International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-10, August 2019

Converting from Service Level Agreement to Probabilistic Temporal Logic Specification

Nur Diana Madinah Ab Hadi, Azlan Ismail, Nur Syazleen Rosli, Suzana Zambri

Abstract*: The need for conversion method exists due to the limitation of manual conversion at design time whenever the interested party must perform some assessments using an existing model checker tool. Manual conversion of the related requirements into the respective specification language is time-consuming especially when the person has limited knowledge and need to do the task repetitively with a different set of Service Level Agreement (SLA) configurations. This paper aims to address the need to automatically capture non-functional requirements specified in the SLA, namely, Service Level Objectives (SLO) and converting them into a specific probabilistic temporal logic specification. We tackle this problem by proposing a conversion method that utilizes a rule-based and template-based approach. The conversion method automatically extracts the required information in SLA based on certain rules and uses the extracted information to replace the elements in the prepared template. We focus on WS-Agreement language for SLA and probabilistic alternating-time temporal logic with rewards specification (rPATL) for the properties specification used in PRISM-games model checker tool. We then implement an initial proof-of concept of a conversion method to illustrate the applicability of translating between targeted specifications.

***Keywords*: Software Conversion, Service Level Agreement, Probabilistic Temporal Logic, Quality of Service**

I. INTRODUCTION

Service Level Agreement (SLA) has been applied in various research areas including cloud computing, where SLA is needed to govern the relationship between the cloud provider and cloud consumer [1]. The agreed SLA is then used by the cloud provider as the key policies in managing its cloud resources. With the notion of autonomic clouds [2], SLA is crucial to enable self-adaptive of cloud in a collaborative environment in order to maintain the quality of service (QoS) level of cloud applications [3]. Assuring QoS of cloud application defined in SLA at runtime is non-trivial computational task. In the case of problematic condition, an appropriate adaptation strategy has to be devised and executed. One of such strategies is the cloud migration that refers to a process of migrating a cloud application from one host to another host. The migration strategy is crucial when the currently deployed host is dealing with resource scarcity and thus requires the application to be migrated to another collaborative host to prevent from SLA violation. As there are more than one host to be chosen, a careful decision has to be made in order to migrate successfully whilst maintaining the expected QoS level.

The mentioned challenge can be tackled from the design time as well as runtime. In this paper, we focus on the design time which concerns of two inter-related aspects. The first concern is to assess a set of migration strategies in order to determine the winning strategies for a SLA specification. Another concern is to find the appropriate SLA configuration to be offered which has higher winning strategies. The second concern is important for a cloud provider in generating a SLA to be offered to the cloud consumers. Hence, the cloud provider can maximize consumer's satisfaction.

One of the potential methods that can be applied to assess the migration strategies is using software verification and synthesis for self-adaptive systems [4]. Software verification is needed to assess the correctness of migration strategy in recovering from failure, whilst synthesis aims to determine its satisfaction against QoS requirements. Both processes are well-studied research area and a few tools, also known as model checking tools have been developed and used by others, such as PRISM [5]. However, performing the assessment using a model checker is not a straight forward activity due to different language specification. In the case of the challenges mentioned above, it involves transforming or converting the cloud migration model and SLA specification into the specification used by the model checker. As a result, it is time-consuming activity especially for a novice user of the tool when dealing with different set of SLA specifications. Therefore, in this paper, we address the need to convert the QoS requirements specified in SLA as properties of a model checking tool. We propose a conversion method to automatically perform the conversion in order to support the assessment activity. The approach consists of extracting non-functional requirements specified in SLA based on WS-Agreement [6] and transforming them into multi-objective properties specified using probabilistic alternating-time temporal logic with rewards specification (rPATL) [7]. The extraction process is designed with a set of rules and focuses on the elements of Service Level Objectives (SLO) in SLA. Meanwhile, the transforming process makes use of a template-based approach, where the respective elements in the template are replaced with the extracted elements of SLO.

Manuscript published on 30 August 2019.
***Correspondence Author(s)
Nur Diana Madinah Ab Hadi, Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA (UiTM), 40450 Shah Alam, Selangor, Malaysia.
Azlan Ismail, Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA (UiTM), 40450 Shah Alam, Selangor, Malaysia.
Nur Syazleen Rosli, Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA (UiTM), 40450 Shah Alam, Selangor, Malaysia.
Suzana Zambri, Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA (UiTM), 40450 Shah Alam, Selangor, Malaysia.

