2021 16th International Conference of System of Systems Engineering (SoSE) | 978-1-6654-4454-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SOSE52739.2021.9497463

Generation of Adaptation Strategies for Dynamic Reconfiguration of a System of Systems

SungJin Lee, Young-Min Baek, Sangwon Hyun, and Doo-Hwan Bae
School of Computing
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea
{sjlee, ymbaek, swhyun, bae}@se.kaist.ac.kr

***Abstract*—A system of systems (SoS) is comprised of constituent systems that interact with each other to achieve a common goal. An SoS operates in an environment with dynamically changing external conditions. Therefore, supporting the resilience of the SoS is crucial to sustain goal achievement at run-time. One way to support this resilience is to generate an adaptation strategy at run-time that guides the dynamic reconfiguration of the SoS. Therefore, adaptation strategies should be effectively generated within a reasonable time at run-time to conduct the dynamic reconfiguration in time. To satisfy this requirement, we propose an adaptation strategy generation approach that consists of two phases. 1) At design time, our approach constructs a knowledge base that contains adaptation strategies for expected environmental conditions. 2) At/during run-time, it generates adaptation strategies by utilizing the knowledge base. Through a case study, we show that our approach can generate adaptation strategies effectively at run-time.**

***Keywords*—System of Systems (SoS), Self-Adaptation, Adaptation strategy, Multi-objective optimization (MOO)**

I. Introduction

Recently, interests in analyzing the behaviors of a system-of-systems (SoS) has been growing because of the increasing complexity of modern systems. An SoS is a goal-oriented system comprised of multiple Constituent Systems (CSs), that have autonomy and independence, and interact with each other to achieve a higher-level common goal [1].

As with other software-intensive systems, supporting the resilience of an SoS is one of the major concerns for SoS engineers trying to effectively deal with dynamic environmental condition changes. Considering the resilience of the SoS, Nielsen *et al.* [1] defines one key dimension of the SoS, which is that an SoS autonomously conducts dynamic reconfiguration by changing its configuration to adapt to external environmental condition changes at run-time. Therefore, SoS managers must be able to guide the dynamic reconfiguration of the SoS to maintain its goal achievement by adapting to environmental condition changes. One promising way to guide the dynamic reconfiguration is to provide the SoS with adaptation strategies that prescribe how the SoS conducts the dynamic reconfiguration.

To generate adaptation strategies at run-time, there are two important requirements that should be considered. First, the generation process should take a reasonable time to have an SoS reconfigure in time. Second, the generated adaptation strategy should be able to properly guide the SoS to an optimal configuration that maximizes the probability of goal achievement. However, there must be a trade-off between these two requirements because the more time is invested in the generation process, the easier it is to achieve the quality requirement.

In conventional approaches, a system developer or a domain expert manually defines adaptation strategies at system design time by predicting the environmental changes of a target system's run-time [2]. However, since predicting all kinds and varieties of environmental changes at run-time is impossible, predefined adaptation strategies would lead to the degradation of system performance when a discrepancy exists between knowledge before the run-time and actual occurrences of run-time events.

To address these concerns and properly reconfigure an SoS according to dynamic changes in environmental conditions, existing studies have proposed to generate an optimal adaptation strategy at run-time [3], [4]. These approaches select a target system as a formal model and generate an optimal adaptation strategy through exhaustive searching using a model checker. However, these approaches impose major computational costs at run-time. Accordingly, they can only satisfy the quality requirement.

To generate effective adaptation strategies within a reasonable time, Coker *et al.* [5] first applied a stochastic search algorithm and demonstrated its applicability to the adaptation strategy generation problem. However, because the stochastic search algorithm also suffers from conflicting requirements, the convergence speed of the search algorithm must also be improved to generate the adaptation strategy promptly.

To satisfy both objectives, this study utilizes previously known adaptation strategies that were (sub)optimally constructed based on past environmental conditions. This approach then performs strategy generation during run-time by improving the convergence speed of the stochastic search algorithm. Previously discovered solutions could be an appropriate starting point for searching for optimal strategies to adapt to run-time environmental conditions. A key idea is to construct a knowledge base that contains adaptation strategies for expected environmental conditions at design time and then utilizes the knowledge and predefined strategies in the knowledge base to facilitate the generation of adaptation strategies during run-time.

