Analysis of Autonomic Service Oriented Architecture

Muhammad Agni Catur Bhakti

Informatics Engineering Department, University of Pancasila
Srengseng Sawah, Jagakarsa, Jakarta 12640, Indonesia
m.agni.cb@gmail.com

Abstract— **Service-Oriented Architecture (SOA) enables composition of large and complex computational units out of the available atomic services. However, implementation of SOA, for its dynamic nature, could bring about challenges in terms of service discovery, service interaction, and service composition. SOA may often need to dynamically re-configure and re-organize its topologies of interactions between the web services because of some unpredictable events, such as crashes or network problems, which will cause service unavailability. Complexity and dynamism of the current and future global network systems require service architecture that is capable of autonomously changing its structure and functionality to meet dynamic changes in the requirements and environment with little human intervention. In this paper, formal models of a proposed autonomic SOA framework are developed and analyzed using Petri Net. The results showed that SOA can be improved to cope with dynamic environment and services unavailability by incorporating case-based reasoning and autonomic computing paradigm to monitor and analyze events and service requests, then to plan and execute the appropriate actions using the knowledge stored in knowledge database.**

Keywords— **Service Oriented Architecture, autonomic computing, case-based reasoning, formal model, Petri Net**

I. INTRODUCTION

As the development of internet technologies has enabled an access to many types of services over the web, networked and distributed systems (providing resources, services, etc.) are nowadays gaining an increasing importance and demand. Hence, the scale and complexity of current distributed systems are also increasing and showing high dynamism [1]. Furthermore, on the base of existing services, large distributed computational units can be built by composing complex compound services out of simple atomic ones [2]. This type of concept and architecture is called Service-Oriented Computing (SOC) and Service-Oriented Architecture (SOA) respectively.

Service-oriented computing is an emerging computing paradigm that utilizes services as the basic constructs to support the development of rapid and easy composition of distributed applications. The visionary promise of SOC is to assemble the application components with little effort into network of services that can be loosely coupled and used to create the flexible dynamic business processes and applications that may span organizational boundaries and computing platforms.

Components of a service-oriented model (data, software, platforms, etc.) should be considered as service that can be used by users through the network, despite of the underlying technologies being used to provide those services. A business process engine can be deployed using service-based integration adapters to access a services based message broker. Service-based business application adapters are used to access several back-end systems, such as databases or legacy systems. The service adapter interface is hence used to unify the interfaces to different kinds of the back-end systems.

Current SOA frameworks offer agility, maintainability, reusability, consistency, efficiency, integration and reduced cost of a service [3]-[5]. Yet, they are still lacking for adaptability and robustness. Schneider et al. [5] stated that technologies and methods are still needed for development of adaptive SOA systems. The results in [6] showed that typical service composition will be complete and correct with an assumption that there are no exceptions or errors occurred from the initiating user to the terminating one. However that is not the case with the current and future complex and dynamic systems.

The work in [1] reported that the scale and complexity of current distributed systems are increasing and showing high dynamism in that the global network systems grow. Future systems also need to be able to cope with unpredictable events that could cause services unavailability, such as crashes or network problems. Therefore, a more robust, more adaptive and autonomous service architecture that can keep up with the dynamic changes in environments and requirements to some extent is required.

An autonomic service oriented architecture based on autonomic computing paradigm [7]-[8] and case-based reasoning (CBR) [9]-[10] has been proposed in [11]–[14]. The autonomic computing paradigm, inspired by the human autonomous nervous system, was proposed as an approach for the development of computer and software systems that are able to manage themselves in accordance with only high-level guidance from administrators. This paradigm has been used in many researches in various domains such as those in [15] and [16] in which the authors adapted autonomic computing paradigm in self-configuration and self-healing software systems.

We incorporated the autonomic computing cycle and case-based reasoning in the proposed framework to introduce learning and adaptability into SOA. The autonomic computing mechanism in SOA will autonomously monitors and analyses service requests, then plans and provides the services. It will also adapt and learn new service profiles leading to better and faster service delivery in the future. The rest of this paper is structured as the following: section 2 elaborates the proposed autonomic SOA framework; section 3 presents the formal model development and analysis of the proposed autonomic SOA; section 4 presents the simulation development of the proposed framework; lastly section 5 presents summary of this paper and direction for future work.

II. PROPOSED AUTONOMIC SOA

Compared to the conventional SOA, the proposed autonomic SOA has additional features which include the addition of autonomic manager and the ability to adapt to changes with the knowledge from a knowledge base. The autonomous SOA will learn and adapt in appropriate ways to solve problems based on the knowledge gained from previous cases, which are stored in the knowledge base, using CBR. It will also be able to suggest services to the users.

