MEDINFO 2001
V. Patel et al. (Eds)
Amsterdam: IOS Press

A Formal Framework for Modelling and Validating Medical Systems

George Eleftherakis

CITY Liberal Studies, Affiliated College of the University of Sheffield, Greece

Abstract

Medical computerised systems which have a major effect on human lives (e.g. those used for diagnosis, therapy, surgery, in the intensive care units, etc) are considered as safety critical systems. Such systems are sometimes responsible for major damages and injuries due to unpredicted malfunction. Misleading user requirements, errors in the specification and in the implementation are the usual reasons responsible for non-safe systems. This paper advocates the use of an integrated formal framework based on a computational machine (X-Machine), in the development of safety critical medical systems. This formal framework gives the ability to intuitively as well as formally model a system, then automatically check if the produced model has all the desired properties, and finally test if the implementation is equivalent to the specification by applying a complete set of test cases. Therefore, the use of this framework in the development of systems in safety critical medical domains can assure that the final product is valid with respect to the user requirements by revealing errors during the whole development life cycle and subsequently add to the confidence of their use. The proposed framework is accompanied by an example, which demonstrates the use of X-Machines in specification, testing and verification.

Keywords:

Medical Informatics; Formal Methods; Safety Critical Systems

Introduction

In *safety critical* domains, such as medical, space, nuclear etc., the existing practical development techniques failed several times to produce a reliable and correct system. Errors in such systems not only cause an increase in the cost of a project, but are also responsible for accidents, even provoking deaths in some rare cases [1]. Among others, the most prominent reasons are:

- Misunderstood user requirements may lead to a potential "correct" specification and implementation, which however does not meet the actual requirements.
- Errors in the specification of the system result to a model different from the one needed.
- Errors in the implementation lead to a different product from the one specified.

The last few decades there was a vivid debate on whether formal methods are needed or not. These debates were triggered by academics and practitioners, adopting extreme positions either for or against them [2]. Over the last years it is recognised internationally that the truth lies somewhere between and that there is a need for use of formal methods in software engineering while there are several cases proving the applicability of formal methods in industrial applications [3].

The application of formal methods in safety critical medical systems could add to the confidence in using the system by revealing errors in both the system's modelling and its implementation [4]. Several examples of successful application of formal methods in medical applications support this argument [5, 6]. Assuming that the user requirements reflect the desired system, there is a need for an integrated formal technique that will allow us to:

- formally specify the system,
- check that the specification has the desired properties,
- check the implementation against the specification.

In this paper, we propose the use of a formal method, namely X-Machines, which can accommodate all three above-mentioned activities. X-Machines is an intuitive, yet rigorous formalism. More specifically, a X-Machine is a general computational machine that is like a Finite State Machine but with a significant difference; transitions are not labelled with simple inputs but with functions that operate on inputs and a memory, allowing the machine to be more expressive and flexible than a simple automaton. X-Machines are able to model both the control and the data part of a system. The framework for formal development proposed in this paper uses X-Machines as a formal

vmodelling language [7], a testing strategy to check the implementation against the X-Machine model [8] and a verification technique to prove the validity of the model [9]. We argue that, by applying the proposed framework, to safety critical systems it is possible to assure that several "safety" properties hold in the final product.

A Framework for Formal Development

The framework suggested to be built around X-Machines is depicted in Figure 1. The grey shaded areas are tasks in which X-Machines are used as the core formal method. First of all, from the user requirements the system is described as an X-Machine model. Then a verification technique for X-Machines (model checking X-Machine models) verifies that certain safety properties hold in the model and feedback is used to adjust the model. The actual implementation task produces the code in a programming language. Finally, the testing of the implementation for correctness with respect to the model takes place and the refinement of the implementation, through the use of a complete test set derived from the X-Machine model.

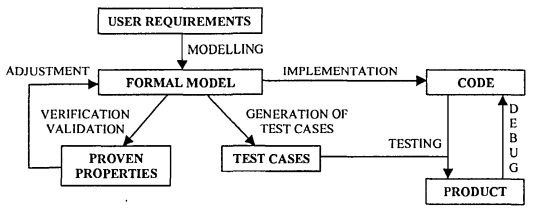

Figure 1 - The proposed formal framework based on X-Machines that supports the development of medical systems

This formal framework which aids in the development of safety critical systems will be demonstrated through a simplistic example of a X-Machine that will be used as a vehicle of study. A medical ray beaming system (i.e. a radiotherapy LINAC) is controlled using three buttons: i) one for charging the machine (a single button press increases the voltage by a 10 mV step), ii) one for the beam activation and iii) one for resetting the machine at any time. The system will only beam if the charge in mV has reached a pre-set maximum, e.g. 30mV. Any attempts to increase the charge of the machine should be rejected, since there is a danger to seriously injure the patient.

