \times M$$

The functions are defined in Table 5. The notation used is that of X-Machine Description Language [15], which is intended to be an interchange language between X-Machine tools [16]. Briefly, the functions take two parameter tuples, i.e. an input symbol and a memory value, and return two new parameter tuples, i.e. an output and a new memory value. A function may be applicable under conditions (if-then) or unconditionally. Variables are denoted by ?. The informative where in combination with the operator <- is used to describe operations on memory values.

<pre> customer_arrives((?c),(?queue, ?b)) = if ?c belongs customer then ((m1),(?new_queue, ?b)) where ?new_queue <- add(?queue, ?c). customer_leaves_0((done),(?c1:nil,false)) = ((m8),(nil, false)). phone_rings((bell_sound), (?queue, ?b)) = ((m2),(?queue, true)). customer_leaves_n((leave),(?c::?q,false)) = if not_is_empty ?q then ((m3),(?q, false)). answer_phone_n((answer),(?c::?q,true)) = ((m4),(?q, false)). serve_another((click), (?queue, false)) = if not_is_empty ?queue then ((m5),(?queue, false)). become_idle ((done), (nil, false)) = ((m6),(nil, false)). answer_phone_0((bell_sound),(nil,false)) = ((m7),(nil, false)). </pre>

Table 5: Definitions of functions

3.2 Generation of Test-Cases

The testing method described in section 2 is used to produce the test-cases using the above X-machine. Since the design for test conditions are satisfied by the developed X-machine, in order to produce the 0-test-set of the associated automaton, a characterisation set and a state cover set are required. Informally, a characterisation set $W \subseteq \Sigma^*$ is a set of input sequences for which any two distinct states of the machine are distinguishable.

<p>Characterisation Set $W = \{\text{phone_rings}, \text{serve_another}\}$ State Cover Set $S = \{\varepsilon, \text{customer_arrives}, \text{customer_arrives phone_rings answer_phone}\}$</p>

Table 6: A characterisation and a state cover set

The state cover $S_{\subseteq \Sigma}^*$ is a set of input sequences such that all states are reachable by q_0 . The W and S sets in the bank cashier X-machine are shown in table 6. Table 7 shows the 0-test-set, which is produced from W and S and the application of the fundamental test function. The same table also shows the output messages for each input sequence. It should be noted that these were generated automatically with the use of the automated tools mentioned above.

No	Input Sequence	Output Sequence
1	customer	m1
2	customer, customer	m1, m1
3	bell_sound	m7
4	customer, bell_sound	m1, m2
5	customer, customer, bell_sound	m1, m1, m2
6	customer, done, bell_sound	m1, m8, m7
7	customer, done	m1, m8
8	customer, bell_sound, answer, customer	m1, m2, m4, m1
9	customer, bell_sound, answer, customer, click	m1, m2, m4, m1, m5
10	customer, bell_sound, answer	m1, m2, m4
11	customer, bell_sound, answer, done, bell_sound	m1, m2, m4, m6, m7
12	customer, bell_sound, answer, done	m1, m2, m4, m6

Table 7: Input and the expected output sequences

4 Implementation and Testing

A deterministic simulation program for the bank cashier problem was written in the C programming language. The system has been developed using the SIMSYS Library [17], which implements the three-phase approach [18]. The simulation program parameters are the customer inter-arrival time (CIAT), the service time (ST), the phone call inter-arrival time (PIAT) and the time required to answer a phone call (AT). The testing simulation program produces the same output messages as the X-machine so that the testing phase is facilitated.

The implementation has been tested with all the test cases described in table 7. In order to show the effectiveness of the approach taken by specifying the problem as X-machine, a number of errors were intentionally introduced in the simulation program. The errors are categorised into two groups: operational errors (labelled as [O]) and transition errors (labelled as [T]) and are shown in Table 8. The test-sequences shown in table 7 were used for the testing of the simulation program. Thus, the program was executed a number of times. Each program execution generated a specific sequence of events that corresponded to a test-sequence. For

each program execution the parameters (i.e. the CIAT, ST, PIAT, AT) and the total simulation time were set so as to produce the desired test-sequence. For example, the test-sequence 2 (table 7) was generated by setting the CIAT, ST, PIAT, AT and total simulated time to 5, 7, 13, 2 and 10. respectively. The output produced by each simulation program execution was checked against the output sequence (table 7). All the introduced errors were identified by the generated input sequences.

Error introduced in the simulation program	Error Type	Identified By test #
Phones that arrive do not set the bell flag to true	O	3
Answering the call does not unset the bell to false	O	12
When customer arrives she does not join the waiting queue	O	7
When one customer arrives, two customer are added in queue	O	7
When a customer leaves, the cashier never becomes idle	T	7
When the cashier serves a customer he remains idle	T	4
Customer leaves but cashier becomes idle later	T	7

Table 8: The seven errors introduced and the input sequence that identified them.

