GenericSOA: a Proposed Framework for Dynamic Service Composition

Mohammed Said[1], Maryam Hazman[1], Hesham Hassan[2], Osama Ismail[2]

[1] Central Lab of Agriculture Expert Systems (CLAES)
Giza, Egypt
[2] Faculty of Computers and Information, Cairo University
Giza, Egypt

Abstract

Many organizations' developers use Service Oriented Architecture (SOA) in building their systems. It provides them by an easy way to re-use their already developed software components. Now, researchers try to enhance the SOA architecture to improve its performance and scalability. This paper, proposes a new SOA architecture called "*GenericSOA*" that allows dealing with legacy systems problem and enhancing SOA elasticity. The proposed architecture aims to easily integrating the newly developed software components. The main idea behind *GenericSOA* is to support its users by a set of predefine task templates. These templates can be used in building the new developed services that can be easily integrated in a loosely coupled way to compose the target system.

Keywords: *Service Oriented Architecture (SOA), Web Service Composition.*

1. Introduction

Due to the great scientific and technological development for working systems, usage of many varied operating systems and different standards in working field, and handling great amounts of data today, this results in an important need for finding a mean to connect these working systems together. Also, it is not easy to develop huge legacy systems which are well tested and free of bugs. It will be very hard and expensive to either rebuild them from scratch or exclude them at all.

Recently, "Service Oriented Architecture (SOA) has become the new architectural style [1][2][3][4]. SOA depends on building applications using existing services. Services are reusable units, which can be used by applications or other services independent of their implementation. So, the functionality of such services is presented within a service description and the services are loosely coupled. SOA gives the advance of offering new functionalities by using various services which are provided by different suppliers [5]. Moreover, the property of loose coupling between services provides dynamic addition and removal of any service at any time [5].

Using services technologies helps in facilitating the development of more complex business cases, and reducing its development time and cost. It provides richness, flexibility and scalability to the enterprises using them [6]. The great challenge for the developers is to ensure the interoperability between the needed services regardless of the programming languages, operating systems and hardware platforms [7]. In general loosely coupled and reusable software components are considered the most important advantage of services technologies [8].

In any organization, there exist many modules that are used by many of its systems and repeated in these systems. The construction of these systems were required rebuilding those small modules again and again in each time the system built. This means wasting time, effort and money, since any updating in a used module within one of the legacy systems does not reflect in the other systems which used this module. In this paper, a GenericSOA framework is proposed to enhance the SOA to facilitate easily modifying and efficiency of services usage. The proposed framework supplies its users by a library of services which they can build their systems by selecting services from them. So, any modification will be done in the selected services will affect in building new systems.

The remainder of the paper is organized as follows. Section 2 introduces the problem that we are trying to solve. Section 3 describes the GenericSOA framework as a proposed solution, and then a brief overview of related work is presented in section 4. Finally, the concluding remarks are presented in section 5.

2. Problem Description

Usually, in any organization there are group of working systems that are well tested. Each system is implemented using its own standalone language. Moreover, the individual system contains various small modules. These modules are linked together to perform some certain task for achieving the whole system goal.

Analyzing some of the existing systems, we observed that those large legacy systems include modules, which are being used by many other systems in the same or even other organizations. These modules are repeated / reused even in other legacy systems within the same organization. This means that the construction of any new system requires rebuilding those small modules again and again each time which means wasting time, effort and money. Also, any updating in a used module within one of the legacy systems does not reflect in the other systems which used this module.

