SERVICE LEVEL AGREEMENT SPECIFICATION, COMPLIANCE PREDICTION AND MONITORING WITH PERFORMANCE TREES

N.J. Dingle W.J. Knottenbelt L. Wang
Department of Computing, Imperial College London, South Kensington Campus, London SW7 2AZ
email: {njd200,wjk,lw205}@doc.ic.ac.uk

ABSTRACT

Service Level Agreements (SLAs) are widely used throughout industry but suffer from specification ambiguities and difficulties in predicting and monitoring compliance. To address these issues, we propose the use of the Performance Tree formalism for the specification and monitoring of Service Level Agreements (SLAs). Specifically, we show how the basic Performance Tree formalism can be adapted to provide a rigorous yet accessible and expressive means to specify common SLA metrics. Using established performance analysis tools that support Performance Trees, this allows system designers to check SLA compliance on formal models of their system before implementation. We also propose an architecture for a system of measurement agents that enables the same performance requirements to be monitored in the context of a live implementation.

KEYWORDS

Service Level Agreements; Performance Trees; Performance analysis; Quality of Service

INTRODUCTION

Many organisations depend heavily on the availability, reliability and performance of key services delivered by internal business units and external organisations. To ensure an adequate quality of service, it is common practice to contractually define the parameters of service provision in the form of a Service Level Agreement (SLA). SLAs specify the type and quality of service to be provided by a supplier in return for a fee paid by a user, as well as any compensation due to the user in the case of sub-standard service delivery by the supplier.

This paper addresses a number of key challenges relating to SLA set-up, compliance prediction using a system model, and monitoring of a live system.

Firstly, it is important to specify SLA metrics in a way that is rigorous and unambiguous yet readily accessible to both supplier and user. Natural language is an obvious and accessible way to specify such metrics, but is neither rigorous nor unambiguous. Mathematical formalisms such as stochastic logics [1, 2] are unambiguous and rigorous but are not accessible; they are also restricted in the range of concepts they can express. Here we propose the use of Performance Trees (PTs) – a recent formalism for the graphical specification of complex performance queries [17, 18] – for SLA metric specification. By extending the basic Performance Tree formalism using its macro feature, we show how PTs are able to provide rigorous yet accessible metric specification without sacrificing expressiveness.

Secondly, it is often the case that service suppliers need to predict SLA compliance for systems that have not yet been implemented. Such design-time intervention helps to avoid the situation whereby a supplier finds that the system they have built does not meet – and cannot feasibly be adapted to meet – agreed SLA requirements. It is also often the case that service suppliers need to predict the effect on SLA compliance of proposed changes to currently operational systems. In both cases, the construction and analysis of a stochastic model provides suppliers with a low-cost means to make the necessary predictions. Indeed, we show how, thanks to recent support implemented in the PIPE tool [3], the compliance of SLA requirements expressed as Performance Trees can be directly evaluated on stochastic models.

Finally, monitoring of SLA compliance in operational systems poses additional challenges in terms of collecting and processing measurement data in a way that ensures the accurate computation of a given SLA metric. To this end, we present an architecture for a system of monitoring agents that can be used for the run-time evaluation of SLA requirements expressed as Performance Trees.

A wide range of separate studies have been carried out in each of the areas of SLA specification, compliance prediction and monitoring. Regarding SLA specification, investigations about formalising SLA frameworks have been carried out on various types of IT services, for example databases, e-commerce systems and technical support operations [7, 10, 11, 14, 20]. Bouman [4] has pointed out the existing problems with present SLA specifications mechanisms in terms of ambiguity, incompleteness and inefficiency, and at the same time suggested general principles in guiding customers and service providers in specifying SLA requirements. More formally, an XML language SLAng [9] based on the Unified Modelling Language (UML) system model has been introduced that provides a rigorous and unambiguous approach for SLA specification. However, it is rather syntactically complicated for non-IT professionals.

Much research has also been carried out on SLA monitoring for a wide range of IT systems [8, 12, 13, 15, 16]. For example, Pereira [15] describes a hierarchical architecture for monitoring quality of service (QoS) parameters to help the enforcement of SLAs between a provider and users. Padgett [14] combines SLA specification and compliance prediction by suggesting a set of service metrics and demonstrating their use in predicting CPU usage in a mathematical model. However, the SLA metrics are limited to only CPU usage, and monitoring of SLA compliance is not considered.