Hence, the outcome of the conversion is the properties which can be further used for verification and synthesis. For this reason, the interested user can make use of PRISM-games model checker [8]. However, the discussion of the verification and synthesis tasks is beyond the scope of this paper. This paper is organized as follows. Section 2 elaborates the fundamental areas for this paper. Section 3 describes the motivation and challenge to be tackled in this work. Section 4 explains the specifications used in this work. Section 5 presents the proposed approach to address the challenge. Section 6 presents a proof-of-concept implementation to illustrate the application of the approach. Section 7 highlights the related works. Finally, Section 8 provides the conclusion and outlines the future work.

II. BACKGROUND

A. SLA-driven Cloud Computing

Cloud computing is defined as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [9]. From the services perspective, the cloud computing can offer Software-as-a-Service (SaaS), Platform-as a-Service (PaaS), Infrastructure-as-a-Service (IaaS) [10].

Service Level Agreement (SLA) has been researched and applied in a few areas, especially in the Web services environment. Several specification languages have been introduced which are based on XML, such as SLAng [11], WSLA [12] and WS-Agreement [6]. In Web services, SLA is a contract to govern the relationship between two parties; Service provider and service consumer. The relationship between them is established whenever both parties agree on the agreement. One of the common frameworks to implement SLA-aware services is SLA@SOI that supports SLA definition language, negotiation, monitoring, violation prediction, and detection, etc [13].

In the area of Cloud computing, SLA needs to exist between two parties as well, namely, between Cloud providers and consumers. These two parties need to agree on a set of terms specified in the SLA document, potentially through a negotiation process [1]. Furthermore, a few research works have attempted to address SLA issues in cloud computing, such as SLA management support [14], a specialized SLA language for the cloud environment, namely, the Cloud SLA specification [15] which was constructed by considering the Open Cloud Computing Interface (OCCI) [16] and the Cloud Computing Reference Architecture of the National Institute of Standards and Technology (NIST) [17].

From the Cloud provider perspective, the established SLA is used as the main requirements to manage its Cloud infrastructure in ensuring the desired services are provisioned to the Cloud consumer [18]. For this reason, existing work [3] proposed a SLA-driven Cloud management that aims to manage the resource management life-cycle through the utilization of agreements which include the establishing of the agreement, feeding the agreement to the cloud resource manager, monitor the agreement satisfaction, and performing the necessary actions to main the quality levels specified in the SLA. The proposed platform has utilized WS-Agreement specification and extended WSAG4J framework [19]. In addition, [20] presented a dynamic resource provisioning, which aims to provision Cloud resources based on the Cloud users' demands while satisfying SLA requirements.

B. Autonomic Computing

Autonomic Cloud Computing is a concept that combines the autonomic computing with cloud computing [2]. The autonomic computing refers to the computing model that brings in the self-managed capability into a software system. The main framework to realize autonomic computing is referred to the MAPE-K framework [21]. The framework contains four key components, namely monitor, analyzer, planner and executor. In the context of autonomic cloud, this framework enables self-managed cloud resources, especially when dealing with varying workload demands. Therefore, the monitoring is used to monitor the current resource usage in relation to the workload demands. The analyzer is used to analyze the impact of the change in the workloads which may potentially cause SLA violation. The planner is responsible to decide the appropriate adaptation action to prevent or minimize the SLA violation. Meanwhile, the executor is required to enforce the necessary action.

C. Quantitative Verification

Quantitative verification is a mathematically-based technique for establishing the correctness, performance, and reliability of systems that exhibit stochastic behavior [22]. Quantitative verification has been proposed to support several stages of software adaptation process in self-adaptive systems [23], such as during analysis and planning of MAPE-K loop [21].

Model checking techniques are the common techniques to realize quantitative verification. In this paper, we are paying a specific attention to PRISM-games model checker [24] [8], a tool for multi-objective strategy synthesis for stochastic games. It is important to note that PRISM-games is an extension PRISM tool [5]. The main activities in performing model checking using a model checking tool are commonly referred to; (i) model specification, (ii) property specification, and (iii) execution of model checking with/without strategy synthesis. The model specification refers to the activity of designing and encoding the systems under study by abstractly representing them using the supported models. In PRISM-games, the users can specify the systems as probabilistic models (e.g. discrete-time Markov chains, Markov decision processes and continuous time Markov chains) and stochastic games. Property specification refers to the activity of designing and encoding one or more objectives to be fulfilled by the specified model. Meanwhile, the execution of model checking task to the execution of respective model checking algorithm to reason and analyze the specified model against the specified properties which results to the model checking outcomes (e.g. satisfied or not, optimized values).