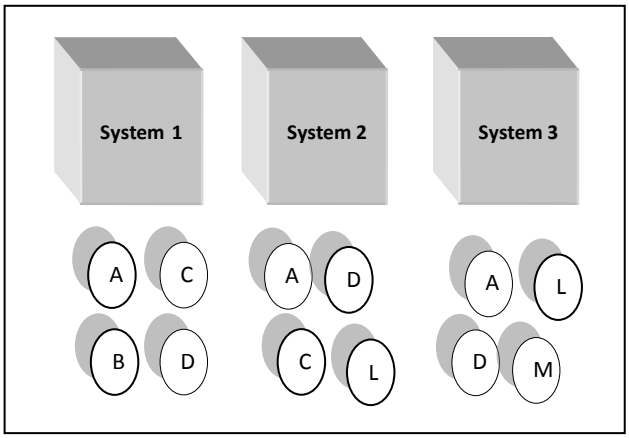

Fig. 1: Legacy System Architecture

Figure 1, illustrates this problem. As shown in this figure, there are three systems already working in the same organization. The first system contains four modules A, B, C, and D. The second one includes four modules A, C, D, and L. While, the third system consists of four modules A, D, L, and M. One can notice that the three systems are isolated, and there is no relationship between their modules. This situation leads to the following problem. If module A in system 1 needs bug fixing then the developer should fix discovered bug in all systems that use module A. The proposed system, which is presented in the following section, tries to solve this problem for saving effort, modification time and ensuring that any modifications will be reflected to all systems using a common component.

3. GenericSOA Framework

We believe that collecting reusable services to form a common library will lead to prevent replication and achieve services maintainability. To explain our idea; suppose we have four services A, C, D, and L in a common library (see figure 2). Suppose that there is a system 4, which is needed to be built, and it consists of three services A, D, and H. Common library can provide its users by A and D services from its common services. The H is the only module that will be built; the other A and D will be used from the common library saving efforts, time and ensuring the used of the modified services.

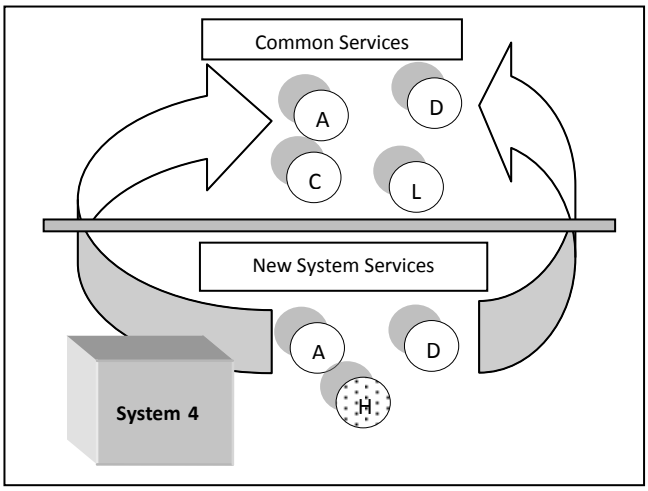

Fig. 2: Proposed system idea

Trying to enhancement SOA and easily modifying services and increase the efficiency of service usage, a proposed GenericSOA is defined. The suggested solution relays on using a services library which contains common used services. This library will be standalone from any legacy systems and any new systems. As shown in figure 3, GenericSOA includes four components; services library, Generic Services Task Model (GSTM) and Generic Services Task Templates (GSTT), as well as ontology. These components are discussed below:

- **Services library:** reusable services repository within an organization. If there is a need for new service which is not included, this service should be built and added to the library.
- **Generic Services Task Model (GSTM)**: connects to services library. So, these services can be employed as a service in any new system instead of reconstructing it from scratch. Also, any modification in these services will be done only in the services library.
- **Generic Services Task Templates (GSTT):** is built using GSTM to achieve a specific task. The building template will be used by developers to generate a specific task in the new developed system.
- **Ontology:** Using the ontology facilitate the communication and cooperation between the services. Developers used different terminologies to describe their input and output variables. Using the services without caring by these terminologies may lead to wrong results. So, the ontology cares by identified these terminologies and its synonyms to provide the users by the right meaning for the inputs, outputs, and task of the services. Ontology can help in understanding to mapping between the same variable

meanings. Researches proposed ontology web language for services and services description [9], for examples OWL-S [10][11].

Moving from the current situation to the new architecture one need time. In this stage, to facilitate converting the old system to new architecture without convert all its functions to services can be done through wrapper module. The wrapper will wrap the coding of the original function in the legacy system to use as a virtual service until moving it to real services in the services library.