Fig. 1 shows the overall architecture of the proposed autonomic SOA that extends a typical SOA infrastructure (i.e. consists of service requestor and service provider) by incorporating the autonomic computing cycle into the business process layer. The architecture is separated into the three tiers:

1. The top that is a presentation tier to provide access to various users through web
2. The mid that is a processing tier to perform and coordinate several jobs and
3. The bottom that is a service / resource tier to enable the utilization of the distributed resources via Web Services.

The service/resource tier refers to service providers in a typical SOA framework. The brokers in processing tier act as service requestors. Here, the functionality of the service registry, by adding a knowledge base as required by the autonomic computing paradigm, is extended. The knowledge base provides the capability to store the previous services profiles (cases) whose features include:

- Name of the service.
- Description of the service.
- The type of service (atomic, composite).
- If the service is a composite service, then the profile will also include profile of the atomic services required to compose the composite service (“ingredients”).
- Where, when, how (sequence) to access (and compose if necessary) the service (“recipe”).

The autonomic computing paradigm is incorporated in the processing tier which has the autonomic manager in it. In the context of autonomic computing paradigm, the autonomic manager will perform the autonomic cycle, i.e. monitoring, analyzing, planning, and executing, which of each is described below.

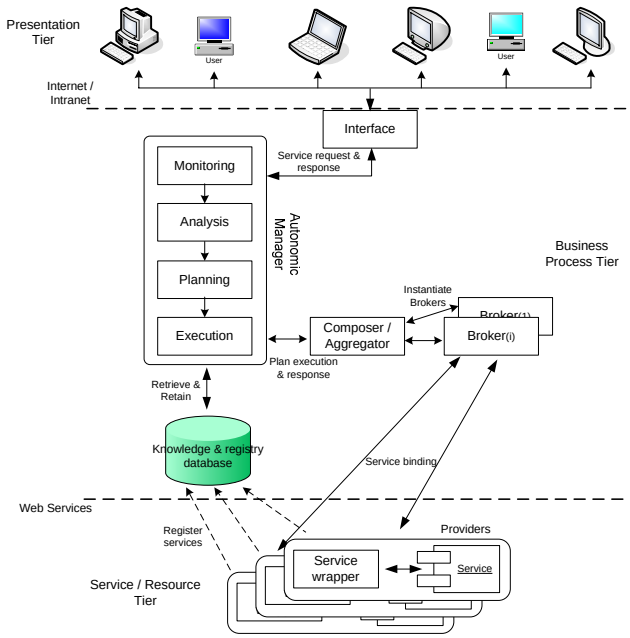

Fig. 1 Architecture of the proposed autonomic SOA framework

A. *Monitoring*

The manager will monitor both its own behaviour and the overall system, including the following:

- The availability of the services
- Addition of new services
- Removal of services
- Request / query from user

A sentinel or monitoring module will provide monitoring services to the SOA elements. Along with service registry, it would provide service discovery. The service monitor continuously monitors the system to detecting and identifying request from user and the status of services. If a service request input is available from user, it will be forwarded into analysis. Then if there is a change in service status, the status of that particular service in knowledge base will be updated. A change in service status will be considered as a new request that will be treated as such (forwarded to analysis module and so forth).

B. *Analyzing*

It means to analyze the requests. The manager will retrieve previous cases from the knowledge base, whose features include description of services, type of service (atomic or composite), their providers, and access to the providers. The cases then will be reused or revised as necessary to provide the (composite) service requested.

Fig. 2 illustrates the adaptation of Case-based Reasoning (CBR) and autonomic computing cycle in the proposed autonomic SOA framework. The analysis process described below is adapted from the CBR cycle (retrieve, reuse, revise, and retain) for adaptive and learning functionality, which include both the analysis and planning processes using the knowledge base as the case base.

1) *Case-Based Reasoning (CBR):* CBR is a process of solving a new problem by remembering a previous similar situation and reusing information and knowledge of that situation [10]. CBR is able to utilize the specific knowledge of previously experienced, concrete problem situations, called 'cases'. In it, a new problem is solved by finding a similar past case, and reusing it in a new problem situation. CBR systems store past experiences as individual problem solving episodes [9]. CBR also refers to an approach to incremental, sustained learning. Since a new experience is retained each time a problem has been solved, CBR comes to be immediately available for future problems. CBR can either mean adapting old solutions to meet new demands, or using old cases to explain new situations, or reasoning from precedents to interpret new situation, or creating equitable solution to a new problem [9]. Kolodner [9] listed the advantages of CBR as the following:

- It allows the reasoner to propose solutions to a problem quickly.
- It allows the reasoner to propose solutions in domains that tare not completely understood by the reasoner.
- It gives the reasoner a means for evaluating solutions when no algorithmic method is available for evaluation.
- Cases are useful in interpreting open-ended and ill-defined concepts.
- Remembering previous experience is useful to help learners to avoid repeating past mistakes.
- Cases help the reasoner to focus on its reasoning on important parts of a problem by pointing out what features of a problem are important ones.

