2016-09-28

Online, library-based visual formal specification monitoring system for monitoring log-files with visible and hidden data

Drusinsky, Doron

SpringerLink

Innovations Syst Softw Eng (2017) 13:67–79
DOI 10.1007/s11334-016-0286-6

ORIGINAL PAPER

Online, library-based visual formal specification monitoring system for monitoring log-files with visible and hidden data

Doron Drusinsky[1]

Received: 3 July 2016 / Accepted: 8 September 2016 / Published online: 28 September 2016

Abstract Runtime Monitoring (RM), also known as Runtime Verification (RV), is the process of monitoring and verifying the sequencing and temporal behavior of an underlying application and comparing it to the correct behavior as specified by a formal specification pattern. Hidden Markov Model (HMM)-based RM enables the monitoring of systems with both visible and hidden data, using the same formal specifications used by deterministic RM. This paper describes an online library of formal specification oracles and an accompanying toolset for the runtime monitoring log-files that contain hidden and visible data.

Keywords Runtime monitoring · Formal specifications · UML · Hidden Markov models

1 Introduction

Formal Specifications (FS) are mathematically based languages for assisting with the implementation and assurance of systems and software. Numerous FS languages have been proposed over the past four decades, primary of which are temporal logics [4,15,16] and diagrammatic languages, such as statechart assertions [5], the FS formalism used by this paper. There are two primary categories of temporal logics, linear time and branching time, with Propositional Linear time Temporal Logic (PLTL) [17] being the better known linear time FS language, and Computational Tree Logic (CTL and CTL*) being the better known variant of branching time logic. FS languages do not necessarily enjoy the same descriptive power. They differ in their domain of discourse: branching time logics assert about computation trees, whereas other languages assert about sequences. They also differ in descriptive power: PLTL is strictly sub-regular and, therefore, weaker than any finite state machine formalism; hence enters Regular LTL [14], which combines PLTL with regular expressions. Nevertheless, as described by the V&V tradeoff cuboid of [10], some formal verification techniques use weaker FS languages to achieve greater verification coverage. Section 2 overviews this tradeoff cuboid in greater detail.

Runtime Verification (RV) of formal specification assertions is a class of methods for monitoring the sequencing and temporal behavior of an underlying application and comparing it to the correct behavior as specified by a formal specification pattern [22]. In [21], the authors describe the application of RV using PLTL for system health management of real-time systems. [13] describes Runtime Verification with Particle Filtering (RVPF), a method for controlling the tradeoff between uncertainty and overhead in runtime verification. In [5,6], Drusinsky describes the application of RV using statechart assertions to the verification of U.S Department of Defense (DoD) and NASA applications, and to those of the Brazilian Space agency. The StateRover [23] is a set of Eclipse IDE plugins for UML statechart-based modeling, specification, validation, and runtime verification.

A Hidden Markov Model (HMM) can be considered as a state machine in which state transitions and state outputs, or observations, are probabilistic. HMMs are used to learn and classify sequences of observables. HMM technology has been used successfully in a diverse set of applications, such

This research was funded by a grant from the Defense Threat Reduction Agency (DTRA). The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

✉ Doron Drusinsky
ddrusins@nps.edu

[1] Computer Science Department, Naval Postgraduate School, Monterey, CA 93943, USA

as speech recognition [18], Gene prediction [Rä], and Crypt-analysis [24].

In [7,9], the author describes an RM technique and algorithm for systems with hidden and visible inputs and outputs. The technique uses deterministic statechart assertions as formal specification. These specifications contain no probability measures, i.e., the end user is not required to obtain a sample space, or otherwise generate probability measures, for the requirements. Rather, the end user needs to create a spreadsheet representation of the HMM outputs as a function of visible artifacts; the tool set then automatically generates the HMM from this spreadsheet and uses it to perform probabilistic verification.

This paper described the process and tools used to create an RM oracle for checking log-files with hidden and visible data against supervising formal specifications. The toolset consists of two primary parts:

1. The rules4business.com [19] online specification patterns and RM site is described in Sect. 2. It consists of a library of generic Natural Language (NL) rules, their corresponding formal specifications in the form of statechart assertions. This service performs RM by executing customized rule instances in the presence of comma-separated (csv) log-file data. The service also provides a visual audit for the RM process. The rules4business.com service does not cater for hidden artifacts within the log-file data sets; hence enters part 2 below.
2. The DTRA toolset is described in Sect. 3. It enables the enhancement of rules created and tested within the rules4business environment by enabling RM of comma separated (csv) log-files with both visible and hidden data.

This two-tier approach is useful in that the end user can first specify and test his or her rule instances assuming all inputs are visible. Once the end user has gained confidence in the correctness of his or her resulting rule instances they proceed to the second part where some of the data are hidden.

Section 4 describes a few application domains of the combined toolset.

[3] describes an approach for learning Temporal Logic formal specification properties of a dynamic system using statistical methods. Our approach does not learn such properties, but rather uses a library of pre-defined properties. In addition, for reasons discussed in Sect. 2, we use visual statechart assertion properties as our formal specification language of choice.

[1,2] describe an approach Adaptive Runtime Verification (ARV), where overhead control, runtime verification with state estimation, and predictive analysis are all combined. This technique uses HMMs in the loop, as we do, but differs from our approach in that: (i) it is tailored for state estimation,

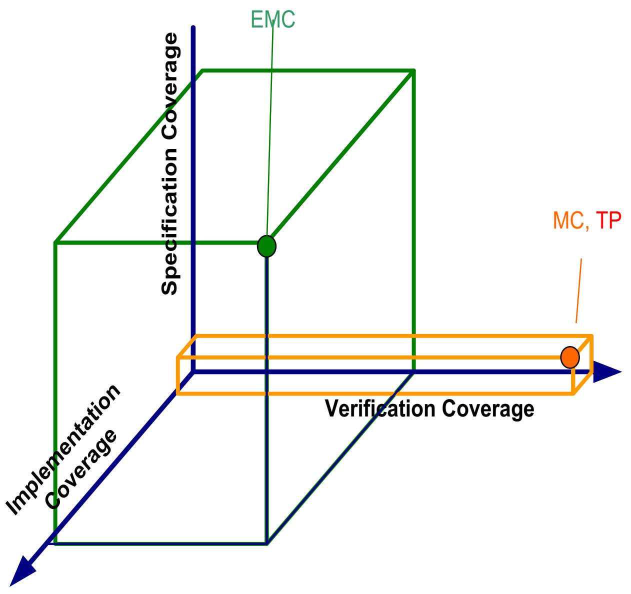


Fig. 1 The coverage tradeoff cuboid [10]

while our approach performs RM on general log-file, and (ii) this technique has no accompanying rule library, online service, or visual audit. In fact, their technique is similar to an earlier technique published by the author [8], where formal specifications and HMMs in the loop are used to monitor a Kalman Filter.

2 An online, library-based, visual formal specification Oracle

In [10], the authors present a visual tradeoff space—depicted in Fig. 1—called the Formal Verification and Validation (FV&V) tradeoff cuboid, for software engineers to discuss the various tradeoffs (e.g., cost, coverage, etc.) between different FV&V approaches to select the appropriate techniques for the FV&V of high-integrity software-intensive systems. The dimensions of this tradeoff space are: *Verification coverage* (qualitatively measuring a given FV&V technique's coverage of all branches and paths a program can execute), *Specification coverage* (measuring a specification language's expressive power), and *Program coverage* (qualitatively measuring the types of programs a given FV&V technique can be applied to). It follows from the cuboid that many FV&V techniques settle for a less expressive specification language (e.g., PLTL, known to be sub-regular) to achieve higher verification coverage. In contrast, RM, being a form of executable verification, does not need to compromise on the descriptive power of its chosen specification language and can, therefore, use a more expressive language. Consequently, rules4business uses statechart assertions as its formal specification language of choice; they are Turing equivalent, yet visual and intuitive, and also enable visual debugging. Note that although rules4business uses a library of seemingly pre-defined generic rules, those rules are actually programmable in the following two ways. First, as discussed in the sequel, all events, time bounds, and counting constraints are programmed (instantiated) using a script language (Javascript or Python). In addition, additional rules can be added to the library using the StateRover tool [6,23], the tool used to create the current rules in the library. Both capabilities are described in greater detail in the sequel.

The rules4business online rule specification and RM service contains a library of three-dozen generic rules that enable the specification of sequencing and temporal rules. Each such rule is provided in two forms: NL, and a corresponding statechart assertion formal specification. The following list is a sample of the generic rules available on rules4business:

- Rule 7 (8): Flag whenever event Q occurs (does not occur) every cycle between events P and R.
- Rule 9 (10): Flag whenever some pair of consecutive E events is less (more) than time T apart.
- Rule 11 (12): Flag whenever event P with (with no) eventual event Q within time T after P.
- Rule 16 (17): Flag whenever event P and eventually Q outside (within) time interval T1 T2 after P.
- Rule 19: Flag whenever more than N events E within time T after Q.
- Rule 20 (23): Flag whenever event Q occurs more (fewer) than N times between events P and R.
- Rule 21 (24): Flag whenever event Q occurs more (fewer) than N times between some pair of consecutive E events.
- Rule 27 (29): Flag whenever more (fewer) than N events E occur within one of a series of consecutive T intervals.
- Rule 34 (35): Flag whenever more (fewer) than N events E within time T prior to Q.

