Runtime monitoring of contract regulated web services

Alessio Lomuscio[1], Wojciech Penczek[2,3], Monika Solanki[1] and Maciej Szreter[2]

[1] Department of Computing
Imperial College London, UK
[2] Institute of Computer Science
PAS, Poland
[3] University of Podlasie, Poland

Abstract. We investigate the problem of locally monitoring contract regulated behaviours in web services. We encode contract clauses in service specifications by using extended timed automata. We propose a *non intrusive* local monitoring framework along with an API to monitor the fulfilment (or violation) of contractual obligations. We illustrate our methodology by monitoring a service composition scenario from the vehicle repair domain, and report on the experimental results.

1 Introduction

Web services (WS) are now considered one of the key technologies for building new generations of digital business systems. Service level agreements (SLAs) provide a useful mechanism to establish agreed levels of service provision when interactions are invoked within certain parameters. Although SLAs are useful, they can represent only basic agreements of service provision. Applications running complex, human-like activities require more general and sophisticated declarative specifications certifying legal-like agreements among the parties.

A useful concept from the legal domain in this sense is the one of *contract* as found in human societies. Should a contract be broken by one of the parties, additional rights and/or obligations (e.g., penalties to be paid) may be applicable to some party. In this paper, we study the problem of monitoring runtime behaviours of *contract regulated web services*. While contracts are usually negotiated offline, it is of interest to monitor at runtime whether interactions between WS are complying to the contracts stipulated between the parties. We put forward a "symbolic" solution to the problem above. We represent both all possible behaviours and the contractually-correct ones as an appropriate timed automata [1] at local web-service level. Specifically we present a local contract runtime monitor (CRM) based on the symbolic toolkit Verics [5], a symbolic model checker for timed-automata. CRM checks the input at runtime against the symbolic representations provided, and reports to the service (or directly to the engineer) any mismatch, or *violation*, between the contract-compliant behaviours originally prescribed and the ones actually received in the input stream.

The significant advantage of the approach is that we do not need to keep the whole state space of the possible and the contract-compliant behaviours in memory but we can simply call the timed-automata engine at runtime to match moves against the stream of events coming from the input. The paper is structured as follows: In Section 2 we briefly

introduce the formalism of timed automata as used here. Section 3 presents our monitoring framework. We analyse a motivating case study in 4 and discuss the monitoring results. Section 5 presents related work and conclusions.

2 Monitoring via Timed Automata

Let $\mathbb{N}$ denote the set of naturals (including 0), $\mathbb{Z}$ - the set of integers, $\mathbb{Q}$ - the set of rational numbers, $\mathbb{R}$ ($\mathbb{R}_+$) - the set of (non-negative) reals, and V be a finite set of integer variables. By a *variable valuation* we mean any total mapping $\mathbf{v} : V \longrightarrow \mathbb{N}$. We extend the mapping $\mathbf{v}$ to expressions of $Ex(V)$ in the usual way. The satisfaction relation ($\models$) for the boolean expressions is also standard.

Given a variable valuation $\mathbf{v}$ and an instruction $\alpha \in Ins^L(V)$, we denote by $\mathbf{v}(\alpha)$ the valuation $\mathbf{v}'$, obtained after executing α at $\mathbf{v}$, which is formally defined as follows:

- if $\alpha = \epsilon$ then $\mathbf{v}' = \mathbf{v}$,
- if $\alpha = (v := ex)$, then $\mathbf{v}'(v) = \mathbf{v}(ex)$ and $\mathbf{v}'(v') = \mathbf{v}(v')$ for all $v' \in V \setminus \{v\}$,
- if $\alpha = \alpha_1\alpha_2$, then $\mathbf{v}' = (\mathbf{v}(\alpha_1))(\alpha_2)$.

