

## A GUI-driven prototype for synthesizing self-adaptation decision

Azlan Ismail<sup>1</sup>, Susanti Intu<sup>2</sup>, Suzana Zambri<sup>3</sup>

<sup>1,2,3</sup>Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA Shah Alam, 40450, Selangor, Malaysia

<sup>1</sup>Knowledge and Software Engineering Research Group, Universiti Teknologi MARA Shah Alam, 40450, Selangor, Malaysia

---

### Article Info

#### Article history:

Received Aug 21, 2019

Revised Dec 28, 2019

Accepted Jan 23, 2020

---

#### Keywords:

Autonomic clouds  
Model checking  
Self-adaptation  
Stochastic games  
Synthesis

---

### ABSTRACT

The ability to ensure an optimal decision is significant for self-adaptive systems especially when dealing with uncertainty. For this reason, a synthesis-driven approach can be used to capture and synthesize a decision that aims to satisfy the multi-objective properties. Assessing the quality of the synthesis-driven approach is challenging, since it involves a set of activities from modeling, simulating, and analyzing the outcomes. This paper presents the design and implementation of a graphical user interface (GUI)-based prototype for assessing synthesis outcome and performance of an adaptation decision. The prototype is designed and developed based on the component-based development approach that is able to integrate the existing and related libraries from PRISM-games model checker for the synthesis engine, JFreeChart libraries for the chart presentation, and Java Universal Network/Graph Framework libraries for the graph visualization. This paper also presents the implementation of the proposed prototype based on the cloud application deployment scenario to illustrate its applicability. This work contributes to provide a fundamental work towards automated synthesis for self-adaptive systems.

*This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.*



---

### Corresponding Author:

Azlan Ismail,  
Faculty of Computer and Mathematical Sciences,  
Universiti Teknologi MARA Shah Alam, 40450,  
Selangor, Malaysia.  
Email: azlanismail@uitm.edu.my

---

## 1. INTRODUCTION

Self-adaptation is a promising strategy used to address the uncertainty, dynamicity and complexity of modern software systems and their environments [1-3]. It enables a software system which consists of an autonomic manager to monitor the targeted system, detect the need for adaptation, make an adaptation whenever needed, and execute the plan to change the behavior of the targeted system from unwanted state to the required state. The notion of self-adaptation has been promoted and approached in many areas, among them, service-oriented systems [4, 5] and cloud-based system [6], and cyber-physical systems [7].

In the context of cloud computing, self-adaptation is useful especially for Autonomic Clouds [6] which introduces the notion of self-adaptive of multi-cloud environment. Each cloud has an autonomic manager that is capable to perform self-adaptation depending on the management tasks (e.g. allocation, scheduling, migration, scaling). For instance, in the case of application deployment, self-adaptation is needed to migrate an application from a cloud to another cloud when dealing with resource scarcity and to ensure the Service Level Agreement of the cloud application can be fulfilled.

One of the key techniques to realize self-adaptation is based on the software synthesis. The common aim of synthesis for SASS is to generate a correct system behavior (correct-by-construction) that satisfies certain goals as a result of self-adaptation. Generating a correct system behavior manually requires a substantial development effort and a thorough knowledge of the application domain. Hence, automated synthesis is needed to guarantee the generation of a system design with the targeted system behavior. Many approaches have been proposed to realize synthesis method for SASS as discussed in [8], namely, synthesis of service compositions [9], synthesis of self-adaptive connectors [10], parameter synthesis [11], and synthesis for self-adaptation [12].

In this work, we focus on the synthesis for self-adaptation that aims to either analyze best- or worst-case scenarios of alternative designs for self-adaptation mechanisms with the assumptions of the system behavior in relation to the operating environment. The key challenge to perform synthesis for self-adaptation lies on the complexity of synthesis tasks, in particular to explore the generated adaptation strategies with different combination of quality attributes and uncertainty of the environment. For this reason, an automated synthesis for self-adaptation is very much needed. The underlying technique to realize self-adaptation and its relation to the operating environment, is based on model checking of stochastic multiplayer games (SMGs) [13]. This technique is suitable since it allows capturing the uncertainty and variability of the environment, and the competitive behavior between the self-adaptive system and its environment.

