Formal Modelling Languages to Specify Real-Time Systems: A Survey

Aftab Ali Haider and Aamer Nadeem

***Abstract*—Real-time systems demand functional as well as temporal correctness. Complex interactions with the environment and strict adherence to time bounds are main characteristics of these systems. The use of formal methods is a natural choice for real-time system development. Formal models are more rigorous in nature and ensure completeness. If a system is highly sensitive to time delays, time behaviour of real-time system must be specified through formal languages. Petri nets, state charts and UML can be used to represent time behaviour, but these techniques are lesser effective than formal languages. The strength of formal languages depends upon completeness and possibility of partial mathematical verification. In order to study time behaviour, we have surveyed existing formal languages used to model real-time systems. We have selected typical languages that are supposed to represent entire set of real-time formal languages. Some parameters are defined to critically evaluate these languages. We have analysed and compared formal languages for real-time systems using our defined evaluation criteria. As a result of our analysis, we conclude that all languages incorporate special constructs to capture time behaviour. Effectiveness of these constructs can be compared on the basis of the capability of the languages to specify time domain. We also conclude that there is an ardent need to standardise formal languages used to specify real-time systems.**

***Index Terms*—Formal languages, real-time systems, temporal logic, first order logic, mathematical verification, time domains, semantics and ontology.**

I. Introduction

A real-time system is one whose functionalities are bounded to timely response. Performance and effectiveness of such systems is dependent on time constrained behaviour. Occurrence of certain specified functionality is not the only objective; this occurrence must lie within specified time-frame. Timely occurrence of response guarantees success and any deviation from time constrained occurrence may cause unbearable loss. Controller of nuclear power plants is one of such examples where any delayed response may cause catastrophe. Modelling of real-time systems requires well defined languages and methods [30]. A language used to specify real-time systems must be capable to explicitly represent time dependent behaviour. An ideal formal language must also have additional features of readability, simplicity and non-ambiguity. Real-time systems can be categorised as hard real-time systems and soft real-time systems. However, degree of criticality of the real-time systems is directly related to its time dependent behaviour. In soft real-time systems, a delay may affect performance but hard real-time systems can never tolerate any such delay and may lead to disastrous conditions [22]-[24].

With the increasing use of real-time systems, need to carefully specify and model these systems to avoid hazards have also increased. Real-time systems are hard to model with informal methods due to implicit, complicated and critical timing relationships involved. Formal modelling is a natural choice for these systems. The reason is that formal methods are more rigorous in nature and explore every possibility to ensure completeness and correctness. Formal techniques mainly include process algebras, temporal logic, automata theoretic techniques, petri-nets and partial order models. [27]-[30].

In this study we have selected RML[3][14], Telos[15], ERAE[30], ForSpec Temporal Logic [2], TRIO [6], TRIO+[10]-[12], ASTRAL[18], [19], RT-Z, TCOZ [1], [26] and mode charts[8][36] for evaluation and comparative study. RML and Telos specify time dependent behavior by incorporating time constructs as extensions. Other languages are excluded either due to similarity with these languages or inability to specify real-time systems. RT-UML and petri nets are excluded due to undefined semantics [37], [38].

II. Evaluation and Critical Analysis

A brief introduction of each modelling languages is given prior to their comparison and are later analysed and evaluated against pre-defined frame work. Table 1 describes the detailed comparison of formal languages.

A. RML

RML provides an object-oriented frame work where objects are grouped into classes and these classes are instances of meta-classes. An ontology consisting of entity, activity and assertion concepts is used in RML for requirement modelling. Concept assertion basically makes RML a formal modelling language. Binary relations (constructs) are used to relate these objects. These objects are further grouped into generalised or specialised classes. Formal Semantics for RML contain set of assertion in first order predicate calculus. In order to represent time, RML uses linear, dense model of time points and history-oriented modeling of an application [3].

RML assumes that entity, activity, and assertion are built-in features of language. This factor reduces the expressiveness of RML to customise the language to specific classes. This can be achieved if notions other than entity, activity and assertion can be defined by dispelling the impression of “fixed” world. These inadequacies of RML were addressed in its successor Telo [18]. RML is unable to model non-functional requirements. It has adopted interval based ontology to model instead of temporal logic [3].