For its benefits, features, and successful implementation in the systems found in the following works, CBR here becomes the chosen method in the analysis and planning processes:

- Montani & Anglano [16] used CBR in developing self-healing software system
- Cheetham [17] and Morgan [18] deployed CBR applications at GE Plastics and General Motors work places respectively
- Manufacturing [19]
- Engineering sales support [20]
- Wireless networks management [21]
- Project management [22]
- Fault diagnosis [23]

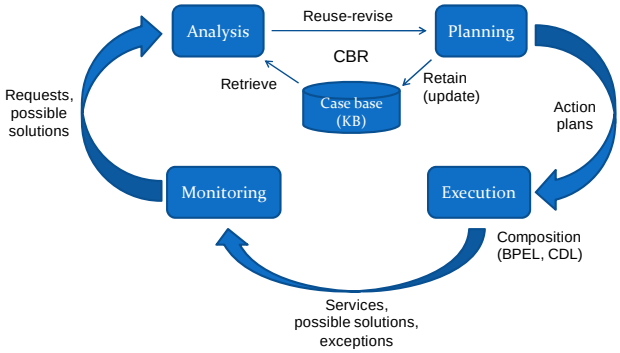

Fig. 2 Adaptation of autonomic cycle and CBR in the proposed autonomic SOA

2) The analysis process: the process is described as the following:

1. Once receiving a request of service, the system starts by first searching for that particular service profile (as represented by a case) in knowledge base / case base. If that particular service profile is available, then it is retrieved for action planning.
2. If there is no service profile of that particular service in the knowledge base, then cases that are having similar properties / features would be retrieved. Various metrics can be used to calculate the similarity distance. For example, the work by [16] used heterogeneous Euclidian-overlap metric (HEOM) [24]. The distance calculation returns a value which is typically in the range of 0..1 with 0 value means zero distance, i.e. x = y.
3. The similar cases found shall be used for action planning (by revising them). The new case afterward will be used for action planning and then added to the knowledge base.
4. If there are no similar previous cases, the monitoring module will search for the composite service in service registry (or search for atomic services that could be composed into the requested service). For scalability, the system should also be able to search in other service registries (e.g. online service registry on the internet or other service ecosystems) if the local service registry does not have the

services needed. The new service profile will then be used for action planning and added (retained) to the knowledge base.

5. The autonomic manager will also suggest other services to the users which are related to the requested services (e.g. other services that are also typically used) based on the previous cases in the knowledge base.

The mechanisms of the CBR in the autonomic SOA are described in the following algorithm:

- Overall CBR mechanism:
  The system will retrieve every record from Knowledge Base (KB) by firstly trying to find exact match of the current case in those records. If an exact match is found, the solution then is forwarded to the next phase, yet if not, the system will select cases that are similar with the current case. The solutions of those selected cases (list of possible solutions) are forwarded to the next phase. However, if there are no similar cases found, the system will search for the service at external / remote service registries.
- Retrieve mechanism:
  Retrieving every record in KB (and put them in an array / list).
- Reuse mechanism:
  Calculate the distance between every record and current case. Find an exact match by comparing every record with the current case (i.e. find the case with zero distances to the current case; because when the distance is 0, it means that it is an exact match). If it is found, then return that record's solution as the current solution.
- Revise mechanism:
  If there is no case with 0 distances, select cases with distances below the distance threshold and save their solutions as a list of possible solution, and forward it to the next phase.
  Eventually, solution that is accepted by users (i.e. used by many users, high usage numbers) will be retained, while other solutions with low usage numbers will be discarded from KB.
- Retain mechanism:
  Record new or updated cases and service status.
  At the end of the retain mechanism, there will be a status update process if there are new cases to be retained.

C. *Planning*

Autonomic manager will plan actions to provide the requested composite service. It plans the suitable actions for the requested service. If it is a composite service, then the action plans will include the following:

- The list of available atomic services needed to compose the required composite service
- Where and how to access the atomic service
- The sequence of accessing the atomic service

It will also update the knowledge base if new action plan is created (or revised from the previous ones) so that these plans can be readily available and prepared faster when the same composite service is re-requested in the future. After receiving the service information from analysis module, the planning module will either create an action plan to invoke the service solution or it will create several action plans of the previous similar cases. The action plan(s) will then be forwarded to execution module.

D. *Executing*

Autonomic manager will execute a plan to provide a requested service, and brokers will assist in interacting and negotiating with the service providers to obtain the required services, including translating messages from the formal messaging protocol of the sender to the formal messaging protocol of the receiver if necessary (in the case where sender and receiver are using different platforms). Upon receiving action plan, the execution module will execute it utilizing the brokers as necessary to interact with service providers.