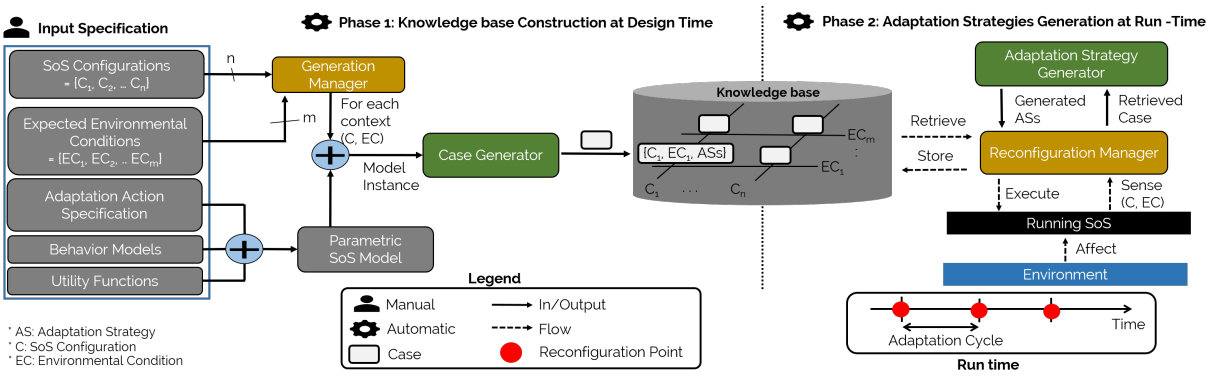

Fig. 1: Overall Approach

The contributions of this study are summarized as follows:

- We propose a dynamic adaptation strategy generation approach at run-time followed by constructing a knowledge base at design time.
- Through a case study, we show that the proposed approach can generate adaptation strategies effectively at run-time.

II. Related Work

Over the last two decades, many approaches have been proposed to support the resilience of systems in the autonomic computing domain. In conventional approaches, systems adapt to environmental condition changes by following adaptation strategies manually defined by system developers at design time [2], [6]. Common limitations of these approaches are that the systems can only adapt to stationary environmental condition changes, which can be foreseen. Furthermore, they require painstaking and time-consuming processes for system engineers to define the adaptation strategies.

To address this concern and make systems adapt to unexpected environmental condition changes, many approaches generating an adaptation strategy at run-time have been proposed. In approaches based on probabilistic model checking [3], [4], the general process is to express a target system as a formal model containing under-specified adaptation decisions and then get an optimal strategy (sequence of adaptation decisions) through an exhaustive search of the probabilistic model checker. The main limitation of these approaches is that they take a long time to generate an adaptation strategy because of the exhaustive search of probabilistic model checking at run-time.

Coker *et al.* first applied a stochastic search algorithm to generate adaptation strategies at run-time [5]. Their search algorithm finds well-established adaptation strategies in a reasonable time by balancing exploration and exploitation for a search space during the search process. Like this work, our approach also uses a stochastic search algorithm with a seeding mechanism that utilizes previously known solutions to support an efficient convergence. Kinneer *et al.* applied a *seeding* technique to generate adaptation strategies [7]. To support the efficient generation of strategies, the seeding approach generates an initial population based on previously known solutions instead of randomly initializing the initial population, and it facilitates the convergence speed of the stochastic search algorithm. They showed that utilization of previously known adaptation strategies and knowledge can generate more effective adaptation strategies than generating them from scratch. When applying the seeding technique, the main concern is now how to generate adequate seeds for an initial population. However, the authors made seeds by heuristically processing past solutions, with an assumption that past solutions already existed. Accordingly, it is difficult to directly apply this approach to other systems if it is difficult to get proper initial seeds. Our approach differs in that it automatically constructs a knowledge base containing good seeds for expected environmental conditions at design time, and it enables retrieval of the seeds from the knowledge base at run-time.

III. Approach

A. Overall Approach

Fig. 1 describes an overview of the proposed approach. The approach is largely divided into two phases: 1) Construction of an initial knowledge base at design time. 2) Generation of adaptation strategies with the seeding at run-time. The key idea behind generating adaptation strategies is to employ a parametric SoS model that takes SoS configuration and environmental conditions as parameters before the NSGAII algorithm generates (sub)optimal adaptation strategies for a given context.

