Runtime monitoring of contract regulated web services (Extended Abstract)

A. Lomuscio
Imperial College London
a.lomuscio@doc.ic.ac.uk

M. Solanki
Univ. of Leicester
m.solanki@mcs.le.ac.uk

W. Penczek
ICS PAS and
Univ. of Podlasie
penczek@ipipan.waw.pl

M. Szreter
ICS PAS
mszreter@ipipan.waw.pl

ABSTRACT

We investigate the problem of locally monitoring contract regulated behaviours agent-based 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. FRAMEWORK

When services are combined, a significant challenge is to regulate their interactions. Service level agreements (SLAs) provide a useful mechanism to establish agreed levels of service provision when interactions are invoked within certain parameters. In web services the traditional notion employed for similar purposes is the one of service level agreement (SLA) [3]. Although SLAs are useful, they can represent only basic agreements of service provision. Applications running agent-based activities require more general and sophisticated declarative specifications certifying legal-like agreements among the parties. Additionally, agents maximising their own utilities may indeed choose to violate these agreements for a better payoff differently. While this is unavoidable, we may still wish to monitor the executions and track at run time the agreements that are being fulfilled and violated.

In this short paper, we survey our approach to runtime monitoring for local behaviours in *contract regulated web services*. We represent both all possible behaviours and the contractually-correct ones as appropriate timed automata [1] at local web-service level. We present a local contract runtime monitor (CRM) based on the symbolic toolkit Verics [2], a symbolic model checker for timed-automata. CRM checks the local service's execution at runtime against the symbolic representations provided, and reports back 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.

Cite as: Runtime monitoring of contract regulated web services (Extended Abstract), Alessio Lomuscio, Wojciech Penczek, Monika Solanki and Maciej Szreter, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

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.

We assume a slightly modified definition of timed automata with discrete data (TADD) [5]. Our approach for local monitoring, Runtime Monitoring for Contract-based Services, is illustrated in Figure 1. For each agent to be monitored all its possible behaviours (contract-compliant and otherwise) are represented as a TADD and stored in the checker. The BMC based monitoring engine checks the snapshots against their TADD specification and reports back whether the actual runtime behaviours are in compliance with the contractually prescribed behaviour as specified in the TADD, or, if not, states the clause that has been violated in the present transition.

A significant feature of our framework is that we do not place any restriction on service implementation in terms of development infrastructure and execution platforms.

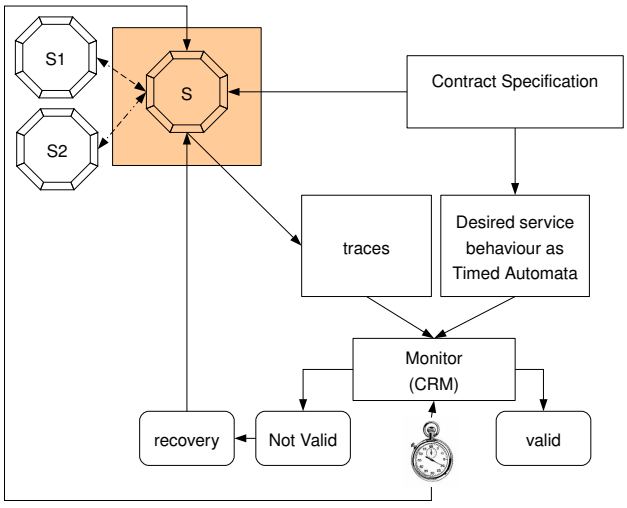

Figure 1: The general architecture and methodology

We use the XML format generated by the model checker UPPAAL [4] for representing the TADD. Our choice is motivated by the fact that UPPAAL provides a user friendly GUI. The TADD specification encodes all possible desired behaviours for a service. Typically, the full set of behaviours for a contract regulated service can be derived from:

- its contractually compliant behaviours. These behaviours encapsulate contractual obligations for the service.
- behaviours that are classified as violations of the contract.
- behaviours that define a recovery from incurred violations.

The monitoring engine is the core component responsible for testing the conformance of runtime service behaviour against the prescribed TADD specification of the service. For each execution step, the answer returned by the monitoring engine is one of the following:

- **GREEN** - This represents the fact that the step is conforming with the specification, i.e., there is a contract compliant transition between the source and target states.
- **RED** - This represents the fact that 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** - This represents the fact that the step is not conforming with the specification, i.e., there is no such transition, neither contract compliant or otherwise.
- **ERROR** - This represents the fact that 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. In case of contract compliant transitions, the service can continue executing as per the orchestrated workflow. For contract violating transitions, the service administrator may impose on the service to execute one of the prescribed recovery transition. In other cases the administrator may choose to override the violations reported and allow the service to continue execution. For a continuous contract violating transition being reported, the service may be stopped. Finally, the outputs generated may be stored in a log file for future offline analysis.

2. A CASE STUDY: CONTRACTS FOR VEHICLE REPAIRS

We consider a service composition scenario that defines a repair contract between a client (C) and a vehicle repair company (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 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.

In order to validate our methodology, we implemented the above case study and monitored several runtime execution steps for the service. To provide an indication of the number of variables the toolkit can monitor at the same time we scaled the example described above by parametrising the number of cars in the contract. The experiments show the approach can monitor effectively several hundreds of variables. This is sufficient for very complex monitoring of key aspects of a service. We did not optimise the monitoring process in any way; we expect our results to improve significantly by tailoring the approach to a particular problem we wish to monitor. Indeed, observe that the methodology above could be parallelised over several engines on top of the web service with each engine monitoring different independent contracts or clauses in a contract.

3. CONCLUSIONS

In this short paper we presented a symbolic approach based on timed automata for the runtime monitoring of contract regulated agent based WS. Several previous efforts have investigated various formalisms and frameworks for the monitoring of functional and non-functional properties of services. Our approach is different from explicit approaches in that state 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.

While verification is still an aspect of systems validation we are not aware of symbolic attempts to the runtime monitoring of these notions. It seems to us that it may be of interest to investigate whether this could be achieved in ways related to the technique presented here.

Acknowledgment. The third and the fourth authors acknowledge partial support from the Polish project ITSOA.

4. REFERENCES

[1] R. Alur. Timed Automata. In *Proc. of CAV'99*, LNCS 1633, pages 8–22. Springer-Verlag, 1999.

[2] 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 TACAS'03*, LNCS 2619, pages 278–283, Springer-Verlag, 2003.

[3] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. J. Netw. Syst. Manage., 2003.

[4] P. Pettersson and K. G. Larsen. Uppaal2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, 2000.

[5] A. Zbrzezny and A. Półrola. SAT-based reachability checking for timed automata with discrete data. *Fundamenta Informaticae*, 79(3-4):579–593, 2007.