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

III. MOTIVATION AND CHALLENGE

The work proposed in this paper is motivated by the need to perform automated quantitative verification with strategy synthesis for the application migration strategies in autonomic clouds environment. Thus, we begin by discussing the application migration problem followed by the need to have automated quantitative verification and synthesis for application migration strategies.

A. Application Migration in Autonomic Cloud Environment

Application migration problem has become of the studied issue in the autonomic cloud environment, specifically related to the Science Cloud Platform (SCP) with the ASCENS project [2]. SCP contains multiple computing nodes or clouds which each cloud is associated with an autonomic manager based on the MAPE-K framework. That means each cloud has to have the ability to adjust its resource autonomously and proactively. The main role of these clouds is to collaboratively and voluntarily share their resources in order to provision cloud application [25]. During runtime, any cloud may be dealing with resource over utilization or resource failures which results in SLA violation. Therefore, the cloud may have to migrate the application under its responsibility to other cloud collaborators. For this reason, it has to analyze and decide its migration strategies that satisfy its migration goals. In this context, a strategy refers to which cloud collaborator to choose in order to migrate the cloud application. Existing work has applied a reputation-based method to choose the right cloud collaborator [26].

B. Assessing Cloud Application Migration Strategies

We are concerned with the need to assess the application migration strategies at design time. This kind of need may be the interest of the cloud developer analyst. Specifically, the interested parties would like to evaluate the following:

- Determine the appropriate migration strategies in relation to a set of predefined SLA parameters.
- Identify the appropriate threshold values to be included in SLA.
- Explore the effect and impact of executing migration strategies within a certain number of cloud collaborators.

To perform the assessment, one of the approaches is to utilize the existing model checking tool to perform verification and synthesis process repetitively. This kind of tool provides the features and functionalities that ease the verification and synthesis task. However, the key challenge is to capture the application migration model and related non-functional requirements. To capture this information, the person has to be familiar with the syntax of the specification.

Therefore, in this work, we focus on capturing the non-functional requirements which are typically specified in the SLA. Such requirements can be related to the performance, cost, reliability, etc. For instance, the response time to migrate the application and/or the response time to execute the application on the collaborator's environment should be less than a threshold. Another example is the cost of executing the application on the collaborator's environment does not beyond a certain threshold.

To capture the requirements, the conventional approach is to manually map and encode the requirements as properties in a model checking tool. However, this approach requires the knowledge and skill to understand both specifications, the source specification (e.g. WS-Agreement) and the destination specification (e.g. probabilistic temporal logic). Hence, in this paper, we address this limitation by providing an automatic approach for the mapping and encoding the requirements via a conversion method. The details are discussed in the next section.

IV. SPECIFICATION LANGUAGES

In this section, we introduce the specifications used in this research, in particular, to represent SLA and probabilistic temporal logic.

A. WS-Agreement

WS-agreement can be defined as a runtime extensible mark-up based language and protocol that used is to monitor and communicate between client and server based on the initial offer [27], [28]. Example of a SLA specification is shown in Figure 3.

The Agreement Context and Agreement Term Type are two main components within WS Agreement where each of them required performing their own task. The agreement context will state data which describe the properties and services involved while the agreement term type is for specifying the functional and non-functional requirements. Under Agreement term there are two main substructures which are named as Service Description Terms (SDTS) and Guarantee Terms [29]. The specialty of the Guarantee Term is it can have single or multiple scopes of an operation. Under the guarantee terms, there are four sub-elements termed as Obliged, Validity, Expression, and Evaluation Event. Each of these sub-elements can have their own further sub-elements. For instance, in the case of Validity, we can define the start and end time of the guarantee terms. Meanwhile, for Expression, we can define the Predicate, SLA parameter, and Value. For more details about the agreement, readers can refer to [6].