There are limited works that attempted to address automated synthesis for self-adaptation decision using model checking techniques, namely, Camara *et al.* [12], Franco *et al.* [14], Pandey *et al.* [15], Camara *et al.* [16]. The review on these works result in several conclusion. Firstly, there is only [12] applied SMG, whilst other applied DTMC (in [14 and 16]), and MDP (in [15 and 16]) as the focused model. All of them made use of PRISM model checker [17], except [12] that utilized PRISM games model checker [18] for the synthesis engine. However, it is not clear how far the work by [12] has integrated their simulation with PRISM games as the backend. Recent work by Ismail and Kwiatkowska [19] addressed synthesis for self-adaptation within the context of autonomic cloud. However, the simulation part was not targeted for GUI-driven prototype.

Therefore, this work extends the previous work of [19] to provide GUI support in dealing with repetitive exploratory and experimenting activities that include configuration settings, dataset generation, creation of specification, model synthesis, and visualizing the outcomes. In particular, this work contributes to ease three main activities: (i) the parameters configuration, (ii) the execution of a series of simulation for the (re)synthesis that is integrated with the libraries of PRISM games model checker [18], (iii) the aggregation and presentation of performance results using JFreeChart [20], (iv) and the exploration of synthesis outcomes using JUNG framework [21]. We utilize the adaptation decision problem of autonomic clouds application deployment scenario of Science Cloud Platform (SCP) [6] to illustrate the feasibility of the prototype. The rest of this paper is organized as follows. Section 2 introduces the exemplar scenario for this work. Section 3 present the design of the proposed prototype. In Section 4, we discuss the implementation of the prototype. We then conclude the paper in Section 5.

## 2. EXAMPLAR SCENARIO

In this section, we introduce the exemplar scenario relates to the autonomic clouds environment as proposed in [6]. In this environment, there are a group of clouds, also known as ensemble [22] that need to work collaboratively to provision resources while having different roles, such as initiator and deployer. Each cloud can join and leave the group. Furthermore, each cloud is embedded with an autonomic manager [23] that can monitor its current resource, make decision if an adaptation is required (i.e. limited resource, virtual machine failure) and perform the planned action. In this scenario, the adaptation decision problem is to choose the best cloud to take the responsibility to deploy the application. Meanwhile, the decision maker is the cloud that is having resource scarcity and not able to continue deploying the cloud application. The challenges in making the decision are influenced by several factors that include multi-objectives (i.e. minimize cost and maximize reliability), uncertainty of adaptation outcomes (i.e. fail to deploy due to the selected cloud becomes unavailable), and the resource variation in terms of timeslots. The details can be referred to [19]. Hence, the GUI-driven prototype should be able to ease the experiment activities for exploring and evaluating the quality of synthesized adaptation decision.

## 3. APPROACH DESIGN

In this section, we introduce the architectural model and the process flow of the proposed prototype.

### 3.1. Architectural model

We view its architectural from input, components, and output perspective. The objective of the proposed support tool is to ease the assessment and exploration of synthesis-driven component. The input data focuses on representing the runtime scenario. Therefore, we divide the inputs as those which can be set manually and those that are generated randomly. The data that can be set manually refers to the number of simulation cycle, the range number of cloud collaborators, and the range number of variation for each cloud that we are interested in the assessment. Meanwhile, the data to be generated randomly refers to the quantitative information of provisioning an application onto a specific resource (i.e. for each variation of each cloud), specifically, the execution time, the reliability, and the cost.

The output data can be classified into two stages, intermediate output data and the presentable output data. The intermediate output data is generated after completing execution of synthesis-driven component per simulation cycle. The intermediate output data can be classified into two types; (i) the outcome of the synthesis-driven approach stored in two files i.e. state-transition and strategies file, and (ii) the behavior of the synthesis-driven behavior stored in the log files. The state-transition file contains the state-transition model as a result of pre-processing of model checking. The strategies file contains the winning strategies as a result of performing model checking with strategy synthesis. It is important to note that, we only keep the version of both files of the last simulation cycle for the sake proof-of-concept. Meanwhile, the log files contain the performance behavior of synthesis-driven approach according to the configuration setting. These log files are kept for every simulation cycle. The presentable output data can only exist, once the intermediate output data is generated. Furthermore, it is generated upon request, by the system user.