5 Discussion and Further Work

X-machine is a formal specification method that possesses sufficient expressive power for the development of the conceptual model required during a simulation study. The method, in contrast to other existing modelling approaches, provides direct support for the testing of the simulation program.

The X-machine may be extended to model real-life systems in a more natural way. To this end, research concerning the development of communicating X-machines is currently carried out [19]. The expressive power of X-machines is equal to that of the communicating X-machines. The latter resemble more naturally the description of a system since they allow the decomposition of a system into sub-systems and thus, it facilitates the conceptual model development phase. A second research direction concerns the incorporation of the notion of time in the X-machine model, so that simulation programs may be automatically derived from the X-machine specification. Finally, work is under progress [20] in model checking [21] X-machines. By proving that certain desired properties are satisfied by the model, we will eventually encapsulate the analysis and

modelling phase and the computer programming and implementation phase as well as the corresponding validation and verification processes under a single framework.

6 Conclusion

X-machines can be viewed as an appropriate alternative framework for modelling and simulation problems. The power of X-machines lies on three characteristics: (a) they extend the finite state machine model, which is a diagrammatic and intuitive model for specifying the control of a system; (b) they model the data of system by means of a memory attached to all states of the system and (c) they can be used to produce all test cases for a system. By means of an example problem it is shown that the X-machine model can be employed in all stages of the simulation program development, ranging from the specification of the conceptual model to the verification of the software.

References:

- [1] Sargent R.G., "An Overview on Verification and Validation Models", In *Proceedings of the 1985 Winter Simulation Conference*, Society for Computer Simulation, 1985.
- [2] Knepell P.L. and Arangno D.C., *Simulation Validation: A Confidence Assessment Methodology*, IEEE Computer Soc. Press, 1993.
- [3] Balci O., "Validation, Verification and Testing Techniques Throughout the Life Cycle of a Simulation Study", In *Annals of Operations Research*, vol. 53, 1994, pp. 121-175.
- [4] Law A. and Kelton W., *Simulation Modelling and Analysis*, McGraw-Hill, 1991.
- [5] Kehris E., "Computer-Aided verification of simulation programs based on their trace", *10th European Simulation Symposium*, 1998.
- [6] Page E., *Simulation modelling methodology: principles and etiology of decision support*, PhD Thesis, Dept. of Computer Science, Virginia Polytechnic Institute, 1994.
- [7] Clarke E., Wing J. M., "Formal Methods: State of the Art and Future Directions", *ACM Computing Surveys*, Vol.28, No.4, December 1996, pp. 626-643.
- [8] Eilenberg S., *Automata Machines and Languages*, Vol. A, Academic Press, 1974.
- [9] Holcombe M., "X-machines as a basis for dynamic system specification", *Software Engineering Journal*, Vol.3, No.2, 1988, pp. 69-76.
- [10] Ipaté F. and Holcombe M., "Specification and testing using generalised machines: a presentation and a case study", *Software Testing, Verification and Reliability*, Vol.8, 1998, pp. 61-81.
- [11] Holcombe M. and Ipaté F., *Correct Systems: Building a Business Process Solution*, Springer Verlag, London, 1998.
- [12] Chow T.S., "Testing Software Design Modeled by Finite-State Machines," *IEEE Transactions on Software Engineering*, Vol.SE-4, No.3, 1978, pp.178-187.
- [13] Ipaté F. and Holcombe M., "An integration testing method that is proved to find all faults," *International Journal of Computer Mathematics*, Vol.63, No.3, 1997, pp. 159-178.
- [14] Pidd M., *Computer Simulation in Management Science*, Wiley, 1998.
- [15] Kefalas P. and Kapeti E., "A Design Language and Tool for X-Machines Specification", (to appear) In *Advances in Informatics* edited by D.I. Fotiadis, S.D. Nikolopoulos, World Scientific Publishing Company, 2000.
- [16] Kefalas P., "Automatic Translation from X-Machines to Prolog", Technical Report TR-CS01/2000, Dept. of Computer Science, CITY Liberal Studies, January 2000
- [17] Crookes J. G., *Simulation using C*, In: *Computer modelling for discrete simulation* Ed. M. Pidd, Wiley, 1989.
- [18] Tocher K., *The Art of Simulation*, Van Nostrand Company, Princeton NJ, 1963.
- [19] Balaneascu T., Cowling A., Gheorgescu H., Gheorghe M., Holcombe M. and Vertan C., "Communicating Stream X-machines Systems are no more than X-machines", *Journal of Universal Computer Science*, Vol.5, No. 9, 1999, pp. 494-507.
- [20] Eleftherakis G., "Model Checking and X-Machines", Technical Report TR-CS02/2000, Dept. of Computer Science, CITY Liberal Studies, January 2000.
- [21] Clarke E. M., Emerson E. A., and Sistla A. P., "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications", *ACM Transactions on Programming Languages and Systems*, Vol.8, No.2, 1986, pp. 244-263.