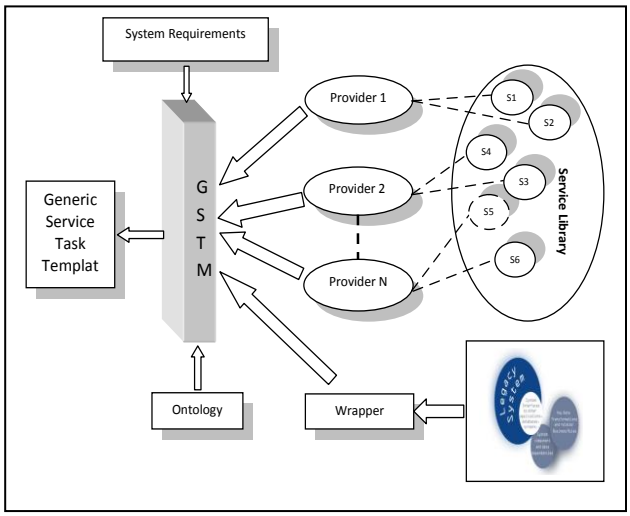

Fig. 3: GenericSOA framework

As shown in figure 3, the suggested GenericSOA framework use the system required from the user, services library, in additional to the ontology as input to its GSTM to generate its output which is the generic services task template GSTT. When needed the GSTM uses the output function from the wrapper as input service to be used in generate GSTT. The wrapper used the legacy system as input to generate the needed function to be used in new systems. The workflow of the system will be found in the following section.

While continuing the process of constructing GSTT library, building new system is done by collecting different GSTTs from it together without necessity to build services from scratch.

3.1 GSTM Description

As shown in figure 4, GSTM consists of three main components that help the system's developers to build generic service task template from different services. The following is a summarization for these components:-

- ***Service Selector:*** this component searches in the services library to choose candidate services that are suitable for performing job/duty in the required template task. The candidate services are selected according to the system requirements.

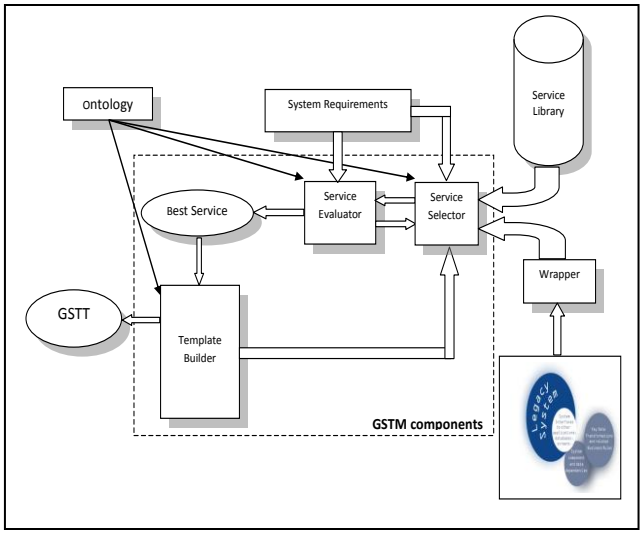

Fig. 4: GSTM Components

- ***Service Evaluator:*** this component evaluates the candidate services set which is the output from the services selector component. The best service to perform the job/duty will be the output from this component.
- ***Task Template Builder*:** the responsible of this is to build the generic service task template that can be used to solve the general problem. If a built task needs any loop, or condition part, the Task Template Builder allows its user to add it in the template body.

