Expanding Database Systems into self–verifying Entities

Kåre J. Kristoffersen and Yvonne Dittrich

IT University of Copenhagen*
DK-2300 S
Denmark
{kjk,ydi}@itu.dk

Abstract. The paper presents work-in-progress aiming at deploying runtime verification techniques to observe whether state changes in a database system conform with temporal business rules. A high level language for tailoring enterprise database systems with temporal business rules is defined. Furthermore we present an algorithmic framework for checking temporal business rules at runtime, i.e. we recommend on-line checking of data in the system as opposed to post-checking, i.e. off–line processing. A prototypical implementation of a runtime verifier (called Verification Server) based on this algorithmic framework is presented and discussed.

1 Introduction

Runtime verification is a branch of verification in which a running program is supervised by a concurrrently running *verifier*. In ths paper we shall employ *timed runtime verification*, in which time will be an improtant parameter in the task of the verifier. Our idea is to use such a mechanism to monitor a running database system and hereby at runtime check whether a sequence of states of a traditional database obey a set of temporal business rules. In [5] an interesting framework for proving temporal properties of a database *prior* to execution is presented. That task is much harder and thus the proof of correctness cannot be automated. Our approach can.

Temporal Business Rules and corresponding mechanism to check their success or failure might be hard coded in business systems. In fact, many such systems exists, like e.g. in a library where customers are (automatically) having their borrowed material reclaimed after one month. However, this approach restricts the flexibility to re-define temporal rules at all or at least to anticipated areas. Using runtime verification techniques provides the possibility to formulate and change general temporal business rules and to check them without changing the business application.

There are a number of issues that make databases an interesting and challenging object of runtime verification as compared to checking ordinary programs. First, the temporal logic needs to include explicit constructs for real time. This is not a complication itself, but a lot of extra care has to be taken in the verification server. Second, inour case

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

the verification server is not really checking just one timed trace, but rather a trace for each of the monitored objects. In the VAC example the verification server has one instance of the rule for each of the many thousand car customers, that it must keep track of. Third, in most administrative databases it should be possible for the rule manager to change rules on the run, and to enable/disable checking of rules. Last, but not least important, the rule manager should have an interface to the system which does not assume a Ph.D. in temporal logic and program verification.

The next section provides an example as a motivation for our approach. Section 3 introduces the formal language for temporal business rules and the algorithmic base we developed for the checking these at runtime. Section 4 presents the design of an early prototype of the Verification Server. Finally, in Section 5 we sum up our finding so far and present our future line of research on the subject.

2 Example: A vehicles assistances company

Vehicle Assistance companies (VACs)[1] go out and assist subscribing vehicle owners whenever they have a problem with their vehicle, typically assisting in starting engines which are low on electrical power or out of gas. Being an insurance company VACs should keep their expenses low. This means that each customer should be assisted as little as possible. More precisely, a *good* customer is one who is self-aided and does not require much assistance. A *bad* customer, on the other hand, is one which tend to use available services too often. In this section we shall illustrate how our runtime verification setup can can aid the VAC manager by showing how good behaviour may be specified as a temporal business rule, whose validity may be monitered by the Verification Server.

SQL queries and manual evaluation The back end enterprise database system of our VAC logs information about assistances in a table Assistances = (Date, CustomerID, TypeOfAssistance, ...). After a period of business time, e.g. one year perhaps, the management might be interested in getting an overview of the transactions with respect to a certain customer C to see whether he/she has required two assistances with less than a month in between. A very simple way to do this could be by executing a simple select statement like the following, and then visually try to capture too frequent assistances requested by customer C.

```
SELECT    *
FROM      Assistances
WHERE     CustomerID=C
ORDER BY Date;
```

The above query would return a chronologically ordered list of all assistances given to C. Such a list could be processed manually to identify instances of too frequent assistances. Based on the manual evaluation the company can decide on appropiate actions to take in that case. The analysis above could be automated a little, for instance by placing the selected information in a cursor and then automatically fetching pairs of elements

[1] In Denmark examples of such are Falck and Dansk Autohjælp.

that occur within a month. Though being more automatic this last approach does not change the processing from being off–line as opposed to being on–line.

