Quality-driven Formal Modelling of the Travel Planner Application

Fatma Kachi[1], Chafia Bouanaka[1]

[1]*LIRE Laboratory, University of Constantine 2-Abdelhamid Mehri*

Abstract

Service-based systems (SBSs) have been widely used in recent years, however ensuring adequate service quality is a significant challenge. To that end, managing the properties of service-based systems through dynamic service selection is primordial. It is an important research problem in SBSs, which aims to select proper services to meet user requirements. However, the dynamics of these systems make their design extremely difficult and leading engineers to pursue new solutions and approaches to model and maintain their Quality of Services. Quality-driven self-adaptation is a promising strategy for managing such complexity while also allowing underlying software systems to accomplish functional and/or quality of service requirements by autonomously adapting their behavior at runtime. In this regard, we defined a formal approach for modeling quality-driven self-adaptive systems and ensuring that quality objectives are constantly met in a previous work. In this paper, we aim to apply the proposed approach to a concrete service-based system, the Travel Planner application, to assess the scalability and generality of our approach, and show its effectiveness.

Keywords

Formal methods, HLPNs and PPNs, Quality-driven Self-adaptive systems, Uncertainty, Travel Planner application.

1. INTRODUCTION

Service-based Systems [1] are highly dynamic software systems composed of several web services. SBSs are widely used in e-commerce, online banking, e-health and many other applications. In these systems, services offered by third-party providers are dynamically composed into workflows delivering complex functionality [2]. Unlike other types of systems, service-based systems rely on service providers to ensure that their web services meet the agreed quality of service. Providing adequate service quality is a crucial challenge and must be monitored in the various activities of the service-based system lifecycle. An exemplar of these SBSs is the Travel Planner Application (TPA) [3, 4] that provides services to search for flights, tourist attractions, accommodation arrangements, and rental cars or bicycles. All these services are interdependent, as the user cannot benefit from a service without having successfully benefited from the service that precedes it, with the exception of the attraction information service that is independent of the rest of the services. Due to this close interdependence between services, the failure of one

RIF'21: The 10th Seminary of Computer Science Research at Feminine, March 08, 2021, University of Constantine 2-Abdelhamid Mehri, Algeria

fatma.kachi@univ-constantine2.dz (F. Kachi); chafia.bouanaka@univ-constantine2.dz (C. Bouanaka)
0000-0001-7714-4246 (F. Kachi); 0000-0003-1746-834X (C. Bouanaka)

service leads to the suspension of the whole system. For example, a failure in the flight booking service systematically leads to the impossibility of benefiting from the remaining services, as hotel reservations and car rentals.

In order to satisfy the user demands and to obtain an optimal functioning of the system, it is necessary to ensure a continuous satisfaction of the quality of service, i.e., at least service availability. Quality of service (QoS) refers to the non-functional characteristics of a service, such as price, response time, reliability and availability [5, 6, 7, 8]. Besides, users of the Travel Planner Application may have different budget limits and different response time requirements for accessing the service. As such, the service needs to ensure that the demands of different users are properly met. In other words, if a service fails, it needs to be replicated quickly. Therefore, quality-driven self-adaptation dynamically changes the system behaviour.

In a previous work [9], we have defined a formal model for quality-driven decision-making in Self-Adaptive Systems (SASs). We mainly leveraged a MAPE-K [10] loop-based model, combining high-level Petri nets (HLPNs) and plausible Petri nets (PPNs), for the design of quality-driven self-adaptive systems under uncertainty. In this model, the adaptation logic is achieved through the adaptation actions that change the system configuration by updating its properties. In the case of SBS, the adaption logic is limited to a simple action, which consists of replicating the failed service.

Our aim in the present paper is twofold: to show the effectiveness of the proposed model for guaranteeing a continuous satisfaction of the quality objectives of the Travel Planner Application, and to establish the genericity of the model through its application in various fields. The model allows achieving self-adaptation and ensuring the optimal functioning of the Travel Planner Application. It uses HLPNs to define the data flows and quantify the quality criteria considered for each service, and adopts PPNs to determine which service will replace the faulty one through the concept of decision plausibility with respect to the adaptation strategy envisaged for this system.