Modelling

Modelling a system means to create an appropriate descriptive adequate specification. *Formal modelling* is the procedure of describing a system and its desired properties precisely, by using a language with rigorously defined syntax and semantics. Logic and sets often form the basic theory behind such mathematical languages, in order to avoid ambiguity and allow automated verification techniques on the model. The majority of formal languages facilitate the modelling of either the data processing, or the control of the system [10]. X-Machine is a general computational machine [8] that, being a blend of diagrams and simple formalisms, it is capable to model both the static and the dynamic part of a system. X-Machines employ a diagrammatic approach of modelling the control by extending the expressive power of FSM. Transitions between states are no longer performed through simple input symbols but through the application of functions. 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. Thus X-Machines can model more complex data structures, in contrast to finite state machines that lack this ability and can model only trivial data structures .

The system described in the previous section is formally specified and the X-Machine model is presented starting with the visual part (a state transition diagram) which shows the different states of the machine and the transitions between these states in Figure 2.

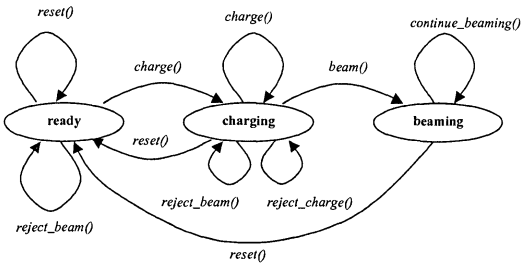

Figure 2 - The X-Machine model of the ray beaming system

Using a formal language based on state transition diagrams, a system is viewed as a set of different states that the system could be found. Several events trigger a change from one state to another. The ray beaming system specified as X-Machine could be in three different states; `ready` (idle) waiting for an event to get it going, the system current charge is 0 mV and the maximum allowed value is 30 mV. Another state recognised as `charging` means that the current charge is more than zero and the beamer is charged to reach the maximum value. The last state is `beaming` where the system actually beams the patient until an event resets the machine.

An informal textual description of the system as a X-Machine follows. The model of the system using the X-Machine notation is formally presented in table 1. Initially the system is in state `ready`. If the user presses the button for charging the machine, the event `charge_button` changes the system state into `charging` but also changes the current value to 10 mV (`charge` function is applicable). In the cases of the events `beam_button`, `reset_button` the system stays in the `ready` state and there is not any change in the charge value.

Table 1. Formal Specification of the ray beamer as a X-Machine model

The **X-Machine** is defined as $M = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:	
Input set Σ	Σ = {charge_button, beam_button, reset_button}
Output set Γ	Γ = { MachineCharging, ChargeRejected, BeamRejected, MachineBeaming, ContinueBeaming, MachineReseting } × N_0
Set of **states Q**	Q = { ready, charging, beaming }
Memory M	M = (MaxCharge , CurrentCharge) where MaxCharge is a variable holding the maximum accumulating voltage accepted by the machine, e.g. MaxCharge ∈ {30}, and CurrentCharge is a variable holding the current voltage.
Initial state q_0	q_0 = ready
memory m_0	m_0 = (30, 0)
Next state function F	F: $Q \times \Phi \rightarrow Q$ shown diagrammatically in figure 2.
Type Φ of the machine is the set of functions φ: $\varphi : \Sigma \times M \rightarrow \Gamma \times M$ The functions defined next using the notation: φ(σ,m) = (γ,m') if condition	**charge**(charge_button, (MaxCharge, CurrentCharge)) = ((MachineCharging, CurrentCharge+10), (MaxCharge, CurrentCharge+10)) if CurrentCharge+10 < MaxCharge **charge**(charge_button, (MaxCharge, CurrentCharge))= ((MachineCharging,MaxCharge),(MaxCharge,MaxCharge)) if CurrentCharge+10 ≥ MaxCharge ∧ MaxCharge ≠ CurrentCharge **reject_charge**(charge_button,(MaxCharge,MaxCharge))= ((ChargeRejected,MaxCharge),(MaxCharge,MaxCharge)). **reject_beam**(beam_button,(MaxCharge,CurrentCharge))= ((BeamRejected,MaxCharge),(MaxCharge,CurrentCharge)) if CurrentCharge < MaxCharge **beam**(beam_button, (MaxCharge, MaxCharge))=((MachineBeaming,0),(MaxCharge,0)) **reset**(reset_button,(MaxCharge,CurrentCharge))= ((MachineReseting,0), (MaxCharge,0)). **continue_beaming**(button, (MaxCharge,0))= ((ContinueBeaming,0), (MaxCharge,0)) if button ∈ {beam_button,charge_button}