To the best of our knowledge, therefore, the work presented in the present paper is the first time a unified environment been proposed for SLA specification, compliance prediction and monitoring.

The remainder of this paper is organised as follows. In the next section, we discuss relevant background including SLA metrics, performance analysis techniques and the Performance Tree formalism. In the following section we introduce a Voting System model that is used as a running example throughout the rest of the paper. We then show how Performance Trees can be used to specify various common SLA metrics, including availability, mean time between failures and mean time to repair, response time percentiles, resource utilisations, throughputs and system productivity. Finally, we present our architecture for an online PT-based SLA monitoring system before concluding and discussing opportunities for further work.

BACKGROUND

Service Level Agreement metrics

In this paper we concentrate on metrics related to availability, response times, resource utilisations, throughputs and system productivity. This ensures that we are able to support a superset of the metric-related concepts covered by the QoS specification language QML [6].

We define these concepts as follows:

- **Availability** is the proportion of time during which service is provided to users. Availability is in turn dependent on two further metrics: **Mean Time Between Failures (MTBF)** and **Mean Time To Repair (MTTR)**.

- **Response time** is the time from a user sending a request to receiving a response or the time from job submission to job completion. SLAs are often concerned with means, variances, and percentiles of response times. For example it may be required of postal service that “90% of first class post is delivered within one working day of posting”.

- **Resource utilisation** is the proportion of time for which a given resource is used by a given service. For example it may be required that a service does not utilise more than 10% of available network bandwidth.

- **Throughput** is the average rate at which a given set of activities occurs. For example, it may be required that a system processes a minimum of 3 000 transactions per second.

- **Productivity** is a weighted sum of the throughput of a number of activities, where the weights are user-specified rewards associated with completion of each activity. If the unit of the reward is financial, this measures a system’s **profitability**.

Performance Analysis

Performance is a vital consideration for system designers and engineers. Indeed, a system which fails to meet its performance requirements can be as ineffectual as one which fails to meet its correctness requirements. Ideally, it should be possible to determine whether or not this will be the case at design time. This can be achieved through the construction and analysis of a performance model of the system in question, using formalisms such as queueing networks, stochastic Petri nets and stochastic process algebras.

Having created a stochastic model of the system, it needs to be decided what performance measures are of interest. It is possible to capture such requirements in logical formulae using a language such as Continuous Stochastic Logic (CSL) [1, 2]. These languages provide a concise and rigorous way to pose performance questions and allow for the composition of simple queries into more complex ones. Such logics can be somewhat daunting for non-expert users, and there still remains the problem of correctly converting informally-specified requirements into logical formulae.

Performance Trees

Performance Trees are an intuitive graphical formalism for the quantification and verification of performance properties. They were proposed to overcome the problems associated with logical stochastic property specification highlighted in the previous section. They combine the ability to specify performance requirements, i.e. queries aiming to determine whether particular properties hold on system models, with the ability to extract performance measures, i.e. quantifiable performance metrics of interest.

The concepts expressible in Performance Tree queries are intended to be familiar to engineers and include steady-state and passage time distribution and densities, their moments, transition firing rates, convolutions and arithmetic operations. An important concern during the development of Performance Trees was ease of use, resulting in a formalism that can be straightforwardly visualised and manipulated as hierarchical tree structures.

A Performance Tree query is represented as a tree structure consisting of nodes and interconnecting arcs. Nodes can have two kinds of roles: operation nodes represent performance-related functions, such as the calculation of a passage time density, while value nodes represent basic concepts such as a set of states, an action, or simply numerical or Boolean constants. Table 1 presents a summary of Performance Tree nodes used in this paper.