The remainder of the paper is structured as follows. Section 2 introduces the Travel Planner Application (TPA), and its expected qualities. Section 3 briefly recalls the main concepts used in this work. Section 4 formally describes the TPA modelling. Section 5 discusses and compares the proposed TPA model regarding the existing ones. Finally, Section 6 concludes the paper.

2. TRAVEL PLANNER APPLICATION RUNNING EXAMPLE

The TPA is composed of the following five services [11]:

- Attraction information service, which searches for tourist attractions.
- Flight booking service, which searches for available flights.
- Hotel booking service, which seeks hotels and makes reservations.
- Bicycle rental service, for renting bikes.
- Car rental service, which allows renting cars.

In this example, searching for attractions may be done in parallel with a flight and an accommodation booking. Once the search and booking process is completed, the car or bicycle rental service will be invoked. The quality criteria considered for each service are: cost, reliability, and availability [8].

- Price: the execution price is the amount of money that a service requester has to pay to execute it.
- Reliability: is the probability that a request is correctly responded within the maximum expected time frame. The reliability value is computed from historical data concerning past invocations using the expression $q_{rel}(s) = N_c(s)/K$, where $N_c(s)$ is the number of times that the service s has been successfully delivered within the maximum expected time frame, and K is the total number of invocations [8].
- Availability: is the probability that the service is accessible. the availability value of a service s is computed using the following expression $q_{av}(s) = T_a(s)/\theta$, where T_a is the total amount of time in which service s is available during the last θ seconds [8].

3. BACKGROUND

Petri nets have been initially proposed to model the behavior of a dynamic system with discrete events [12]; they have then undergone several extensions and variants [13, 14, 15, 16] to cover more concerns in system modelling and analysis. In what follows, first, we present two types of Petri nets to be used in this work, high-level Petri nets (HLPNs) and plausible Petri nets (PPNs). Then, we define the concept of uncertainty.

3.1. High-Level Petri Nets (HLPNs)

HLPNs are a well-defined semi-graphical technique for the specification, design and analysis of systems. An HLPN is made up on a set of nodes (i.e. places and transitions) and a set of arcs connecting places to transitions and vice-versa as in an ordinary Petri net but is extended in the following way:

- each place is associated with a place type and can contain a collection of tokens corresponding to that place type.
- Transitions are dotted with boolean expressions called guards.
- Additionally, arcs are inscribed with expressions called arc annotations. Whenever an expression evaluates to true, a multiset of tokens is produced in the output places of the corresponding transition according to arc's weights and types.

3.2. Plausible Petri Nets (PPNs)

The combination of the principles of Petri nets with the foundations of information theory resulted in a new model for Petri nets, called plausible Petri nets (PPNs) [14, 17]; which are a hybrid variant of Petri nets composed of two types of places and transitions, namely, symbolic and numerical, in order to describe both discrete and continuous behaviours of a system. In the symbolic subnet, the discrete behavior is described using regular tokens, while in the numerical subnet, continuous or numerical behavior is described with tokens that carry information about the states of variables. A state of information about a given variable $x \in \chi$ is the probability density function (PDF) of x over χ [18], where χ is the state space of a stochastic variable x. For more details on PPNs, the reader is referred to [19].

The main feature of PPNs resides in their efficiency to jointly consider the evolution of a discrete event system together with uncertain information about the system state using states of information [14]. They provide a mapping between the possible numerical values of a state variable and their relative plausibility, hence giving greater versatility for representing uncertain knowledge using a more principled approach [19].

3.3. Uncertainty

Uncertainty in a software system is defined as the circumstances in which the behavior of the system deviates from expectations due to the dynamics and unpredictability of various factors within these systems [20]. Unpredictable changes and missing knowledge about the environmental changes during system deployment refers to epistemic uncertainty. Researchers link uncertainty in SASs to decision making. Therefore, it means the lack of sufficient information or knowledge to make certain adaptation decisions.

4. TRAVEL PLANNER APPLICATION MODELLING

This section is dedicated to TPA formal modeling. First, we briefly describe the SAS formal model [9]. Then, we present the necessary parameters to be identified to apply the model. Finally, we present the resulting TPA formal model and illustrate it through an execution scenario.