Manuscript received September 19, 2012; revised October 14, 2012.

The authors are with Center for Software Dependability, Muhammad Ali Jinnah University (MAJU), Islamabad, Pakistan (e-mail: aftab775@yahoo.com, anadeem@jinnah.edu.pk).

TABLE I: A COMPARISON OF FORMAL LANGUAGES

Parameters	ForSpec	RML	Telos	Mode Charts	TRIO and TRIO+	ASTRAL	TCOZ and RT-Z	Parameters	ForSpec
Partial Mechanical Verification	A decidable subset is verifiable	Does not exist	Does not exist	Partially verifiable	A decidable subset is verifiable	Partially verifiable	Does not exist	Partial Mechanical Verification	A decidable subset is verifiable
Time Domain representation	All Time domains	Dense	Discrete	Discrete	Discrete and continuous	Discrete and continuous	Dense	Time Domain representation	All Time domains
Scalability	yes	No, simple systems with simple time dependent behavior can be modeled	Yes	Get complicated with complex systems, these are scalable to some extent	Scalable after inclusion of TRIO+	yes	Yes, Time CSP can be implemented for complex behavior However RT-Z is not	Scalability	yes
Object Oriented Frame Work Supportivity	Does not support	Yes	Yes	No	Yes	No, however after translating into TRIO and using TRIO+ possible	Yes	Object Oriented Frame Work Supportivity	Does not support
Executable Specifications	Does not contain	Does not contain	Does not contain	Does not contain	Yes	Yes, after translating into TRIO+	Does not contain	Executable Specifications	Does not contain
Time Constructs	Prop, Future(b), Past(b,n), next, wnext, eventually, globally, until and wuntill	Hold , Overlap	Equal, meets, before, overlaps, during, starts, ends	Mode and transitions are used to specify time	alwF, alwP, somF, somP, sometimes, always, lasts, lasted, until, since, untilw, sincew, before	State, events and Transition are used to specify time	Timed CSP construct along with Deadline and waituntil	Time Constructs	Prop, Future(b), Past(b,n), next, wnext, eventually, globally, until and wuntill
Formal Notation and Model used	Purely formal notations	ERAE Model	ERAE Model	State Charts	Pure Formal Notations	State Machines	Objects and classes with formal notations	Formal Notation and Model used	Purely formal notations
Partial Mechanical Verification	A decidable subset is verifiable	Does not exist	Does not exist	Partially verifiable	A decidable subset is verifiable	Partially verifiable	Does not exist	Partial Mechanical Verification	A decidable subset is verifiable
Time Domain	All Time	Dense	Discrete	Discrete	Discrete	Discrete	Dense	Time Domain	All Time

RML is an assertion based language, which uses first order predicate calculus and represents dense time domain only. Constructs used in RML are Hold and Overlap through Boolean connectives. RML supports object-oriented framework. Due to simplicity of the time related constructs and inability to express more complex systems, degree of ease to test is quite high for RML.

B. Telos

Telos is a successor of RML and has the ability to specify complex systems. ERAE model is successor of Telos and more expressive than Telos [30].

Telo has seven constructs to specify time which are equal, meets, before, overlaps, during, starts, and ends. “Hold” construct of RML is not present in Telos. However, Telos is more expressive in terms of capturing time related behaviour than RML. Telos uses first order logic with existential and universal quantifiers and is based on linear time temporal logic. Telos supports reason with deductive rules and axioms. Telos can be used to specify complex systems. However, the ease to test is high for Telos as well. Telos can specify discrete time domain. There are no mechanical verification tools to partially verify Telos [15][30].

C. TRIO and TRIO+

TRIO builds on events and relationship between these events. There are classes in TRIO specification and these classes are expressed with formulae. These formulae can be either axioms or derived and are grouped into classes. TRIO has first order, linear time temporal logic. TRIO specifications suit small systems and complicated systems are specifications require certain extensions in it. TRIO is not object oriented and can’t implement inheritance, abstraction and information hiding. TRIO+ extends TRIO, is object-oriented, more expressive and is suited for large and complicated specifications. The constructs of TRIO+ have narrowed the gap between early requirements and the high level design [6]-[16].