The formalism also supports macros, which allow new concepts to be created with the use of existing operators, and an

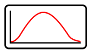
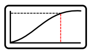
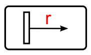
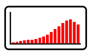
Textual	Graphical	Description
?	?	The result of a performance query.
PTD		Passage time density, calculated from a given set of start and target states.
Perctl		Percentile of a passage time density or distribution.
ProbInInterval	Pr($[t_1, t_2]$ ●→●)	Probability with which a passage takes place in a certain amount of time.
Moment	$E(X^n)$	Raw moment of a passage time density or distribution.
FR		Mean occurrence of an action (mean fi ring rate of a transition).
SS:P		Probability mass function yielding the steady-state probability of each possible value taken on by a StateFunc when evaluated over a given set of states.
Macro	Macro	User-defi ned performance concept composed of other operators.
⋈	<,≤,==,≥,>	Arithmetic comparison of two numerical values.
⊕	+,-,*,/,^	Arithmetic operation on two numerical values.
Num	Num	A real number.
Actions	Actions	A set of system actions.
States	States	A set of system states.
StateFunc	StateFunc	A real-valued function on a set of states.

Table 1: Selected Performance Tree nodes

abstract state-set specification mechanism to enable the user to specify groups of states relevant to a performance measure in terms of the corresponding high-level model.

Performance Trees have been integrated into the Platform Independent Petri net Editor (PIPE), thus allowing users to design Generalised Stochastic Petri Net (GSPN) models and to specify relevant performance queries within a unified environment. PIPE communicates with an Analysis Server which employs a number of (potentially parallel and distributed) analysis tools to calculate performance measure [5]. These include steady-state measures, passage time densities and quantiles, and transient state distributions.

To offer greater ease for constructing performance queries, we have recently developed an alternative query construction mechanism called the Natural Language Query Builder (NLQB) [19]. The NLQB guides users in the construction of performance queries in an iterative fashion, presenting at each step a range of natural language alternatives that are appropriate in the query context.

RUNNING CASE STUDY

Fig. 1 shows a GSPN model of an electronic voting system[1] which will be used throughout this paper. In the model there are several voters, *CC*, a limited number of polling booths, *MM*, and a smaller number of central servers, *NN*. Voters vote asynchronously, moving from place p_0 to p_1 as they do

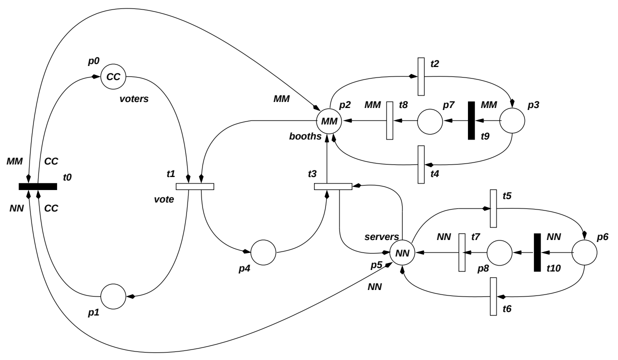


Figure 1: The Voting System model

so. Polling booths which receive their votes transit t_1 from place p_2 to place p_4. At t_3, the vote is registered with as many servers as are currently operational in p_5.

The system is considered to be in a failure mode if either all the polling booths have failed or all the central servers have failed. If either of these complete failures occurs, then a high priority repair is performed that resets the failed units to a fully operational state. If some (but not all) of the booths or servers fail, they attempt self-recovery via transitions t_4 and t_6 respectively. The system is considered to be available and will continue to function as long as at least one polling booth and one server remain operational.

To facilitate the reasoning about failures and repairs, we augment the state vector of the system model with two boolean

[1]`http://www.doc.ic.ac.uk/~njd200/voting.mod`

components: *just-repaired* which is set to true by any transition which moves from a failed to an available state, and *just-failed* which is set to true by any transition which moves from an available to a failed state.

Note that the rate at which failures (recoveries) occur depends on the number of currently operational (failed) units and so the transitions t_2, t_4, t_5 and t_6 are modelled with marking-dependent rates. Also, the rate at which voters vote depends on the current number of voters and available polling booths, and the rate at which polling booths register cast votes depends on the number of available servers, and so t_1 and t_3 are also modelled with marking-dependent rates.

The system we study features 100 voters, 10 polling booths and 10 servers, and its underlying Continuous Time Markov Chain (CTMC) contains $93\,274$ states.