If the requested service is an atomic service, then the service will be simply provided by the service provider. Meanwhile if it is a composite one, then the autonomic manager will execute the action plan and then provide the composite service, which is by composing the atomic services that can be based on Business Process Execution Language (BPEL) or Choreography Description Language (CDL).

III. FORMAL MODELS

By using Petri Nets which provides further insight on the behaviour of the autonomic SOA, especially in situations where actual system testing is not applicable, formal modelling and analysis of the proposed architecture are conducted.

A. *Petri Nets based Validation Methodology*

Petri Nets [25]-[26] based functional validation framework is used to analyze the SOA framework proposed in this research. This framework was introduced in [6] to validate service composition in SOA. Fig. 3 shows the functional validation methodology. Later on, the state transitions using Petri Nets modeling will be analyzed for enabling the process of validation on service's behavioral correctness and other properties.

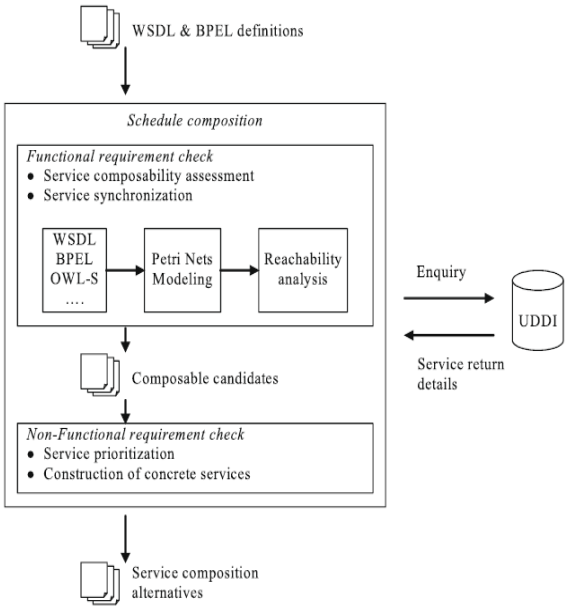

Fig. 3 Petri Nets based functional validation methodology [6]

Coloured Petri Nets (CPNs) [27] is a modelling language developed for systems in which communication, synchronization, and resource sharing play an important role. CPNs combine the strengths of ordinary Petri nets with the strengths of a high-level programming language. Petri nets provide the primitives for a process interaction, while the programming language provides the primitives for the definition of data types and the manipulations of data values.

CPN models can be made with or without explicit reference to time:

- Untimed CPN models are usually used to validate the functional/logical correctness of a system.
- Timed CPN models are used to evaluate the performance of the system.

CPNs also offer more formal verification methods, i.e. state space analysis (reachability, boundedness, home properties, liveness, and fairness) and invariant analysis.

B. *Modelling Web Services*

Details about web services are based on information from Web Service Description Language (WSDL) descriptions. Thus to model a web service it is necessary to provide the following WSDL data:

- the name of the web service
- contents of the XML message sent to the external web service (types and names of arguments)
- contents of the response XML message from the external web service (types and names of arguments)
- exceptions for the web service

To invoke a web service and to get a result, the XML messages are used, which contain names and values of input parameters or responses. Meanwhile, to model these XML messages in Petri Nets, appropriate colour sets have to be declared. Record type is used, for enabling mapping names and values as defined in a WSDL description of messages.

A web service composition involves three main interactions; namely invoking, sending, and receiving [28]. In the colored Petri nets those interactions are modeled as transitions, thus in this work those three subsets of transitions to represent those operations are derived and enhanced from [28] to cope with exceptional and no response messages and to support the CBR processes, which are:
$T_{invokeWS}$, T_{sendWS}, and $T_{receiveWS}$.

A transition *t* that represents an *invoke* operation can be defined as the following:

$$t \in T_{invokeWS} \text{ iff } (t \in T) \wedge (size(In(t)) = 1) \wedge (size(Out(t)) >= 2) \wedge (\exists p \in In(t) : C(p) \rightarrow inMsg) \wedge (\exists p1 \in Out(t) : C(p1) \rightarrow outMsg) \wedge (\exists p2 \in Out(t) : C(p2) \rightarrow Revise)$$

where:

- *T* is a set of all transitions in a net,
- *In* and *Out* are functions that map a node to its input and output nodes, respectively,
- *size* refers to a size of a set,
- *C* maps a place into its color set,
- $\rightarrow$ maps WS messages into record types,
- *inMsg* and *outMsg* represent accordingly all input and all output messages defined in a WS description for a web service.

The definition shows that a transition modelling an *invoke* operation has one input place with the colour set mapped from a WSDL input message, and at least two output places - one with the colour set mapped from a WSDL output message and another with the unit colour set (it represents "no response" type of output). The size of the set of output can be bigger than two as in WSDL description it is possible to have fault messages, each of which is modelled as an output place. Fig. 4 shows the Petri net model of the invoke operation.