The architecture consists of several components. Firstly, the interface panel to enable some configurations, initiate certain tasks (i.e. simulation, results presentation and visualization) and to present a chart as well as to visualize graph. Secondly, the simulator to support data generation process by controlling the amount of simulation cycle and to compute some data for displaying purpose. Thirdly, the synthesis controller to control the execution of synthesis-driven approach. Fourthly, the stochastic games engine or libraries to execute the stochastic games model checking with synthesis algorithm, the Pareto set computation algorithm, and to export the state-transition and strategies profiles.

### 3.2. Overall process flow

We divide the process flow into three stages, namely, data generation, chart presentation, and graph visualization. We present the process flow as a sequence diagram. Figure 1 illustrates the data generation part with simulation as the core task. The sequence diagram for the chart presentation and graph visualization is not illustrated due to the page constraint.

The data generation begins with the initiation activity from the system user via the interface panel. It involves inserting data into the required configuration parameters (1) and making a simulation request i.e. through pressing button (2). This will trigger the simulation task of the simulator which executes based on number of simulation cycle (3). For each cycle, it starts with the generation of random values to represent the runtime information (4). Then, the simulator initiates the synthesis task via synthesis controller (5) that results in encoding the model and properties specification (6) followed by pre-processing task (7). After that, the synthesis controller initiates the model checking process (8) via stochastic games engine that performs the model checking algorithm with strategy synthesis based on multi-objective properties (9). The successful completion of model checking results in the generation of log data, state-transition and strategies files (10|13). In the case of failure, re-synthesis is performed. It starts with the computation of Pareto set followed by executing the model checking task (12). The re-synthesis will be done within a certain threshold. The status of this generation is returned to the synthesis controller (11|14).

Once the simulation cycle has been completed, the system user can choose to either explore the performance behavior of the synthesis-driven component via chart presentation or explore the state-transition and strategies via graph visualization. The chart presentation or graph visualization begins when the user makes the request through interface panel (15|20) which triggers the request to the simulator (16|21). If the data has not been generated yet, then an error message will be displayed (19|24). Otherwise, the logged/gathered data is computed for the chart presentation (17) or prepared for the graph visualization (22), and displayed to the user (18|23).