SLA SPECIFICATION WITH PERFORMANCE TREES

To demonstrate the use of Performance Trees to describe and predict compliance with SLA requirements, we have constructed some common SLA metrics using parameterised macros and evaluated them for the Voting System model. Our methodology is not limited to those metrics described in this section, however, as the extensible nature of Performance Trees allows the user to construct macros for the metrics which are most relevant to them.

Availability

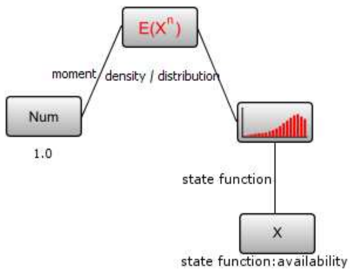


Figure 2: Availability macro definition

Fig. 2 shows the PT macro definition for availability. This takes as input a parameter, X, that is a state function that returns 1 if the system is available in the state, and 0 otherwise. Evaluating the SS:P operator yields a probability mass function (pmf), the domain of which is 0 and 1 (corresponding to system non-availability and availability respectively) and the range of which is the steady-state probability of each domain value. Computing the expected value (via the first moment) of the pmf yields the steady-state probability of finding the system in an available state.

Suppose the SLA for the Voting System specifies that it shall have an availability greater than 99%. The corresponding Performance Tree definition of this requirement, which makes use of the PT macro defined above, is given in Fig. 3.

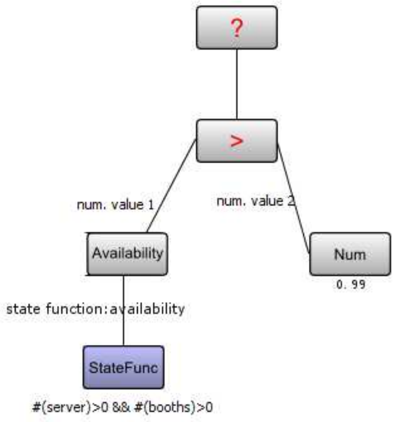


Figure 3: Availability example

According to the model description the system is considered to be available if at least one server and one polling booth are operating. The corresponding state function therefore evaluates to 1 if *#(servers)* $>$ *0* $\wedge$ *#(booths)* $>$ *0*, and to 0 otherwise. By evaluating this query on the model (using PIPE's Performance Tree Query Editor and associated evaluation environment) we find that the system is predicted to achieve 99.88% (to 2 d.p.) availability and therefore to meet its SLA requirement.

Mean Time Between Failures

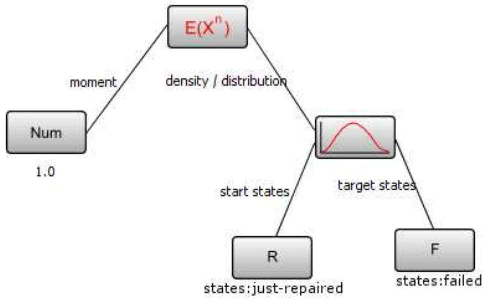


Figure 4: Mean Time Between Failures macro definition

Another important measure is the *Mean Time Between Failures* (MTBF), for which the corresponding Performance Tree macro definition is shown in Fig. 4. This requires two sets of states as input: the states where the system has just been repaired, *R*, and the failure states, *F*. These states can be defined using the evaluator's built-in state set definition tool. Evaluating the PTD operator yields the pdf of the first passage time from state set *R* to state set *F*. Computing the expected value (via the first moment) of the pdf yields the MTBF.

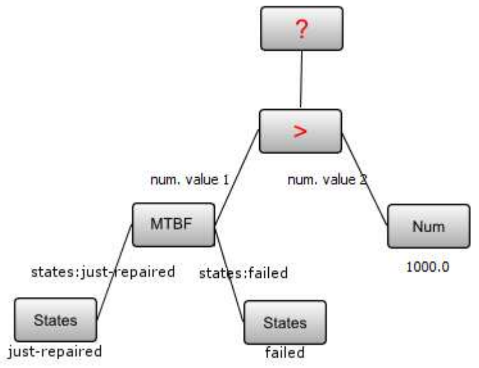


Figure 5: Mean Time Between Failures example

Fig. 5 shows the instantiation of this macro for an SLA requirement that the MTBF of the Voting System is greater than 1000 time units. From the model description above we have that the system is considered to have failed if either all servers or all booths have failed. By evaluating this query on the model we find that the MTBF is 661.69 time units (to 2 d.p.) and we can therefore conclude that the system will not meet its MTBF requirement.