Being in state charging and accepting a charge_button event the system stays in the same state but if the current charge is less than the maximum value it is increased by 10 mV. This is possible through the function charge which is triggered by the event charge_button if the corresponding if-condition guard relevant to the X-Machine's memory is satisfied. Otherwise, the current charge is not modified. Identifying the event reset_button the system is reset to the initial state ready and current charge becomes 0 mV. With the event beam_button the system either changes state and moves to beaming if the current charge is equal to the maximum value or it stays in state charging (if the current charge is less than 30 mV) without changing anything. In state beaming receiving the event beam_button or charge_button leaves the system in the same state. Finally with a reset_button the system moves to state ready making the current charge 0 mV. All these events trigger the corresponding functions as depicted in Figure 2.

The next step in the proposed framework is to verify that the produced model of the system satisfies several safety properties, before starting the implementation based on this model.

Verification

In order to prove that the system modelled as a X-Machine model, has the desired properties, a model checking technique for X-Machines is used [9]. Thus having the system as a X-Machine model, it is possible to query that model if it has the desired properties by exhaustively searching the state space in order to verify that the properties are satisfied. In order to express such queries, there is a need for a mathematical language with built-in notion of time that can describe the ordering of events in time without introducing time explicitly. Temporal logic [11] seems to be the most appropriate. A variation of temporal logic with more expressive and flexible operators is the Computational Tree Logic (CTL) [12]. With the use of combinations of CTL operators, i.e.: the universal (A) and the existential (E) path operators and the eventually (F) and globally (G) state operators, it is possible to express if a desired property is valid in the whole model or in part of it, starting from the initial state. In the example used previously in this paper the logic proposition: AG (CurrentCharge ≤ 30), expresses that *it is impossible the voltage to become greater than the maximum value, which in this case is 30*, meaning that the machine will never be charged with more than the permitted value, avoiding the chance to injure the patient. The translation of this temporal

formula is that the requirement expressed with that formula holds in the model if in every state of the state space the following is true; the voltage is less or equal to 30 (the property p ⇔ CurrentCharge ≤ 30, is true in every state). More examples of temporal CTL formulae which express safety properties that could be checked using the model checking technique proposed in [9] are shown in table 2.

Table 2. Examples of CTL formulae

Example of Property p	Temporal Operators in CTL	Explanation
CurrentCharge ≤ 30	AG p	For every path and for every state in the path, the property p is valid
CurrentCharge < 30	AF p	For every path, there exists at least one state where p is valid
CurrentCharge = 0	EG p	There are some paths (at least one), where in every state of these paths p is valid
CurrentCharge = 30	EF p	There are some paths (at least one), where in some states of these paths p is valid

Research is contacted for the development of an extension of temporal logic that will facilitate the model checking of X-Machines, and will help the user to create intuitively the queries to the model.

The verification of the model assures that in the produced model all the safety requirements expressed as temporal formulae hold, thus in the next phase which is the implementation of the system, confidence is added that the correct system is the one that is being implemented.

Testing

After the implementation of the system, it is necessary to test the final product and prove that the produced implementation is equivalent to the model. Ipate and Holcombe [13] presented a testing method for X-Machines which finds all faults in an implementation [14], as long as the model satisfies some design for test conditions (completeness and output distinguishability) [13].

Using X-Machines and this testing method, a complete set of test cases is produced from the model. Each test case, is a sequence of inputs and a sequence of outputs is expected from that, according to the formal model. Feeding all these test cases to the implementation and comparing the output sequences produced, with the ones expected, it is possible to test if the implementation is equivalent to the original model. For example a test case produced with the application of this testing method to the X-Machine model of the above specified ray beamer was: <charge_button, reset_button, charge_button>. In this case if the implementation fails to produce the same output sequence to that of the specified model, then the system would probably have a fault in resetting the ray at any given time, as the model imposes. Thus using the whole test set it is possible to test the implementation and prove that it is correct with respect to the specified model.

X-Machines is not only a formal method to model a system but also offer a strategy to test the implementation against the model and find faults in the final product [15]. With the addition of the model checking verification technique, a framework that uses formal methods is established, that facilitates the "correct" development of a safety critical system.