For the purpose of RM, the end user must instantiate (customize) one or more generic rules, as follows:

1. Customize event names, such as E, P, or Q, by associating them with the csv log-file; more details about this step are provided in the sequel.
2. Customize time bounds, such as T, T1, and T2. For example, rule 11 requires the specification of T; the end user customizes T in the rules4business.com web page by writing $T = n$, n being an integer number, and also selecting a time unit (year, month, week, day, hour or minute).
3. Customize counting constraints, such as specifying $N = 3$ for an instance of rule 27.

Every generic rule in the rules4business library is accompanied by a statechart assertion formal specification. A statechart assertion [5] is a UML statechart with a special Boolean flag. The assertion flags when an input scenario causes the statechart to reach the Flag state. Figure 2 depicts the statechart assertion diagrams for rules 9, 21, and 34 of the rules4business library. It also depicts timeline diagrams showing a scenario for each respective diagram. Note that the statechart diagram for rule 34 is non-deterministic; non-deterministic statechart assertions have the semantics of a Non-deterministic Finite Automata (NFA), i.e., using an existential acceptance rule given all possible NFA computations [12], with the *Flag* state being the equivalent of an NFA's accepting state [5].

The domain of discourse for the rules4business oracle RM is a log-file in the form of a comma delimited (csv) spreadsheet. Figure 3 depicts a snippet of the csv file used throughout this section.

Consider an example where log-files are bank statement csv files. In this context, consider generic rule 21 of Fig. 2c, whose Natural Language (NL) specification is: "Flag whenever event Q occurs more than N times between some pair of consecutive E events". After selecting this generic rule, the end user customizes Q, E, and N. An example of the customization of Q is "description.indexOf("Credit Card$''$) $>=$ 0 and amount $>$ 500". Q evaluates to true for every csv file row that contains the string "Credit Card" in the *description* column and also has an amount greater than 500 in the *amount* column. An example for E is "description.indexOf("Payroll") $>=$ 0", namely E evaluates to true for every csv file row that contains the word "Payroll" in the description column. An example for N is 1. Hence, the customized rule effectively means R21a: "flag whenever there is more than one credit card charge transaction of more than \$500 between two consecutive payroll transactions".

The rules4business oracle monitors (checks) an uploaded csv file against the end user's rule, thereby effectively performing RM using formal statechart assertion specifications. A visual audit of the oracle's decision (i.e., *Flag* or *no Flag* conclusion) is provided in the form of step-by-step animation of both the formal statechart assertion of the rule under investigation, and the uploaded csv file; the end user can move forwards and backwards in time while doing so. Figure 4 depicts the visual audit of rule 21 while evaluating the csv log-file shown in Fig. 3.

The purpose of visual audit is to help identify specification errors and to enhance trust in the service, not unlike the way programmers use a debugger in a conventional IDE.

Rules4business rules are temporal and sequencing rules with time and counting constraints. Such rules are typically not specifiable by classical formal specification languages such as Propositional Linear Time Temporal Logic (PLTL) [10]. They also include rules that specify intervals (e.g., rule 27) and past time (e.g., rule 34).

Some rules4business rules enable data-specific pattern matching. Consider, for example, generic rule 25, depicted

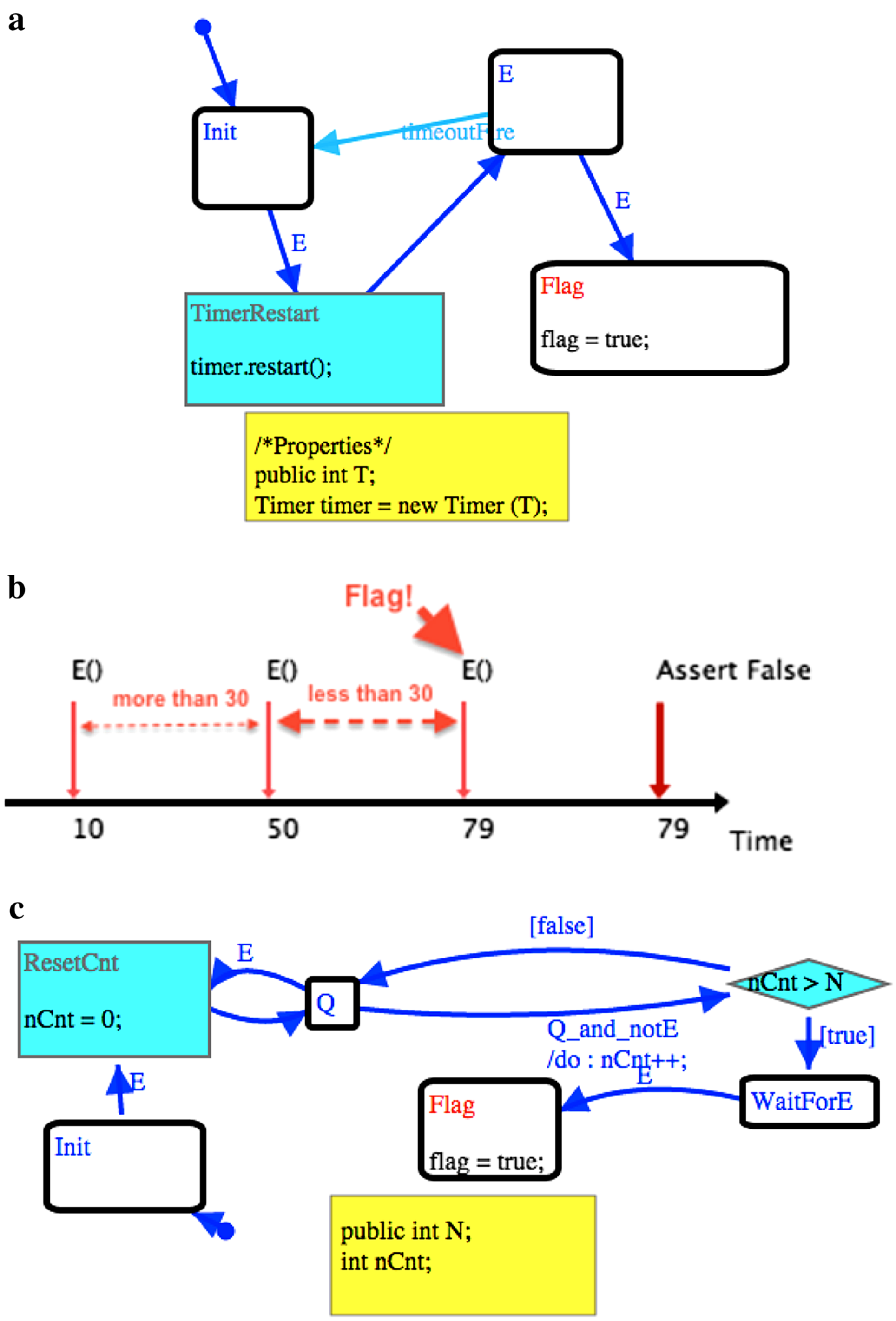


Fig. 2 Statechart assertions for rules 9, 21, and 34. **a** Statechart assertion for generic rule 9; its Natural Language (NL) specification is: "Flag whenever some pair of consecutive E events is less than time T apart". **b** Timeline diagram depicting a scenario for rule 9, using $T = 30$. **c** Statechart assertion for generic rule 21; its NL specification is: "Flag whenever event Q occurs more than N times between some pair of consecutive E events". **d** Timeline diagram depicting a scenario for rule 21, using $N = 3$. **e** Statechart assertion for generic rule 34; its NL specification is: "Flag whenever more than N events E within time T prior to Q". Note that the statechart is non-deterministic. **f** Timeline diagram depicting a scenario for rule 34, using $T = 30$ and $N = 3$

in Fig. 5. Its NL specification is: "Flag whenever more than N data-specific events E occur within time T after Q"; the following example explains the meaning of the term "data specific". Suppose we want a rule that flags when there are four or more expenses per week charged to the *same* organization, being one of PEETS, WHOLEFDS, or TARGET. In other words, the rule should *not* flag when there are two expenses to PEETS and two to TARGET, or four expenses to SAFEWAY, etc., rather, it should flag when there are four to TARGET, or four to PEETS, or four to WHOLEFDS. To that end, we customize the data construct, which stands for "data specific"; hence the customization parameters for rule 25 are now:

- E = *amount* > 0,
- Q = *amount* > 0, data=*description*.match('PEETS|WHOLEFDS|TARGET')[0].