In the first phase, the knowledge base that contains cases is constructed. Each case generation process is as follows. A *generation manager* instantiates the SoS parametric model through a given pair of SoS configurations and environmental conditions. Thereafter, the NSGA-II in a *case generator* takes the parametric SoS model and generates non-dominated adaptation strategies. Lastly, the *case generator* generates the case that contains (sub)optimal adaptation strategies in a given context.

In the second phase, whenever the environmental condition is changed every adaptation cycle, the adaptation strategy generation process is triggered. The overall process is sum-

marized as follows: 1) ***Monitor:*** A *reconfiguration manager* monitors the current context. 2) ***Analysis:*** The *reconfiguration manager* retrieves the most similar adaptation strategies to the monitored context from the knowledge base. 3) ***Plan:*** An *adaptation strategy generator* generates adaptation strategies with the *seeding*. 4) ***Execute:*** The *reconfiguration manager* executes a selected adaptation strategy on the running SoS and stores generated adaptation strategies in the knowledge base.

B. Input Specification

SoS Configuration and Environmental Condition

- SoS configuration (C) specifies the number of CSs in the SoS and is defined as an integer array. Cardinality means how many CS types exist, and each value of SoS configuration variable (cv_i) specifies the number of CSs in the SoS model.
- Environmental condition (EC) specifies an external environmental condition and is defined as a double array. Cardinality means how many external environmental condition variables exist, and each value of environmental condition variable (ecv_i) specifies the unspecified variables (transition probability or sojourn time) of behavior models.

SoS Parametric Model It consists of three components: 1) a network of behavior models, 2) an adaptation action specification, and 3) utility functions. The SoS parametric model is simulated in a discrete-time simulation, which assumes that the system changes only at each discrete time tick. The behavior model is based on the PTA (Probabilistic Timed Automata) [8] to describe the stochastic behavior of CSs and internal environment interacting with CSs. The detailed explanation is on the this link[1].

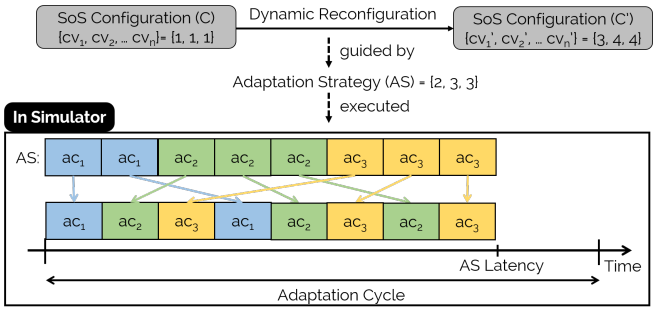

Fig. 2: Dynamic reconfiguration

Adaptation strategy is a sequence of adaptation actions that is expressed as an integer array. Adaptation action is a primitive reconfiguration action that adds or removes one of the CSs from the SoS while taking the time. Each integer value denotes how many CSs are added or removed from the same index of the SoS configuration. Fig. 2 describes an example where an SoS configuration {1, 1, 1} is reconfigured to {3, 4, 4} by executing an adaptation strategy expressed as {2, 3, 3}.

[1]https://github.com/SungJin1212/AdaptationStrategyGenerator

For the evaluation, the simulator simulates the SoS model and the adaptation strategy concurrently for a predefined adaptation cycle. Through the simulation, the adaptation strategy latency and values specifying an SoS level goal achievement defined in the SoS model are calculated.

The utility function maps the objective value defined in the SoS parametric model to the utility function value. The quality of the generated adaptation strategy is evaluated by the weighted-sum of the utility function values.