Let $\mathcal{X} = \{x_1, \ldots, x_{n_\mathcal{X}}\}$ be a finite set of real-valued variables, called *clocks*. The set of *clock constraints* over $\mathcal{X}$ and V, denoted $\mathcal{C}(\mathcal{X}, V)$, is defined by the grammar: $\mathfrak{cc} ::= true \mid x_i \sim c \mid x_i \otimes x_j \sim c \mid x_i \otimes x_j \sim v \mid x_i \otimes v \sim c \mid v \otimes w \sim x_i \mid \mathfrak{cc} \wedge \mathfrak{cc}$, where $x_i, x_j \in \mathcal{X}$, $v, w \in V$, $c \in \mathbb{N}$, $\otimes \in \{+, -\}$, and $\sim \in \{\leq, <, =, >, \geq\}$. Let $\mathcal{X}^+$ denote the set $\mathcal{X} \cup \{x_0\}$, where $x_0 \notin \mathcal{X}$ is a fictitious clock representing the constant 0. A *clock-to-clock assignment* A over $\mathcal{X}$ is a function $A : \mathcal{X} \longrightarrow \mathcal{X}^+$. $Asg(\mathcal{X})$ denotes the set of all the assignments over $\mathcal{X}$. By a *clock valuation* we mean a mapping $\mathbf{c} : \mathcal{X} \longrightarrow \mathbb{R}_+$. The satisfaction relation ($\models$) for a clock constraint $\mathfrak{cc} \in \mathcal{C}(\mathcal{X}, V)$ under a clock valuation $\mathbf{c}$ and a variable valuation $\mathbf{v}$ is defined as:

- $(\mathbf{c}, \mathbf{v}) \models (x_i \otimes v \sim c)$ iff $\mathbf{c}(x_i) \otimes \mathbf{v}(v) \sim c$,
- the other cases are defined similarly.

In what follows, the set of all the pairs $(\mathbf{c}, \mathbf{v})$, composed of a clock and a variable valuation, satisfying a clock constraint $\mathfrak{cc}$ is denoted by $[\![\mathfrak{cc}]\!]$. Given a clock valuation $\mathbf{c}$ and $\delta \in \mathbb{R}_+$, by $\mathbf{c} + \delta$ we denote the clock valuation $\mathbf{c}'$ such that $\mathbf{c}'(x) = \mathbf{c}(x) + \delta$ for all $x \in \mathcal{X}$. Moreover, for a clock valuation $\mathbf{c}$ and an assignment $A \in Asg(\mathcal{X})$, by $\mathbf{c}(A)$ we denote the clock valuation $\mathbf{c}'$ such that for all $x \in \mathcal{X}$ we have $\mathbf{c}'(x) = \mathbf{c}(A(x))$ if $A(x) \in \mathcal{X}$, and $\mathbf{c}'(x) = 0$ if $A(x) = x_0$. Finally, by $\mathbf{c}^0$ we denote the *initial* clock valuation, i.e., the valuation such that $\mathbf{c}^0(x) = 0$ for all $x \in \mathcal{X}$. In this paper we assume a slightly modified definition of timed automata with discrete data [17], which extend the standard timed automata of Alur and Dill in the following way:

Definition 1. *A* timed automaton with discrete data *(TADD) is a tuple* $\mathcal{A} = (\Sigma, L, l^0, V, \mathcal{X}, \mathcal{E}, \mathcal{I})$, *where*

- Σ *is a finite set of* labels (actions),
- L *is a finite set of* locations,
- $l^0 \in L$ *is the* initial location,
- V *is the finite set of integer variables,*
- $\mathcal{X}$ *is the finite set of clocks,*

– $\mathcal{E} \subseteq L \times \Sigma \times Bool(V) \times \mathcal{C}(\mathcal{X}, V) \times Ins^L(V) \times Asg(\mathcal{X}) \times L$ *is a* transition relation*, and*
– $\mathcal{I} : L \longrightarrow \mathcal{C}(\mathcal{X}, \emptyset)$ *is an* invariant function*.*

The invariant function assigns to each location a clock constraint (without integer variables[4]) expressing the condition under which $\mathcal{A}$ can stay in this location.

The semantics of a TADD $\mathcal{A}$ is given below.