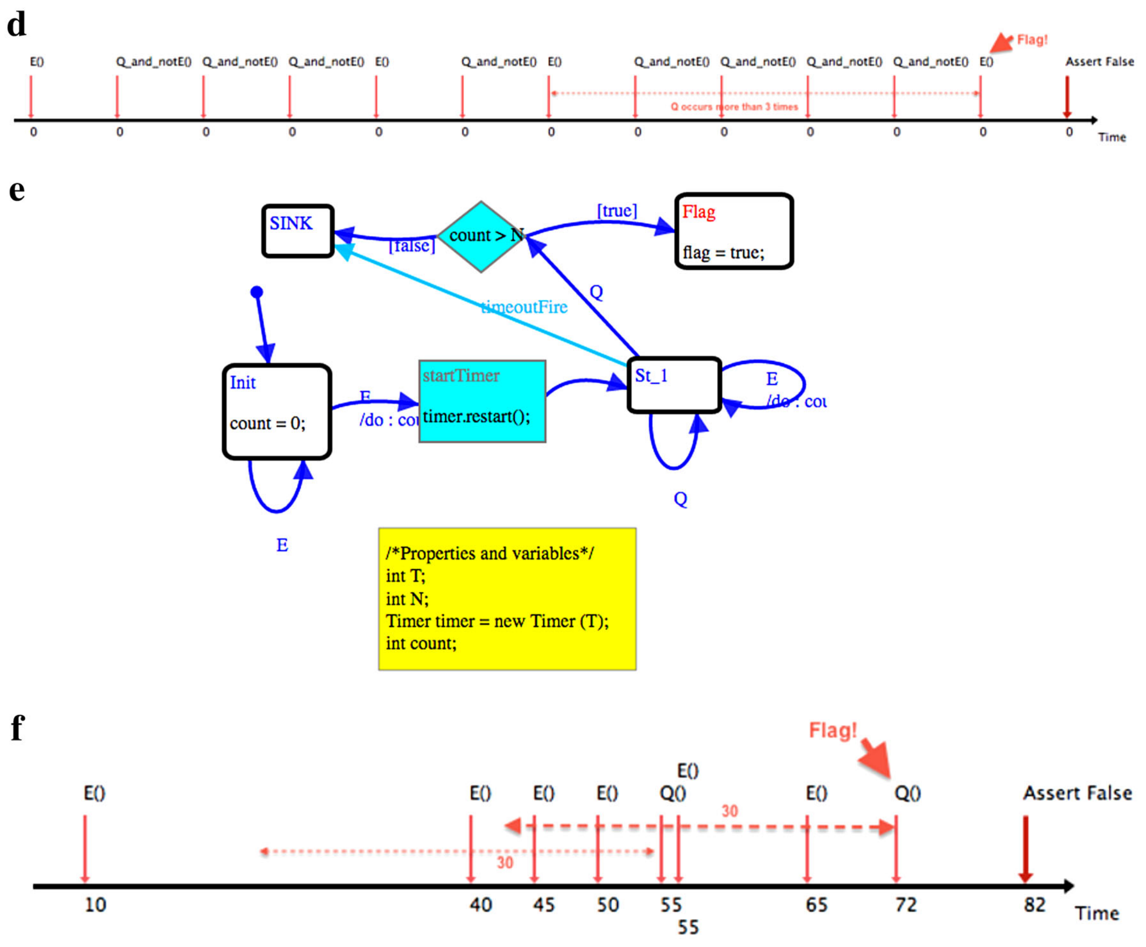
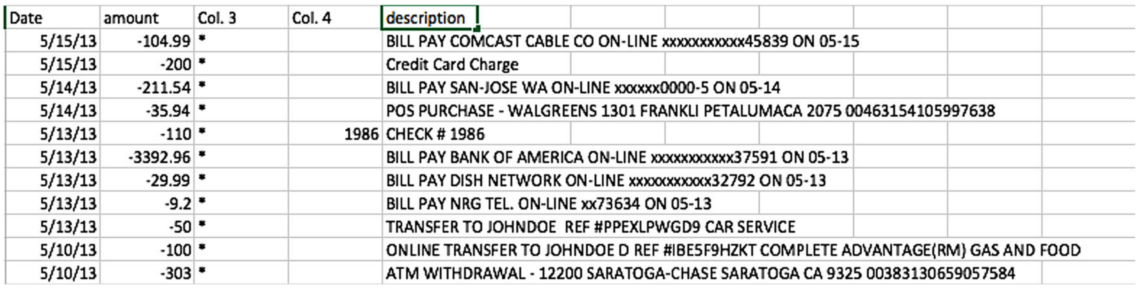


Fig. 2 continued



Date	amount	Col. 3	Col. 4	description
5/15/13	-104.99	*		BILL PAY COMCAST CABLE CO ON-LINE xxxxxxxxxxx45839 ON 05-15
5/15/13	-200	*		Credit Card Charge
5/14/13	-211.54	*		BILL PAY SAN-JOSE WA ON-LINE xxxxxx0000-5 ON 05-14
5/14/13	-35.94	*		POS PURCHASE - WALGREENS 1301 FRANKLI PETALUMACA 2075 00463154105997638
5/13/13	-110	*	1986	CHECK # 1986
5/13/13	-3392.96	*		BILL PAY BANK OF AMERICA ON-LINE xxxxxxxxxxx37591 ON 05-13
5/13/13	-29.99	*		BILL PAY DISH NETWORK ON-LINE xxxxxxxxxxx32792 ON 05-13
5/13/13	-9.2	*		BILL PAY NRG TEL. ON-LINE xx73634 ON 05-13
5/13/13	-50	*		TRANSFER TO JOHNDOE REF #PPEXLPWGD9 CAR SERVICE
5/10/13	-100	*		ONLINE TRANSFER TO JOHNDOE D REF #IBE5F9HZKT COMPLETE ADVANTAGE(RM) GAS AND FOOD
5/10/13	-303	*		ATM WITHDRAWAL - 12200 SARATOGA-CHASE SARATOGA CA 9325 00383130659057584

Fig. 3 A snippet of a sample csv log-file used by the rules4business oracle

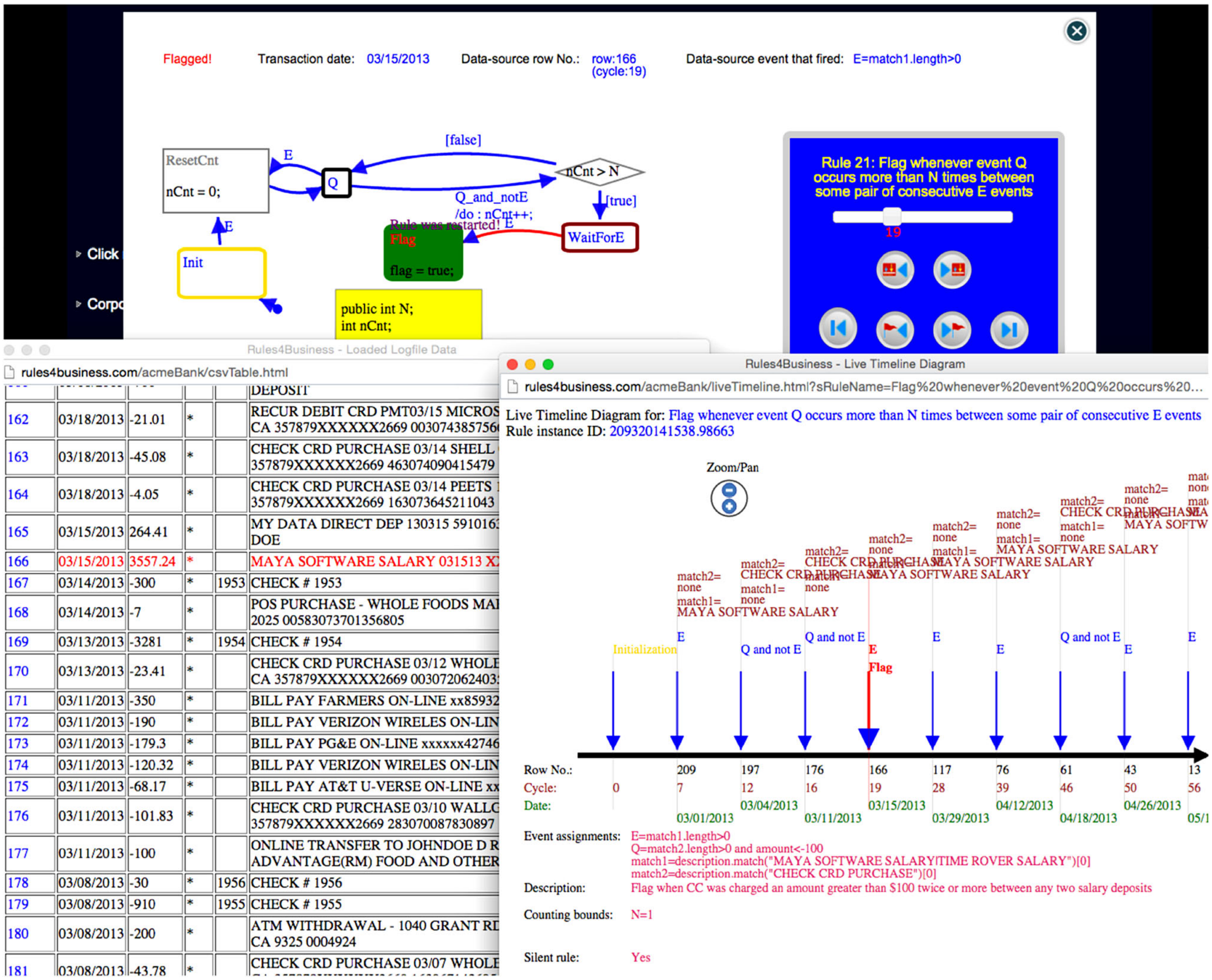
- N=3,
- T=1-week,

2.1 Rules4business service architecture

There are two actors that interface with the rules4business system: a rule specification expert who creates generic rules, and subject matter expert who uses generic rules by customizing them and uploading csv log-files for monitoring and verification purposes.