C. Phase1: Knowledge base Construction at Design time

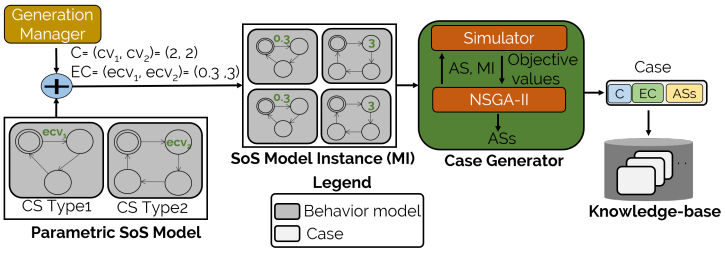

Fig. 3: Case generation process example

For each pair of SoS configurations and environmental conditions, the case generation process is summarized as follows. 1) The parametric SoS model is instantiated by the SoS configuration and environmental condition. 2) Non-dominated adaptation strategies are generated through the NSGA-II. 3) Each case is generated and added to the knowledge base by the *case generator*.

Fig. 3 illustrates an example of the case generation process. The SoS model has two CS types and two unspecified variables. An ecv_1 specifies the probability of the transition, and an ecv_2 specifies the sojourn time of the state. Once the SoS model is instantiated, it will have two CSs for each CS type and unspecified variables are specified according to the C and EC, respectively. The NSGA-II then takes the SoS model and generates adaptation strategies. Lastly, the case is generated and stored in the knowledge base.

D. Phase2: Adaptation Strategies Generation at Run-time

Analysis: Retrieve a Most Similar Case

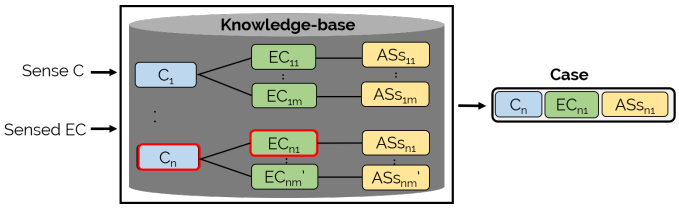

Fig. 4: Retrieve procedure

The retrieve process is summarized as follows. 1) For all SoS configurations in the knowledge base, the most similar configuration with a minimal *Euclidean distance* for a sensed SoS configuration is retrieved. 2) For all environmental conditions with the same SoS configuration as the retrieved

SoS configuration, the most similar environmental condition with a minimal *Euclidean distance* for a sensed environmental condition is retrieved. 3) A case that has retrieved an SoS configuration and an environmental condition is retrieved, and adaptation strategies that will be seeds in the planning step are then extracted in the retrieved case.

Fig. 4 describes an example. If a C_n is the most similar SoS configuration, and an EC_{n1} is the most similar environmental condition, then the case that has C_n and EC_{n1} is retrieved.

Planning: Generate Adaptation Strategies with *Seeding*

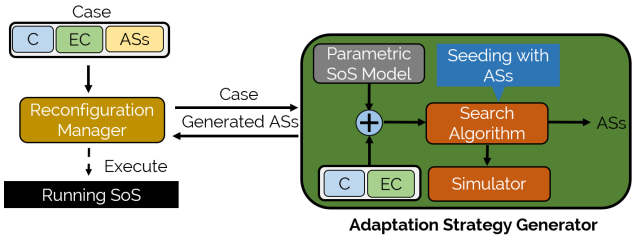

Fig. 5: Process of generating adaptation strategies at run-time

Fig. 5 illustrates a process of generating adaptation strategies. In the *adaptation strategy generator*, once the SoS model reflecting the sensed context has been instantiated, the search algorithm sets an initial population as retrieved adaptation strategies with randomly generated adaptation strategies because the results of past research that uses *seeding* in other domains show that using the whole previous solution could be inferior to starting from scratch [9]. If the size of adaptation strategies is less than the number of seeds, then whole adaptation strategies are set to seeds because the number of Pareto-fronts could be less than the seed size. Once adaptation strategies are generated, one of the adaptation strategies with the maximal utility value is selected and executed.

IV. Case Study

A. MCIResponse SoS

Mass casualty incident (MCI) refers to large-scale incidents emerging in a relatively large number of patients with injuries [10]. MCI response processes typically require multiple collaborative responders to save patients as much as possible. Accordingly, an MCIResponse SoS (MCIR SoS) is composed of autonomous responders (CSs) contribute to an SoS level goal (save patients) by collaborating with one another. When the MCI occurs and is reported, the MCIRSoS assesses the situation and dispatches responders to rescue and save patients. In this scenario, we assume that the MCI is caused by a tsunami emerging, with the patients both at sea and on the ground, and the tsunami occurring continuously for every adaptation cycle. In one adaptation cycle, 500 patients emerge at sea and on the ground, respectively. The reconfiguration problem in this scenario is that the MCIRSoS should decide how many responders to add or remove from the MCI scene when the environmental condition influencing rescue or save activities is changed. If the environmental condition worsens, then the MCIRSoS has to dispatch more CSs at the MCI scene. On the contrary, removing CSs from the MCI scene is necessary when environmental conditions get better because CSs consume operational costs for each tick.