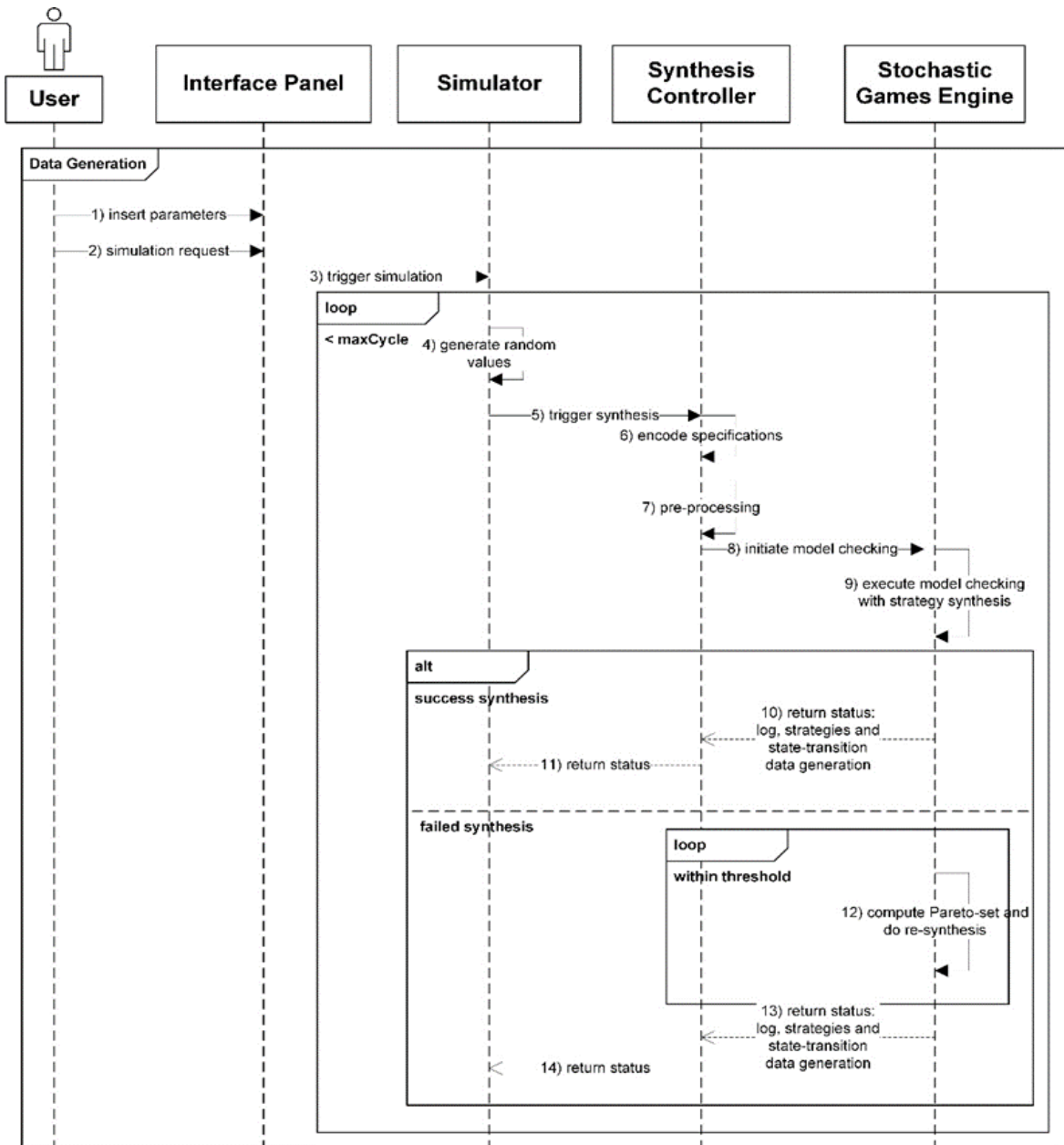


Figure 1. Data generation process

**4. IMPLEMENTATION**

We have implemented a proof of concept prototype for addressing cloud application deployment problem discussed earlier. The main interface of the prototype system is shown in Figure 2. It provides the configuration panel for the user to enter the respective parameters, the button to execute the synthesis process, the chart panel to show the performance results, and the button to visualize the state-transition graph and the strategies graph.

The parameters that can be set by the system users are meant to define the cloud deployment model as well as the simulation setting. Specifically, the items are as follows. Number of Cloud that is used to define a group of clouds that is currently in the joint collaboration. Number of Variation that is needed to define a series of resource variation in terms of timeslots for each cloud. A timeslot determines the expected resource utilization by a cloud. Number of Simulation Cycle that is used to set the number of cycle per combination of cloud and its variation. It is important to obtain the mean value. Value Assignment that is required to set when to assign the generated random values into the behavioral model specification, either during the creation of model specification, or during the creation of a model instance. This parameter is relevant for the user to explore the performance of synthesis in relation to the values

assignment stage. Finally, Objective Properties with three objectives, specifically, cost, time, and reliability. We choose these three objectives since they are commonly used to evaluate the quality goals of self-adaptive systems, e.g. [24, 25]. For each objective, the system user can define the threshold values and its comparator symbol (i.e.  $\$, >, =$ ).