The rules4business service consists of four main components:

1. *Generic rule creation*. Generic rules are created by the rule specification expert using the StateRover IDE and automatic code generator [5,23]. The service already contains three-dozen rules but those can be changed. StateRover diagram files are then manually uploaded to the rules4business server where they go through automatic code generation, resulting in an executable Java implementation and corresponding Java class file.



Flagged! Transaction date: 03/15/2013 Data-source row No.: row:166 (cycle:19) Data-source event that fired: E=match1.length>0

Rule 21: Flag whenever event Q occurs more than N times between some pair of consecutive E events

Rules4Business - Loaded Logfile Data

rules4business.com/acmeBank/csvTable.html

					DEPOSIT
162	03/18/2013	-21.01	*		RECUR DEBIT CRD PMT03/15 MICROS CA 357879XXXXXX2669 00307438575
163	03/18/2013	-45.08	*		CHECK CRD PURCHASE 03/14 SHELL 357879XXXXXX2669 463074090415479
164	03/18/2013	-4.05	*		CHECK CRD PURCHASE 03/14 PEETS 357879XXXXXX2669 163073645211043
165	03/15/2013	264.41	*		MY DATA DIRECT DEP 130315 591016 DOE
166	03/15/2013	3557.24	*		MAYA SOFTWARE SALARY 031513 X
167	03/14/2013	-300	*	1953	CHECK # 1953
168	03/14/2013	-7	*		POS PURCHASE - WHOLE FOODS MA 2025 00583073701356805
169	03/13/2013	-3281	*	1954	CHECK # 1954
170	03/13/2013	-23.41	*		CHECK CRD PURCHASE 03/12 WHOLE CA 357879XXXXXX2669 00307206240
171	03/11/2013	-350	*		BILL PAY FARMERS ON-LINE xx85932
172	03/11/2013	-190	*		BILL PAY VERIZON WIRELES ON-LIN
173	03/11/2013	-179.3	*		BILL PAY PG&E ON-LINE xxxxxx42746
174	03/11/2013	-120.32	*		BILL PAY VERIZON WIRELES ON-LIN
175	03/11/2013	-68.17	*		BILL PAY AT&T U-VERSE ON-LINE xx
176	03/11/2013	-101.83	*		CHECK CRD PURCHASE 03/10 WALLG 357879XXXXXX2669 283070087830897
177	03/11/2013	-100	*		ONLINE TRANSFER TO JOHNDOE D R ADVANTAGE(RM) FOOD AND OTHER
178	03/08/2013	-30	*	1956	CHECK # 1956
179	03/08/2013	-910	*	1955	CHECK # 1955
180	03/08/2013	-200	*		ATM WITHDRAWAL - 1040 GRANT RD CA 9325 0004924
181	03/08/2013	-43.78	*		CHECK CRD PURCHASE 03/07 WHOLE

Rules4Business - Live Timeline Diagram

rules4business.com/acmeBank/liveTimeline.html?sRuleName=Flag%20whenever%20event%20Q%20occurs%20...

Live Timeline Diagram for: Flag whenever event Q occurs more than N times between some pair of consecutive E events
Rule instance ID: 209320141538.98663

Row No.:		209	197	176	166	117	76	61	43	13
Cycle:	0	7	12	16	19	28	39	46	50	56
Date:		03/01/2013	03/04/2013	03/11/2013	03/15/2013	03/29/2013	04/12/2013	04/18/2013	04/26/2013	05/1

Event assignments: E=match1.length>0
Q=match2.length>0 and amount<-100
match1=description.match("MAYA SOFTWARE SALARY|TIME ROVER SALARY")[0]
match2=description.match("CHECK CRD PURCHASE")[0]

Description: Flag when CC was charged an amount greater than $100 twice or more between any two salary deposits

Counting bounds: N=1

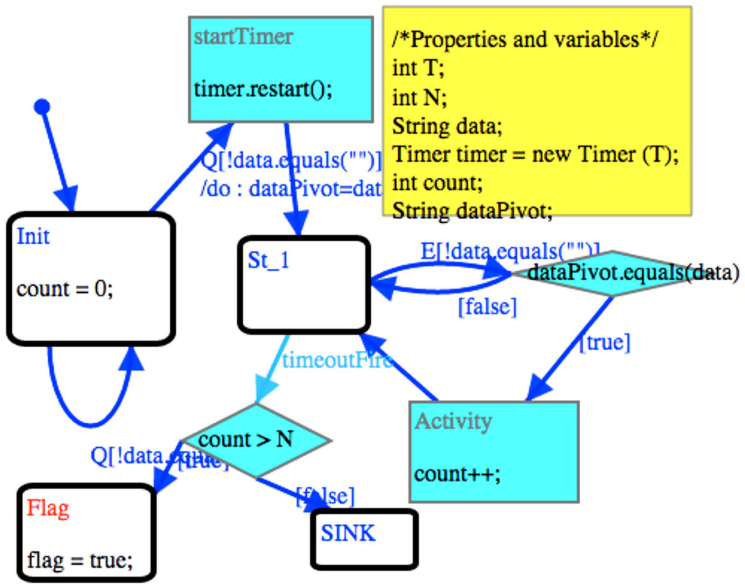
Silent rule: Yes

Fig. 4 A visual audit of rule 21's evaluation. It consists of three synchronized animated views: the statechart assertion, a timeline diagram, and the uploaded csv log-file

In addition, all diagrams are converted to SVG vector graphics, a form of graphics supported by modern web browsers.

2. *Rule customization service*. As described earlier, the end user manually customizes rules supplying small Javascript or Python snippets to rules4business, using a Web user interface. These snippets, along with time and counting constraints, and the ID of the generic rule being customized, constitute a *rule instance*; they are stored in a MySQL database (DB).
3. *Rule execution service*.
   (a) The engine (automatically) locates the class file per each of the end user's stored rule instances, and loads it dynamically.
   (b) As depicted in Fig. 6, when the end user uploads a csv log-file, that file is evaluated read row by row, each row inducing possible events, as follows. Suppose a rule instance I contains the event handler Q, customized as "description.indexOf("Credit Card$''$) $>=$ 0 and amount $>$ 500". This Javascript code is automatically executed on the server using the data in the *description* and *amount* cells of the current csv row; if the snippet evaluates to *true* then an event Q is fired on I (Q is implemented as a class method—firing Q is but a method call).
4. *Visual audit service*. While evaluating the csv file, the server stores a log of all statechart events that fired along with their corresponding csv row information. Upon request, these data are automatically downloaded to the end user's browser in JSON format. In addition, as depicted in Figs. 4 and 5b, Javascript on the browser/client side then automatically animates the SVG representation of the statechart assertion, a timeline diagram, and a browser copy of the input csv file, all in a