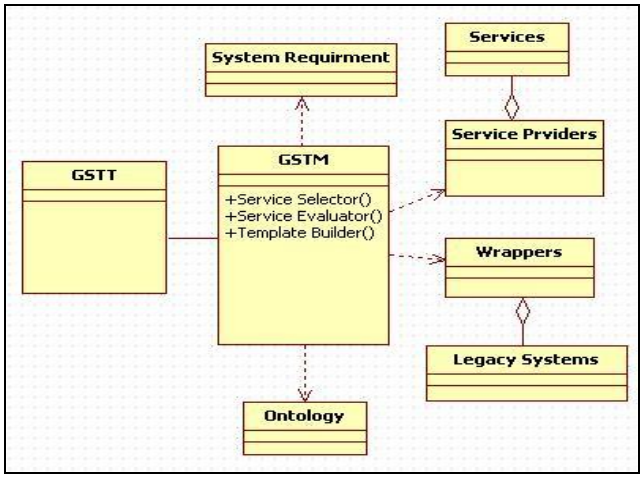

Fig. 5: GSTM Components using UML

Figure 5 illustrates the SGTM component using the UML. As shown in this figure, the GSTM connects to various service providers and wrapper which provide it by all the

existing services in the organization. According to the user requirement the GSTM generate the suitable GSTT.

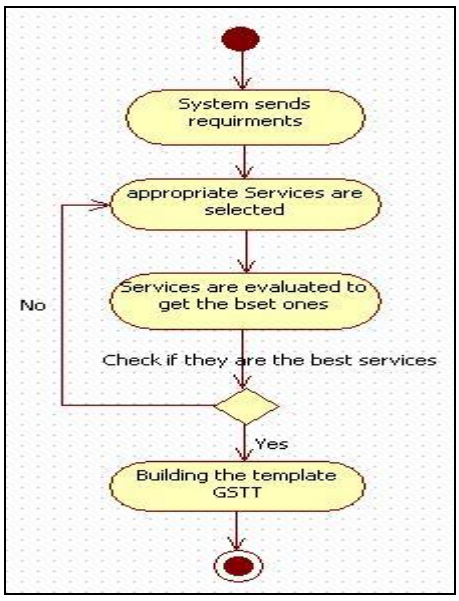

Fig. 6: GSTM workflow using UML

The GSTM receives the system requirement from its user as shown in figure 6. According to this required it select the appropriate services, which is evaluated based on some characteristic like, the time of execution, the availability of needed input parameters, and the accuracy of the service output. The service evaluator selects the best services for job or duty according to the system requirements. Finally, a GSTT is ready to be used in solving specific problem or performing special task.

The output from GSTM is a task template which can be used to generate a specific task which is used in building the organization system. Figure 7 shows the proposed representation for a GSTT.

GSTT Name:
GSTT Description:
GSTT Input:
GSTT Output:
GSTT Pre-Condition:
GSTT Post-Condition:
Task:
<Call Service N>
<if statement>
<While statement>
<Repeat statement>

Return GSTT Output

Fig. 7: GSTT Representation

As Shown in figure 7, the GSTT has a name, description of job done by it, and the needed input and the output from this GSTT. If the task needs some specific situation to execute, the Pre-condition part will include the needed situation to be achieved. Also, if it needs to ensure about some results or parameters it will be put in the Post-condition. The execution of the task can be primitive or complex. The primitive task just execute the used services with a specific order. Complex task needs to execute loops or an alternative execution of the services. In complex task, a developer add any needed condition statement (if, repeat, while).

3.2 GenericSOA Properties

There are some properties for GenericSOA which make it easy to be used:

1. GenericSOA contains ontology that facilitates the cooperation between the services and makes standardization between them to help GSTM in building the GSTT.
2. Any service in Library can be either primitive service (cannot be divided into smaller services) or composite service (contains smaller services in it).
3. Any GSTT can has pre-conditions which must be fulfilled in order to execute it.
4. Any GSTT can has post-conditions to ensure that its result is the right result.
5. Any GSTT can have some workflow to achieve its goal. This workflow contains organized services which can be either obligatory (must be execute to complete the task) or optional (its execution does not effect in task execution, but may be affect in the quality of the result).

4. Related Works

As we are saying before, SOA developers depend on building their applications using composite services. A composite service is "*a service developed by aggregating the existing services to realize a new value-added functionality*" [8]. Static and dynamic service compositions are the most popular approaches of service composition [12]. In static service composition, all the required components are determined before it is composited. This approach is preferred when the business partners are fixed and the business requirements are stable for a little while. Therefore, static approach is not suitable in case of fast changing environments [13]. On the other hand, the dynamic service composition approach is more suitable in such dynamic situations (e.g., daily added services, changing of service environment, continuous spreading, and increasing in services providers) [14]. Shortly speaking dynamic service composition is more

flexible in modifying, adjusting and expanding changes during the process runtime [15].