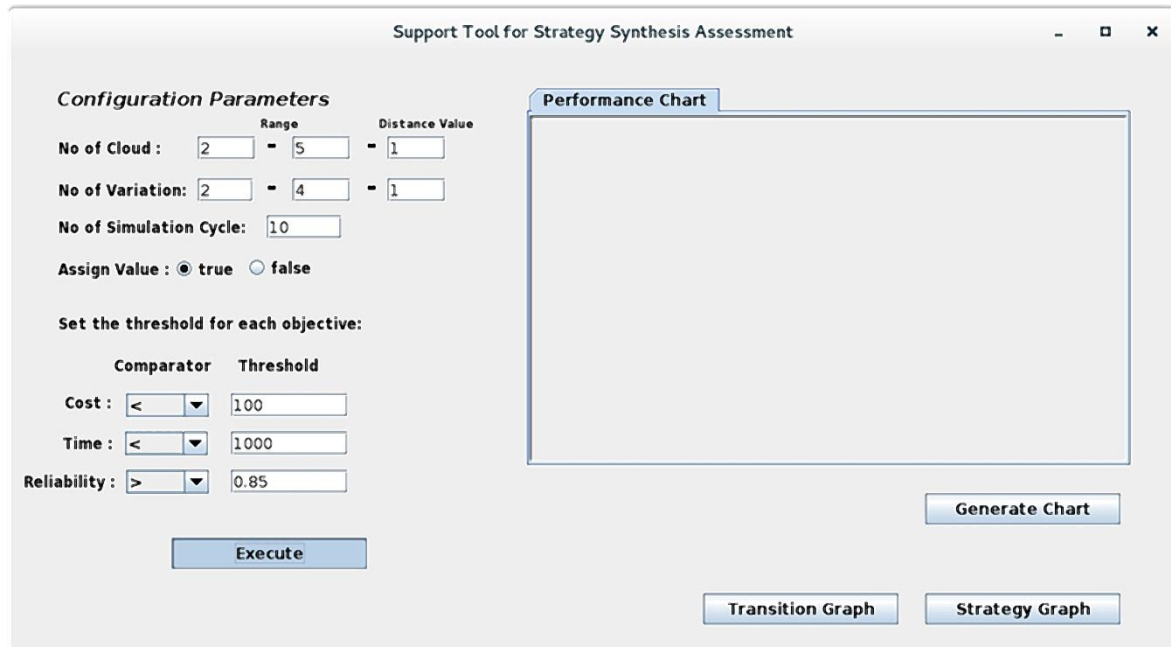


Figure 2. Main interface of the prototype

The parameters that can be set by the system users are meant to define the cloud deployment model as well as the simulation setting. Specifically, the items are as follows. Number of Cloud that is used to define a group of clouds that is currently in the joint collaboration. Number of Variation that is needed to define a series of resource variation in terms of timeslots for each cloud. A timeslot determines the expected resource utilization by a cloud. Number of Simulation Cycle that is used to set the number of cycle per combination of cloud and its variation. It is important to obtain the mean value. Value Assignment that is required to set when to assign the generated random values into the behavioral model specification, either during the creation of model specification, or during the creation of a model instance. This parameter is relevant for the user to explore the performance of synthesis in relation to the values assignment stage. Finally, Objective Properties with three objectives, specifically, cost, time, and reliability. We choose these three objectives since they are commonly used to evaluate the quality goals of self-adaptive systems, e.g. [24, 25]. For each objective, the system user can define the threshold values and its comparator symbol (i.e.  $\$, >, =$ ).

Besides that, there are three main buttons for executing the core functionalities. Execute button to perform the synthesis-driven method for a set of simulation cycle. For each cycle, it generates a decision behavioral model specification as shown in Figure 3 and multi-objective properties specification. The generation takes the configuration parameters defined earlier. Then, the model checking is performed which produces the state-transition and strategies profile. In general, the state-transition profile contains all possible paths that can be executed by the decision maker. Meanwhile, the strategy profile contains the selected path(s) that can satisfy multi-objective properties as shown in Figure 4. Transition and Strategy graph button to map the data in the state-transition and strategies profile into a graph visualization.

Example of graph visualization for strategies profile is illustrated in Figure 5 (note: the state-transition graph is not shown in this paper). It represents the exploration for a decision maker which leads to the goal state. It contains the selected and explored paths as a subset of the state-transition profile. The states from both files are mapped to the nodes, whilst the transitions (i.e. the action) are mapped to the edges. From the sample of strategies graph, it illustrates four nodes and four transitions. State 0 is the initial state of the decision maker, and State 4 is determined as the goal state. The exploration begun with choosing Cloud 0 (labeled as  $r_0$ ), and resulted in moving to State 7. Then, it had explored two branches. One of the branches (a)