B. Probabilistic Temporal Logic

The properties are formulated as the probabilistic alternating-time temporal logic with rewards specification (rPATL) [7]. It is introduced for expressing quantitative properties for reasoning stochastic multi-player games (SMGS) model [30] using PRISM-games model checker. SMGS model is useful to model probabilistic and competitive behavior. In the context of a single autonomic cloud, the competitive behavior of two players can be represented as a controlled and uncontrolled environment [31]. The controlled aspect refers to the action to be executed by the autonomic component, whilst the uncontrolled environment can be referred to the behavior of the network which affects the performance of the action.

Syntactically, rPATL is a CTL-style branching-time temporal logic that distinguishes between path formulae and state formulae. The path formulae consider the composition of multiple nodes in the model, whilst the state formulae are associated with specific nodes in the model.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

As rPATL is used to support the reasoning multi-player stochastic games, it enables a specification of a coalition of players that has a strategy which can ensure the probability of an event's occurrence or that an expected reward measure meets some thresholds, irrespective of the actions of the other players.

There are two probabilistic operators that can be expressed to form a property in rpatl, namely, the probabilistic operator P and the reward operator R. In this work, we only focus on the reward operator, but we introduce both for providing the background information.

For both types of operators, we can define a threshold =? To require the computation of the actual probability of reward. For instance, as follows:

$$<< P_1,P_3 >> P_{max=?}[F \text{ “end”}] \quad (1)$$

The property expresses the need to find the maximum probability with which players P1 and P3 can guarantee that an end-state is reached. In the above example, F is an example of path formula.

rPATL can also be used to reason about the total reward [C], mean payoff [S] or a long-run ratio of two rewards. In the case of total reward, the formulated games model must have reachable terminal states with zero rewards.

Furthermore, the properties defined as rPATL can be of two types, single objective, and multi-objective properties. Conventional rPATL is used to state a single objective property, whilst multi-objective is a boolean combination of single objective properties. The multi-objective properties must be of the same type (i.e. Either all probabilistic or reward-based) and the threshold must be specified. An example of multi-objective is as follows:

$$<<P >> (R\{r_1\} <X[C] \& R\{r_2\} >Y [C]) \quad (2)$$

This property expresses that the players in coalition P must ensure a strategy where the accumulated reward r1 is less than X is guaranteed and the accumulated reward r2 is greater than Y is guaranteed, independently of the strategies of other players. More examples of properties specification can be referred to [32].

V. THE DESIGN

In this section, we explain the proposed conversion approach and method which aims to address the assessment activity during design time. The runtime stage is considered as future work.

A. Conversion Approach

The conversion approach is meant to support the interested party (e.g. Software developer) to quickly obtain the properties based on required probabilistic temporal logic specification. It consists of four main phases, namely:

(1) Importing SLA documents -The approach should allow the users to import a SLA document (i.e. WS-Agreement) and display it back to the user.

(2) Converting into properties -The approach should ease the conversion process by enabling the users to obtain the required properties automatically.

(3) Exporting properties -The approach should enable the users to export the properties into .props file automatically, which is the format that is compatible with PRISM-games tool.

(4) Supporting validation of the properties -The approach should support the users to check the correctness of the generated properties via PRISM-games tool.

The approach assumes that the interested party or user has access to a SLA document. Thus, it begins by importing the SLA document into the conversion application. The system will display the contents of SLA. Then, the user initiates the conversion process that results in properties based on probabilistic temporal logic specification. It is important to note that currently, the validation of the converted properties is supported by allowing the user to export/save them into a .props file. Then, this file can be opened and checked using PRISM-games model checking tool.

B. Conversion Method

In this section, we describe the conversion method in terms of algorithm that converts from SLA to the probabilistic temporal logic specification as shown in Algorithm 1.

```
Require: SLA specification
Ensure: rPATL specification
1: Initialize parser and assign path to SLA file
2: Get the root element of SLA
3: for all element in SLA do
4: Find the SLO section
5: if the SLO section is found then
6: Retrieve the name, predicate, parameter,
and value
7: end if
8: end for
9: Get the template file
10: Replace the elements in the template with
the respective elements from SLO
11: Write and store into a result file
```