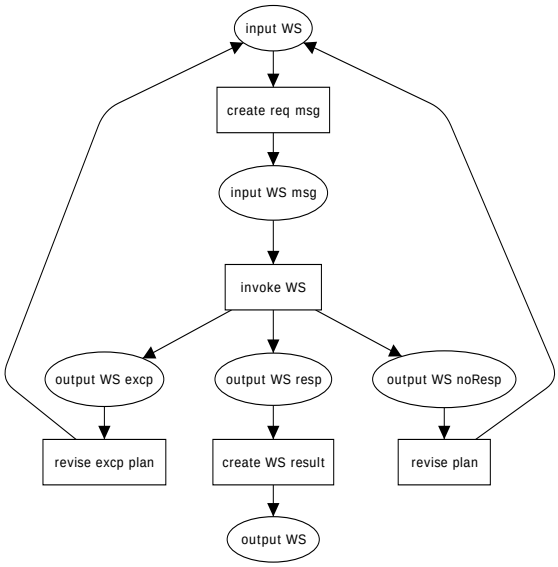

Fig. 4 Petri Net model of the invoke operation

A transition *t* that represents a *send* operation can be defined as the following:

$t \in T_{sendWS}$ *iff* $(t \in T) \wedge (size(In(t)) = 1) \wedge (size(Out(t)) = 1) \wedge (\exists p \in In(t) : C(p) \rightarrow inMsg) \wedge (\exists p1 \in Out(t) : C(p1) \rightarrow reqMsg)$

Different from *invoke* operation, in *send* operation there is no any different output type but only the request service message (*reqMsg*) colour set. Fig. 5 shows the Petri net model of the *send* operation.

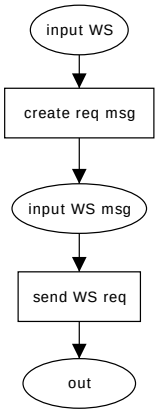

Fig. 5 Petri Net model of the send operation

A transition *t* that represents a *receive* operation can be defined as the following:

$t \in T_{receiveWS}$ *iff* $(t \in T) \wedge (size(In(t)) = 1) \wedge (size(Out(t)) >= 2) \wedge (\exists p \in In(t) : C(p) \rightarrow respMsg) \wedge (\exists p1 \in Out(t) : C(p1) \rightarrow outMsg) \wedge (\exists p2 \in Out(t) : C(p2) \rightarrow Revise)$

The difference between this definition and the *invoke* operation is that for input there is the *respMsg* colour set. Thus, an input message is not modelled. Fig. 6 shows the Petri net model of the *receive* operation.

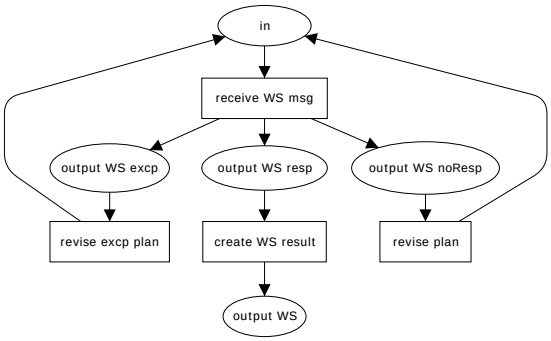

Fig. 6 Petri Net model of the receive operation

The set of all interactions for composite web service can be defined as the following:

$$T_{WS} = T_{invokeWS} \cup T_{sendWS} \cup T_{receiveWS}$$

One of the Petri Nets analysis methods are occurrence graphs which in this work are used to analyze composite web services to identifying how failures of required web services may influence the overall SOA execution. An occurrence graph is a graph with a node for each reachable marking (a distribution of tokens between places) and an arc for a transition and its binding (called binding elements). This graph is the basis for checking whether composite web service can be successfully executed even if one or more used web services do not respond or give out exceptional message, which is modelled as "no response" and "exceptional" type of output respectively, and in the colored Petri Nets as output place of an interaction with the unit color set. To perform such checking it is necessary to infer the reachability of a marking representing a success of composite web service composition from markings representing different outputs from external web services. This analysis was also extending the work by Zurowska & Deter [28].

The nodes and markings in an occurrence-equivalence graph (OE-graph), that represent exceptional or no response types of output for each used external web service, are identifiable in the research. Then it is followed by checking the reachability of

$M_{success}$ from all those states. If the success is reachable, it enables to execute composite web service even if there is an exception or no response; otherwise in case of a failure of a component, composite web service in conventional SOA framework could not be successfully executed. The additional revise node makes the proposed framework potentially able to reach $M_{success}$ even in the case where exceptional error message is received from the web service WS_n. The framework will revise the composition plan and it will invoke the next web service (WS_{n+1}) instead. Fig. 7 shows the occurrence graph with equivalence classes for web service composition, in which the successful marking is represented by node 9. Thus it can be concluded that even if the external web services is not responding or giving exceptional messages, the service composition is likely will still be successful.