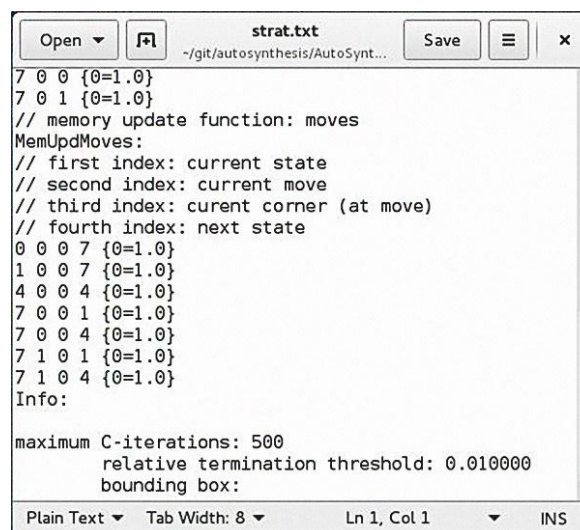
was to State 1 by choosing timeslot 0 (labeled as rs0) of Cloud 0 (labeled as n0). However, this branch was expected to fail and thus the exploration moved back to State 7. Another branch (b) was to State 4 by choosing timeslot 1 (labeled as rs1) of Cloud 0 (labeled as n0). State 4 is actually the goal state. This means, the selection of branch (b) at runtime can lead to a successful adaptation. Furthermore, as a system user, we can conclude that Cloud 0 is the best option based on the defined configuration if the application to be migrated and deployed executes within timeslot 1.

```

152 const double n4rs3_rel = 0.9226748058577241; //rel
153 const int n4rs3_time = 850; //time
154 const double n4rs3_res = 0.3401740479962477; //time
155
156
157 //=====Global Parameters=====
158 global t:[0..1] init 0; //to control the turn
159 global goal : bool init false; //(absorbing state)
160 global n:[-1..MXN] init -1; //number of computing node
161
162 //=====Module for Player 1=====
163 module planner
164 //P1's coordinator :
165 [end] (t=0) & (goal=true) -> true; //to end the selection
166 //P1 moves :
167 [r0] (t=0) & (goal=false) -> (n'=0) & (t'=1);
168 [r1] (t=0) & (goal=false) -> (n'=1) & (t'=1);
169 [r2] (t=0) & (goal=false) -> (n'=2) & (t'=1);
170 [r3] (t=0) & (goal=false) -> (n'=3) & (t'=1);
171 [r4] (t=0) & (goal=false) -> (n'=4) & (t'=1);
172 endmodule
173
174 //=====Module for Player 2=====
175 module environment
176 //P2 moves for single or sequential pattern:
177 [n0rs0] (t=1) & (n=0) & (app_rs + n0rs0_res <= mx_rs) -> n0rs0_rel:(goal'=true) &
178 [n0rs1] (t=1) & (n=0) & (app_rs + n0rs1_res <= mx_rs) -> n0rs1_rel:(goal'=true) &
179 [n0rs2] (t=1) & (n=0) & (app_rs + n0rs2_res <= mx_rs) -> n0rs2_rel:(goal'=true) &
180 [n0rs3] (t=1) & (n=0) & (app_rs + n0rs3_res <= mx_rs) -> n0rs3_rel:(goal'=true) &
181
182 [n1rs0] (t=1) & (n=1) & (app_rs + n1rs0_res <= mx_rs) -> n1rs0_rel:(goal'=true) &
183 [n1rs1] (t=1) & (n=1) & (app_rs + n1rs1_res <= mx_rs) -> n1rs1_rel:(goal'=true) &
184 [n1rs2] (t=1) & (n=1) & (app_rs + n1rs2_res <= mx_rs) -> n1rs2_rel:(goal'=true) &
185 [n1rs3] (t=1) & (n=1) & (app_rs + n1rs3_res <= mx_rs) -> n1rs3_rel:(goal'=true) &
186

```

Figure 3. Sample of decision behavioral model based on SMG models in PRISM games model checker



```

Open  strat.txt  Save  x
-/git/autosynthesis/AutoSynt...
7 0 0 {0=1.0}
7 0 1 {0=1.0}
// memory update function: moves
MemUpdMoves:
// first index: current state
// second index: current move
// third index: curent corner (at move)
// fourth index: next state
0 0 0 7 {0=1.0}
1 0 0 7 {0=1.0}
4 0 0 4 {0=1.0}
7 0 0 1 {0=1.0}
7 0 0 4 {0=1.0}
7 1 0 1 {0=1.0}
7 1 0 4 {0=1.0}
Info:
maximum C-iterations: 500
relative termination threshold: 0.010000
bounding box:
Plain Text  Tab Width: 8  Ln 1, Col 1  INS