TRIO can be used for dense or discrete time domain and is un-decidable. However, a subset of TRIO, PLTLB Propositional Linear Time Temporal Logic is decidable. TRIO allows mechanical verification through automatic or semi automatic approaches, e.g., SPIN. The semantic of TRIO is Model Parametric Semantic. TRIO has the ability to cope with any time model [6].

The environment for TRIO can be viewed as a machine

level language rather than specification language. Therefore, TRIO is very hard to read or understand. In TRIO, specifications can neither be modularized nor has means to represent it as levels of hierarchies. TRIO is best suited for machine level representations to be used in specifications [6].

The concept of assumption about the environment, ability to specify critical requirements and proof system for modularized specifications are not found in TRIO [18][19].

Degree of ease to test TRIO is very high due to presence of extra payload carried to make the specifications executable. These specifications are helpful to remove any inconsistency between the requirements and model. Basic version of TRIO was not scalable, however, with the advent of TRIO+ issue of modelling complex and large systems has been resolved. TRIO uses first order logic, can specify all types of domain models, and has linear time temporal logic. It has the ability to specify object oriented framework. A decidable subset as discussed earlier can be verified mathematically through tableaux method. It deviates from classical temporal logic in the sense that it allows quantitative reasoning about time. Constructs used in TRIO are alwF, alwP, somF, somP, sometimes, always, lasts, lasted, until, since, untilw, sincew, before [6], [11], [12].

D. ASTRAL

ASTRAL emerged from ASLAN [13] and RT-ASLAN [9]. There is a strong influence of these languages on ASTRAL. ASTRAL has borrowed not only the language but also the proof checking from RT-ASLAN. A state machine model is used in ASTRAL which comprises of variables, invariants, transitions and constraints. The concept of assumption about the environment, ability to specify critical requirements and proof system for modularized specifications are found in ASTRAL [4].

ASTRAL is not a machine level language and can specify the higher level design. Model of ASTRAL comprises a single global specification and collection of state machines. ASLAN has the means to model hierarchy and modularity in specifications. ASTRAL has also inherited this from ASLAN.

In order to make the specifications of the ASTRAL executable, it can be translated into TRIO. Logic and time representations of ASTRAL are borrowed from TRIO. ASTRAL lacks the ability to specify object oriented concepts. There is a need to extend ASTRAL to incorporate these concepts. However, ASTRAL can be used to model large and complicated systems and is scalable. ASTRAL supports layered compositional and executable specifications [18], [19].

ASTRAL is partially verifiable and uses the proof checking of RT-ASLAN. It can specify all types of time domains and is more expressive than TRIO.

E. Forspec Temporal Language (FTL)

ForSpec uses linear time logic (LTL) and is based on higher order logic. It can captures all time domains and is quite expressive than classical Linear time logic. It allows temporal connectivity through time windows addresses inability of LTL to specify past connectivity. FTL takes past connectivity as boolean rather than temporal connectivity. Constructs used are prop, future (b), past (b,n), next, wnext, eventually, globally, until and wuntill and connectives used are !, ||, &&, implies, iff [2].

ForSpec is unable to specify object-oriented framework and this area requires further efforts to make the language compatible with other contemporary languages. ForSpec is difficult to test and requires high level of skill for understanding the logic contained and to generate test paths and test data. ForSpec is partially verifiable.

F. RT-Z and TCOZ

RT-Z is real-time Object-Z whereas Time Communicating Object-Z (TC-OZ) resulted after blending Object-Z and Timed-CSP. Timed-CSP is an algebraic specification language. Timed-CSP is an extension to CSP which has the ability to specify real-time systems. RT-Z has Time Refined Calculus of Z specifications to incorporate time. RT-Z has great similarity to VDM++. It is an object oriented modelling language. It has the ability to modularize the specifications [23]-[26].

Time CSP is far better than RT- Z to describe process control and has the ability to handle timing delays and timeouts with much more simplicity. RT-Z is much complicated than TCOZ. The sequence of the events is well determined by Time CSP than RT- Z. Then preconditions are not calculated in TCOZ. Similarly there is no requirement to have in depth reasoning mechanism for Time CSP [1].

G. Mode Charts

Mode charts are graphical specification language and use state charts. These have the ability to specify discrete time model and have linear time temporal logic. Semantics of mode charts are defined and are discussed in [36] and there exists mechanical verification of mode charts [8]. These characteristics make Mode charts a formal language .

