2011 International Conference on Telecommunication Technology and Applications
Proc .of CSIT vol.5 (2011) © (2011) IACSIT Press, Singapore

Formal Modelling of an Autonomic Service Oriented Architecture

M. Agni Catur Bhakti [1] and Azween B. Abdullah [1 +]

[1] Department of Computer and Information Sciences, Universiti Teknologi PETRONAS
Bandar Seri Iskandar 31750 Tronoh Perak, Malaysia

Abstract. Service oriented architecture (SOA) is improving conventional software architecture by enabling composition of large and complex system out of simpler software services. However, implementation of SOA brings about some challenges, including its adaptability and robustness. A more robust service architecture that is capable of changing its structure and functionality autonomously, i.e. with little human intervention, is required due to the increased complexity and dynamism of the current network systems. In this paper, we propose the adaptation of autonomic computing paradigm into SOA to improve its robustness, and we also elaborate the formal modelling of the proposed autonomic SOA.

Keywords: service oriented architecture, autonomic computing, formal model.

1. Introduction

Service Oriented Architecture (SOA) is the main architectural concept in the field of service oriented computing. Using SOA, large distributed computational units can be built based on existing services by composing complex composite services out of simple atomic ones [1]. Definitions of SOA are given by several international bodies / organizations, including the Organization for the Advancement of Structured Information Standards (OASIS) in [2] and the World Wide Web Consortium (W3C) in [3].

The current SOA frameworks offer service reusability, consistency, efficiency, and integration. However they are still lacking adaptability and robustness. Conventional service composition will be complete and correct with the assumption that there are no exceptions or errors occur from the initiating user to the terminating user. That is not the case with the current complex and dynamic systems. It is reported in [4] that the scale and complexity of current distributed systems are increasing and showing high dynamism as the global network systems grow. They 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 is required.

We proposed the adaptation of autonomic computing paradigm presented by IBM Research Centre [5], [6] into SOA domain in order to meet the challenges mentioned above. We incorporated the autonomic computing cycle and case-based reasoning (CBR) [7] in our framework to introduce learning and adaptability into SOA. The autonomic mechanism in SOA will autonomously monitors and analyzes service requests, then plans and provides the optimum 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 presents brief backgrounds on autonomic computing paradigm and case-based reasoning as we are adapting those technologies into our proposed framework; section 3 elaborates the proposed autonomic SOA; section 4 describes the formal description the proposed architecture, followed by its formal modelling and analysis; lastly section 5 presents summary of this paper and direction for future work.

+ Corresponding author. Tel.: + 6053687507; fax: + 6053656180.
E-mail address: azweenabdullah@petronas.com.my.

2. Backgrounds

2.1. Autonomic computing

The autonomic computing paradigm, inspired by the human autonomous nervous system, was proposed by IBM 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 [6]. This paradigm has been used in many researches in various domains such as those in [8] and [9] in which the authors adapted autonomic computing paradigm in self-configuration and self-healing software systems.

The autonomic systems consist of autonomic elements, whose behaviour is controlled by autonomic manager, which will relieve the human responsibility of directly managing the managed elements by monitoring these elements and its external environment to construct and execute plans based on the analysis of the gathered information. Thus, the autonomic managers will carry out the autonomic computing cycle, i.e. monitor, analyze, plan, and execute, using its knowledge base. Eventually, a system needs to exhibit four aspects of self-management, i.e. self-configuration, self-optimization, self-healing, and self-protection [6] in order to fully achieve the essence of autonomic computing, i.e. to self-manage.

2.2. Case based reasoning

Case-based reasoning (CBR) [7] is the process of solving a new problem by remembering a previous similar situation and by reusing information and knowledge of that situation. CBR is able to utilize the specific knowledge of previously experienced, concrete problem situations, called ‘cases’. In CBR, a new problem is solved by finding a similar past case, and reusing it in the new problem situation. CBR also is an approach to incremental, sustained learning, since a new experience is retained each time a problem has been solved, making it immediately available for future problems.

Benefits of using CBR approach include the following: reasoning by re-using past cases is a powerful and frequently applied way to solve problems (inspired from human’s problem solving); CBR favours learning from experience, since it is usually easier to learn by retaining a concrete problem solving experience than to generalize from it; and CBR is also known to be well suited for domain where formalized and recognized background knowledge may not be available, and that is the case in autonomous service architecture. A new problem is solved by retrieving one or more previously experienced cases, reusing the case in one way or another, revising the solution based on reusing a previous case, and retaining the new experience by incorporating it into the existing knowledge-base (case-base).