Algorithm 1: Conversion Algorithm

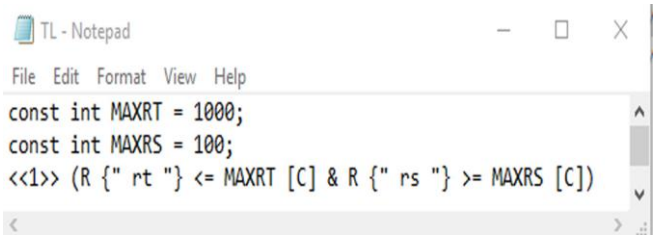

Figure 1. Sample Of Properties Template For Temporal Logic Specification

The algorithm takes SLA specification as the input and produced rPATL specification as the output. It begins by extracting the elements of SLA using XML parser that returns a Document Object Model (DOM) document. Then, it searches the entire elements in DOM to find the interested elements specifically those that are defined within Service Level Objectives (SLO) section. Herein, we are interested in extracting the name of SLO, its predicate, parameter, and value. Whenever this information is obtained, the algorithm then generates a template file. Then, the elements retrieved from SLO is used to replace the elements in the template.

Finally, the updated template file is written and stored in a result file. A sample of the template file is shown in Figure 1. In the template, it specifies two default parameters, MAXRT to hold the maximum value for the time-related requirement, and MAXRS to hold the maximum value for the cost related requirement. Then, it specifies multi-objective properties as a boolean combination of time and cost related objectives. Hence, the outcome of the conversion will replace the default parameters with its values and the condition parameters. Example of the converted specification is shown in Figure 4.

VI. IMPLEMENTATION

In this section, we present the proof-of-concept implementation of our conversion approach. We have developed a conversion program based on Java language. The interface of the program is shown in Figure 2.

Figure 2. Main interface of the conversion program

The program offers five functionalities and two main paragraph fields, namely, for displaying/editing the content of WS-Agreement and for displaying/editing the converted properties. The first functionality refers to the open button to enable opening a WS-Agreement file and import the content in one of the paragraph fields. The second functionality is the save button to allow saving any changes made to the WS Agreement file. The third functionality is the clear button to empty the contents of the field. Then, the convert button that implements the algorithm introduced in Algorithm 1. Finally, the save result button that can save the converted properties into a new file. With this program, the interested party/user can perform that conversion automatically. The process begins by importing the respective SLA file into the program such as shown in Figure 3. Then, the user can execute the conversion and the converted result is as shown in Figure 4.

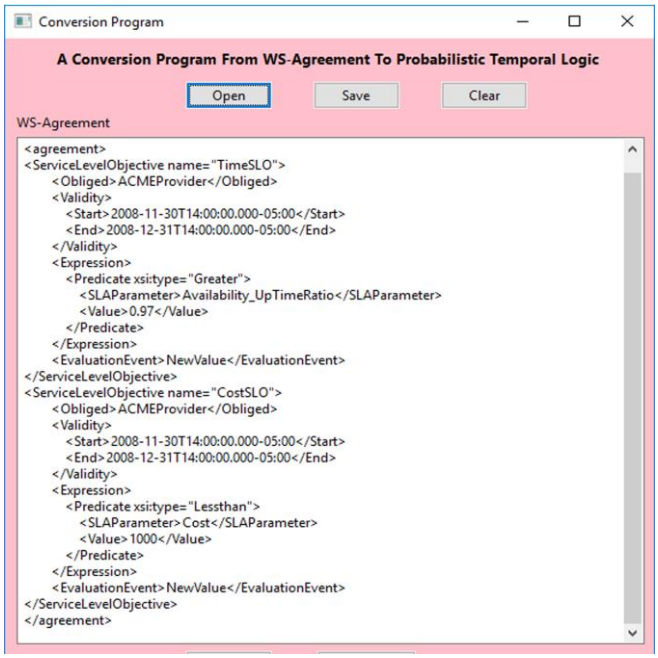

Figure 3. A Sample Of SLA Imported Into The Conversion Program

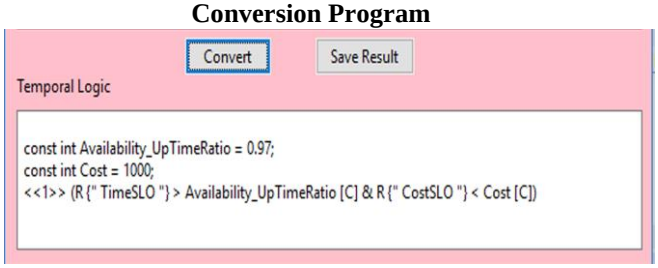

Figure 4. Sample Of Converted SLA As Temporal Logic Specification

After having the converted properties, the user can export and save it into a specific format. In the case of PRISM-games tool, the required format for the properties is .props. Given with the properties file, the respective user can perform the required verification with or without synthesis using PRISM-games model checker. Readers can refer to [8] for further details.