Definition 2. *The* semantics *of* $\mathcal{A} = (\Sigma, L, l^0, V, \mathcal{X}, \mathcal{E}, \mathcal{I})$ *for an initial variable valuation* $\mathbf{v}^0 : V \longrightarrow \mathbb{Z}$ *is a labelled transition system* $\mathcal{S}(\mathcal{A}) = (Q, q^0, \Sigma_{\mathcal{S}}, \longrightarrow)$*, where:*

– $Q = \{(l, \mathbf{v}, \mathbf{c}) \mid l \in L \wedge \mathbf{v} \in \mathbb{Z}^{|V|} \wedge \mathbf{c} \in \mathbb{R}_+^{|\mathcal{X}|} \wedge \mathbf{c} \models \mathcal{I}(l)\}$ *is the set of states,*
– $q^0 = (l^0, \mathbf{v}^0, \mathbf{c}^0)$ *is the initial state,*
– $\Sigma_{\mathcal{S}} = \Sigma \cup \mathbb{R}_+$ *is the set of labels,*
– $\longrightarrow \subseteq Q \times \Sigma_{\mathcal{S}} \times Q$ *is the smallest transition relation:*
  - *for* $a \in \Sigma$*,* $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{a} (l', \mathbf{v}', \mathbf{c}')$ *iff there exists a transition* $t = (l, a, \beta, \mathfrak{cc}, \alpha, A, l') \in \mathcal{E}$ *such that* $\mathbf{v} \models \beta$*,* $(\mathbf{c}, \mathbf{v}) \models \mathfrak{cc}$*,* $\mathbf{v}' = \mathbf{v}(\alpha)$*,* $\mathbf{c} \models \mathcal{I}(l)$*, and* $\mathbf{c}' = \mathbf{c}(A) \models \mathcal{I}(l')$ (action transition)*,*
  - *for* $\delta \in \mathbb{R}_+$*,* $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{\delta} (l, \mathbf{v}, \mathbf{c} + \delta)$ *iff* $\mathbf{c} \models \mathcal{I}(l)$ *and* $\mathbf{c} + \delta \models \mathcal{I}(l)$ (time transition)*.*

Intuitively, in the initial state all the variables are set to their initial values, and all the clocks are set to zero. Then, at a state $q = (l, \mathbf{v}, \mathbf{c})$ the system can either execute an action or time transition.

2.1 TADD Semantics for RMCS

Inspired by related work in the formal representation of states of compliance and violation [10], we partition the set of global states Q of $\mathcal{S}(\mathcal{A})$ for $\mathcal{A} = (\Sigma, L, l^0, V, \mathcal{X}, \mathcal{E}, \mathcal{I})$ into two subsets G and R such that $G \cap R = \emptyset$[5]. The set G represents *green* (or *ideal*) states, whereas R represents the *red* (or *non-ideal*) ones. Intuitively, G contains the states of compliance and R contains the states of violation with respect to the contract, i.e., the whole set of clauses being included. Figure 1 illustrates the intuition behind the semantics.

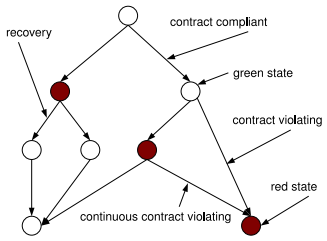

Fig. 1. Partitioning of states and transitions in a TADD

Based on the above partitioning each action transition (q, a, q') of $\mathcal{S}(\mathcal{A})$ can be one of the following four types of transitions:

[4] To ensure the monotonicity of the timed successor relation.
[5] This partition is obtained "location-wise" from a partition of the set of locations L of $\mathcal{A}$.

– **Contract compliant**: between green and green states, i.e., $q, q' \in G$. These transitions occur when the observed behaviour is in compliance with the prescribed behaviour of the contract.
– **Contract violating**: between green and red states, i.e., $q \in G$ and $q' \in R$. These transitions occur when the observed behaviour violates the prescribed behaviour of the contract.
– **Recovery**: between red and green states, i.e., $q \in R$ and $q' \in G$. These transitions occur when a recovery action is taken by the service after a violation of the prescribed behaviour is recorded.
– **Continuous contract violating**: between red and red states, i.e., $q, q' \in R$. The transitions occur when no recovery results from a previous violation.

We say that there is a *step* from state q_1 to q_2 in $\mathcal{A}$ if $q_1 \xrightarrow{\delta_1} q_1' \xrightarrow{a} q_2' \xrightarrow{\delta_2} q_2$, for some states $q_1', q_2' \in Q$, $\delta_1, \delta_2 \in \mathbb{R}_+$, and $a \in \Sigma$.