3. Autonomic SOA

We initiated our research with goals to provide new concepts of adapting autonomic computing paradigm into SOA and to develop a more adaptive and robust SOA prototype based on these concepts. Compared to the conventional SOA, the proposed autonomic SOA has additional features which include the addition of autonomic agent / manager and the ability to adapt to changes with the knowledge from a knowledge base. The autonomic capability can 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.

Figure 1 shows the overall architecture of the proposed autonomic SOA that extends a typical SOA infrastructure (service requestor and service provider), e.g. web services model in [10], by incorporating the autonomic computing cycle. The architecture is separated into three layers:

- On the top is a presentation layer that provides interfaces to various users.
- At the middle is a processing layer that performs and coordinates several jobs.
- At the bottom is a resource layer that enables utilization of the distributed resources via web services.

The resource/service layer is a typical SOA framework which consists of service providers, service registry, and the broker agents act as service requestors in processing layer. We extend the functionality of the service registry by adding a knowledge base as required by the autonomic computing paradigm. Knowledge base provides the capability to store the previous services profiles (cases) whose features including:

- Name and description of service.
- Type of service (atomic or composite).
- If the service is a composite service, then the profile will also include profile of the atomic services needed to compose the composite service (the “ingredients”).
- Where, when, how to access (and compose) the service (the “recipe”). The recipe and ingredients are the solution of the case.
- Service usage and usage with other services.

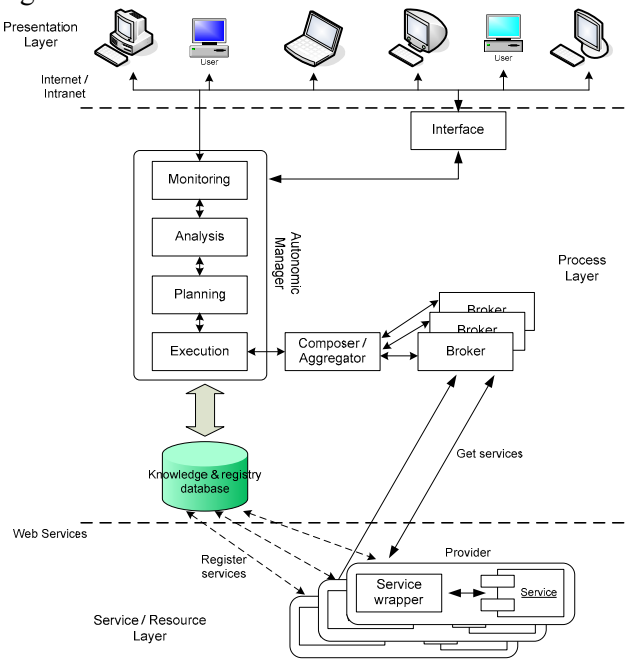

Fig. 1: Overall architecture of the proposed autonomic SOA.

The autonomic computing paradigm is incorporated in the processing layer which has the autonomic manager in it. In the context of autonomic computing paradigm, the autonomic manager will perform the autonomic cycle as described in the following sections.

3.1. Monitoring

Monitor the overall system which shall include the monitoring of the requests from users, sensing the availability of the services, addition of new services, removal of services, etc. A sentinel or monitoring agent will provide this monitoring service. It will continuously monitor the system in order to detect and identify request from user and the status of services. The status checking can be set at user-defined frequency. When a request is detected, the monitor will forward the request to analysis module. When there is change in service status (added, removed, available, or unavailable), it will update that service’s status in knowledge database.

3.2. Analysis

Analyze incoming service requests. It will retrieve related information (service profiles) from knowledge base and will use and/or revise the information as necessary to provide the requested service. It starts by first searching for information of the requested service in the knowledge base. If that particular service profile is available in the knowledge base, then this information is used to plan appropriate actions. However if it is not found in the knowledge base, the agent will look for similar ones and will reuse and/or revise if necessary to plan/create action plans.

We adapted the CBR cycle, i.e. 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. CBR is chosen based on its features and also successful implementation in the self-healing system [9]. Figure 2 illustrates the adaptation of CBR and autonomic computing cycle into SOA. The CBR process is described below.

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 in [9] used heterogeneous Euclidian-overlap metric [11]. The similar cases found shall then be used for action planning (by revising them).

If there are no similar previous cases, the monitoring agent 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 needed services. The new service profile will then be used for action planning and added (retained) to the knowledge base.