VII. RELATED WORK

The automated conversion that aims to translate between two different specification languages has been studied by a few works. The common reason for enabling this automated conversion is to support formal analysis offered by the existing analysis techniques and tools. In this section, we highlight these works that have focused on the design time and conclude to the novelty of the work proposed in this paper.

A few general conversion approaches between two different specifications have been proposed, specifically as follows. The work by Baresi et al. [33,34] addressed the need to analyze Interaction Overview Diagrams. Thus, they proposed a conversion approach from IOD to TRIO temporal logic [35] based on the transformation formulae. With this translation, IOD can be analyzed and verified via Zot bounded model checker [36].

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Meanwhile, Meziani et al. [37] focused on assisting software developer to understand the outcome of Colored Petri Nets (CPN) analysis using CPN model checker [38]. For this reason, they proposed a conversion method from CPN to Unified Modeling Language (UML) diagrams.

Furthermore, a few works have targeted the conversion approach to PRISM model checker. The work by Gallotti et al. [39] aimed to support QoS analysis of service composition using PRISM model checker [5]. Hence, they proposed a model based transformation approach called ATOP from XMI specification (i.e. Activity Diagrams) to PRISM model (e.g. Markov Decision Process (MDP)). They also mentioned about non-functional requirements (NFR) presented as Probabilistic Computation Tree Logic (PCTL) specification [40]. However, the automated conversion from NFR specification to PCTL is not discussed.

The work by Krostsiani et al. [41] addressed SLA validation using PRISM model checker, especially for SLA monitoring. In this context, it is crucial to check for possible SLA violation events and related actions. To enable this, a transformation approach is proposed that converts extended WS-Agreement into PRISM model (i.e. Continuous-Time Markov Chain model (CTMC). The extension enables more details about the actions to be taken upon the violation of specific guarantee terms. The conversion approach differs from our work since it focuses on the model aspect while we focus on the properties aspect.

VIII. CONCLUSION

We have presented a proof-of-concept implementation of conversion approach from SLA specification (i.e. WS-Agreement) to rPATL specification. Our motivation for the proposed approach is based on the need to analyze the cloud migration process for autonomic cloud environment at design time. The approach enables the respective user to import the SLA file, convert it into the rPATL specification, and export it as .props file. The conversion method utilizes a rule-based technique and a template-based approach. The rules are used to extract the required elements from SLA which is then replaced the elements specified in the template. The exported .props file can be used as the properties to analyze a PRISM-games model (i.e. Stochastic Multi-player Games) using PRISM-games model checker.

As our future work, we plan to consider a complex SLA as well as different SLA specification language to be converted into the respective probabilistic temporal logic specification. We also consider combining this properties-driven conversion with the model-driven conversion to enhance its benefit. We also plan to integrate with the syntax checking module provided in PRISM games model checker to bring this conversion capability into the runtime environment

ACKNOWLEDGMENT

Azlan Ismail acknowledges the support of the Fundamental Research Grant Scheme, 600-IRMI/FRGS 5/3 (214/2019), funded by Ministry of Education Malaysia.