a



b

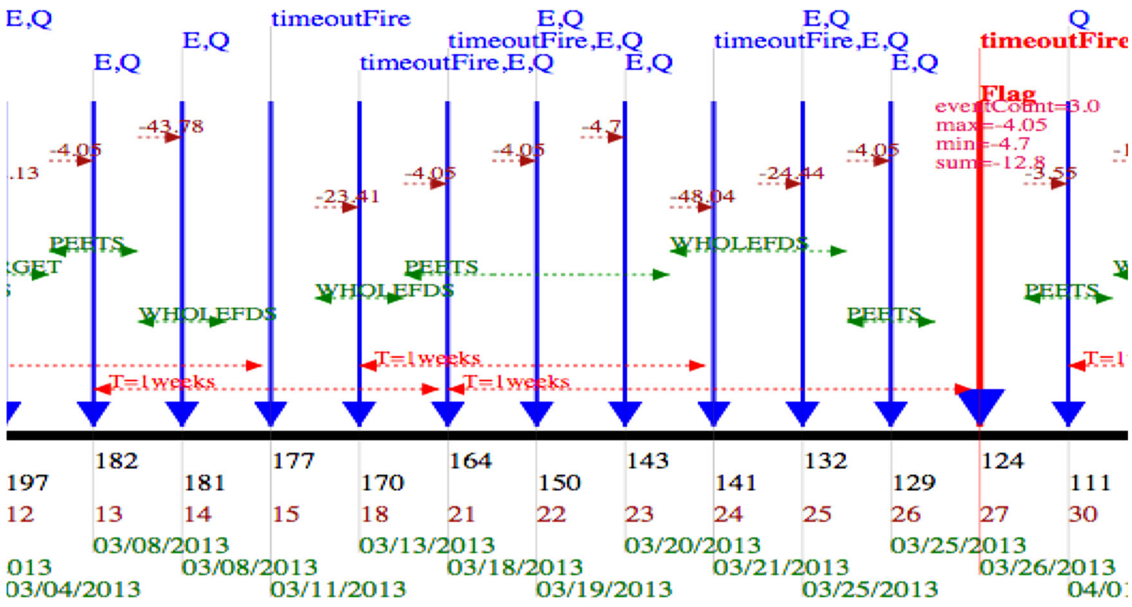


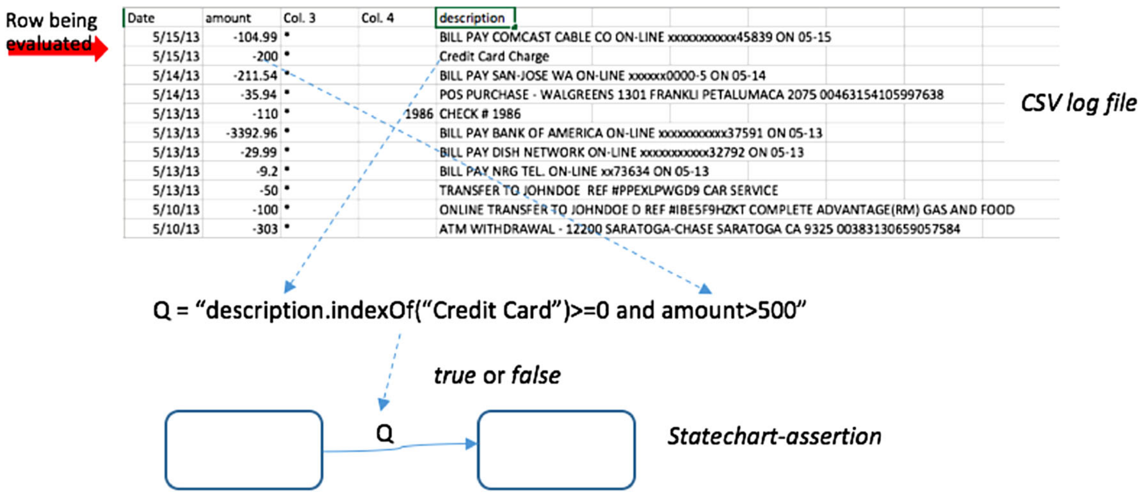
Fig. 5 The statechart assertion and timeline diagram for rule 25. **a** Statechart assertion for rules4business generic rule 25. **b** A snippet of the rules4business timeline diagram visual audit for the evaluation of generic rule 25 per the log-file in Fig. 3. It show four PEETS purchases on the week prior to 03/26

synchronized manner, thereby providing the end user a visual audit. The purpose of this audit is to help identify specification errors and to enhance trust in the service, not unlike the way programmers use a debugger in a conventional IDE.

3 Monitoring systems with hidden data

As discussed in the introduction, the rules4business.com service discussed in Sect. 2 caters for RM of visible data only. The second primary component of our toolset supports RM of data that are partially or all hidden. In this part, the end user uses the rule instances she/he created and tested in the rules4business.com part.

The underlying technique in this part is based on HMM's, overviewed below. The algorithms used for RM of statechart assertion rules in the presence of hidden data are detailed in [9]. The main components of the toolset are described in this section.



Date	amount	Col. 3	Col. 4	description
5/15/13	-104.99	•		BILL PAY COMCAST CABLE CO ON-LINE xxxxxxxxxx45839 ON 05-15
5/15/13	-200	•		Credit Card Charge
5/14/13	-211.54	•		BILL PAY SAN-JOSE WA ON-LINE xxxxxx0000-5 ON 05-14
5/14/13	-35.94	•		POS PURCHASE - WALGREENS 1301 FRANKLI PETALUMACA 2075 00463154105997638
5/13/13	-110	•	1986	CHECK # 1986
5/13/13	-3392.96	•		BILL PAY BANK OF AMERICA ON-LINE xxxxxxxxxxx37591 ON 05-13
5/13/13	-29.99	•		BILL PAY DISH NETWORK ON-LINE xxxxxxxxxxx32792 ON 05-13
5/13/13	-9.2	•		BILL PAY NRG TEL. ON-LINE xx73634 ON 05-13
5/13/13	-50	•		TRANSFER TO JOHNDOE REF #PPEXLPWGD9 CAR SERVICE
5/10/13	-100	•		ONLINE TRANSFER TO JOHNDOE D REF #IBE5F9HZKT COMPLETE ADVANTAGE(RM) GAS AND FOOD
5/10/13	-303	•		ATM WITHDRAWAL - 12200 SARATOGA-CHASE SARATOGA CA 9325 00383130659057584

Fig. 6 The RM process using customization script and a csv log-file

3.1 Hidden Markov models

A discrete emission HMM, of the kind used in this paper, is a variant of a finite state machine with a set of states S and a set of discrete emissions (also referred to as observables or outputs) O, as follows:

1. All pairs of states are connected by a transition.
2. For every transition $t : s1 \rightarrow s2, s1, s2 \in S : t$ is labeled with a probability measure, indicating the probability of that transition firing. There are no other labels annotating transitions.
3. For every state $s \in S$: the sum of probabilities annotating transitions from s to other states is 1.
4. For every state $s \in S$: s emits all outputs $o \in O$, each with a prescribed probability $b_s(o)$, where the sum of probabilities of all emissions in s is 1.

In a traditional HMM use case, the states are hidden and the emissions, also referred to as outputs, are considered visible. An HMM is typically created in a partially automatic manner, where the number of states and the output set O are given, and a subsequent learning phase algorithm [20] calculates the remaining HMM parameters, being: transition probabilities and output probabilities. A typical HMM use case is *classification*, where a sequence of observables is mapped to one of a plurality of candidate HMM's using a best fit algorithm. In such a use case, there is usually no interest in assigning a meaning to individual HMM states; rather the HMM itself is either selected or not selected as a best fit. As explained in the sequel, our use case is somewhat different in that the end user subject matter expert identifies the states by name and meaning.

3.2 Asserting about hidden data

While Sect. 2.1 identifies two actors that interface with the rules4business service, here we identify an actor too; usually this is the second actor of Sect. 2.1, but she/he could be a third person. This actor identifies hidden states and populates a HiddenState column in a learning phase csv file, as discussed below.

The NL rules and corresponding statechart assertions discussed in Sect. 1 assert about *visible* data within log-files, such as contents of the *amount* column or the *description* column of the csv file of Fig. 3. Consider the following variant of the banking-statement log-file oracle where each statement transaction (row) can be in one of three hidden states: Benign (B), Suspicious (S), or Fraud (F). These states are hidden because they do not appear in the log-file of Fig. 3. Rather, an HMM whose states are B, S, and F is learned from learning-phase log-files and is later used in runtime for the purpose of monitoring [9]. We now specify a variant of rule R21a, R21b: "flag whenever there is more than one *suspicious* credit card charge transaction of more than \$500 between two consecutive payroll deposits". R21b is modeled in rules4business using a customization of Rule21 where:

- Q is: description.indexOf("Credit Card") >= 0 and amount > 500 and HiddenState==S
- E is: description.indexOf("Payroll")>=0
- N is: 1

Note how Q refers to two visible csv columns (*amount* and *description*) as well as a hidden column named *HiddenState.* The hidden column exists only for learning purposes—in a learning phase csv log-file, i.e., when the HMM is learned. In runtime, RM is performed using csv log-files that has the same structure as the files used by rules4business service, i.e., files with no HiddenState column; instead, HiddenState information is derived from the HMM.

Section 3.2 describes the architecture and tools for implementing such RM.

3.3 A tool-kit for runtime monitoring of systems with hidden data

The toolset relates to two phases of the RM process: (i) HMM learning phase, and (ii) RM phase. The tools in the toolset are named after the sponsor, DTRA.

3.3.1 HMM learning phase tool

Recall that an HMM is a probabilistic finite state machine in which states (the *hidden states*) have probabilistic outputs. The learning-phase csv log-file consists of one *HiddenState* column (populated by a subject matter expert), and all other columns are considered to be the visible output columns. In other words, the learning-phase log-file describes the hidden state associated with a row (a tuple) of visible outputs. A sample of a learning-phase table is depicted in Table 1.