4.1. A Petri Net-based formal model for SASs

We defined a formal model that is a combination of HLPNs and PPNs with the goal of mitigating uncertainties to ensure the continuous satisfaction of SAS QoSs and assisting the decision-making process while selecting the proper adaptation plans (PPNs). It is made up of two layers: the monitored one, which is modeled by HLPN, and the control one, composed of the emulator for encoding and emulating the monitored layer to interact with the API and modeled as a HLPN. The API is made up of a set of read/write primitives represented by the HLPN transitions, and the managing system, which is a MAPE-K loop-based model that combines HLPN and PPN. HLPNs contribute in the definition of data flows by using expression and annotation principles to quantify the observed qualities among the model's various components. They allow the model to express more sophisticated data structures in a dynamic system. PPNs are used to help and improve the decision-making process in presence of uncertainty, allowing for the most appropriate adaption plans to be determined using the principle of decision plausibility.

4.2. Model parameters

The Petri Net-based formal model proposed in [9] is generic and can be applied in several areas and on different cases. However, to achieve our objectives and be able to model a quality-driven SAS, we need to identify the necessary parameters of the control layer by performing the following steps (see Figure 1):

1. Identify the overall qualities of the system;
2. Determine characteristics that allow quantifying system qualities as well as the contextual uncertainties affecting the system behavior;
3. Model the managed system with the use of HLPN;
4. Parameterize the MAPE-K loop used in the control layer. This later is generic and parameterized by: the functional system or layer, the quality objectives and the adaptation actions.

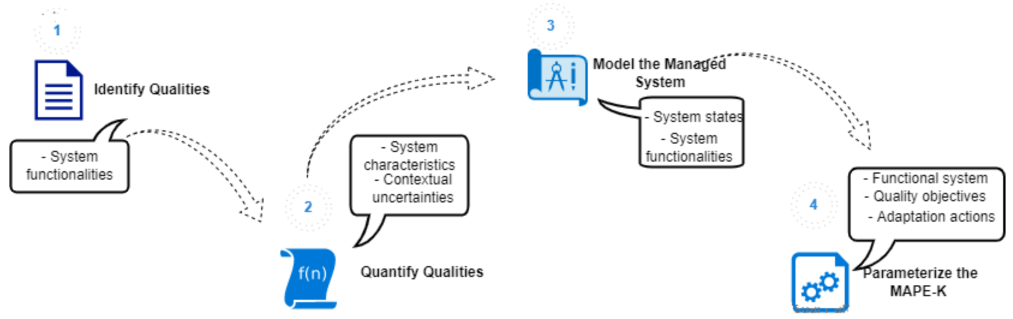

1
Identify Qualities
- System functionalities
2
- System characteristics - Contextual uncertainties
Quantify Qualities
3
Model the Managed System
- System states - System functionalities
4
- Functional system - Quality objectives - Adaptation actions
Parameterize the MAPE-K

Figure 1: Modelling a quality-driven SAS process.

The monitored layer represents the model of the managed system, it is defined in step 3. Generally, this modelling describes the overall functioning of the system.
The quality objectives define requirements that the system aims to maintain and ensure, such as reliability, availability, security and performance. These qualities are quantified using properties already defined in step 2. The adaptation actions are the possible solutions and configurations that restore the system quality and ensure the continuous satisfaction of the quality objectives.

4.3. The TPA formal model

The running example is a service-based system, which means that its quality depends on the services it provides. Kotler [21] suggested that a service is any activity or benefit that one party can give to another that is essentially intangible and does not result in the ownership of anything. According to this definition and the functioning of the TPA, we model a service as a HLPN place where tokens represent service instances. A service-QoS is determined regarding its cost, reliability value and availability rate. In this example, we adapt the TPA by selecting the more plausible service on the basis of a comparison of the QoS values of all services, since the quality objectives are defined via thresholds in our approach. Besides, we focus on modelling the system states/context elements and actions to be performed by the system, but in this case, we are more interested with the system functioning process. The TPA architecture is given in Figure 2.

