Checking Temporal Business Rules

Kåre J. Kristoffersen, Christian Pedersen, and Henrik R. Andersen

IT University of Copenhagen*
DK-2400 NV
Denmark
{kjk,cp,hra}@itu.dk

Abstract. In this paper we take the position that formal methods and especially the rather new area of runtime verification must be taken seriously into account in developing the *next* generation of Enterprise Ressource Planning Systems. First we illustrate how temporal logic may be used to specify temporal business rules. Next we present a verification server which can monitor a running ERP system with respect to satisfaction of the business rules given in the specification. The algorithm which forms the body of the verification server works by rewriting, for each event, the timed LTL formula into a residual formula that takes into account the time and system state at the occurence of the event. The residual formula will be the requirement for the timed system in the future, to be further rewritten at the occurrence of the next event.

1 Motivation

Runtime verification is a branch of verification in which a running program is supervised by a concurrently running *verifier*. Figure 1 illustrates the situation comparing a state-based "polling scheme" with an event-driven timed verifier. Properties are formulated in a real-time logic called LTL_t. The logic LTL_t is an extension of LTL (Linear Time Logic) with real time constructs embodied by a freeze quantifier together with atomic clock constraints making it possible to express real time logical properties of a system (following [7]). This logic is suitable for expressing timed safety, liveness and fairness properties. An example of a *bounded* liveness property for a bank account could be: *If Balance is negative then within 10 days the Balance should be positive or zero.* In LTL_t this could be written as follows (assuming the resolution and scale of time is days):

$$\Box(Balance < 0 \Rightarrow x.(t \leq x + 10\ U\ Balance \geq 0))$$

In the above expression the variable t is the global time, or more precisely the time of the observation trace generated by the system being monitored. In this paper we use states with discrete time. This corresponds to time stamps with real time at a given accuracy (for instance, milliseconds). The semantics of the *freeze* formula $x.\phi$ is that when evaluated, the value of x is replaced by the current time.

* This work is supported by the project NEXT which is a joint effort between Microsoft Business Solutions and the IT University of Copenhagen.

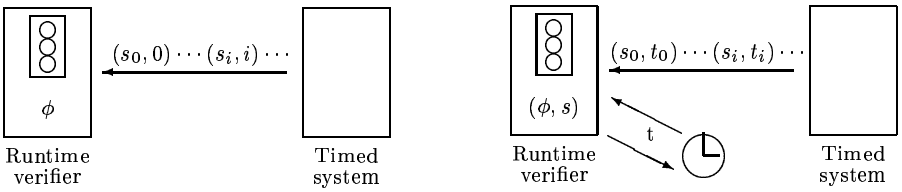

Fig. 1. A runtime verifier supervising a timed system. In the figure to the left, state information is computed and transmitted from the timed system at each discrete time point $0, 1, 2, \ldots$. The runtime verifier computes a new residual formula ϕ at each step, resulting in one of three situations, indicated by the three-coloured traffic light. "Red" corresponds to the situation where an error has been detected, i.e., the formula has reduced to false. "Green" corresponds to acceptance, i.e., the formula has reduced to true and the verifier can stop. "Yellow" corresponds to the situation where so far no error has been found but the verifier cannot yet stop. In the figure to the right the verifier only computes when an event happens, at the time points $t_0 < t_1 < \cdots$. An event is either a *state change* in the timed system or a timeout. Timeouts are set by the verifier and computed as what we shall refer to as the *smallest interesting timepoint* as given by the formula ϕ. A prototype, called TIMECHECKER, corresponding to the system at the right handside has been made.

The traditional field of application for formal methods and especially verification is within safety critical and embedded systems for which correctness is of vital importance. Errors are either fatal or they are costly. Therefore a lot of energy has been put into developing tools for checking such systems at design time, that is, prior to execution i.e. SPIN [4]. As an alternative, attempts have been made to check a running java program in Java PathExplorer [1]. Common for these efforts is that what is verified is a *program*, and typically a program taking input from a very restricted set of possibilities. This implies that the program under investigation can be regarded as a closed system, which may be checked alone.

Financial applications and business software (ERP Systems) as well as traditional back end databases are a class of applications that have attracted very little attention from the formal methods community. What needs to be verified in such a system is not the system software itself, but the data the system is managing. These data can be given a semantics which resemples that of timed traces, namely a sequence of states where each state consists of predicates true in this state and then a *time stamp* explaining *when* the state is measured. One reason that this have not gained very much interest from the verification community is that it is typically not of a safety–critical nature and hence correctness is of less importance compared with traffic control or production systems. At the same time ERP Systems differ from the beforementioned in that they can (and should be able to) consume unboundedly many different input data. As such they suffer severely from the well–known "State Explosion Problem" and consequently exhaustive verification is not feasible.

There could be several angles of attack on solving the timed run-time verification problem. One line could be to translate the timed LTL-formula into a timed automaton. The main challenge here is how to deal with the freeze operator. The logical clocks should correspond to clocks of the automaton, however, there is no simple bound of the number

of needed clocks. Consider for instance a *punctuality* property like

$$\Box(p \Rightarrow x.\Diamond(q \wedge t = x + 1000)),$$

expressing that always, when p holds, then after exactly 1000 time units, q should hold. If p change as often as possible then it might change 500 times before the first time q should be verified to be true. All these time points must be remembered indicating that the automaton will need up to at least 500 clocks. It means that it would be impractical to try to compute a full automaton in advance and an on-the-fly construction employing a variation of alternating automaton seems to be a way to find a solution. In [11] this problem is avoided by limiting the expressive power of the timed logic to upper-bounds. They avoid having any acceptance condition and there is no blow-up in the number of neeeded clocks. We work with the full logic.