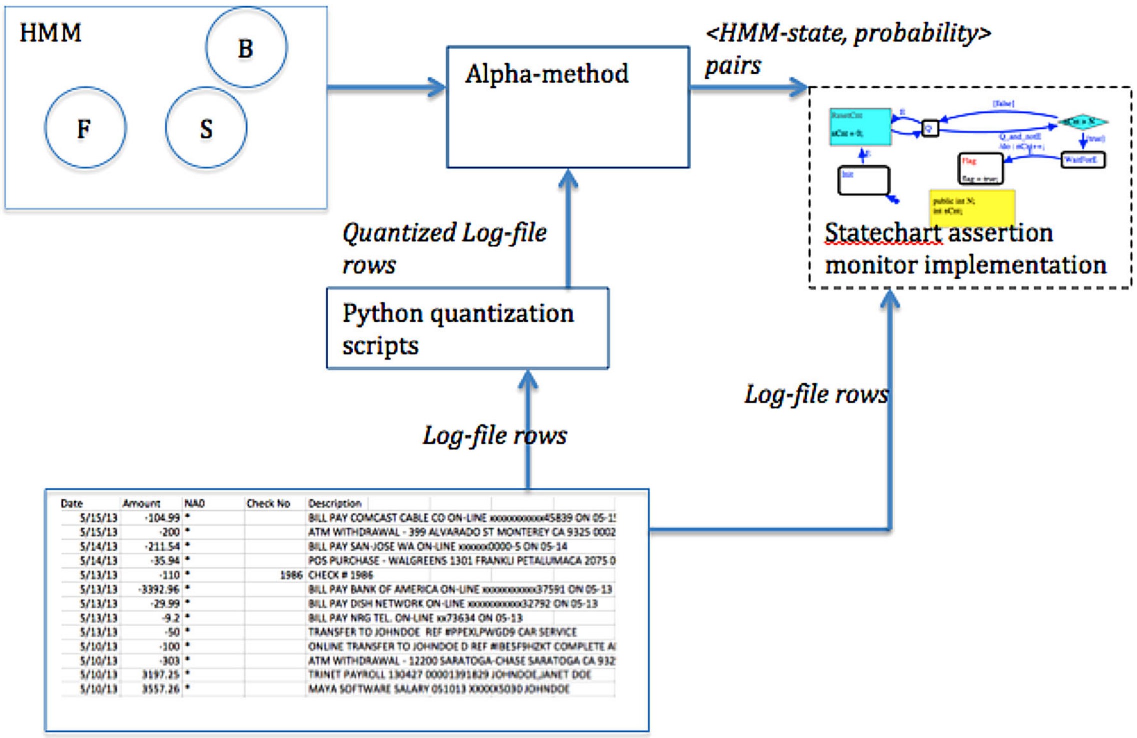
The *dtrahmm* tool creates an HMM from a learning-phase log-file, using the algorithm described in [9]. In a nutshell, this algorithm creates an HMM using frequency analysis, such as the ratio of rows where HiddenState="B" and where the next row has HiddenState="S" to the total number of rows minus one constitutes the probability of the HMM transition B $\rightarrow$ S.

Note that for the HMM induced by the learning-phase log-file of Table 1, an HMM output is an element of the Cartesian Product O $= Amount \times Description$, where A*mount* and *Description* are sets that consist of all possible values of the respective columns in the learning-phase log-file. Hence, for example, *amount* consists of all possible Dollar amounts that can possibly be listed in the Amount column. Clearly, the cardinality of O is too large to enable effective learning of the HMM. Hence, *dtrahmm* uses a quantized version of each column in the learning-phase log-file. For example, *amount'*, the quantized version of *amount*, consists of three possible values: *amountLT0*, *amount0to500*, and *amountGT500*, representing $amount < 0, 0 \leq amount \leq 500$, and $amount > 500$, respectively. The end user specifies these quantized values using a simple Python script that is provided to *dtrahmm* as an argument.

Table 1 Snippet of a learning phase log-file

Initial state	Date	Amount	Hidden state	Description
	5/15/13	−104.99	B	BILL PAY COMCAST CABLE CO ON-LINE xxxxxxxxxxx45839 ON 05-15
	5/15/13	−200	B	ATM WITHDRAWAL - 399 ALVARADO ST MONTEREY CA 9325 0002812
	5/14/13	−211.54	S	BILL PAY SAN-JOSE WA ON-LINE xxxxxx0000-5 ON 05-14
	5/14/13	−35.94	S	POS PURCHASE - WALGREENS 1301 FRANKLI PETALUMACA 2075 00463154105997638
	5/13/13	−110	F	CHECK # 1986
	5/13/13	−3392.96	B	BILL PAY BANK OF AMERICA ON-LINE xxxxxxxxxxx37591 ON 05-13
	5/13/13	−29.99	B	BILL PAY DISH NETWORK ON-LINE xxxxxxxxxxx32792 ON 05-13
	5/13/13	−9.2	B	BILL PAY NRG TEL. ON-LINE xx73634 ON 05-13
	5/13/13	−50	B	TRANSFER TO JOHNDOE REF #PPEXLPWGD9 CAR SERVICE
	5/10/13	−100	B	ONLINE TRANSFER TO JOHNDOE D REF #IBE5F9HZKT COMPLETE ADVANTAGE(RM) GAS AND FOOD
	5/10/13	−303	B	ATM WITHDRAWAL - 12200 SARATOGA-CHASE SARATOGA CA 9325 00383130659057584
	5/10/13	3197.25	B	TRINET PAYROLL 130427 00001391829 JOHNDOE,JANET DOE
	5/10/13	3557.26	B	MAYA SOFTWARE SALARY 051013 XXXXX5030 JOHNDOE
	5/9/13	−675	B	CHECK # 1980
	5/9/13	−78.47	B	CHECK CRD PURCHASE 05/08 WHOLEFDS FRK 10044 PETALUMA CA 357879XXXXXX2669 284852125152774 ?MCC=5457
	5/9/13	−44.15	B	CHECK CRD PURCHASE 05/08 CHEVRON 00090030 PETALUMA CA 357879XXXXXX2669 463139107653054 ?MCC=7642
	5/8/13	−4.05	S	CHECK CRD PURCHASE 05/06 PEET'S #10102 PALO ALTO CA 357879XXXXXX2669 163426855403466 ?MCC=5614

The HiddenState column describes the HMMs' hidden state; it is populated by a subject matter expert. All other columns are considered the visible outputs


HMM
B
F
S
Alpha-method
<HMM-state, probability> pairs
Statechart assertion monitor implementation
Quantized Log-file rows
Python quantization scripts
Log-file rows
Log-file rows

Date	Amount	NAO	Check No	Description
5/15/13	-104.99	*		BILL PAY COMCAST CABLE CO ON-LINE xxxxxxxxxxx45839 ON 05-1
5/15/13	-200	*		ATM WITHDRAWAL - 399 ALVARADO ST MONTEREY CA 9325 0002
5/14/13	-211.54	*		BILL PAY SAN-JOSE WA ON-LINE xxxxxx0000-5 ON 05-14
5/14/13	-35.94	*		POS PURCHASE - WALGREENS 1301 FRANKLI PETALUMACA 2075 0
5/13/13	-110	*	1986	CHECK # 1986
5/13/13	-3392.96	*		BILL PAY BANK OF AMERICA ON-LINE xxxxxxxxxxx37591 ON 05-13
5/13/13	-29.99	*		BILL PAY DISH NETWORK ON-LINE xxxxxxxxxxx32792 ON 05-13
5/13/13	-9.2	*		BILL PAY NRG TEL. ON-LINE xx73634 ON 05-13
5/13/13	-50	*		TRANSFER TO JOHNDOE REF #PPEXLPWGD9 CAR SERVICE
5/10/13	-100	*		ONLINE TRANSFER TO JOHNDOE D REF #IBESF9HZKT COMPLETE A
5/10/13	-303	*		ATM WITHDRAWAL - 12200 SARATOGA-CHASE SARATOGA CA 932
5/10/13	3197.25	*		TRINET PAYROLL 130427 00001391829 JOHNDOE,JANET DOE
5/10/13	3557.26	*		MAYA SOFTWARE SALARY 051013 XXXXX5030 JOHNDOE

Fig. 7 The architecture of the HMM-based runtime Oracle

3.3.2 RM phase

The runtime toolset implements the process described in [9], also illustrated in Fig. 7. As seen there, the process uses an HMM generated in a learning phase. When observing a runtime log-file, the Alpha method described in [9] processes the HMM and a runtime log-file and generates a list of $<$ HMM-state, probability $>$ pairs, one for every cycle, where a cycle corresponds to a row of the csv log-file (i.e., row No. 1 is cycle 1, row No. 2 is cycle 2, etc.). This list of pairs is provided to the RM algorithm implementing the statechart assertion.