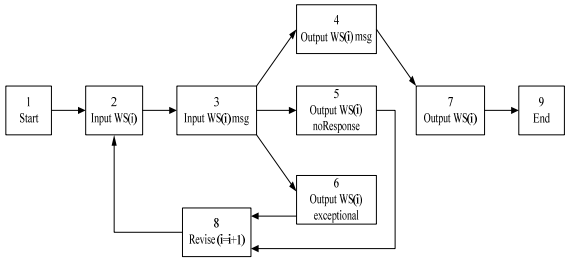

Fig. 7 Occurrence graph of web service composition in autonomic SOA

IV. Simulation Development

To simulate and test the framework proposed in this research, soapUI, a Java-based free and open source cross-platform testing solution for SOA, is used. Equipped with a graphical interface, and enterprise-class features, soapUI allows users to create and execute automated functional, regression, compliance, and load tests. In a single test environment, soapUI provides complete test coverage and supports all standard protocols and technologies, including SOAP and REST-based Web services, JMS enterprise messaging layers, databases, and Rich Internet Applications.

A. *Currency Conversion Services*

A test environment whose goal is to show the ability of the proposed framework to cope with unavailable services was developed by using the following WSDL files available in the internet:

1. Currency Convertor web service [29]
2. Currency Service web service [30]

This case study will show the ability of the autonomic SOA framework to cope with erroneous or unavailable atomic services. These currency converter services were selected as they are freely available on the internet, and they provide the equal atomic service, i.e. providing conversion rate for a given two currencies.

It is then followed by creating the mock services of those services. Mock services can be used to create a proof of concept, either as a wire frame or as a demo for the proposed framework. This is a powerful means and provides a good ground for decision-making of the framework.

The simulation program was executed several times for the following conversion:

- US Dollar (USD) to Malaysia Ringgit (MYR)
- Euro (EUR) to Malaysia Ringgit (MYR)
- Malaysia Ringgit (MYR) to Indonesia Rupiah (IDR)

The web service providers were simulated to be down (unavailable) alternatingly. Table 1 shows the currency converter simulation results.

TABLE I
CURRENCY CONVERSION SIMULATION RESULTS

Conversion	***Currency Convertor***	***Currency Service***	**Conversion Result Reachable?**
USD to MYR	Output message	No response	Yes
USD to MYR	No response	Output message	Yes
EUR to MYR	Output message	No response	Yes
EUR to MYR	No response	Output message	Yes
MYR to IDR	Output message	No response	Yes
MYR to IDR	No response	Output message	Yes

The results showed that the proposed autonomic SOA framework was able to keep providing currency conversion rate service to the user every time. The autonomic SOA will seamlessly switch and access the *CurrencyConvertor* when *CurrencyService* was unavailable and vice versa, thus increasing the overall system robustness and reliability.

Without the autonomic feature activated, erroneous web service in the simulation, e.g. *CurrencyConvertor* service, will produce the socket time out exception message after the system tried for some times to connect to the web service. The simulation stopped and user must create new request to try to re-connect or try other service provider. However, with the autonomic feature activated, when the *CurrencyConvertor* web service was unavailable, the system was still able to provide the currency conversion rate by seamlessly switch to the other service provider, i.e. *CurrencyService*.

B. *Travel Scheduling / Vacation Planner*

To compare the research as peer-to-peer, the following works that also used Petri Nets modeling are chosen and presented. This case study will show the ability of the proposed autonomic SOA framework to cope with unavailable services in service composition.

1) *Travel scheduling:* Yoo et al. [6] used travel scheduling as a case study. The conditions of their work are the following:

The validation conditions:

- Visit = *AirlineBooking* & *HotelReservation* & *CarRental*
- Initial input = *TravelInfo*
- Final output = *TravelSchedule*
- Final status = Success (Accept) | Failure (Reject)

Their results stated that "*the service composition is complete and logically correct if no exception / error occurs from the (initiating) user to the (terminating) user*" [6]. In this aspect, the autonomic SOA framework in this research is better for being able to cope for non responsive atomic services, exceptions and errors messages happened in service composition as shown in the formal models and analysis in the previous section.

If any error or exception is raised in service composition, it will be captured by the monitoring module and the CBR process will analyze the error and plan action to overcome the error accordingly. The action plan may include usage of other service provider (in case of web service provide error or unavailability) or usage of other channel of communication (in case of network problem).

2) *Vacation planner:* Zurowska and Deter [28] used vacation planner as a case study in their work. Their result showed that their framework was able to void interactions with optional components (web services) that are not working. However, in the case when the faulty web service is compulsory to successfully execute composite web service (like *FindFlight* in their example), the system was unable to overcome it. This is shown in their reachability analysis in Table 2.