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

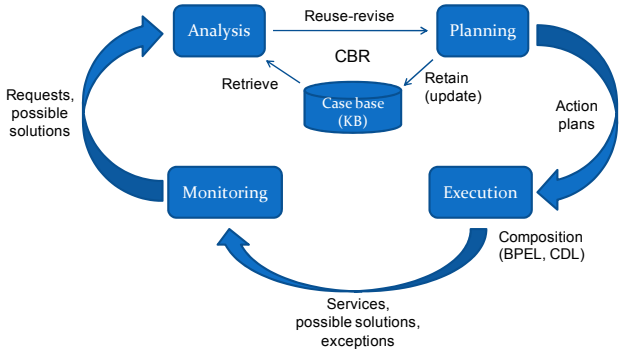

Fig. 2: Adaptation of autonomic cycle and CBR cycle in the proposed autonomic SOA.

3.3. Planning

Plan the suitable actions for the requested service. If the requested service is a composite service then the action plan will include the list of available atomic services needed to compose the required composite service, where and how to access them, and the sequence of accessing them. It will also later update the knowledge base if new action plan is created (or revised from old ones) so that these plans can be readily available when the same composite service is requested again in the future.

3.4. Execution

Autonomic manager executes the composed plan to provide the requested service. Broker or negotiator agent will assist in interacting and negotiating with the applications to obtain the required atomic services, and the service composer / aggregator will compose the composite service out of the atomic services (e.g. using BPEL, CDL).

4. Formal Model

4.1. Formal definitions

SOA is "*a set of components which can be invoked, and whose interface descriptions can be published and discovered*" [12]. The services are published by service providers. Service requestors then can discover (via service registry) and invoke those services. Thus we can define SOA formally as the following:

Definition 1: *SOA* is a four-tuple, consisting of service, *S*, service provider, *SP*, service requestor / consumer, *SC*, and service registry, *SR*:

$$SOA = \langle S, SP, SC, SR \rangle$$

Service, *S*, is an abstract resource that represents a capability of performing tasks [12]. Service can be thought of as a container for a set of system functions that have been exposed to the web based protocols. There are two types of services:

- Composite service, *sc*, i.e. a service whose implementation calls other services and is a result of composition function, *c*, result of other services
- Atomic service, *sa*, i.e. self-contained service and does not invoke any other services

Definition 2: $sc = c(s_1, \ldots, s_n)$, where $\{sa, sc, s_1, \ldots, s_n\} \in S$

Service registry is an authoritative, centrally controlled store of service information [12], which may be used by service providers to publish their services and service requestors to discover services using Web Service Description Language (WSDL). Thus:

Definition 3: $SR = \{desc(s_1), \ldots, desc(s_n)\}$

where $desc(s)$, is description of service, $s \in S$

Service provider is the entity that is providing service [12]. Thus it can be considered as a set of service.

Definition 4: $SP = \{s\}, s \in S$

The autonomic manager will carry out the autonomic computing cycle: monitor, analyze, plan, and execute, via its knowledge base [6]. Hence:

Definition 5: Let autonomic manager, *AM,* be a 5-tuple, consisting of monitoring, *M*, analysis, *A*, planning, *P*, execution, *E*, and knowledge base, *KB:*

$$AM = \langle M, A, P, E, KB \rangle$$

Autonomic systems consist of autonomic elements, whose behaviour is controlled by autonomic manager [6]. Therefore the Autonomic SOA, *ASOA*, is an SOA with autonomic manager, *AM*:

Definition 6: $ASOA = \langle SOA, AM \rangle$

Using definition 1, 5, and 6, we obtain:

Definition 7: $ASOA = \langle SP, S, SR, SC, M, A, P, E, KB \rangle$

In the proposed framework, *AM* will retrieve *cases* from *KB*, whose features include:

- Name of the service
- Type of service (atomic, composite, etc)
- Description of service (WSDL-based)
- Number of usage (to measure the usability of the service)
- Related services
- The "recipe" as the solution, including the list of service providers or "ingredients" of a service; and how to access the service providers and compose a composite service

Thus:

Definition 8: $case = \langle name, type, desc, usage\_no, related\_serv, solution \rangle$

4.2. Snapshot mechanism

In order to achieve more robust service oriented architecture and to reduce service searching time, we will need a mechanism to determine global status of services in the ecosystem of autonomic elements. To determine a global status, an autonomic element, *ae*, must enlist cooperation of other elements that must record their own local services status and send the recorded local status to *ae*. We adapted and enhanced the work in [13] to suit our autonomic SOA framework.