Using triggers The first approach to solve the VAC managers problem in an on–line fashion is by using triggers when inserting a new data set in the related table of the database. We could trigger on insertion of a new assistance and chech whether this is within 30 days of the most recent assistance. This can be done in PL/SQL as follows:

```
define trigger TooFrequentAssistance
on insert in assistances A
   if new A.date > ( select date from assistances
                     where customerID=C and rowid=1 order by date) - 30
   then insert into TooFrequent values(A.date, A.CustomerID, A.name,...)
```

Each assistance occuring too soon after another is registred in a table named `TooFrequent`. There is the clear drawback to this solution; to change the rule a database consultant has to be called in to change the trigger. This will be expensive and also less flexible than a solution where the user him- or herself can change the business rule.

The ideal solution The ideal solution would allow the administrator responsible for developing and maintaining the business rules of the company to enter the rule in some kind of notation. A part of the system would then check whether the changes in the database are in agreement with that rule. In this case it would yield an event if a customer asks for assistance twice within 30 days. The business expert could then define what should happen in case a violation occurs. E.g. an e-mail could be sent to the responsible person, or one could start a business process in the companies ERP system. Such a system should be able to handle more than one business rule. Real life temporal business rules can be based on different time units. In our case, we are interested in days as basic time units. The rule that a customer should not wait for more than one hour for assistance would have to be formulated based on hours or minutes as a time unit. As we do not know in advance what data exactly our temporal business rules might be based on, we would like to be able to use any kind of attributes and attribute changes as a base for formulating the temporal business rules. This requires the interface for the business expert to have access to the model of the real database, or - in case of the database being too complex for a business expert to handle - a view that provides the data that temporal rules can be based on. In the next sections we present our bid on the solution.

3 Temporal Business Rules

A state of a database is a pair (s, t) where s is a discrete component representing a snapshot of all the data (or more precisely those data that are *relevant* for the temporal business rules)and t is a time stamp. Using s, boolean constraints may be decided, ie. if a new assistance has occured, or the balance of a bank account is below a certain treshold. A temporal business rule for a database is a then specification on how the internal *state* of the database may evolve over time, and what should happen when a rule is satisfied (or the opposite) by the data in the database. This can be done as follows

$$\text{IF TC THEN Action}_1\text{ELSE Action}_2,$$

where TC is a temporal condition and Actions 1 and 2 are some action to be performed. The core of our temporal conditions are given by *Timed LTL*, LTL_t, see [1], in the following abstract syntax

$$\begin{aligned}\phi ::= p \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \texttt{ALWAYS}\ \phi \mid \texttt{ALWAYS}_\texttt{c}\ \phi \\ \mid \texttt{EVENTUALLY}\ \phi \mid \texttt{EVENTUALLY}_\texttt{c}\ \phi.\end{aligned}$$

where $p \in AP$ and $c \in I\!N$.

The syntactic elements are: atomic propositions which can be the occurence of insert, update, delete or that an attribute in the database is above/below a certain treshold. Further, logical connectives and then temporal operators `ALWAYS` and `EVENTUALLY` both of which may be equipped with a time bound c. Intuitively they mean the following: Where `ALWAYS` ϕ denotes that the formula ϕ must hold in all timepoints, $\texttt{ALWAYS}_\texttt{c}\ \phi$ only requires ϕ to hold in the coming c time units. Conversely, the formula $\texttt{EVENTUALLY}_\texttt{c}\ \phi$ requires that formula ϕ is satisfied *before* c time units have passed, and thus it is a more restrictive operator than `EVENTUALLY` which only requires the sub–formula to hold at some point arbitrarily long away in the future. We shall use the standard abbreviation such as expressing implication etc. as logical connectives, e.g. $\phi_1 \Rightarrow \phi_2$ for $\neg\phi_1 \vee \phi_2$ and to use *true* instead of $\neg p \vee p$.

Using this language we may define temporal business rule in which the temporal condition is the following:

$$\texttt{ALWAYS}\ (new(C.Assistance) \Rightarrow \texttt{ALWAYS}_{30}\ \neg new(C.Assistance))$$

In other words, now we have a business rule which is broken in case assistances for a customer happens too often. And, it is a rule which is not built into the database itself, rather it is going to be governed by the Verification Server.

Checking Temporal Business Rules The algorithm in the verification server works by a rewriting principle. For each new state encountered from the database the algorithm *rewrites* the temporal constraints to a new formula which states what should hold from now. Such a new formula is called a residual formula. In the algorithm below the residual of formula ϕ with respect to a timed state σ_i is denoted by ϕ/σ_i.

Algorithm: Runtime Verification procedure

Let $\sigma = \sigma_0\sigma_1\ldots$ be a timed trace, let ϕ be a formula, let $exists(\sigma_i, time)$ be a predicate which is true exactly when σ contains σ_i and $t_i = time$ and let forceEvaluation() be a method which returns the systems current state, at the current time.