TABLE II
REACHABILITY ANALYSIS FOR THE VACATION PLANNER [28]

"From" marking	"To" marking	Reachable
Output_FindAttractions_No + Output_FindFlight_Msg	End	Yes
Output_FindAttractions_Msg + Output_FindFlight_No	End	No

In their case study, the *FindFlight* web service is a compulsory service and the *FindAttractions* web service is an optional one. If there is a valid output message from the *FindFlight* web service and no valid output from the *FindAttractions* web service, the end state is still reachable. However if there is no valid output from the *FindFlight*, even if there is a valid output from *FindAttractions*, the end state will be unreachable.

This case study was simulated using the WSDL descriptions given by [28] in soapUI environment. Table 3 shows simulation results of the vacation planner in the proposed autonomic SOA.

TABLE III
VACATION PLANNER SIMULATION RESULTS

FindAttractions	***FindFlight***	**Vacation Booking Result Reachable?**
No response	Output message	Yes
Output message	No response	Yes

From this aspect, the proposed autonomic SOA framework of this research is also better compared to the conventional SOA framework analyzed by [28], because it is still able to reach success end state ($M_{success}$) even if there is no valid output from *FindFlight* (no response or exceptional message) as shown in the simulation results and also described in the formal analysis in section 3. This is true due to the ability of the framework to revise its action plan and look for other services similar to what *FindFlight* provides, either within the service ecosystem or searching at other service ecosystems.

V. CONCLUSIONS

The Petri Net analysis showed that in the proposed autonomic SOA framework, web service composition will still be successful even if the atomic web services are not responding or giving error messages. The revise process makes the proposed framework potentially able to reach successful end marking even in the case where exceptional error message is received from web service provider. The framework will revise the composition plan and it will invoke the next service provider instead.

The simulation results showed the ability of the proposed framework to work around unavailable services and seamlessly provide user with the same type of service from different service providers. Therefore the framework will improve the success rate of providing not only atomic service, but also composite service since it improves the availability and reliability of the atomic services. If all the required atomic services to compose a composite service are obtainable, then the service composition will be successful since the service composition process itself executed internally within the business process layer of the framework. Thus it can be concluded that the proposed framework will also improve the success rate of providing a composite

service by ensuring the availability and reliability of its atomic services.

The proposed autonomic SOA framework is yet to be implemented in real world system applications. In this research, the proposed framework has been modeled and simulated. Yet to comparing it with other SOA implementation equally, it needs to be implemented in real applications. Future works could focus on implementing the proposed framework in a specific application domain, then analyzing and benchmarking it with other SOA implementations.

The presented research analysis on the proposed framework also has not included a thorough quantitative evaluation and analysis to measure the quantitative improvements over conventional SOA framework, especially in term of Quality of Services (QoS) and its usage in Service Level Agreement (SLA). Further quantitative study is needed after the proposed framework has been fully implemented.

REFERENCES

[1] A. Montresor, H. Meling, and O. Babaoglu, "Toward self-organizing, self-repairing, and resilient large-scale distributed systems," Technical Report UBLCS-2002-10, Sep. 2002.

[2] A. Lazovik and F. Arbab, "Using Reo for service coordination," in *Proc. of ICSOS 2007*, LNCS 4749, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 398-403.

[3] M. Bell, *Service-Oriented Modeling: Service Analysis, Design, and Architecture*, John Wiley & Sons, Inc., Hoboken, New Jersey, 2008.

[4] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, *Applied SOA: Service-Oriented Architecture and Design Strategies*, Wiley Publishing, Inc., Indianapolis, Indiana, USA, 2008.

[5] D. Schneider, C. Bunse, and K. Schmid, "Towards Adaptive Service Engineering", in *Proc. of the International Workshop on the Foundations of Service-Oriented Architecture*, Special Report CMU/SEI-2008-SR-011, June 2008.

[6] T. Yoo, B. Jeong, and H. Cho, "A Petri Nets based functional validation for services composition," *Expert Systems with Applications* 37 (2010), pp. 3768–3776, Elsevier, 2009.

[7] *Autonomic computing: IBM's perspective on the state of information technology*, IBM, USA, October 2001.

[8] J. O Kephart and D. M. Chess, "The vision of autonomic computing," in *Computer*, vol. 36, No. 1, IEEE Computer Society, pp. 41-50, Jan. 2003.

[9] J. L. Kolodner, "An Introduction to Case-Based Reasoning", *Artificial Intelligence Review*, vol. 6, pp. 3-34, 1992.

[10] A. Aamodt and E. Plaza, "Case-based reasoning: foundational issues, methodological variations, and system approaches," in *AI Communications*, 7:39-59, 1994.