Theorem 1: The snapshot algorithm will determine a global status of autonomic element *ae*.

Proof: As soon as any snap input occurs at autonomic element *ae(i)*, that element records the state of services in *ae(i)* and sends out markers on all its output channels. As soon as any other autonomic element *ae(j)* receives a marker on any channel, it records the state of services in *ae(j)* and sends out markers on all its output channels if it has not previously done so. With the assumption that the network graph is a mesh network, as in the case of internet, where every node can propagate the data / state in the network, markers will eventually propagate to all autonomic elements, and all elements record their local status. Also every autonomic element will eventually perform a report output. Thus, the global status of the autonomic elements is obtained.

4.3. Service composition

There are three main interactions in web service composition, they are: invoke, send, and receive [14]. In the colored Petri nets they are modelled as transitions, thus three subsets of transitions are required to represent those operations, they are: T_{invoke}, T_{send}, and $T_{receive}$. We adapted and extended the work in [14] to accommodate the CBR process.

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

$t \in T_{invoke}$ *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* is 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 that models an invoke operation has one input place with the color set that is mapped from a WSDL input message, and it has at least two output places: one with the color set that is mapped from a WSDL output message and one with the unit color set (it represents "no response" type of output). The size of the set of output places can be bigger than 2, because we can have fault messages in WSDL description, each of which is modelled as an output place. In the case of no response or fault response, the system will revise the plan (indicated by *Revise* output place).

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

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

The difference with invoke operation is that we do not have different output types but only the request service message (*reqMsg*) color set.

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

$t \in T_{receive}$ *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* color set, so we do not model an input message. And the system will revise the plan (indicated by *Revise* output place) in the case of no response or fault response, similar with the invoke transition.

5. Summary and Future Work

This paper presented and discussed an approach to achieve an autonomic SOA by applying autonomic computing paradigm and case-based reasoning. We have presented the design and elaborated the proposed architecture with its descriptions, formal definitions and models. These formal models serve as foundation for the next phases of our research.

We have developed an initial prototype of the proposed architecture using Axis2 SOA and web services framework [15], autonomic computing paradigm, and case-based reasoning techniques. The initial prototype development that supports atomic services has successfully completed. We will extend the prototype to benchmark it with the available SOA framework. It is expected that our framework will be able to overcome exceptions or faulty web services due to its adaptive feature.

We are also working on formal analysis, i.e. modelling, verification, and formal simulation of the proposed architecture using CPN Tools [16] which will provide further insight on the behaviour of the autonomic SOA, especially in situations in which actual system testing and implementation might not be applicable.

6. References

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

[2] OASIS Reference Model for Service Oriented Architecture. OASIS Standard. 12 Oct. 2006.

[3] H. Haas and A. Brown (Editors). Web Services Glossary. W3C Working Group Note 11 Feb. 2004.

[4] A. Montresor, H. Meling, and O. Babaoglu. Toward Self-Organizing, Self-Repairing, and Resilient Large-scale Distributed Systems. Technical Report UBLCS-2002-10. Department of Computer Science, University of Bologna, Italy. September 2002.

[5] IBM. Autonomic Computing: IBM's Perspective on The State of Information Technology. http://www.research.ibm.com/autonomic/manifesto/

[6] J.O. Kephart and D.M. Chess. The Vision of Autonomic Computing. *Computer*. IEEE Computer Society. Jan. 2003, **36** (1), pp. 41-50.

[7] A. Aamodt and E. Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*. 1994, 7, pp. 39-59.

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

[9] S. Montani and C. Anglano. Achieving Self-Healing in Service Delivery Software Systems by Means of Case-Based Reasoning. *Applied Intelligence*. Springer Netherland. April 2008, **28** (2), pp. 139-152.

[10] M.N. Huhns and M.P. Singh. Service-oriented computing: key concepts and principles. *IEEE Internet Computing*. Jan-Feb. 2005, pp. 75-81.

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

[12] D. Booth, H. Haas, and A. Brown. Web Services Glossary. Technical Report, World Wide Web Consortium (W3C). 2004. http://www.w3.org/TR/ws-gloss/

[13] K.M. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transaction on Computer Systems*. February 1985, **3** (1), pp. 63-75.

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

[15] Apache Web Services – Axis. http://ws.apache.org/axis/

[16] CPN Tools – Computer Tool for Coloured Petri Nets. http://wiki.daimi.au.dk/cpntools/cpntools.wiki