REFERENCES

1. Pichot A, Wieder P, Waldrich O, et al. Dynamic SLA-negotiation based on WS-Agreement; 2007. Technical Report TR-0082. Available from: http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0082.pdf.
2. Mayer P, Velasco J, Klarl A, et al. The Autonomic Cloud. In: Wirsing M, H¨olzl M, Koch N, et al., editors. Software engineering for collective autonomic systems: The ascens approach. Cham: Springer International Publishing; 2015. p. 495–512. Available from: http://dx.doi.org/10.1007/978-3-319-16310-916.
3. Garc´ıa Garc´ıa A, Blanquer Espert I, Hern´andez Garc´ıa V. SLA-driven dynamic cloud resource management. Future Generation Computer Systems. 2014;31(1):1–11.
4. Calinescu R, Autili M, C´amara J, et al. Synthesis and Verification of Self-aware Comput¬ing Systems. In: Self-aware computing systems. Cham: Springer International Publishing; 2017. p. 337–373. Available from: http://link.springer.com/10.1007/978-3-319-47474-811.
5. Kwiatkowska M, Norman G, Parker D. PRISM 4.0: Verification of Probabilistic Real-Time Systems. (Lecture Notes in Computer Science; Vol. 6806); 1. Springer; 2011. p. 585–591. Available from: http://dx.doi.org/10.1007/978-3-642-22110-147.
6. Andrieux A, Czajkowski K, Dan A, et al. Web services agreement specification (WS-Agreement). Vol. 128; 2007. p. 216.
7. Chen T, Forejt V, Kwiatkowska M, et al. Automatic verification of competitive stochastic systems. Formal Methods in System Design. 2013 8;43(1):61–92. Available from: http://dx.doi.org/10.1007/978-3-642-28756-522 http : //dx.doi.org/10.1007/s10703 − 013 − 0183 − 7.
8. Kwiatkowska M, Parker D, Wiltsche C. PRISM-games 2.0: A Tool for Multi-Objective Strategy Synthesis for Stochastic Games. In: TACAS; 2016. Available from: http://www.prismmodelchecker.org/papers/tacas16pg2.pdf.
9. Mell P, Grance T. The NIST Definition of Cloud Computing. National Institute of Standards and Technology; 2009. Available from: https://citadel-information.com/wp¬content/uploads/2012/08/NIST-SP800-145-definition-of-cloud-computing.pdf.
10. H¨ofer C, Karagiannis G. Cloud computing services: taxonomy and comparison. Journal of Internet Services and Applications. 2011;:1–14Available from: http://dx.doi.org/10.1007/s13174-011-0027-x.
11. Lamanna DD, Skene J, Emmerich W. SLAng: a language for defining service level agreements. In: Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of; 2003. p. 100–106.
12. Keller A, Ludwig H. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management. 2003; 11(1):57–81. Available from: http://link.springer.com/10.1023/A:1022445108617.
13. Theilmann W, Yahyapour R. SLA@SOI -SLAs Empowering a Dependable Service Economy; 2010. Available from: https://ercim-news.ercim.eu/en83/special/slasoi-slas¬empowering-a-dependable-service-economy.
14. Patel P, Ranabahu A, Sheth A. Service Level Agreement in Cloud Computing; 2009.
15. Serrano D, Bouchenak S, Kouki Y, et al. SLA guarantees for cloud services. Future Generation Computer Systems. 2016;54:233–246. Available from: http://www.sciencedirect.com/science/article/pii/S0167739X15000801.
16. Metsch T, Edmonds A. Open cloud computing interface-infrastructure; 2011. Available from: https://redmine.ogf.org/attachments/220/infrastructure.pdf.
17. Bohn RB, Messina J, Liu F, et al. NIST Cloud Computing Reference Architecture. In: 2011 IEEE World Congress on Services; 7. IEEE; 2011. p. 594–596. Available from: http://ieeexplore.ieee.org/document/6012797/.
18. Buyya R, Calheiros R, Li X. Autonomic cloud computing: Open challenges and architectural elements. Emerging Applications of. 2012; Available from: http://ieeexplore.ieee.org/xpls/absall.jsp?arnumber = 6407847.
19. WSAG4J -WS-Agreement for Java; ???? Available from: http://wsag4j.sourceforge.net/site/index.html.