The monitored layer model is organised in 4 phases: flight reservation, hotel reservation, tourist attractions search, and rental phase. These phases enclose all services provided by the TPA with the rental phase containing two services: car rental service and bike rental service. The control layer monitors the service failures and adapts the system using the API primitives, which are used inside the MAPE-K control loop to simulate the sensing and actuating actions upon the monitored layer. All services are represented in the knowledge zone and used by the MAPE elements. For readability raisons, we present the model of each layer separately.

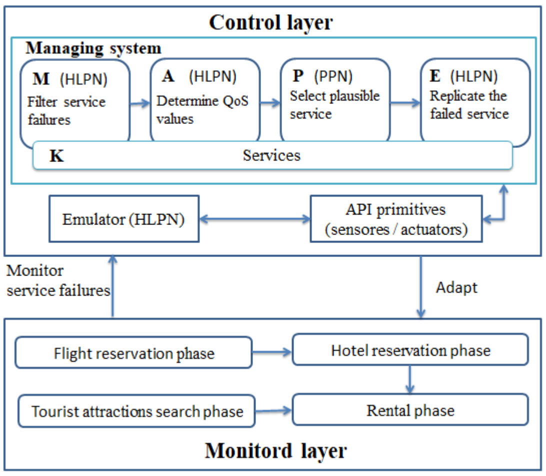


Figure 2: The TPA architecture.

4.3.1. Monitored layer

Figure 3 represents the monitored layer model of the TPA, which is organised in 4 phases: *flight reservation, hotel reservation, tourist attractions search*, and *rental phase*. Places represent the states of user requests; transitions represent user actions and services failure (ex: *fialedSearchFlight, failedSearchAttraction, failedSearchHotel…*). The user may initiate the flight reservation phase, tourist attractions search phase, or both. Whenever he requests the flight reservation service his request may be treated successfully by firing the transition *flightReservation* producing a token that contains the flight information in the place *flightReserved*, this will enable him to move to the next phase; or the service may fail and its identifier is passed to *serFailed* place. This place contains the identifier of the failed services (*flight, hotel* and *attraction service*). Service functioning is supervised by the monitor through the *rentFailed* place, which contains the identifier of the failed car rental service and bike rental service, in order to perform adaptation. After a successful flight reservation phase, the user can start the 2^{nd} phase by requesting the hotel reservation service, which may also fail or be successfully executed. The tourist attractions search phase allows users to consult tourist attractions; finishing this phase successfully produces a token containing information about the searched tourist attraction in the *rentingService* place. This place also contains tokens representing the reserved hotel information. At the end, the user may execute the rental phase to rent a car or a bicycle whenever the accommodation arrangements have been made through the hotel service. In addition, he can rent them after requesting the attraction service directly.