3 Runtime monitoring framework

Our architecture for local monitoring, RMCS, is illustrated in Figure 2.

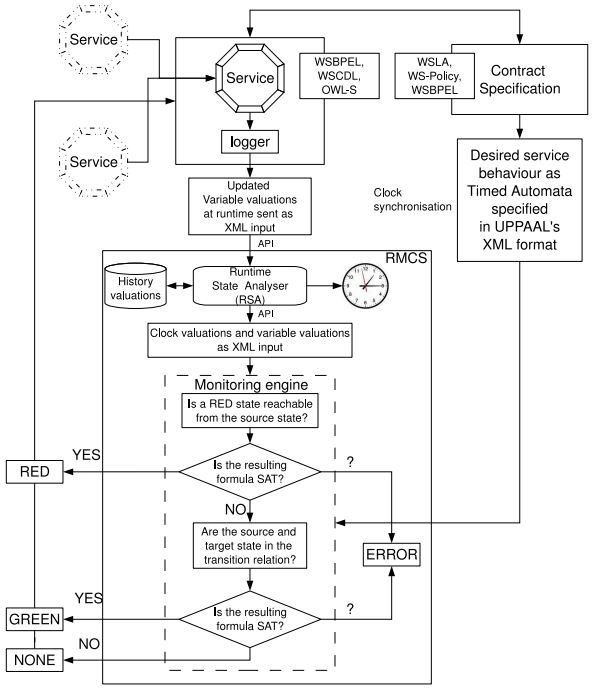

Fig. 2. The general architecture and methodology

Agents implementing WS are the primary entities within our framework. Service behaviour and contracts associated with them may be specified at a high level using WS

standards, e.g., WSBPEL [13] and contracts, e.g., WSLA [7]. The TADD specification for the service is engineered from these interface representations. The specification of service behaviour used by RMCS is the TADD representation described in Section 2. We use the XML format generated by the model checker UPPAAL for representing the TADD. The runtime state analyser interfaces with the logger for receiving snapshots of the latest variable valuations generated by the service. Snapshots are passed to the RSA via the logging framework. RSA is also responsible for updating clocks by querying the system hardware, in accordance with the granularity of a *tick* chosen by the service. The monitoring engine is the core component responsible for testing the conformance of runtime service behaviour presented as an input from the RSA, against the prescribed TADD specification of the service.

Each execution step passed to the engine is encoded and its conformance to the TADD specification is tested. Our SAT-based verification method does not need to construct the complete model for $\mathcal{A}$, which could be unfeasible for both the explicit-state as well as BDD-based methods. The engine checks at runtime whether the stream of execution steps received as inputs from the RSA, conforms with its symbolic representation of all possible behaviours. For each execution step, the answer returned by the monitoring engine is one of the following facts:

- **GREEN** - the step is conforming with the specification, i.e., there is a contract compliant transition between the source and target states.
- **RED** - a red state is reached as a target of the transition given, i.e., a contract has been violated as a result of the transition. This also signifies the fact that the inputs do not comply with the extended format of the TADD for the service.
- **NONE** - the step is not conforming with the specification, i.e., there is no such transition, neither contract compliant or otherwise.
- **ERROR** - the specification given does not mirror the observed transition so it amounts to an error.

Results reported at runtime may be analysed in several ways.

4 A vehicle repair contract: case study

We consider a service composition scenario that defines a repair contract between a client (C) and a vehicle repair company (RC). A repair contract specifies details concerning a particular repair, i.e., the type of repair to be performed, price, dates, pickup and delivery locations etc. For simplicity we only model the behaviour of RC. Table 1 identifies some of the contract clauses governing the actions taken by RC, the deadlines against which the contracts are monitored, if the clause can be violated, and, if a violation is recorded, whether any recovery is possible. Note that in some cases RC may take an "offline" action, in response to a violation from which no recovery may be possible. For example consider clause 6: "For any violation take recovery action within 3 days". If the recovery action is not taken, C may take an offline legal action against RC.