$\psi := \phi/\sigma_0$ $i := 1$ $time := 1$ $sit := SIT(\psi)$
while $\psi \neq true \wedge \psi \neq false$ **do**
  if $exists(\sigma_i, time) \wedge s_i \neq s_{i-1}$ **then**
    $\psi := \psi/\sigma_i$
    $sit := SIT(\psi)$

$i++$
end if
if $time = sit$ **then**
$\sigma_i := forceEvaluation()$
$\psi := \psi/\sigma_i$
$sit := SIT(\psi)$
$i++$
end if
$time++$
end while
if $\psi = true$ **then**
return "yes"
end if
if $\psi = false$ **then**
return "no"
end if

Occasionally, the satisfaction (or falsification) of a formula is not triggered by a change of state in the database, but rather by the elapse of time alone. For example, in the example above, after 30 days from the last car assistance to customer C, the yellow warning light may be turned off. In this case we say that 30 days is a so–called *interesting time point*. Now, since we do not assume the database to emit its internal state at each and every time point (this would bee far too inefficient!) we have no direct way of knowing the state of the database at such interesting time points. However, to ensure that the algorithm is timely complete we let the verification server compute the coming smallest interesting timepoint (SIT) and then insert an artificial state at such a time point[2]. See [4] for a complete description of the algorithm. The rewrite principle is a timed extension of the one in [3].

4 The prototypes

Based on the notation and algorithm introduced in the last section a verification server was implemented. The verification server is designed as an independent program running parallel to the business systems of the company. A first prototype parsed a rule in LTL notation and verified trace of state changes to a simple (one-table) database complied with the rule. Recently this prototype was extended to handle more than one rule with a more general way to interact with the database of the business system. For each rule an independent thread is started that takes care of the verification of that rule. This also allows the different rules to be based on different time units. In the momentary version, the verification server queries the database for relevant state changes once each of time unit specified as the basic time unit of the rule. Parallel a prototype for a rule builder is developed. We explore different alternatives for the user interface to find a notation for the temporal business rules and to construct them in a way that both leads to correctly

[2] The discrete component of the artificial state is the one appearing in the most recent state emitted by the database.

formulated expressions in LTL and allows for a meaningful interpretation from a business point of view.

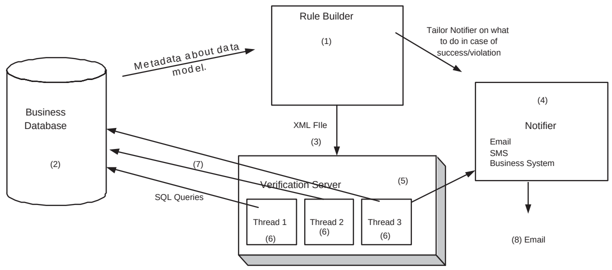

Fig. 1. The system architecture.

The system so far looks as depicted in figure 1. When the business rule expert wants to insert the rule that customers requiring vehicle assistance more than once within 30 days, he opens the *rule builder* (1)[3]. The system provides him with an overview of the attributes he can base his rules on by querying the database for its data model (2). He selects the main table modeling the instances of assistance. When he is satisfied and presses the save button the rule builder constructs an XML file (3) containing the LTL representation of the rule, the time unit the rule is based on and sql statements to access the data defining the states the rule is observing. He then has to define what should happen in case of the violation of this rule in the administrator interface of the *notifier*. (4) He decides that an e-mail should be sent to the person responsible for handling expensive customers. The *verification server* (5) parses the rule and creates and internal representation of the rule for the transformation algorithm. A new thread is started for the rule and parameterised with the respective sql queries and the time unit for the rule (6). Each thread independently accesses the database to obtain the information on state changes that are of interest when verifying the fulfillment of the rule (7). For our example rule the assistance table is accessed. When a rule is violated a notification is sent to the notifier. (4) The notifier sends an e-mail (8) to the one responsible for expensive customers as specified by the business rule expert.

5 Conclusions and future research

The prototypes described in the previous section provide a prove of concept for this kind of general way of checking temporal business rules. Our prototype system can handle several temporal rules based based on the state of different items in the database. The thread based design allows to define different time units for the different rules. It is possible to define different reactions, in case a temporal rule is broken. Besides sending a notification to this part of the system, it continues checking that the database changes

[3] The numbers refer to the numbers in figure 1.

comply with the different rules. However there are several issues that still are subject to future exploration. We plan to set up the verification server with a database to administer student projects here at our university to explore the following issues.