Mean Time To Repair

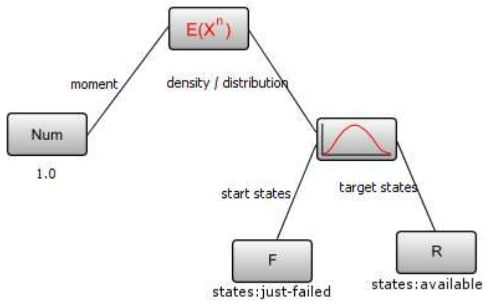


Figure 6: Mean Time to Repair macro definition

Mean Time To Repair (MTTR) indicates how quickly a system recovers from a failure state. As shown in Fig. 6, it requires two set of states as input: the states where the system has just failed, *F*, and the states where the system is repaired, *R*.

Fig. 7 shows the instantiation of this macro for an SLA requirement that the MTTR of the Voting System is less than 5 time units. By evaluating this query on the model we find that the MTTR is 0.96 time units (to 2 d.p.) and we can therefore conclude that the system will meet its MTTR requirement.

We note that an alternative definition of availability is:

$$\frac{\mathrm{MTBF}}{\mathrm{MTBF} + \mathrm{MTTR}}$$

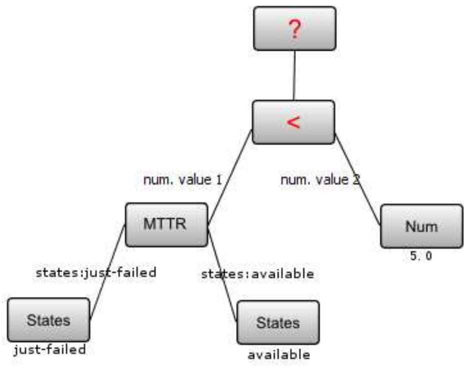


Figure 7: Mean Time to Repair example

Substituting in the above values for MTBF and MTTR, we have that the Voting System's availability is 99.86% (to 2 d.p.), which agrees very well with the value calculated above from the mean of the distribution of the corresponding state function.

Response time percentiles

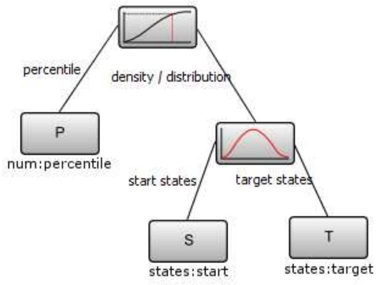


Figure 8: Response time percentile macro definition

Fig. 8 shows the Performance Tree definition of a response time percentile requirement while Fig. 9 shows the Performance Tree corresponding to the SLA requirement that *"the system shall be capable of processing all voters within 30 time units 95% of time"*. The graph in Fig. 10 shows the cumulative distribution function of the time taken to process all voters with the 95% confidence interval at $t = 25.5$ time units marked. As the SLA requires this time to be less than 30 time units, we can conclude that the system will meet this response time percentile requirement.

Resource utilisation

Fig. 11 shows the Performance Tree macro definition for an SLA resource utilisation requirement. This is similar to the Availability macro, save for the fact that its input state func-