B. MCIRSoS Configuration and Environmental Condition

There are three CS types in the MCIRSoS: a firefighter CS, a helicopter CS, and an ambulance CS. The main role of the firefighter CS and helicopter CS is to rescue the patients on the ground and in the sea, respectively. The role of the ambulance CS is to transfer rescued patients to the hospital. We assume that all CSs can be dispatched to the MCI scene by up to 50. The detailed models are on the this link[1].

TABLE I: Environmental Condition Specification

Variable	Influence	Interval	Min	Max
ecv_1: *WeatherCondition*	Probability of searching patients	0.2	0.1	0.7
ecv_2: *RoadCondition*	Transfer time	2	1	7

Table I shows the environmental condition values. The *RoadCondition* influences the patient transfer time of the ambulance CS. Accordingly, if the *RoadCondition* worsens, the MCIR-SoS should dispatch the ambulance CS additionally. The *WeatherCondition* influences the patient searching probability of the firefighter and helicopter CS. If the *WeatherCondition* worsens, the MCIR-SoS should dispatch the firefighter and helicopter CS additionally. In this scenario, there are 16 environmental condition states, and the environmental condition state is randomly changed with the same probability for every adaptation cycle.

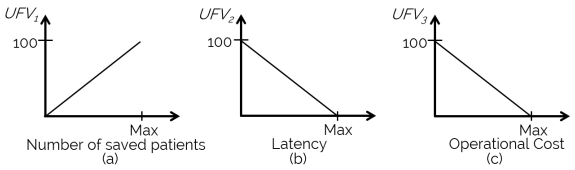

Fig. 6: Utility functions

C. Utility Functions and NSGA-II Specification

TABLE II: Adaptation Strategy Objective Variables

Objective variable	Calculation method
Number of saved patients	Count saved patients
Operational cost	Sum of operational cost of dispatched CSs
Latency	Sum of latency of adaptation actions

The adaptation strategy has three objectives. Table II shows the calculation methods for each objective value. The search algorithm has three objectives: maximize the *Number of saved patients*, minimize the *Latency*, and minimize the *Operational cost*. Fig. 6 contains functions mapping objective values onto the utility function value.

In the MCIRSoS, we calculate the utility value by the weighted-sum of utility function values, as follows:

$$UV = \sum_{i=1}^{N} w_i \times UFV_i$$

where *UV* is the utility value of the adaptation strategy, w_i is the weight of the *i*th objective, UFV_i is the utility function value of the *i*th objective, and *N* is the number of objectives. We set the weight of the *Number of saved patients* to 0.6, the *Operational cost* to 0.3, and *Latency* to 0.1.

TABLE III: NSGA-II Specification

NSGA-II Parameter	Parameter Value
Population Size	20
Seed Size	10
Generation at design time	10
Generation at run time	3
Crossover / Rate	Uniform Crossover / 0.9
Mutation / Rate	Uniform Mutation / 0.1

Table III contains the parameter specifications of the NSGA-II search algorithm. We set the seed size as 10 because other domain studies that use the seeding methodology results show that 50% to 60% of the population size is the best [11]. The other parameter values are chosen by a parameter sweep.

V. Evaluation

A. Experimental Setup

We implement the NSGA-II using a MOEA framework [12], which provides well-known multi objective optimization search algorithms. The experimental environment includes Windows 10 Pro (64-bit) PC with Intel Core i7-7700 3.60GHz processor and 16GB memory.