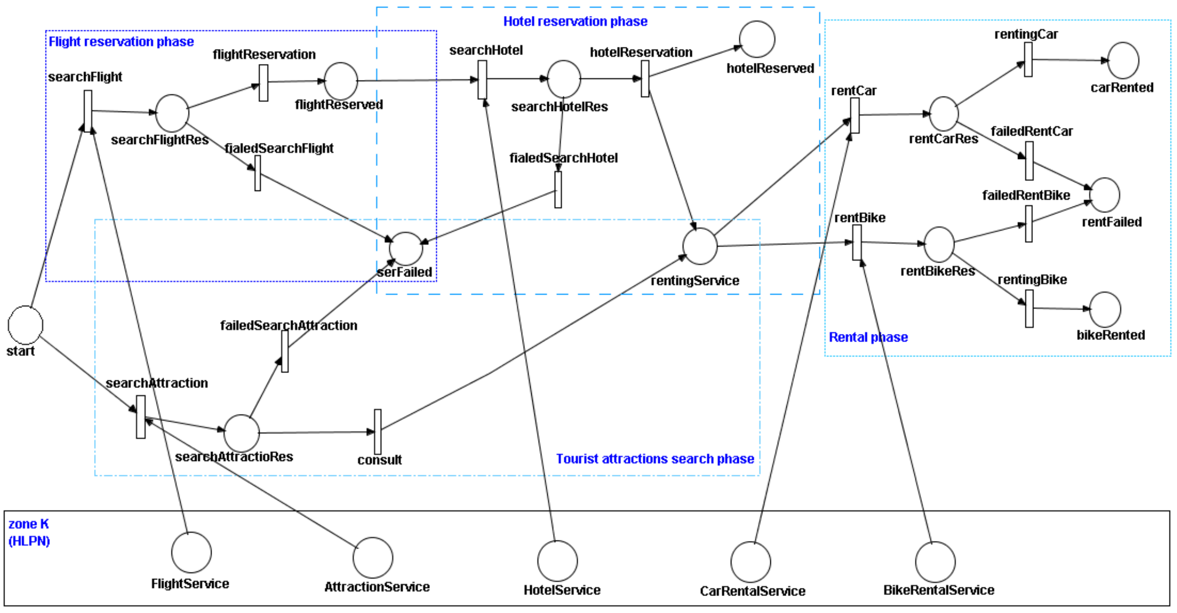


Figure 3: Travel planner application model.

4.3.2. Control layer

In this section, we detail the TPA adaptation logic represented by the managing system of Figure 2; whose model is given in Figure 4 where the black transitions represent the cross-zone ones, and the blue places and transitions represent the plausible ones. We are limited in this example to the following adaptation strategies: if the flight booking service or the hotel reservation service fails, the corresponding service will be replaced by the service with a higher reliability value; if the attraction information service fails, it will be replaced by the service with the lower cost value.

1. **Knowledge**

Since TPA is a service-based system, its services are represented by places in the knowledge base (*HotelSrvices, FlightServices, AttreactionServices*...). Each service instance is represented as a token in the corresponding place. Each token is a tuple containing the necessary data to identify the service, which are the service identifier s, its cost, the number of times that the service s has been successfully delivered (N_c), the number of invocations (K), and the total amount of time in which it is available during the last θ seconds (T_a).

2. **Monitor**

Since a service may fail at any time while treating a user request, the monitor element is responsible of the recovering of the failed service information by monitoring the place *serFailed* using the transition *getFailedService*, which is a read primitive of the API. Then, it filters the result to determine the failed service data and passes them to the analyser element.

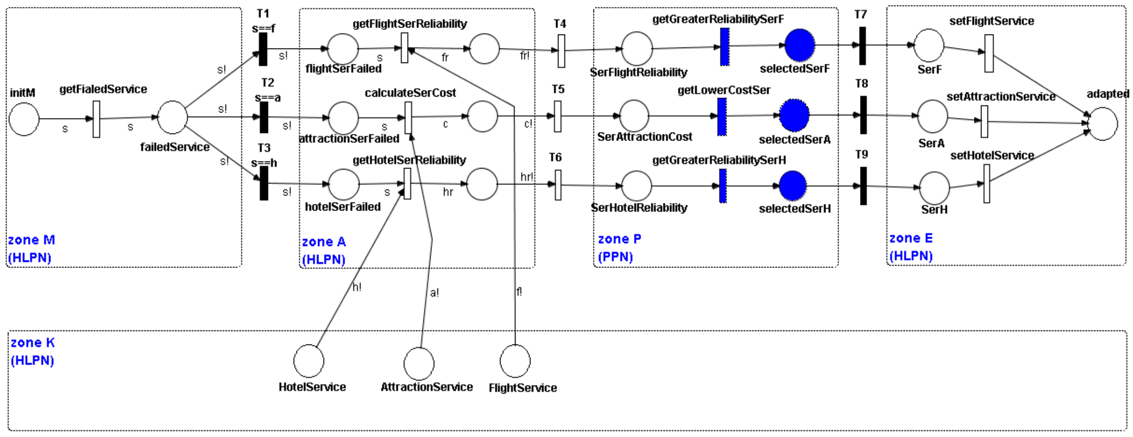


Figure 4: Travel planner application adaptation logic.

3. Analyser

It calculates the reliability of the service in the case of the flight reservation/hotel reservation services failure, by firing transitions *getFlightSerReliability* and *getHotelSeReliability*, respectively, and calculates service cost in the case of attraction information service failure; these values are calculated using formulas presented in Section 2. The calculated values are transmitted to the planner element.