Mode charts get complicated when specifying complex and large systems. Mode charts result from state charts and have same degree of ease to test. Mode charts use modes and transitions to specify time dependent behaviour [8][36].

III. Conclusion and Future Work

Temporal correctness can never be guaranteed, however, with formal models these systems can be made more reliable and fault tolerance can be provided. With more and more use of software dependent real-time systems; there is more need to carefully model such systems.

In this study, it is found that RT-Z and TCOZ are both based on Z formal methods. In order to eliminate redundant efforts, an effort can be made to find semantic relation between these languages. If TCOZ is more expressive than RT-Z and subsumes RT-Z then there exists possibility to focus efforts on TCOZ. However, in case of contradiction to the above assumption, RT-Z can be considered for future works instead of TCOZ.

References

[1] K. Araki, A. Galloway, K. Taguchi, B. Mahony, and J. S. Dong, "Overview of the semantics of TCOZ," *Integrated Formal Methods*, York, UK, pp. 66–85, Springer-Verlag, June 1999.

[2] R. Armoni, L. Fix, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. M. Haim, A. Tiemeyer, E. Singerman, M. Y. Vardi, and Y. Zbar, "The ForSpec temporal language: A new temporal property- specification language," In *Proc. of 8th Int'l Conf. on Tools and Algorithms for the*

Construction and Analysis of Systems (TACAS'02), Lecture Notes in Computer Science 2280, pp. 296–311. Springer-Verlag, 2002.

[3] S. Greenspan, J. Mylopolous, and A. Borgida, "On formal requirements modeling languages: RML revisited," in *Proc. of 16th Int. Conf. on Software Eng*., 1994, pp. 135-147.

[4] C. Ghezzi and R. A. Kemmerer, "ASTRAL: An Assertion Language for Specifying Real-Time Systems," in *Proc. of ESEC'91- 3rd European Softwre Engineering Conference*, LNCS 550, Springer-Verlag, 1991.

[5] A. C. Porisini, C. Ghezzi, and R. Kemmerer, "Specification of Realtime Systems Using ASTRAL," *Technical Report 96-30, Computer Science Department, University of California*, Santa Barbara, US, January 1997.

[6] C. Ghezzi, D. Mandrioli, and A. Morzenti, "TRIO: A Logic Language for Executable Specifications of Real-Time Systems," *J. Systemsand Software*, June 1990.

[7] A. C. Porisini, P. S. Pietro, and R. Kemmerer, "Formal Semantics Definition for ASTRAL," *Report No. TRCS 94-15, Dept. of Computer Science, Univ. of California*, Santa Barbara, Sept. 1994.

[8] F. Jahanian and A. K. Mok, "Modechart: A Specification Language for Real-Time Systems," *IEEE Trans. Software Eng.*, vol. 20, no. 10, pp. 879–889, Oct. 1994.

[9] B. Auernheimer and R. A. Kemmerer, "A Specification language for real-time systems," *IEEE Trans. Software Eng*., vol. 12, no. 9.

[10] L. Briand and S. Morasca, "Software Measurement and Formal Methods: A Case Study Centered on TRIO+ Specifications," *ICFEM'97*, Hiroshima, Japan, November 12-14, 1997.

[11] A. Morzenil and S. Pietro, "An object oriented logic specification language," *Tech. Rep. ENEL/CRA*, Jan. In Itahan.

[12] A. Morzenil and S. Pietro, "Object oriented logical specification of time-critical systems," *ACM Trans. S'oftuJ. Eng*., pp. 56–98.

[13] B. Auernheimer and R. A. Kemmerer, "ASLAN User's Manual," *TRCS84-10, Dept. of Computer Science, Univ. of California*, Santa Barbara, Apr. 1992.

[14] S. Greenspan, A. Borgida, and J. Mylopoulos, "A Requirements Modeling Language and Its Logic," *Information Systems*, vol. 11, no. 1, pp. 9-23, 1986

[15] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis, "Representing Knowledge about Information Systems in Telos," In *Proc. of Database Application Engineering with DAIDA*, research reports ESPRIT, project 892, DAIDA, vol. 1, Springer-Verlag, Berlin, 1993.