```

Figure 4. Text-based synthesized strategy

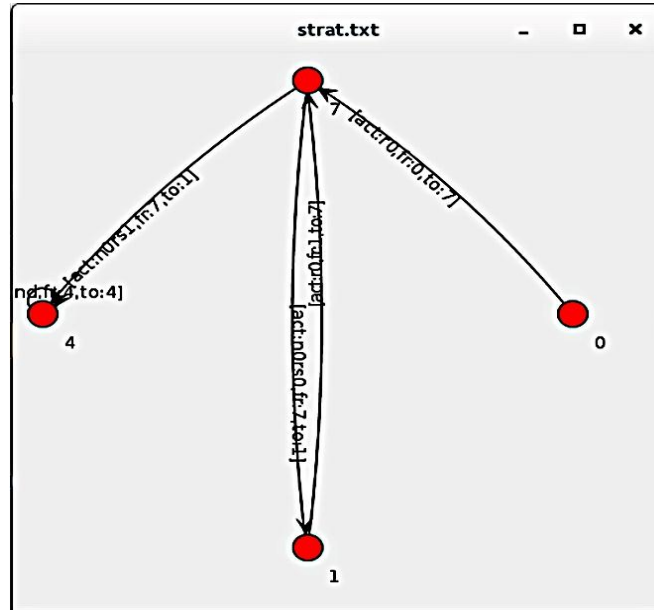


Figure 5. Visualization of synthesized strategy

Finally, Generate chart button that is used to aggregate the performance data collected for each simulation cycle. Then, a chart is visualized such as shown in Figure 6. In this figure, it presents three lines of data distribution, where each line represents the synthesis mean time of a specific variation number with increasing number of clouds. As presented in the chart, the higher number of variation with higher number of cloud takes longer time. In addition, we can also notice unusual pattern from the chart. For instance, the mean time of synthesis at four variation is taking a bit longer when there are four clouds as compared to five clouds. This situation may be caused by re-synthesizing cycle due to unsatisfactory of specified multi-objectives. As another example, we can see the minimum mean time of two clouds at four variation as compared to two and three variation. This condition may be due to having more options for making the decision that reduces the need for re-synthesizing and results in less synthesis time.

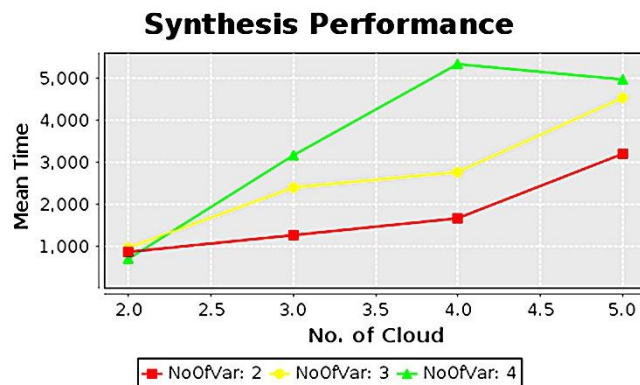


Figure 6. Sample of performance chart

## 5. CONCLUSION

We have presented a GUI-driven prototype for supporting the synthesis of self-adaptation decision in the autonomic clouds environment. It allows the users to configure certain parameters, execute simulation, generate performance chart, and explore the synthesis outcomes via graph visualization. The current limitation of the prototype can be viewed from two aspects, namely, the assessment scenario, and the functionality. In terms of assessment scenario, it is currently meant for the cloud application

deployment problem. Furthermore, it only limits to the synthesis-driven for self-adaptation decision. Hence, further study can focus on generalizing the problem as coordination problem of collective self-adaptive systems that involves decentralized decision making. From the functionality perspective, it currently limits the number of properties to three objectives. In addition, the graph visualization does not store the state-transition and strategies profile for each simulation cycle of each configuration. These limitations can be extended depending on the needs and can be taken as future work.

Despite the limitation, we believe that this prototype opens for more opportunities to be enhanced to support the investigation and exploration of self-adaptation decision process. The short term enhancement include to provide a flexibility in defining different number of objective properties, and to allow a predefined model and properties to be imported into the prototype environment, besides the one that can be generated automatically. The long term enhancement may include the ability to execute different synthesis techniques with learning ability for investigating the behavior of self-adaptation decision.