The informal behaviour of RC is described as follows. When RC receives a request from C to undertake a repair job, it sends a repair proposal. In response, C sends an acceptance or rejection message. If accepted, RC sends a contract initiation message to C. RC then waits for the vehicle to arrive, failing which it sends two reminders to C. If the vehicle fails to arrive, it takes an offline action. As per the contract, RC is *obliged* to

clause	Contract regulated actions	Deadline	Violation	Recovery
1	Receives a repair request by C	5 days	-	-
2	Sends a repair proposal to C	7 days	-	-
3	Assess damage to the vehicle	3 days	yes	yes
4	Execute repair	30 days	yes	yes
5	Send repair report to C	5 days	yes	yes
6	For any violation take recovery action	3 days	yes	no (take offline action)

Table 1. Some contract regulated actions for RC

assess the damage, repair the vehicle and send a report to C. On receiving the report, C is *obliged* to send payment to RC. If the payment is not sent, RC sends two reminders to C and then takes an offline action.

The actions taken by RC in response to messages sent by C are monitored to meet the deadlines set for various activities as per the contract. Failure to meet deadlines is considered a violation of the contractual obligations. In some cases a recovery from the violation may be possible.

4.1 Monitoring the runtime behaviour of the Repair Company

The full set of behaviours of the repair company is represented by a TADD[6]. As described in Section 4, deadlines for various activities are decided during contract negotiation between the parties. Deadlines are defined in terms of number of days. For example consider a contract clause to be monitored: *If C sends a damaged vehicle to RC, RC assesses the damage to the vehicle within 3 days -clause (3) in table 1. A snippet of the TADD for the clause is shown in the Figure 3.* Figure 3 describes the timeline in number

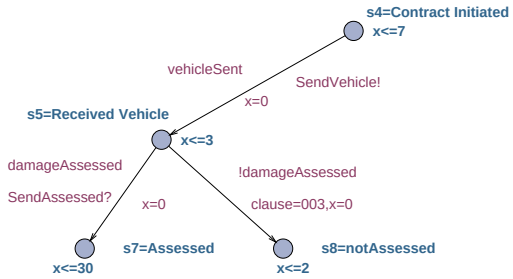

Fig. 3. TA specification of clause (3)

of days for clause (3), a snapshot passed to RSA at $x = 0$ from the logger when a vehicle for repair arrives, snapshots sent to the monitoring engine by the RSA and the results from monitoring. As per the contract, once a damaged vehicle has arrived the damage has to be assessed within 3 days. A snapshot is again sent by the logger to the RSA at $x = 5$. The snapshot taken at $x = 0$ and at $x = 5$ are sent by the RSA as a pair - or as a

[6] The complete TADD for the example is too large to be shown here.

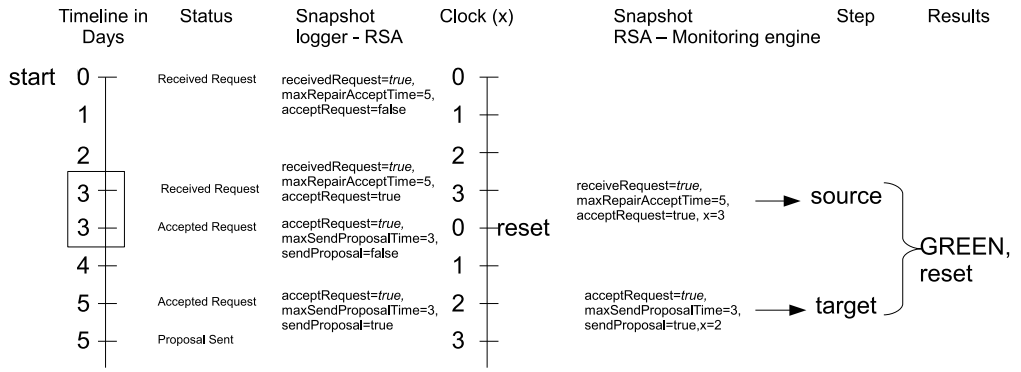

Fig. 4. Runtime valuations for clause (2)

"step" to RMCS. The results returned by the monitoring engine are $\{RED, reset, 003\}$. RED signifies that a violation has occurred, i.e., the damage was not assessed within the deadline, $reset$ indicates that the clock has been reset and 003 indicates the clause index that has been violated.