[16] R. M. Smullian, *First order Logic*, Springer Verlag, 1968.

[17] N. Rescher and A. Urquhart, *Temporal Logic*, Springer Verlag, Vienna-New York, 1971.

[18] C. Ghezzi and R. A. Kemmerer, "Executing Formal Specifications: The ASTRAL to TRIO Translation Approach," in *Proc. Symp. Testing, Analysis, and Verification*, Victoria, B.C., Canada, Oct. 1991.

[19] A. Morzenti, D. Mandrioli, and C. Ghezzi, "A Model Parametric Real-Time Logic," *ACM Trans. Programming Languages and Systems*,vol. 14, no. 4, pp. 521–573, Oct. 1992.

[20] S. Gerhardt, D. Craigen, and T. Ralston, "An International Survey of Industrial Applications of Formal Methods," *Department of Commerce, Technology Administration, National Institute of Standards and Technology*, Computer Systems Laboratory, Gaithersburg, MD 20899, USA, March 1993

[21] A. Hall, "Seven Myths of Formal Methods," *IEEE Software*, vol. 7, no. 5, pp. 11–19, September 1990.

[22] E. M. Clarke and J. M. Wing, "Formal methods: state of the art and future directions," *ACM Computing Surveys*, vol. 28, no. 4, pp. 626–643, 1996.

[23] G. Smith and I. Hayes, "Towards Real-Time Object-Z. In Integrated Formal Methods," pp. 49-55, Springer-Verlag, June 1999.

[24] J. M Spivey, "The Z notation: A reference Manual 2nd," *Prentice Hall*, London 1992.

[25] S. Schneider and J. Davies, "A brief history of Time CSP," *Theoretical Computer Science*, vol. 138, 1995.

[26] B. P. Mahony and J. S. Dong, "Blending Object-Z and Timed CSP: An introduction to TCOZ," In *The 20th International Conference on Software Engineering (ICSE'98). IEEE Computer Society Press*, April 1998.

[27] J. S. Ostroff, "Survey of Formal Methods for the Specification and Design of Real-Time Systems," *Tutorial on Specification of Time*, 1992.

[28] R. Alur and T. A. Henzinger, "Logics and Models of Real-time Systems: A Survey," In *REX Workshop – Real-time: Theory in Practice*, LNCS, Springer Verlas, 1992.

[29] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw, "Comparing Different Approaches for Specifying and Verifying Real-Time Systems." In *Proc. of 10th IEEE Wsh.Real-Time Operating Systems and Software*, pp. 122–129, New York, 1993.

[30] R. A. Kemmerer and P. Z. Kolano, "Formally specifying and verifying real -time systems," In *proceedings of the 1st IEEE International Conference on Formal Engineering Methods*, Hiroshima, Japan, pp. 12-14, Nov, 1997.

[31] E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert, and A. Rifaut, "A Knowledge Representation Language for Requirements Engineering," in *Proc.of IEEE*, vol. 74, no. 10, Oct. 1986, pp. 1431 –1444.

[32] A. Pnueli, "The Temporal Logic of Programs," *Proc. 18th IEEE Symp. Foundations of Computer Science*, Providence, pp. 46-57,1977.

[33] L. Lamport, "Sometimes is sometimes "not never" - on the temporal logic of programs," In *Proc. 7th ACM Symp. on Principles of Programming Languages*, pp.174–185, January 1980.

[34] A. Pnueli, "Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends," In *Proc. of Advanced School on Current Trends in Concurrency*, pp. 510–584, Berlin, Springer-Verlag, 1985.

[35] M. Gordon, "HOL: A Proof Generating System for higher order logic," *VLSI Specification, Verification, and Synthesis*, Kluver, 1987.

[36] F. Jahanian, R. S. Lee, and A. K. Mok, "Semantics of Modechart in realtime logic," in *Proc. of 21st Hawaii Int. Conf. Syst. Sci*., Jan. 1988.

[37] M. Felder, D. Mandrioli, and A. Mozenti, "proving properties of real-time systems through logical specifications and petri nets," *IEEE Trans. Software Engg*, vol. SE -17, pp. 259-273, Mar1991.

[38] L. Lavazza, G. Quaoroni, and M. Veturelli, "Combining UML and formal notations for modelling real-time systems," in *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software*.