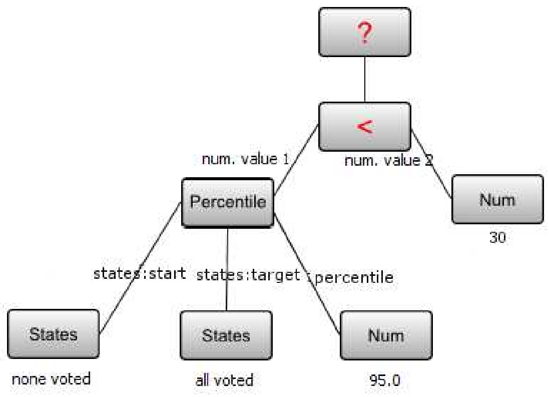


Figure 9: Response time percentile example

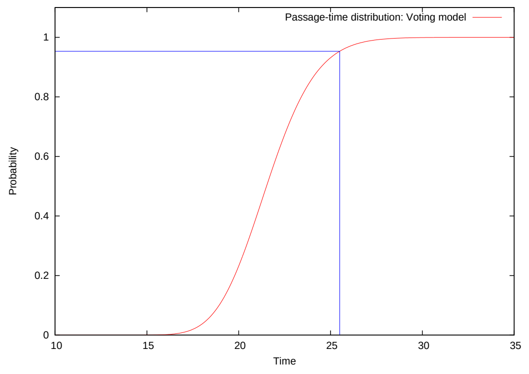


Figure 10: Result of evaluating the query in Fig. 9

tion describes when a particular resource is said to be busy rather than when the system is available.

Fig. 12 poses the question for the Voting System *"For what proportion of time is at least one polling booth in use?"*. By evaluating this query on the model we find that the proportion of time for which this is the case is 99.93% (to 2 d.p.).

Throughput

Throughput is the measure of the rate at which certain activities occur, for example the amount of data transferred over a communications link per second. It is also a fundamental element for evaluating the productivity of a system. In Performance Tree terms, throughput is described as the mean occurrence rate of actions in the corresponding system model. The corresponding macro simply consists of one *Firing Rate* node.

Fig. 13 shows the Performance Tree query asking for the average number of votes being cast per time unit. By evaluating this query we find that the Voting System achieves a throughput of 0.075 voters per time unit (to 3 d.p.).

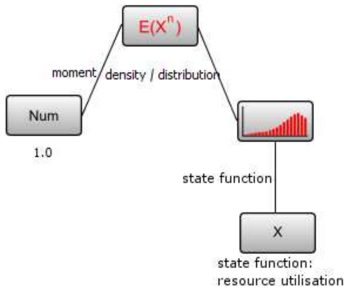


Figure 11: Resource utilisation macro definition

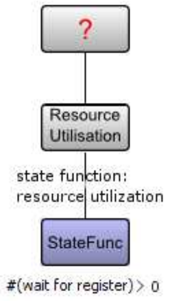


Figure 12: Resource utilisation example

Productivity/Profitability

Based on the definition of throughput, Performance Trees can be used to provide a measurement of the productivity of a system. As shown in Fig. 14, for an SLA productivity can be defined as a sum of the products of the mean occurrence rate of certain actions and their corresponding impulse rewards, which can be positive or negative.

Fig. 15 shows the Performance Tree of the SLA requirement that the Voting System *"shall make a profit of more than 20 currency units per time unit, based on earning 2 currency units from each vote successfully cast but paying 100 currency units for each high-priority server repair and 50 currency units for each high-priority polling booth repair"*. In this example, the first action/reward pair is made up of the rate at which votes are cast and the earnings from one vote; the second pair consists of the rate of high-priority server repairs and their cost, and the third pair consists of the rate of high-priority polling booths repairs and their cost.

Calculating the throughputs of the three relevant transitions and multiplying by the rewards for each reveals that the total profitability of the system is -0.02 currency units per time unit (to 2 d.p.). From this, we can conclude that the system does not meet its SLA requirement for profitability and indeed is predicted to cost more to run than it generates in revenue.