The approach taken here is that of computing a residual formula, taking into account the states seen sofar, and expressing the property to hold for the timed system in the future. This is a bit like computing "weakest pre-conditions" but in a forwards manner: a "strongest post-condition", as it has been classically done for program verification in the pioneering work of Dijkstra and Hoare.

1.1 Verificationserver

The verification server shown in Figure 2 is a j2EE implementation of the runtime verifier depicted in Figure 1. The server contains the original specifications to be checked as well as the dynamicaly computed residuals. It has an interface through which it gets information about updates of the ERP system being monitored.

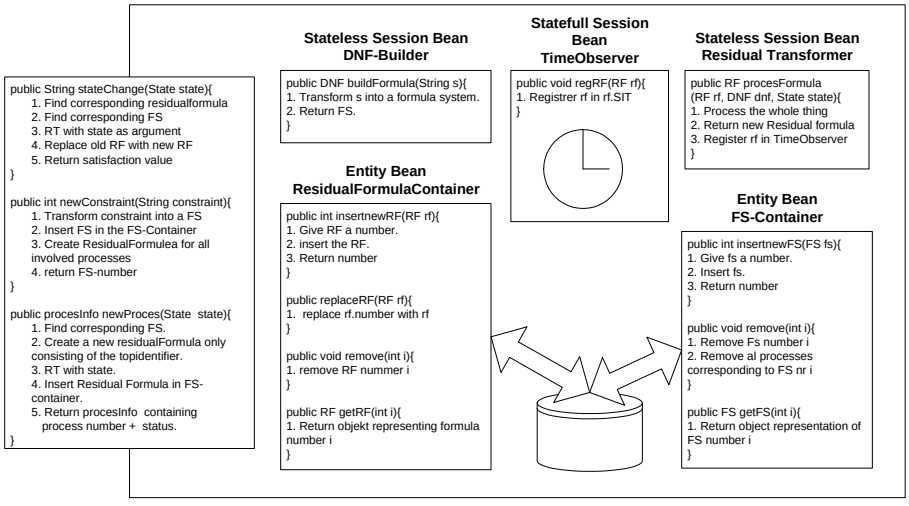

Fig. 2. The verification server.

References

1. K. Havelund, G. Ruso. Monitoring Java Programs with Java PathExplorer. First Workshop on Runtime Verification (RV'01), Paris, France, 23 July 2001. Electronic Notes in Theoretical Computer Science, Volume 55, Number 2, 2001
2. D. Giannakopoulou, K. Havelund. Automata-Based Verification of Temporal Properties on Running Programs. Automated Software Engineering 2001 (ASE'01), San Diego, California, 26-29 November 2001, IEEE Computer Society.
3. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple onthe -fly automatic verification of linear temporal logic. Proc. 15th International Symposium on Protocol Specification, Testing and Verification (PSTV XV), pp 318, Chapman and Hall, 1995.
4. G. Holzmann. The Model Checker SPIN. IEEE Trans. on Software Engineering, Vol. 23, No. 5, May 1997, pp. 279-295.
5. K. Kristoffersen, C. Pedersen and H. R. Andersen. Runtime Verification of Timed LTL using Disjunctive Normalized Equation Systems. Appears in Preliminary Proceedings of Third International Workshop on Runtime Verification RV '03, Boulder, Colorado, USA. July 13, 2003. (Technical Report MS-CIS-03-21, University of Pensylvania). To appear in Issue 89.2 of Electronic Notes in Theoretical Computer Science.
6. Time-Rover Corp. Temporal Business Solutions. The DB Rover. 2000.
7. R. Alur, T. Henzinger. Logics and Models of Real Time: A Survey. Real Time: Theory in Practice, Lecture Notes in Computer Science 600, Springer-Verlag, 1992, pp. 74-106.
8. T. Henzinger. It's About Time: Real-Time Logics Reviewed. Proceedings of the Ninth International Conference on Concurrency Theory (CONCUR), Lecture Notes in Computer Science 1466, Springer-Verlag, 1998, pp. 439-454.
9. K. Havelund, G. Ruso. Monitoring Programs using Rewriting. Automated Software Engineering 2001 (ASE'01), San Diego, California, 26-29 November 2001, IEEE Computer Society.
10. M.C.W. Geilen, D.R. Dams. An on-the-Fly Tableau Construction for a Real-Time Temporal Logic. Proceedings of the Sixth International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT2000, 20-22 September 2000, Lecture Notes in Computer Science Vol.1926 Pune, India, Ed. M. Joseph, pp. 276-290, Springer-Verlag, Berlin, 2000.
11. M.C.W. Geilen. An improved on-the-fly tableau construction for a real-time temporal logic. Computer Aided Verification, CAV 2003, Proceedings. Boulder, Colorado, USA, July 8 – 12, 2003.
12. Z. Chaochen, C.A.R. Hoare, A.P. Ravn: A Calculus of Durations, Information Processing Letter, 40, 5, pp. 269-276, 1991.
13. J. S. David. Three Events that Defines an REA Methodology for Systems Analysis, Design and Implementation. 2001.