4.2 Experimental results and Discussion

In order to validate our methodology, we implemented the above case study and monitored several runtime execution steps for the service. RMCS successfully monitored 8 execution steps per second depending on the number of variables defined for the steps. Violated contracts and clock resets were reported by RMCS. Note that this could be useful in the context of monitoring SLAs, where typically large number of execution steps need to be monitored to ensure a reliable Quality-of-Service.

5 Related work and conclusions

In this paper we presented a symbolic approach based on timed automata for the runtime monitoring of contract regulated agent based WS. Monitoring service behaviour has been an active area of research. Several efforts have investigated various formalisms and frameworks for the monitoring of functional and non-functional properties of services. The monitoring problem has been considered for several formalisms in papers [16, 2, 4, 14, 12, 11, 9, 3]. Timed automata have been used in earlier work such as [8] on monitoring and fault diagnosis of systems, while [15] presents an approach which also uses timed automata for monitoring SLAs. The aims of the above approaches are however quite different from our objectives in this paper. However [8, 15] are not concerned with local monitoring of contract-based executions.

An attractive feature of our approach over those mentioned above is that histories and pending contracts are not stored in memory during the monitoring. This positively impacts the scalability of the approach and is particularly useful when monitoring multiple and long running contracts between several services. As a case study we presented the monitoring of contracts for a repair company. Although the TADD for the service

is not large enough to exploit the full capabilities of RMCS, we believe it is still sufficiently significant to demonstrate the methodology and scope of the proposed approach. Experiments demonstrate larger scenarios would be handled just as well by the technique.

Much work remains to be done. An important part of our future work is the translation to TADDs from high level specification standards such as WSBPEL.

References

1. R. Alur. Timed Automata. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 8–22. Springer-Verlag, 1999.
2. Fabio Barbon, Paolo Traverso, Marco Pistore, and Michele Trainotti. Run-time monitoring of instances and classes of web service compositions. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*.
3. Luciano Baresi, Carlo Ghezzi, and Sam Guinea. Smart monitors for composed services. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*. ACM.
4. Domenico Bianculli and Carlo Ghezzi. Monitoring conversational web services. In *IW-SOSWE '07: 2nd international workshop on Service oriented software engineering*. ACM.
5. P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Pólrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS: A tool for verifying Timed Automata and Estelle specifications. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, LNCS. Springer-Verlag.
6. N. Eén and N. Sörensson. MiniSat. http://minisat.se/MiniSat.html.
7. Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *J. Netw. Syst. Manage.*, 2003.
8. M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *11th International SPIN Workshop on Model Checking of Software (SPIN'04), Barcelona, Spain*, LNCS.
9. Zheng Li, Yan Jin, and Jun Han. A runtime monitoring and validation framework for web service interactions. In *ASWEC '06: Proceedings of the Australian Software Engineering Conference (ASWEC'06)*. IEEE Computer Society.
10. A. Lomuscio and M. Sergot. Deontic interpreted systems. *Studia Logica*, 75(1):63–92, 2003.
11. G. Mahbub, K.; Spanoudakis. Run-time monitoring of requirements for systems composed of web-services: initial implementation and evaluation experience. In *ICWS'05, IEEE International Conference on Web Services*.
12. Carlos Molina-Jimenez, Santosh Shrivastava, Ellis Solaiman, and John Warne. Contract representation for run-time monitoring and enforcement. *cec*, 2003.
13. OASIS Web service Business Process Execution Language (WSBPEL) TC. Web service Business Process Execution Language Version 2.0, 2007.
14. Marco Pistore, F. Barbon, Piergiorgio Bertoli, D. Shaparau, and Paolo Traverso. Planning and monitoring web service composition. In *AIMSA*, pages 106–115, 2004.
15. F. Raimondi, J. Skene, L. Chen, and W. Emmerich. Efficient monitoring of web service slas. Technical report, UCL, London, 2007.
16. Monika Solanki. *A Compositional Framework for the Specification, Verification and Runtime Validation of Reactive Web Service*. PhD thesis, De Montfort University, Leicester, UK, October 2005.
17. A. Zbrzezny and A. Półrola. SAT-based reachability checking for timed automata with discrete data. *Fundamenta Informaticae*, 79(3-4):579–593, 2007.