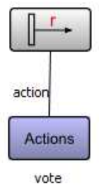


Figure 13: Throughput example

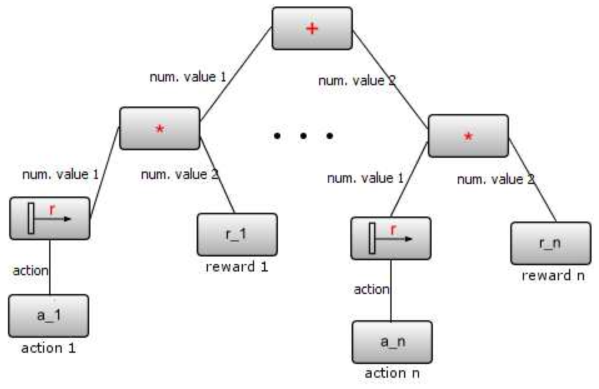


Figure 14: Productivity macro definition

ARCHITECTURE FOR ONLINE SLA MONITORING

In this section we propose a unified architecture to enable PTs to be used to monitor SLAs in a real system as well as to evaluate SLA requirements on a system model. This will provide users with an accurate and accessible approach to evaluate actual system performance against the same SLA specifications for which the system model has been analysed. The general structure of the architecture is shown in Fig. 16. To gather data on the system's behaviour, client-side event logging agents are installed that collect performance data relevant to the metrics specified in the PT operator nodes of the corresponding SLA definition. For example, if the SLA contains a passage time density operator, such agents would record the time taken for each passage of the system from the start to target states. Once enough passages have been observed it is possible to construct an approximation of the density using a histogram. The data thus collected can be then stored in an online database that is accessed through an intuitive user interface which permits users to verify the actual system against SLAs using the same PT's SLA specification interface as for model analysis. This is done by constructing their SLA requirement query and submitting it to the SLA monitor. The SLA monitor extracts the relevant data from the online database according to the request received and generates a compliance report.
This architecture can also be used to verify the system model against the real implementation to check if it correctly reflects the behaviour of the live system. This will allow investigation of new system configurations to be conducted to determine their likely effectiveness. For example, users could investigate adding an extra server or increasing the bandwidth to their system to predict the likely performance improvement. They could then decide whether or not the improved performance warranted the expense of implementing such changes.

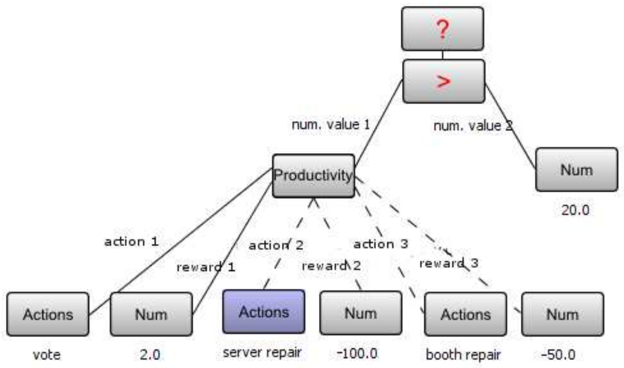


Figure 15: Productivity example

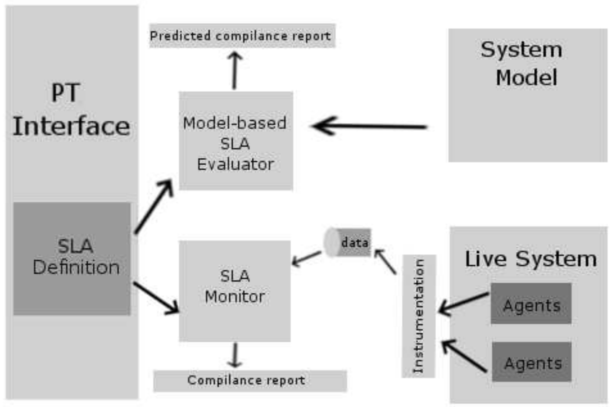


Figure 16: SLA prediction and monitoring architecture

CONCLUSION

In this paper we have demonstrated how Performance Trees can be used to reason about SLAs. Using macro functionality, we have presented PT-based definitions for common SLA concepts such as availability and productivity, and we have demonstrated the analysis of such metrics using a case study of a Voting system. We have also proposed an architecture based on the use of monitoring agents to collect information about the performance of the system once implemented. This can then be used to monitor compliance with an SLA using the same Performance Tree framework employed for pre-implementation analysis.

In the future we intend to extend the work presented here in a number of ways. We will add support for the expression of SLA metrics in natural language by augmenting our existing Natural Language Query Builder; this will make the construction of SLA-specific Performance Trees even more intuitive. We will also implement the on-line monitoring architecture and demonstrate its applicability to a real-world example.