20. Jamshidi P, Pahl C, Mendonca NC. Managing Uncertainty in Autonomic Cloud Elasticity Controllers. IEEE Cloud Computing. 2016 5;3(3):50–60. Available from: http://ieeexplore.ieee.org/document/7503491/.
21. Kephart JO, Chess DM. The vision of autonomic computing. Computer. 2003;36(1):41– 50.
22. Kwiatkowska M. Quantitative verification: models techniques and tools. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering -ESEC¬FSE '07; 9; New York, New York, USA. ACM Press; 2007. p. 449. Available from: http://dl.acm.org/citation.cfm?id=1287624.1287688.
23. Calinescu R, Ghezzi C, Kwiatkowska M, et al. Self-adaptive software needs quantitative verification at runtime. Communications of the ACM. 2012;55(9):69–77.
24. Chen T, Forejt V, Kwiatkowska M, et al. PRISM-games: A model checker for stochastic multi-player games. Springer; 2013. p. 185–191.
25. Mayer P, Klarl A, Hennicker R, et al. The Autonomic Cloud: A Vision of Voluntary, Peer-2-Peer Cloud Computing. In: 2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops; 9. IEEE; 2013. p. 89–94. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6803264.
26. Celestini A, Lluch Lafuente A, Mayer P, et al. Reputation-Based Cooperation in the Clouds. In: Zhou J, Gal-Oz N, Zhang J, et al., editors. Trust management viii: 8th ifip wg 11.11 international conference, ifiptm 2014, singapore, july 7-10, 2014. proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg; 2014. p. 213–220. Available from: http://dx.doi.org/10.1007/978-3-662-43813-815.
27. Ludwig H. Web Service Level Agreement (WSLA) Language Specification Foundations of SLA Management for services and utility computing View project Zenith View project; 2003.
28. Frankova G, Malfatti D, Aiello M. Semantics and Extensions of WS-Agreement. Journal of Software. 2006 7;1(1).
29. Dan AA, Ludwig H, Rofrano J. WS-Agreement Structure. 2004;(January).
30. Chen T, Forejt V, Kwiatkowska M, et al. On Stochastic Games with Multi¬ple Objectives. Springer, Berlin, Heidelberg; 2013. p. 266–277. Available from: http://link.springer.com/10.1007/978-3-642-40313-225.
31. Ismail A, Kwiatkowska M. Synthesizing Pareto Optimal Decision for Autonomic Clouds Using Stochastic Games Model Checking. In: 2017 24th Asia-Pacific Software Engineering Conference (APSEC); 12. IEEE; 2017. p. 436–445. Available from: http://ieeexplore.ieee.org/document/8305966/.
32. PRISM-games -Property Specification; Available from: https://www.prismmodelchecker.org/games/properties.php.
33. Baresi L, Morzenti A, Motta A, et al. From Interaction Overview Diagrams to Temporal Logic. Springer, Berlin, Heidelberg; 2011. p. 90–104. Available from: http://link.springer.com/10.1007/978-3-642-21210-99.
34. Baresi L, Morzenti A, Motta A, et al. A logic-based semantics for the verification of multi-diagram UML models. ACM SIGSOFT Software Engineering Notes. 2012 7;37(4):1. Available from: http://dl.acm.org/citation.cfm?doid=2237796.2237811.
35. Ciapessoni E, Mirandola P, Coen-Porisini A, et al. From formal models to formally based methods: an industrial experience. ACM Transactions on Software Engineering and Methodology. 1999 1;8(1):79–113. Available from: http://portal.acm.org/citation.cfm?doid=295558.295566.
36. Pradella M, Morzenti A, San Pietro P. The symmetry of the past and of the future. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering ¬ESEC-FSE '07; New York, New York, USA. ACM Press; 2007. p. 312. Available from: http://portal.acm.org/citation.cfm?doid=1287624.1287669.
37. Meziani L, Bouabana-Tebibel T, Bouzar-Benlabiod L. From Petri Nets to UML Model: A New Transformation Approach. In: 2018 IEEE International Conference on Information Reuse and Integration (IRI); 7. IEEE; 2018. p. 503–510. Available from: https://ieeexplore.ieee.org/document/8424751/.
38. Westergaard M. CPN Tools 4: Multi-formalism and Extensibility. Springer, Berlin, Heidelberg; 2013. p. 400–409. Available from: http://link.springer.com/10.1007/978-3-642¬38697-822.
39. Gallotti S, Ghezzi C, Mirandola R, et al. Quality prediction of service compositions through probabilistic model checking. Springer; 2008. p. 119–134.
40. Baier C, Katoen J, Larsen K. Principles of model checking. ; 2008. Available from: https://books.google.com.my/books?hl=enlr=id=5dvxCwAAQBAJoi=fndpg=PR13dq=+Principles+of+model+c
41. Krotsiani M, Spanoudakis CK, George. Validation of Service Level Agreements Using Probabilistic Model Checking. In: 2017 IEEE International Conference on Services Computing (SCC); 6. IEEE; 2017. p. 148–155. Available from: http://ieeexplore.ieee.org/document/8034979/.