## ACKNOWLEDGEMENTS

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] B. Cheng *et al.*, "Software engineering for self-adaptive systems: A research roadmap software engineering for self-adaptive systems," *Software Engineering for Self-Adaptive Systems*, vol. 5525, pp. 1-26, 2009.
- [2] R. Laddaga, "Active software," *IWSAS, Proceedings of the first international workshop on Self-adaptive*, vol. 1936, pp. 11-26, April 2000.
- [3] R. De Lemos, *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," *Springer*, vol. 7475, pp. 1-32, January 2013.
- [4] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," in *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387-409, May-June 2011.
- [5] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola, "MOSES: A framework for QoS driven runtime adaptation of service-oriented systems," in *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1138-1159, Sept.-Oct. 2012.
- [6] P. Mayer, *et al.*, "The autonomic cloud," in *Software Engineering for Collective Autonomic Systems*, vol. 8998, pp. 495-512, 2015.
- [7] A. Musil, J. Musil, D. Weyns, T. Bures, H. Muccini, and M. Sharaf, "Patterns for self-adaptation in cyber-physical systems," in *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, pp. 331-368, May 2017.
- [8] R. Calinescu, *et al.*, "Synthesis and verification of self-aware computing systems," in *Self-Aware Computing Systems*, pp. 337-373, January 2017.
- [9] M. Autili, D. Di Ruscio, A. Di Salle, and A. Perucci, "CHOReOSynt: enforcing choreography realizability in the future internet," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering-FSE 2014*, pp 723-726, November 2014.
- [10] A. Di Marco, P. Inverardi, and R. Spalazzese, "Synthesizing self-adaptive connectors meeting functional and performance concerns," *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, San Francisco, CA, pp. 133-142, 2013.
- [11] T. Quatmann, C. Dehnert, N. Jansen, S. Junges, and J.-P. Katoen, "Parameter synthesis for markov models: faster than ever," *International Symposium on Automated Technology for Verification and Analysis*, vol. 9938, pp. 50-67, September 2016.
- [12] J. Cámara, D. Garlan, B. Schmerl, and A. Pandey, "Optimal planning for architecture-based self-adaptation via model checking of stochastic games," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing-SAC '15*, pp 428-435, April 2015.
- [13] M. Svoreňová and M. Kwiatkowska, "Quantitative verification and strategy synthesis for stochastic games," *European Journal of Control*, vol. 30, pp. 15-30, July 2016.
- [14] J. M. Franco, F. Correia, R. Barbosa, M. Zenha-Rela, B. Schmerl, and D. Garlan, "Improving self-adaptation planning through software architecture-based stochastic modeling," *Journal of Systems and software*, vol. 115, pp. 42-60, May 2016.
- [15] A. Pandey, G. Moreno, J. Cámara, and D. Garlan, "Hybrid planning for decision making in self-adaptive systems," *2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, Augsburg, pp. 130-139, 2016.
- [16] J. Cámara, D. Garlan, and B. Schmerl, "Synthesizing tradeoff spaces with quantitative guarantees for families of software systems," *Journal of Systems and Software*, vol. 152, pp. 33-49, June 2019.
- [17] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," *International Conference on Computer Aided Verification-CAV 2011*, vol. 6806, pp. 585-591, 2011.



- 
- [18] M. Kwiatkowska, D. Parker, and C. Wiltsche, "PRISM-games 2.0: A tool for multi-objective strategy synthesis for stochastic games," in *TACAS-International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, vol. 9636, pp. 560-566, April 2016.
- [19] A. Ismail and M. Kwiatkowska, "Synthesizing pareto optimal decision for autonomic clouds using stochastic games model checking," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, Nanjing, pp. 436-445, 2017.
- [20] Andreas Viklund, "JFreeChart." [Online]. Available: <http://www.jfree.org/jfreechart/>. [Accessed: 02-May-2018].
- [21] "JUNG-Java Universal Network/Graph Framework." [Online]. Available: <http://jung.sourceforge.net/> [Accessed: 02-May-2018].
- [22] A. Klarl, P. Mayer, and R. Hennicker, "Helena@Work: Modeling the science cloud platform," *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, vol. 8802, pp. 99-116, 2014.
- [23] R. Buyya, R. Calheiros, and X. Li, "Autonomic cloud computing: Open challenges and architectural elements," *2012 Third International Conference on Emerging Applications of Information Technology*, Kolkata, pp. 3-10, 2012.
- [24] S. Cheng, D. Garlan and B. Schmerl, "Evaluating the effectiveness of the rainbow self-adaptive system," *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Vancouver, BC, 2009, pp. 132-141.
- [25] D. Weyns and R. Calinescu, "Tele Assistance: A Self-Adaptive Service-Based System Exemplar," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Florence, pp. 88-92, 2015.