4. Planner

Its task is to select the proper service instance from the existing ones with regard to adaptation strategy requirements. It uses the concept of plausibility to select the most appropriate service. For example, the transition *getGreaterReliabilitySerF* allows determining the flight service with the greatest reliability value. When the planner task selects the best service, the executor is triggered.

5. Executor

It updates the failed service with the selected one by firing transition according to the service (*setFlightService, setAttractionService, or setHotelService*), representing the write primitive of the API. In fact, it updates the corresponding place marking.

4.4. Execution Scenario

The proposed model is simulated and validated using the extended PNemu[1] framework; where we defined a new class "HLPN" to model the managed system and its environment by an HLPN; we have modified the emulator class to be able to emulate models specified as HLPN; then we have redefined some primitives of the API to capture the HLPN concepts, such as *getTokens* and *setTokens* primitives.

[1]PNemu has been released as an open source software, available at https://github.com/SELab-unimi/pnemu.

Table 1
TPA services information.

Service	Identifier	Cost	$N_c(s)$	K	T_a
Attraction service	AS1	18	21	25	250
	AS2	16	35	40	192
	AS3	20	10	16	180
	AS4	14	19	20	290
Flight service	FS1	15	20	30	280
	FS2	17	47	50	295
	FS3	13	16	20	199
	FS4	25	15	25	245
Hotel service	HS1	15	30	33	285
	HS2	19	16	16	275
	HS3	21	33	40	264
	HS4	25	19	25	235

We assume an initial configuration of the system containing 4 instances of each service, which are attraction information service, flight reservation service, hotel reservation service, bicycle rental service, and car rental service. We specify each instance by a token of type tuple containing service instance information; see Table 1. We have considered the same parameters as presented in [3], which are service identifier, cost, the amount of times that the service has been successfully delivered (N_c), the number of invocations (K), and the total amount of time in which it is available during the last θ seconds (T_a); θ is defined by system administrator. Table 1 is limited to services that are modelled in Figure 4; some values are retrieved from the files used in [11] and others are randomly chosen; we assume that θ = 300 seconds. A part of the HLPN instance representing the monitored layer is given in Figure 5; it corresponds to the flight reservation phase, the presence of a token in place *searchFlightRes* means that the list of flights displayed in the user interface are given by this service (FS4).

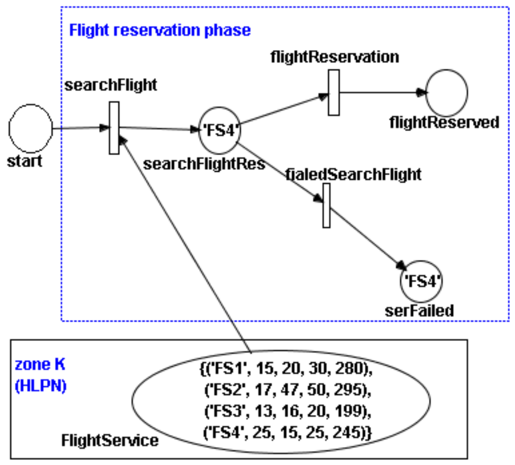
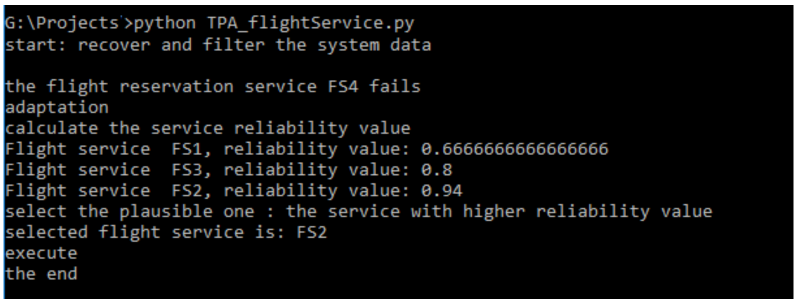


Figure 5: Flight reservation phase model instance.