Using the production Database as is, or defining a specific view The solution the database prototypes implement today offers the whole database to the business expert to define temporal rules. Also the verification server accesses the database without requiring any specific functionality on the database side. This solutions is very general and requires no changes at the database side. However, in a normalised database, the data belonging together from the business point of view is often distributed over several tables. Data might be stored in a specific way due to requirements in the business systems. Production databases often contain a huge amount of tables of which only few might be interesting when defining temporal business rules. Defining a specific view for the verification server would on the one hand allow to provide the user with a concise overview of the kind of data relevant for the formulation of temporal business rules. It would have the additional advantage that it would allow to change the database without interfering with the already defined temporal rules. On the other hand it would constrain the formulation of rules to what is provided in the view. As long as changes in the temporal business rules do not go beyond that scope, the changes can be implemented by the business expert. In case other parts of the database have to be accessed, a database expert has to extend the view.

Checking the whole data of only accessing changes In the momentary implementation of the communication between the database and the verification server, the verification server queries the production table for data that fulfills the different boolean expressions the rules are based on. This way relevant changes will automatically be recognized. For large databases this might lead to performance problems, especially when short time intervals are used as a base for the rules. Also, with the momentary implementation of the interaction between the database and the verification server the example with the VAC would not work: The select statement check for changes violating absolute boundaries in existing relations they do not check the introduction of a new entity.

Another solution would be to only access changes to the data under observation. This would require to create a buffer and specific triggers as part of the database based on the definition of the rules in the rule builder. This should be possible but would require to change the business database itself. The rule builder module would have to keep track with what kind of changes it implemented in the database and would have to undo them in a controlled way when a temporal rule is change or erased. A third possibility would be to implement an observer pattern between the database and the verification server. Here again, the database would have to be adapted based on the rules defined by the user to create the events the verification server needs for checking the temporal business rules.

Finding a suitable user interface Research on End-User Development has shown that users are able to handle complex computations defined in a formal way when the formalisms are presented in a professionally meaningful way. [6][9] We plan to experiment with different kinds of user interfaces in order to find a suitable way to represent the data temporal rules can be formulated about and to support the user in deploying the possibilities that formalism offers. From a technical point of view the notifier and the rule builder are two very different parts of the system. From a business perspective the definition of temporal rules and the definition of what should happen in case of a violation belong together. Here the two tailoring interfaces must be integrated. Another issue that needs more exploration is the requirements such a system poses on the organization: when business rules can be (re-)defined, the organization has to implement procedures to decide what rules to implement.[8][2]

Performance So far we only tested the verification server based on toy examples. Implementing a sharp version together with the project database of our university will give us the opportunity to evaluate the design and performance of the verification server when interacting with a small but realistic database. The performance of the concrete verification algorithm as well as the influence of the different design possibilities for the interaction with the database can then be evaluated under more realistic conditions.

6 Acknowledgments

Thanks to Bente B. Petersen, Divya R. Malemane, Bue Pedersen, Ulrik D. Olsen who together developed the different prototypes the article is based on as part of their Master Theses.

References

1. 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.
2. Y. Dittrich, O. Lindeberg. Designing for Changing Work and Business Practices. In: N. Patel (ed.). Evolutionary and Adaptive Information Systems. IDEA group publishing, 2002.
3. K. Havelund, G. Ruso. Monitoring Programs using Rewriting. Automated Software Engineering 2001 (ASE'01), San Diego, California, 26-29 November 2001, IEEE Computer Society.
4. 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). Appears in Issue 89.2 of Electronic Notes in Theoretical Computer Science.
5. C. H. Kung. A Temporal Framework for Database Specification and Verification. Proceedings of the Tenth International Conference on Very Large Data Bases, Singapore, August, 1984.
6. F. Patern, M. Klann, V. Wulf. Research Agenda and Roadmap for EUD. Deliverables of the Network of Excellence on End-User Development December 2003. (http://giove.cnuce.cnr.it/eud-net.htm)
7. Time-Rover Corp. Temporal Business Solutions. The DB Rover. 2000.
8. R. Trigg, S. Bdker. From Implementation to Design: Tailoring and the Emergence of Systematization in CSCW. Proceedings of the CSCW 94, ACM-Press, New York, 1994, pp. 45-55.
9. O. Stiemerling, H. Kahler, V. Wulf. How to make software softer- Designing tailorable applications. Proceedings of the Designing Interactive Systems (DIS) 1997.