The specific details of the tools used in this process are as follows.

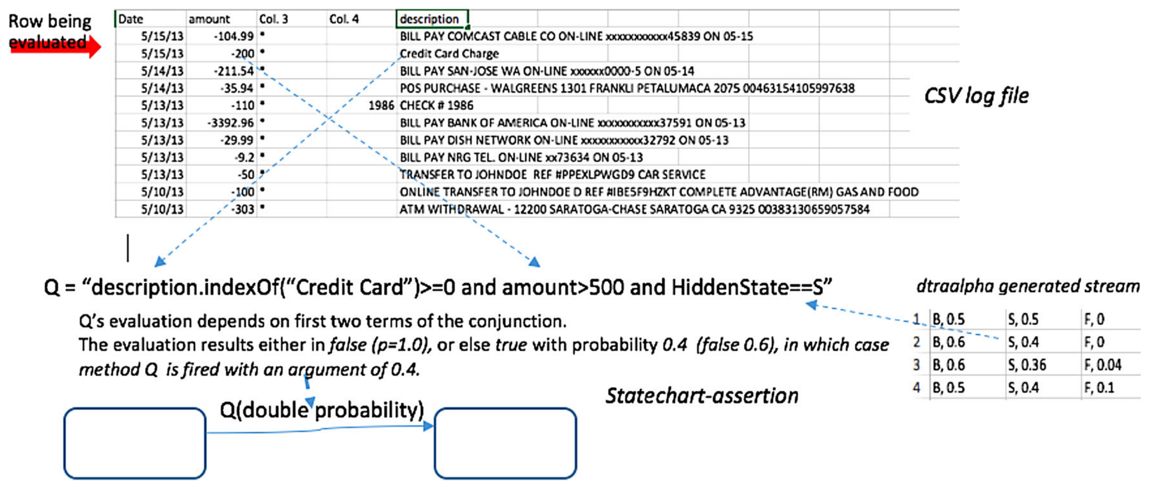
1. *dtraalpha*. This program implements the Alpha method component of Fig. 7. It is the Alpha method detailed in [9] (based on [20]). Its inputs are:
    - The HMM generated by *dtrahmm*.
    - Quantized log-file rows generated from a runtime log-file using the end user's quantization scripts.

Its output is a list of $<$ hidden-state, probability $>$ pairs, as described [9].

2. *dtracg*. This program implements the statechart assertion monitor component of Fig. 7, according to the algorithm of [9]. It takes the Java implementation of a rule, as automatically created by rules4business, and converts it to a monitor that operates on both visible data (rows of the log-files) and hidden data (list of <hidden-state, probability $>$ pairs generated by *dtraalpha*). This step does not execute that monitor; it generates its code only; *dtrarm* below executes the monitor.

In a nutshell, the Java implementation of a rule monitor is an extension of an implementation of an NFA, i.e., it contains a computation object for every NFA computation, where each computation object carries a probability measure (the probability of reaching the Flag state). Each computation is activated by calls to event handler methods, each method being an implementation of a transition such as a method $E(double\ probability)$ for the transition labeled E of Fig. 2a. The probability argument of the message handler is used by *dtrarm* below.

3. *dtrarm*. This program performs RM by executing monitor code generated by *dtracg*, as detailed in [9]. Its inputs are:
    - Java code for the rule's implementation; this is the output from *dtracg*.
    - Runtime log-file; this file contains the deterministic (visible) inputs. Note that the rule does not require a quantized version of the log-file; only the HMM and its Alpha method do.



Date	amount	Col. 3	Col. 4	description
5/15/13	-104.99	•		BILL PAY COMCAST CABLE CO ON-LINE xxxxxxxxxxx45839 ON 05-15
5/15/13	-200	•		Credit Card Charge
5/14/13	-211.54	•		BILL PAY SAN-JOSE WA ON-LINE xxxxxx0000-5 ON 05-14
5/14/13	-35.94	•		POS PURCHASE - WALGREENS 1301 FRANKLI PETALUMACA 2075 00463154105997638
5/13/13	-110	•	1986	CHECK # 1986
5/13/13	-3392.96	•		BILL PAY BANK OF AMERICA ON-LINE xxxxxxxxxxx37591 ON 05-13
5/13/13	-29.99	•		BILL PAY DISH NETWORK ON-LINE xxxxxxxxxxx32792 ON 05-13
5/13/13	-9.2	•		BILL PAY NRG TEL. ON-LINE xx73634 ON 05-13
5/13/13	-50	•		TRANSFER TO JOHNDOE REF #PPEXLPWGD9 CAR SERVICE
5/10/13	-100	•		ONLINE TRANSFER TO JOHNDOE D REF #IBESF9HZKT COMPLETE ADVANTAGE(RM) GAS AND FOOD
5/10/13	-303	•		ATM WITHDRAWAL - 12200 SARATOGA-CHASE SARATOGA CA 9325 00383130659057584

Q = “description.indexOf(“Credit Card”)>=0 and amount>500 and HiddenState==S”

Q’s evaluation depends on first two terms of the conjunction.
The evaluation results either in *false (p=1.0)*, or else *true* with probability *0.4 (false 0.6), in which case method Q is fired with an argument of 0.4.*

1	B, 0.5	S, 0.5	F, 0
2	B, 0.6	S, 0.4	F, 0
3	B, 0.6	S, 0.36	F, 0.04
4	B, 0.5	S, 0.4	F, 0.1

Fig. 8 The RM process of Fig. 6 in the presence of the alpha stream

- Pairs of < hidden-state, probability > generated by *dtraalpha*.

This component executes a log-file the almost same way the rules4business rule execution service (see Sect. 2.1; Fig. 6) does. It does so with two modifications, as illustrated in Fig. 8:

(a) Suppose a rule instance I contains the event handler Q, customized as “description.indexOf(“Credit Card”) $>=$ 0 and amount $>$ 500 and HiddenState==S”. The *description* and *amount* columns are visible; hence, as in section 2.1, *dtrarm* executes the snippet using the *description* and *amount* cells of the current csv row (row i). *HiddenState*, however, is known to be hidden; *dtrarm*, therefore, uses the probability of the i’th symbol of *dtraaplha* input stream (i being the index of the csv log-file row being evaluated) being “S” as follows. Q evaluates to *false* if any visible term (e.g., amount $>$ 500) is *false*. It evaluates to *true* otherwise, but with a probability measure, being the probability of the i’th symbol of *dtraaplha* input stream being equal to “S”.

(b) When an event handler, such as $Q(double\ probability)$ above, fires, it fires with a probability measure. The probability of the computation is then multiplied by this probability measure.

The *dtrarm* program generates a list of probabilities, one per cycle, where an element of the list contains the probability of the rule being flagged in the corresponding cycle. Listing 1 contains the output of *dtrarm* tool for rule R21b. It is often assumed that a probability threshold of 0.5 should be used to convert a probability of Flag to some resulting action, such as declaring a failed test. However, as described in [8], when RM is used for mission or safety critical applications, the threshold will likely be lower.

The reason *dtracg* and *dtrarm* are separate programs is that end users are expected to run *dtrarm* with a plurality of runtime csv log-files using the same implementation of a rule generated by *dtracg*.

```
The following is a list of probability values, one per cycle
(csv file row), indicating the probability of the monitor
reaching the Flag state in that cycle

Row 1: probability of Flag=0.0
Row 2: probability of Flag=0.0
Row 3: probability of Flag=0.0
Row 4: probability of Flag=0.0
Row 5: probability of Flag=0.0
…
Row 38: probability of Flag=0.0
Row 39: probability of Flag=2.220446049250313E-16
Row 40: probability of Flag=0.0
Row 41: probability of Flag=4.440892098500626E-16
Row 42: probability of Flag=2.220446049250313E-16
Row 43: probability of Flag=4.440892098500626E-16
Row 44: probability of Flag=4.440892098500626E-16
Row 45: probability of Flag=2.220446049250313E-16
…
Row 69: probability of Flag=2.220446049250313E-15
Row 70: probability of Flag=1.7763568394002505E-15
Row 71: probability of Flag=0.7982817064429559
Row 72: probability of Flag=0.7982817064429559
Row 73: probability of Flag=0.7982817064429558
Row 74: probability of Flag=0.7982817064429558
Row 75: probability of Flag=0.7982817064429558
Row 76: probability of Flag=0.7982817064429558
Row 77: probability of Flag=0.7982817064429559
Row 78: probability of Flag=0.7982817064429559
..
Row 230: probability of Flag=0.9999819258671948
Row 231: probability of Flag=0.9999819258671948
Row 232: probability of Flag=0.9999819258671948
Row 233: probability of Flag=0.9999819258671948
Row 234: probability of Flag=0.9999819258671948
Row 235: probability of Flag=0.9999819258671948
Done!
```

Listing 1. The output of *dtrarm*tool for rule R21b based on the architecture of Figure 7.