[11] M.A.C. Bhakti, A.B. Abdullah, and L.T. Jung, "Autonomic, Self-Organizing Service Oriented Architecture in Service Ecosystem," in *Proc. of International Conference on Digital Ecosystems and Technologies (IEEE DEST 2010)*, Dubai, United Arab Emirates, 12-15 April 2010.

[12] M.A.C. Bhakti, A.B. Abdullah, "Towards an Autonomic Service Oriented Architecture in Computational Engineering Framework," in *Proc. of the 10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010)*, Kuala Lumpur, Malaysia, 10-13 May 2010.

[13] M.A.C. Bhakti, A.B. Abdullah, "Design of an Autonomic Service Oriented Architecture," in *Proc. of the 4th International Symposium on Information Technology (ITSim 2010)*, volume 2, pp. 805-810, Kuala Lumpur, Malaysia, 15-17 June 2010.

[14] M.A.C. Bhakti, A.B. Abdullah, "Autonomic Computing Approach in Service Oriented Architecture," in *Proc. of IEEE Symposium on Computers and Informatics (ISCI 2011)*, pp. 231-236, Kuala Lumpur, Malaysia, 20-22 March 2011.

[15] H. Arora, T. S. Raghu, A. Vinze, and P. Brittenham, "Collaborative Self-Configuration and Learning in Autonomic Computing Systems: Applications to Supply Chain," in *Proc. IEEE International Conference on Autonomic Computing*, June 2006.

[16] S. Montani and C. Anglano, "Achieving self-healing in service delivery software systems by means of case-based reasoning," *Journal of Applied Intelligence*, vol. 28, No. 2, Springer Netherland, Apr. 2008, pp. 139-152.

[17] W. Cheetham, "Tenth Anniversary of the Plastics Color Formulation Tool", in *Proc. of the 16th Innovative Applications of Artificial Intelligence Conference*, Published by The AAAI Press, California, July 2004.

[18] A. P. Morgan, J. A. Cafeo, K. Godden, R. M. Lesperance, A. M. Simon, D. L. McGuinness, and J. L. Benedict, "The General Motors Variation-Reduction Adviser: Deployment Issues for an AI Application", in *Proc. of the 16th Innovative Applications of Artificial Intelligence Conference*, AAAI Press, California, July 2004.

[19] D. Hinkle and C. Toomey, "Applying Case-Based Reasoning to Manufacturing", *AI Magazine* 16(1), pp. 65-73, Spring, 1995.

[20] I. Watson and D. Gardingen, "A Distributed Case-Based Reasoning Application for Engineering Sales Support", in *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI-*

99), Vol. 1, pp. 600-605, Morgan Kaufmann Publishers, 1999.

[21] M. Barbera, C. Barbero, P. D. Zovo, F. Farinaccio, E. Gkroustiotis, S. Kyriazakos, I. Mura, and G. Previti, "An Application of Case-Based Reasoning to the Adaptive Management of Wireless Networks", in *Proc. of the 6th European Conference on Case-Based Reasoning (ECCBR), Lecture Notes in Artifical Intelligence (LNAI) 2416*, pp. 490–504, Springer-Verlag Berlin Heidelberg, 2002.

[22] K. Xu and H. Muñoz-Avila, "CaBMA: Case-Based Project Management Assistant", in *Proc. of the 16th Innovative Applications of Artificial Intelligence Conference*, Published by The AAAI Press, California, July 2004.

[23] B.-S. Yang, S. K. Jeong, Y.-M. Oh, and A. C. C. Tan, "Case-based reasoning system with Petri nets for induction motor fault diagnosis", *Expert Systems with Applications* 27, pp. 301–311, Elsevier Ltd., 2004.

[24] D.R. Wilson and T.R. Martinez, "Improved Heterogeneous Distance Functions", *J. Artificial Intelligence Research*, 6, pp. 1—34, 1997.

[25] C. A. Petri, "Communication with Automata," New York: Griffiss Air Force Base. Tech. Rep. RADC-TR-65-377, vol. 1, suppl. 1, 1966.

[26] T. Murata, "Petri Nets: Properties, Analysis, and Applications," in *Proc. of the IEEE*, vol. 77, no. 4, April 1989.

[27] K. Jensen, "Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use", Volume 1, Basic Concepts, Monographs in Theoretical Computer Science, Springer-Verlag, 1997.

[28] K. Zurowska and R. Deters, "Overcoming failures in composite web services by analysing colored petri nets," in CPN'07 - Workshop and Tutorial on Practical Use of Coloured Petri Nets and CPN Tools, Denmark, 2007.

[29] Currency Convertor web service. [Online]. Available: http://www.webservicex.net/CurrencyConvertor.asmx?wsdl

[30] Currency Service web service. [Online]. Available: http://www.restfulwebservices.net/wcf/CurrencyService.svc?wsdl