Discussion & Conclusions

This paper presented a formalism (the X-Machine model) and showed how it may support several steps related to the development of medical systems. The formalism possesses characteristics that render it attractive: firstly, it is intuitive (since it employs a graphical notation) and secondly, it is unambiguous (given that it is based on a rigorous notation). A number of tools based on the X-Machine model have already been developed to facilitate modelling using X-Machines [16] and to automate the production of test cases given a specified X-Machine model. Currently work is conducted to develop tools that will facilitate the model verification step [17]. Furthermore, there is a plan to employ a real-life case study in order to demonstrate the practicality of the proposed method and the suitability of the developed tools.

X-Machines contribute to a framework for formal development that is suitable for the production of safety critical systems. The adoption of X-Machines gives all the advantages of a formal modelling method coupled with a verification methodology and a testing strategy, thus making it possible to prove that certain safety-critical properties hold in the final medical system.

Acknowledgements

I would like to thank Dr P Kefalas and Dr E Kehris for their suggestions and proof reading and Prof Mike Holcombe for his valuable comments on my work over the last two years.

References

[1] Leveson NG. *Safeware: System Safety and Computers.* Addison Wesley Longman, 1995.

[2] Young W D. Formal Methods versus Software Engineering: Is There a Conflict?. In: Proceedings of the Fourth Testing, Analysis, and Verification Symposium; 1991 Oct; Victoria, British Columbia ; pp. 188-9.

[3] Craigen D, Gerhart S, and Ralston T. Formal Methods Reality Check: Industrial Usage. *IEEE Trans on Software Engineering* 1995; 21(2): 90-8.

[4] Bowen JP, and Stavridou V. Formal methods and software safety. In Frey H ed. *Safety of Computer Control Systems 1992 (SAFECOMP'92)*, Pergamon Press, 1992; pp. 93--8.

[5] Jacky J, Unger J, Patrick M, Reid D, and Risler R. Experience with Z developing a control program for a radiation therapy machine. In: Proceedings of the 11th International Conference of Z Users ZUM '97: Z Formal Specification Notation; 1997 3-4 Apr; Berlin, Germany: Springer-Verlag; pp. 317-28.

[6] Kasurinen V, and Sere K. Integrating action systems and Z in a medical system specification. In Gaudel MC, Woodcock J, eds. *FME'96: Industrial Benefit and Advances in Formal Methods*, LNCS 1051, Springer-Verlag , 1996; pp. 105--19.

[7] Holcombe M. X-Machines as a basis for dynamic system specification. *Software Engineering Journal* 1988;3(2): 69-76.

[8] Holcombe M and Ipate F. *Correct Systems: Building a Business Process Solution*, Springer Verlag, London, 1998.

[9] Eleftherakis G, and Kefalas P. Model Checking Safety-Critical Systems Specified as X-Machines. *An. Univ. Bucur. Mat. Inform.* 2000; 49(1): 59-70.

[10] Clarke E, Wing JM. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys* 1996;28 (4): 626-43.

[11] Pnueli A. The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science; 1977 31 Oct-2 Nov; Providence, Rhode Island : IEEE; pp. 46–57.

[12] Clarke EM, Emerson EA, and Sistla AP. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. Programming Languages and Systems* 1986; 8(2): 244–63.

[13] Ipate F and Holcombe M. Specification and testing using generalised machines: a presentation and a case study. *Software Testing, Verification and Reliability* 1998; 8: 61-81.

[14] Ipate F and Holcombe M. An integration testing method that is proved to find all faults. *International Journal of Computer Mathematics* 1997; 63(3):159-78.

[15] Kehris E, Eleftherakis G, and Kefalas P. Using X-Machines to Model and Test Discrete Event Simulation Programs. In: Mastorakis N, ed. *Systems and Control: Theory and Applications*, World Scientific and Engineering Society Press, July 2000; pp. 163-168.

[16] Kefalas P and Kapeti E. A Design Language and Tool for X-Machines Specification. In: Fotiadis DI and Nikolopoulos SD, eds. *Advances in Informatics*, World Scientific Publishing Company, April 2000; pp. 134-45.

[17] Eleftherakis G and Kefalas P. Towards Model Checking of Finite State Machines Extended with Memory through Refinement. To appear in: Proc. 5th WSES/IEEE WORLD MULTICONFERENCE ON Circuits, Systems, Communications & Computers (CSCC 2001), Crete, July 2001.

Address for correspondence

George Eleftherakis
Computer Science Department, City Liberal Studies,
Affiliated College of the University of Sheffield
13 Tsimiski Str., 54624 Thessaloniki, Greece

eleftherakis@city.academic.gr

http://www.city.academic.gr/material/academic_staff/computer_science/eleftherakis/