We define a *# of discrete points* (*NDP*) which specifies the number of partitions for each SoS configuration value. We conduct the experiment with various *NDP* because the performance of our approach depends on how many SoS configurations are stored. The more SoS configurations are stored in the initial knowledge base at design time, the higher the probability of getting a similar case to the sensed context. Therefore, the higher the *NDP* used, the better the quality of the retrieved adaptation strategies serving as seeds. If the *NDP* is 2, then 8 SoS configurations are stored in the knowledge base (C_1:(25, 25, 25), C_2:(25, 25, 50) ... C_8:(50, 50, 50)).

TABLE IV: Experiment Setup

Parameter name	Value
Adaptation cycle	150 ticks
Simulation time	3000 ticks
Running SoS initial configuration	(10, 10, 10)
Expected environmental conditions	(0.3, 3), (0.5, 5)

Table IV contains information on the experimental setup. We simulate 20 adaptation cycles to ensure that all the environmental conditions are covered at least once. We set an initial SoS configuration to (10, 10, 10), which means there are 10 firefighters, 10 helicopters, and 10 ambulances initially. We assume two environmental condition states are expected at design time.

B. Evaluation

RQ1. Can our approach guide the dynamic reconfiguration effectively at run-time?

We use the *NoSeed* term to denote an approach that randomly initializes an initial population for each adaptation cycle and compare it with the our approach. We conduct experiments from a small *NDP* (2) to a large (10) one to show that our approach can effectively guide the dynamic reconfiguration regardless of the *NDP*. For evaluation of the **RQ1**, we concurrently run the *NoSeed*. Our approach under the same environmental condition changes during 20 adaptation cycles for each *NDP* setting, and we then report a sum of utility values of selected adaptation strategies for 20 adaptation cycles. We report the average values of the experimental results for 30 runs to assure the generality of the experimental results. We also perform the Wilcoxon signed-rank test to a pair of the sum of utility values for each *NDP* setting to show whether a statistically significant difference exists. If a p-value is less than 0.05, then there is a statistically significant difference between the two samples.

TABLE V: Sum of utility values of selected adaptation strategies for small (2), medium (5), and large (10) *NDP* and *p-values*.

NDP	Approach	Sum of utility values	p-value
2	OurApproach	1598.06	< 0.01
	NoSeed	1576.39	
5	OurApproach	1591.40	< 0.01
	NoSeed	1568.24	
10	OurApproach	1594.81	< 0.01
	NoSeed	1574	

Table V contains the sums of utility values and *p-value*s. The sum of utility values in our approach is better than that of the *NoSeed* approach for all *NDP* cases, and all the *p-value*s are less than 0.01. Therefore, the adaptation strategies generated by our approach can effectively guide the dynamic reconfiguration better than the approach generating adaptation strategies from scratch.

RQ2. Is investing more computational time to construct an initial knowledge base at design time meaningful?

To evaluate the **RQ2**, we conduct experiments for various *NDP* settings under the same environmental condition changes for 20 adaptation cycles. We calculate the sum of utility values and the *p-value* to show that a statistically significant difference exists.

Table VI contains the *p-value* for the smallest *NDP* setting (2) versus the various *NDP* settings and the initial knowledge base construction time. When the *NDP*s are 4 and 6, there is no significant difference for the sums of utility values. In the case of *NDP* 8, we can observe a weak significant difference (*p-value* < 0.1). In the case of the *NDP* 10, we can observe a significant difference between the cases with 10 and 25 cases.

TABLE VI: *p-value* for the sums of utility values for the smallest (2) versus the sums of utility values for various *NDP*

NDP	Sum of utility values	p-value	Construction time
2	1588.50	*None*	3 min.
4	1595.71	0.165	21 min.
6	1597.95	0.404	65 min.
8	1597.64	0.078	176 min.
10	1600.79	0.031	305 min.
12	1598.98	0.106	594 min.

This is because the more SoS configurations are stored at design time by consuming more computation time, the higher the chance of retrieving adaptation strategies whose context information is similar to the current context. However, there is no significant difference in the case of *NDP* 12. This is because a more similar case could be retrieved when we use a low *NDP*, given that a high *NDP* might not include the configuration of a low *NDP*. Therefore, an optimal *NDP* value must be found to improve the quality of the adaptation strategy.

We can thus conclude that investing computation time to construct a knowledge base is meaningful because not only is there a significant difference, but consuming lots of computation time at design time is also affordable.