We assume that the a user requests the flight reservation service, but unfortunately the later fails (FS4). To maintain and ensure a continuous satisfaction of the quality objectives, the system tries to replace it by another service with the higher reliability value, thus the analyser element calculates the reliability value for each instance of the flight reservation service. Then, it sends the results to the planner element in order to select the plausible service, which is the service with the higher reliability value, service FS2 for instance. The selected service is resumed by the executor using the write primitive of the API; it updates the place marking by the new token representing the selected service. Simulation results are illustrated in Figure 6.



```
G:\Projects >python TPA_flightService.py
start: recover and filter the system data

the flight reservation service FS4 fails
adaptation
calculate the service reliability value
Flight service  FS1, reliability value: 0.6666666666666666
Flight service  FS3, reliability value: 0.8
Flight service  FS2, reliability value: 0.94
select the plausible one : the service with higher reliability value
selected flight service is: FS2
execute
the end
```

Figure 6: Simulation result of self-adaptive travel planner application.

5. DISCUSSION

In recent years, some studies have modelled the TPA using different methods and tools. In [3], TPA is modelled by Agent Flow's where authors are interested with cost analysis, execution duration, reliability and reputation quality; they developed a non-formal framework to execute the proposed model, which is specific to SBSs and doesn't deals with uncertainty. In [22], a formal model of TPA is proposed, it is based on parameterized Discrete-Time Markov Chains (DTMC). In this model, authors are interested with analyzing cost and reliability qualities. The developed framework for model execution is also specific to SBSs and doesn't deal with uncertainty. The model is also used in [11] and implemented with lotus@runtime, which is dedicated to SAS, but its applicability has been demonstrated by SBSs. The models proposed in these works do not separate the system from its adaptation logic, this makes the models application-specific and not applicable to other systems, contrary to our model. These shortcomings lead to the limitations of these model applicability and the difficulty of generalizing them to study different cases and in different fields.

In this paper, we applied our formal model to TPA; it combines HLPNs with PPNs to model both self-adaptation and manage uncertainties. The generality of our model is confirmed by this case study; all we have to do is to specify the parameters of the control layer. The model is used to represent several applications in various fields, such as SBSs and aircraft planning in [9]. Furthermore, it takes charge of uncertainty and manages it at different levels of the decision-making process. The above discussion is summarized in Table 2.

Table 2
TPA modelling synthesis.

Model	Formal	Executability	uncertainty	Generality	QoS
Agent Flow's	-	+	-	-	Cost, Reliability, Availability, Reputation
DTMC	+	+	-	-	Cost, reliability
Our model	+	+	+	+	Cost, reliability

6. CONCLUSION

We proposed a formal approach for modeling and analyzing quality-driven self-adaptive systems that evolve under uncertainty while still preserving and assuring the continuous satisfaction of an acceptable quality of service. In this paper, we demonstrate the generality and effectiveness of our approach. The Travel Planner Application case study is a real-world problem that has been modelled with our approach. The findings show how HLPN can be used to model and gather the monitored data to facilitate the autonomous and adaptive service replication decision-making. This example demonstrates how service-based systems can be adapted using a new formalism such as our Petri net-based approach. As a result, our approach is generic and adaptable to any system.

As future work, in the short term we plan to verify some properties of the TPA model. In the middle term, we intend to create a model generator that converts HLPN model to Markov chains model so that we can use model checkers in conjunction with AI techniques to verify system properties and avoid the combinatorial explosion problem.

References

[1] Marc Oriol, Xavier Franch, and Jordi Marco. "Monitoring the service-based system lifecycle with SALMon". In: *Expert Systems with Applications* 42.19 (2015), pp. 6507–6521. DOI: 10.1016/j.eswa.2015.03.027.

[2] Danny Weyns and Radu 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*. IEEE. 2015, pp. 88–92. DOI: 10.1109/SEAMS.2015.27.

[3] Liangzhao Zeng et al. "QoS-aware middleware for web services composition". In: *IEEE Transactions on software engineering* 30.5 (2004), pp. 311–327. DOI: 10.1109/TSE.2004.11.

[4] Liangzhao Zeng et al. "An agent-based approach for supporting cross-enterprise workflows". In: *Proceedings 12th Australasian Database Conference. ADC 2001.* IEEE. 2001, pp. 123–130. DOI: 10.1109/ADC.2001.904473.