Table 2 A snippet of Yilmaz's learning-phase table

followers_count	user_created_at	HiddenState
1002	448	M
845	846	S
363	360	S
119	176	S

4 Application domains and examples

4.1 Monitoring tweets

In [25], Yilmaz investigates the use of RM for identifying malicious Tweets by monitoring big data. He collected 22K tweets from publicly available data of Twitter and used them in testing and validation processes. Yilmaz identifies three hidden states in the tweet data: Benign (B), Suspicious (S), and Malicious (M). An example of a rules4business rule use by Yilmaz is the following customization of rules4business' Generic rule 9: "Flag when two *suspicious* tweets are less than 4 weeks apart", where the suspicious nature of a tweet is its hidden state. Hence, the customization parameters of Rule 9 are: E is: *HiddenState*=="S", T is: 4 weeks.

Table 2 depicts a snippet of Yilmaz's learning-phase csv log-file. Here, a subject matter expert assigned the *HiddenState* column by observing the *followers_count* and *user_created_at* columns (*user_created_at* contains the date of account creation). An example of quantization he used with *dtrahmm* is (for the *followers_count* column):

- *followers_count* $<$ 100: followers_countLE100
- 100 $\leq$ *followers_count* $\leq$ 500: between 100 and 500
- *followers_count* $>$ 500: followers_countGT500

4.2 Power grid stability and reliability

In [11], Galinski investigates the use of RM for modeling and predicting failures in the electrical power grid. Galinski's NL and statechart assertions are classified into four categories: Undesirable Events, Downward Trends, Failure to Recover, and Undesirable Fluctuations.

Galinski identified four hidden states, i.e., information that is not readily available in the log-files: Steady State (S), Loss of a Generator (G), Transmission Line Down (T), and Variable Energy Flow from a Renewable Resource (R).

An example of a Failure to Recover assertion that asserts on both visible and hidden information is: "Flag when PRC falls below 1354 MW (N-1 criterion) while in state G and is not restored to 1354 MW or greater in 15 minutes", where PRC is the Physical Response Capability, a measurement of responsive reserves in the grid. Galinski modeled this assertion using rules4business generic rule 12: "Flag whenever event P with no eventual event Q within time T after P", using P = PRC $<$ 1354 and HiddenState==G, Q = PRC $<=$ 1354, and T = 15 min.

Table 3 contains Galinski's quantization for the PRC column of the csv log-file.

Table 3 Galinski's quantization for the PRC column

PRC state	PRC range
0	PRC $<$ 1354
1	1354 $\leq$ PRC $<$ 1500
2	1500 $\leq$ PRC $<$ 1700
3	1700 $\leq$ PRC $<$ 1900
4	1900 $\leq$ PRC $<$ 2100
5	2100 $\leq$ PRC $<$ 2300
6	PRC $\geq$ 2300

5 Conclusion

This paper described an RM technique that combines the use of rules4business, a customizable and executable library of visual formal specifications with visual audit, with a toolset for performing RM on hidden data. Both components perform RM of csv log-files in the presence of formal specifications. The first tool does so assuming all data are visible, whereas the second extends that RM to monitor hidden data as well.

As described in Sect. 4, two students used the combined approach for performing RM of visible and hidden data in two specific domains, and the author did so for financial systems [7]. We have gained the following insight into the ease of use of the process:

1. While using rules4business component of the process is mostly automatic, the second component of the process (described in Sect. 3) consists of tools that are executed manually. There is a relatively easy fix to this problem using simple Python script that chains the execution of all tools in the toolset.
2. Some of the tools (*dtraalpha*) described in Sect. 3 require the specification of quantization using custom Python scripts to be written by the end user. At present, the quantization levels are defined rather arbitrarily.
3. Debugging a stream of RM outputs, such as those of Table 1 is a challenge. For this reason, it is important to gain trust in the rules while using the deterministic rules4business step, using its visual audit capability.

Some additional application domains we envision the proposed approach can be used in are:

1. Online, automatic monitoring of malicious emails, and network attacks in general. Here, hidden artifacts are expected to be location and other identification information not provided by an IP address.
2. Monitoring telemetric streams, of the kind sent from NASA missions. Here, telemetry data are most often rigid in structure; therefore, some additional information needed for verification purposes might not be included in that data and is, therefore, hidden.
3. Monitoring self-driving cars. Here, hidden artifacts are expected to be the state and intent of human actors, such as other drivers or pedestrians.

Acknowledgements Funding was provided by Defense Threat Reduction Agency (Grant No. ID0EIIAC15).

References

1. Bartocci E, Havelund K, Grosu R, Stoller SD, Seyster J, Smolka SA, Zadok E (2011) Runtime verification with state estimation. In: Proc. of RV'11, the 2nd international conference on runtime verification, San Francisco, Sept 27–30, Lecture notes in computer science, vol 7186. Springer, pp 193–207
2. Bartocci E, Grosu R, Kamarkar A, Smolka SA, Stoller SD, Zadok E, Seyster J (2012) Adaptive runtime verification. In: Proc. of RV 2012: the 3rd international conference on runtime verification, Istanbul, Turkey, September, Lecture notes in computer science, vol 7687. Springer, pp 168–18
3. Bartocci E, Bortolussi L, Sanguinetti G (2014) Data-driven statistical learning of temporal properties. In: Proc. of FORMATS 2014: the 12th international conference on formal modeling and analysis of timed systems, Lecture notes in computer science, vol 8711. Springer, pp 23–37
4. Clarke EM, Emerson EA (1981) Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen D (ed) Proceedings of workshop on logic of programs, LNCS 131. Springer, pp 52–71
5. Drusinsky D (2006) Modeling and verification using UML statecharts-a working guide to reactive system design, runtime monitoring and execution-based model checking. Elsevier, Oxford. ISBN 9780750679497
6. Drusinsky D (2011) Practical UML-based specification, validation, and verification of mission-critical software. DogEar Publishing, Indianapolis
7. Drusinsky D (2012) Behavioral and temporal pattern detection within financial data with hidden information. J Univ Comput Sci 18(14):1950–1966
8. Drusinsky D (2013) Behavioral and temporal rule checking for Gaussian random process–a Kalman filter example. J Univ Comput Sci 19(15):2198–2206
9. Drusinsky D (2014) Runtime monitoring and verification of systems with hidden information. Innov Syst Softw Eng 10(2):123–136
10. Drusinsky D, Michael JB, Shing M-T (2008) A visual tradeoff space for formal verification and validation techniques. IEEE Syst J 2(4):513–519 (ISSN: 1932-8184)
11. Galinski J (2015) Formal specifications for an electrical power grid system stability and reliability. Master Thesis in Computer Science, Naval Postgraduate School, Monterey, CA. https://calhoun.nps.edu/handle/10945/47259
12. Hopcroft J, Ullman J (2001) Theory of formal languages and automata, 2nd edn. Addison Wesley, Boston. ISBN 0-201-44124
13. Kalajdzic K, Bartocci E, Smolka SA, Stoller SD, Grosu R (2013) Runtime verification with particle filtering. In: Proc. of RV 2013: the 4th international conference on runtime verification, INRIA Rennes, France, September, 2013, Lecture notes in computer science, vol 8174. Springer, pp 149–166
14. Leicker M, Sanchez C (2010) Regular linear-time temporal logic. 2010 17th IEEE international symposium on temporal representation and reasoning (TIME), pp 3–5. doi:10.1109/TIME.2010.29
15. Meiera A, Mundhenkb M, Thomasa M, Vollmera H (2008) The complexity of satisfiability for fragments of CTL and CTL*. Electronic notes in theoretical computer science 223(26):201–213
16. Nitsche U (1994) Propositional linear temporal logic and language homomorphisms. In: Proceedings of 3rd international symposium on logical foundations computer science, LNCS 813. Springer, pp 265–277
17. Pnueli A (1977) The temporal logic of programs. Proc.18th IEEE Symp. on foundations of computer science, pp 46–57
18. Pierce JR (1969) Whither speech recognition. J Acoust Soc Am 46(4):1049–1051
19. (2016). http://www.rules4business.com
20. Rabiner LW (1989) A tutorial on hidden Markov models and selected applications in speach recognition, Proc. of the IEEE, vol 77, No. 2
21. Reinbacher T, Rozier KY, Schumann J (2014) Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Ábrahám E, Havelund K (eds) TACAS 2014 (ETAPS). LNCS, vol 8413. Springer, Heidelberg, pp 357–372
22. Rosu G, Thati R (2004) Monitoring algorithms for metric temporal logic specifications. Elsevier, Electronic notes in theoretical computer science
23. (2016). http://www.time-rover.com
24. Singh S. The code book: the science of secrecy from ancient Egypt to quantum cryptography. Fourth Estate, London, pp 143–189. ISBN 1-85702-879-1
25. Yilmaz A (2016) Detecting malicious tweets using runtime monitoring with hidden information. Master thesis in computer science, Naval Postgraduate School, Monterey, CA