VI. Discussion

Discussion of the Scalability The insight of this approach is to take a run-time overhead at design time because a design-time overhead is affordable compared to a run-time overhead. However, if too much time has to be spent to construct the initial knowledge base, then the timing of a system launch could also be delayed. In our computing resources, it took 3 min. for the case where the *NDP* was 2 and 594 min. for the case where the *NDP* was 12. However, because a large *NDP* does not always mean a better result, finding a suitable *NDP* value in the target SoS is necessary.

Limitations Some limitations exist in our approach. First, the scenario used in this study is hypothetical. Even if we generate this scenario by referring to many MCI documents, many real situations or cases could have been abbreviated. Even if we have no choice but to use the hypothetical scenario because there is no benchmark SoS, a more detailed scenario is still necessary. Second, we not only used one search algorithm, but we also adjusted the parameters by trying one by one within the frequently used parameters in other papers. Nonetheless, there might be better search algorithms and parameter values. In other words, to further improve the quality of adaptation strategies, additional experiments with various search algorithms are necessary.

VII. Conclusion

As the SoS will be deployed in an environment where external environmental conditions dynamically change, it is important to support the dynamic reconfiguration of the SoS to have the SoS sustain its goal achievement by adapting environmental changes. To facilitate adaptation strategy generation process at run-time, we used the *seeding* methodology that initializes the initial population of the stochastic search algorithm as previously known solutions. The SoS manager could utilize generated adaptation strategies when dynamic reconfiguration is necessary. Through experimental results, we demonstrated that our approach can generate effective adaptation strategies at run-time better than the approach that does not use the knowledge base. The generated adaptation strategies could help the SoS manager to guide the dynamic reconfiguration. In the future, we are planning to apply the proposed approach to other SoS to generalize our results.

Acknowledgment

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2021-2020-0-01795) and (No. 2015-0-00250, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System) supervised by the IITP(Institute of Information & Communications Technology Planning & Evaluation).

References

[1] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: basic concepts, model-based techniques, and research directions," ACM Computing Surveys (CSUR), vol. 48, no. 2, pp. 1–41, 2015.

[2] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," Computer, vol. 37, no. 10, pp. 46–54, 2004.

[3] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: a probabilistic model checking approach," in Proceedings of the 2015 10th joint meeting on foundations of software engineering, 2015, pp. 1–12.

[4] A. Pandey, G. A. Moreno, J. Cámara, and D. Garlan, "Hybrid planning for decision making in self-adaptive systems," in 2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO). IEEE, 2016, pp. 130–139.

[5] Z. Coker, D. Garlan, and C. Le Goues, "Sass: Self-adaptation using stochastic search," in 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. IEEE, 2015, pp. 168–174.

[6] I. Lanese, A. Bucchiarone, and F. Montesi, "A framework for rule-based dynamic adaptation," in International Symposium on Trustworthy Global Computing. Springer, 2010, pp. 284–300.

[7] C. Kinneer, Z. Coker, J. Wang, D. Garlan, and C. L. Goues, "Managing uncertainty in self-adaptive systems with plan reuse and stochastic search," in Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems, 2018, pp. 40–50.

[8] D. Beauquier, "On probabilistic timed automata," Theoretical Computer Science, vol. 292, no. 1, pp. 65–84, 2003.

[9] M. D. Schmidt and H. Lipson, "Incorporating expert knowledge in evolutionary search: a study of seeding methods," in Proceedings of the 11th Annual conference on Genetic and evolutionary computation, 2009, pp. 1091–1098.

[10] C. T. Born, S. M. Briggs, D. L. Ciraulo, E. R. Frykberg, J. S. Hammond, A. Hirshberg, D. W. Lhowe, and P. A. O'Neill, "Disasters and mass casualties: I. general principles of response and management," JAAOS-Journal of the American Academy of Orthopaedic Surgeons, vol. 15, no. 7, pp. 388–396, 2007.

[11] T. Chen, M. Li, and X. Yao, "On the effects of seeding strategies: a case for search-based multi-objective service composition," in Proceedings of the Genetic and Evolutionary Computation Conference, 2018, pp. 1419–1426.

[12] D. Hadka, "Moea framework-a free and open source java framework for multiobjective optimization. version 2.11," URL http://www. moeaframework. org, 2015.