REFERENCES

[1] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous-time Markov chains. In *Lecture Notes in Computer Science 1102: Computer-Aided Verification*, pages 269–276. Springer-Verlag, 1996.

[2] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous-time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

[3] P. Bonet, C.M. Llado, R. Puijaner, and W.J. Knottenbelt. PIPE v2.5: A Petri net tool for performance modelling. In *Proc. 23rd Latin American Conference on Informatics (CLEI 2007)*, Costa Rica, October 2007.

[4] J. Bouman, J. Trienekens, and M. van der Zwan. Specification of service level agreements: problems, principles and practices. *Software Quality Journal*, 12(1):43–57, March 2004.

[5] D.K. Brien, N.J. Dingle, W.J. Knottenbelt, H. Kulatunga, and T. Suto. Performance Trees: Implementation and Distributed Evaluation. In *Proc. 7th Intl. Workshop on Parallel and Distributed Methods in Verification (PDMC'08)*, Budapest, Hungary, March 2008. Elsevier.

[6] S. Frolund and J. Koistinen. QML: A language for Quality of Service specification. Technical Report TR-98-10, HP Laboratories, Palo Alto, California, USA, 1998.

[7] J-P. Garbani. Best practices for service-level management. Technical report, Forrester Research, Inc., 2004.

[8] D. Greenwood, G. Vitaglione, L. Keller, and M. Calisti. Service level agreement management with adaptive coordination. In *Proc. International Conference on Networking and Services (ICNS 2006)*, pages 45–50, 2006.

[9] D.D. Lamanna, J. Skene, and W. Emmerich. SLAng: A language for defining service level agreements. In *Proc. 9th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, Washington, DC, USA, 2003.

[10] R. Leopoldi. IT services management - a description of service level agreements. Technical report, RL Consulting, 2002.

[11] H. Ludwig, A. Keller, A. Dan, R.P. King, and R. Franck. Web Service Level Agreement (WSLA) language specification. Technical report, IBM T.J. Watson Research Center, 2003.

[12] G. Morgan, S. Parkin, C. Molina-Jimenez, and J. Skene. Monitoring middleware for service level agreements in heterogeneous environments. In *Proc. 5th IFIP conference on e-Commerce, e-Business, and e-Government (I3E '05)*, pages 79–93, Poznan, Poland, 2005.

[13] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, K. Krishnakumar, and A. Meisels. A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. In *Lecture Notes in Computer Science 3470: Proc. Advances in Grid Computing (ECG '05)*, pages 651–660. Springer-Verlag, 2005.

[14] J. Padgett, K. Djemame, and P. Dew. Grid service level agreements combining resource reservation and predictive run-time adaptation. In *Proc. of the UK e-Science All Hands Meeting, Nottingham*, September 2005.

[15] P.R. Pereira. Service level agreement enforcement for differentiated services. In *Lecture Notes in Computer Science 3883: Proc. Second International Workshop of the EURO-NGI Network of Excellence*, pages 158–169, Villa Vigoni, Italy, May 2006. Springer-Verlag.

[16] A. Sahai, V. Machiraju, M. Sayal, L.-J. Jin, and F. Casati. Automated SLA monitoring for web services. Technical Report HPL-2002-191, HP Laboratories, Palo Alto, California, USA, 2002.

[17] T. Suto, J.T. Bradley, and W.J. Knottenbelt. Performance Trees: A New Approach to Quantitative Performance Specification. In *Proc. 14th IEEE/ACM Intl. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2006)*, pages 303–313, Monterey, CA, USA, September 2006.

[18] T. Suto, J.T. Bradley, and W.J. Knottenbelt. Performance Trees: Expressiveness and quantitative semantics. In *Proc. 4th International Conference on the Quantitave Evaluation of Systems (QEST'07)*, pages 41–50, Edinburgh, September 2007.

[19] L. Wang, N.J. Dingle, and W.J. Knottenbelt. Natural language specification of Performance Trees. In *Lecture Notes in Computer Science 5261: Proc. 5th European Performance Engineering Workshop (EPEW 2008)*, pages 141–151, September 2008.

[20] E. Wustenhoff. Service level agreement in the data center. Technical Report 816-4551-10, Sun BluePrints Online, April 2002.