[5] Angus FM Huang, Ci-Wei Lan, and Stephen JH Yang. "An optimal QoS-based Web service selection scheme". In: *Information Sciences* 179.19 (2009), pp. 3309–3322. DOI: 10.1016/j.ins.2009.05.018.

[6] Daniel A. Menasce. "QoS issues in web services". In: *IEEE internet computing* 6.6 (2002), pp. 72–75. DOI: 10.1109/MIC.2002.1067740.

[7] Justin O'Sullivan, David Edmond, and Arthur Ter Hofstede. "What's in a Service?" In: *Distributed and Parallel Databases* 12.2 (2002), pp. 117–133. DOI: 10.1023/A:1016547000822.

[8] Liangzhao Zeng et al. "Quality driven web services composition". In: *Proceedings of the 12th international conference on World Wide Web*. 2003, pp. 411–421. DOI: 10.1145/775152.775211.

[9] Fatma Kachi, Chafia Bouanaka, and Souheir Merkouche. "A Formal Model for Quality-Driven Decision Making in Self-Adaptive Systems". In: *arXiv preprint arXiv:2012.01651* (2020). DOI: 10.4204/eptcs.329.5.

[10] Autonomic Computing et al. "An architectural blueprint for autonomic computing". In: *IBM White Paper* 31.2006 (2006), pp. 1–6. DOI: 10.1021/am900608j.

[11] Davi Monteiro Barbosa et al. "Lotus@ runtime: A tool for runtime monitoring and verification of self-adaptive systems". In: *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE. 2017, pp. 24–30. DOI: 10.1109/SEAMS.2017.18.

[12] Carl Adam Petri. "Kommunikation mit automaten". In: (1962).

[13] International Standard ISO/IEC 15909. "High-level Petri Nets - Concepts, Definitions and Graphical Notation". In: *Final Draft International Standard ISO/IEC* 15909.4 (2002), pp. 1–43. ISSN: 14324350. DOI: 10.1007/BF02679450.

[14] Manuel Chiachıo et al. "A new paradigm for uncertain knowledge representation by Plausible Petri nets". In: *Information Sciences* 453 (2018), pp. 323–345. ISSN: 00200255. DOI: 10.1016/j.ins.2018.04.029.

[15] John C Baez and Jade Master. "Open petri nets". In: *Mathematical Structures in Computer Science* 30.3 (2020), pp. 314–341. DOI: 10.1017/S0960129520000043..

[16] Mohammed Taleb-Berrouane, Faisal Khan, and Paul Amyotte. "Bayesian Stochastic Petri Nets (BSPN) - A new modelling tool for dynamic safety and reliability analysis". In: *Reliability Engineering and System Safety* 193 (2020), p. 106587. ISSN: 09518320. DOI: 10.1016/j.ress.2019.106587.

[17] Manuel Chiachıo et al. "An information theoretic approach for knowledge representation using Petri nets". In: *FTC 2016 - Proceedings of Future Technologies Conference*. IEEE, 2017, pp. 165–172. ISBN: 9781509041718. DOI: 10.1109/FTC.2016.7821606.

[18] Guillermo Rus, Juan Chiachıo, and Manuel Chiachıo. "Logical inference for inverse problems". In: *Inverse Problems in Science and Engineering* 24.3 (2016), pp. 448–464. DOI: 10.1080/17415977.2015.1047361.

[19] Manuel Chiachıo et al. "Plausible Petri nets as self-adaptive expert systems: A tool for infrastructure asset monitoring". In: *Computer-Aided Civil and Infrastructure Engineering* 34.4 (2019), pp. 281–298. DOI: 10.1111/mice.12427.

[20] Sara Mahdavi-Hezavehi, Paris Avgeriou, and Danny Weyns. “A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements”. In: *Managing Trade-Offs in Adaptable Software Architectures*. Elsevier, 2017, pp. 45–77.

[21] Deepak R. Kanthiah Alias and Jeyakumar S. *Marketing management*. Educreation Publishing, 2019.

[22] Radu Calinescu, Kenneth Johnson, and Yasmin Rafiq. “Developing self-verifying service-based systems”. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2013, pp. 734–737. DOI: 10.1109/ASE.2013.6693145.