Researches try to enhance the SOA development by adapting the services at runtime [7] [5]. Denaro and his colleges presented an architecture, which facilitates clients to automatically adapt their performance to alternative web services [7]. The alternative Web services give compatible functionality via different interaction protocols. The infrastructure used in this work discovers the appropriate interactions of the web services, and then builds models that make the interaction protocols closer and push runtime adaptations at client-side [7].

CoBRA (Component Based Runtime Adaptable) is adaptation architecture; it enables dynamic adaptation at runtime to provide a foundation for autonomic, self-managing, self-healing, self-optimizing, self-configuring and self-adaptive applications [5]. The adaptation is done by exchanging in the implementation. So the self-managing of SOA applications to adjust services at runtime is very essential to avoid the confusion of the service availability within the application execution [5].

SOA approaches face the semantic ambiguity problem that faces Web service as general. Lécué and his colleges present a framework to perform dynamic service composition, which uses semantic meanings for services input and output [6]. Their framework uses semantic matchmaking between outputs and inputs of the service to provide interconnection and interaction between services. The main idea of this framework relies on discovering the appropriate semantic meanings between several service descriptions. They apply a composition algorithm which uses semantic graph-based approach. Their algorithm decides the related and most suitable service compositions for some service request by considering functional and non-functional properties of services [6]. While, Solanki and his colleges use ontology to solve the ambiguity problem [16]. Their framework enriches semantic service descriptions which are assumption and commitment assertions. These assertions facilitate reasoning about service composition and verification of their integration [16].

5. Conclusion

Service Oriented Architecture (SOA) depends on building applications using existing services. The new systems are built by use the existing services in the organization's old systems instead of building it from scratch in the SOA environment. Researchers try to enhancement the SOA to overcome the existing problems and to maximize its benefits. GenericSOA, the proposed architecture try to enhance SOA by gathering the needed services in separated templates to be used to achieve a specific task. It consists of four components namely: services library, Generic Services Task Model (GSTM) and Generic Services Task Templates (GSTT), and ontology. It provides its user by a way to build GSTTs to select from them for building new system. Building GSTT is done by select the suitable services to achieve the duty from the services library. The GenericSOA ontology facilitates the communication and cooperation between the services. Using GenericSOA architecture to build the SOA environment will enhance SOA. It enables its users by update and modifies the services only on the library which save time and effort as well as increase the efficiency of service usage.

References

[1] Papazoglou, M. P., Van Den Heuvel, W. J.. "*Service oriented architectures: approaches, technologies and research issues"*, The VLDB Journal, 16(3), pp.389–415, 2007.

[2] Huhns, M. N., Singh, M. P, "*Service-oriented computing: Key concepts and principles*", IEEE Internet Computing, 9(1), pp.75–81, 2005.

[3] Thomas E., "*Service-oriented architecture: concepts, technology, and design"*, Prentice Hall PTR, Upper Saddle River, 2005.

[4] Said, M., Ismail, O., Hassan, H., "*Web Service Composition and Legacy Systems: A Survey*", International Journal of Computer Applications,*69*(17), pp.9-15, 2013.

[5] Irmert, F., Fischer, T., Meyer-Wegener, K., "*Runtime adaptation in a service-oriented component model*", In Proceedings of the international workshop on Software engineering for adaptive and self-managing systems, pp.97-104, ACM, 2008.

[6] Lécué, F., Silva, E., Pires, L. F., "*A framework for dynamic web services composition*", In Emerging Web Services Technology, Birkhäuser Basel, Volume II, pp.59-75, 2008.

[7] Denaro, G., Pezzé, M., Tosi, D., Schilling, D., "*Towards self-adaptive service-oriented architectures*", In Proceedings of the workshop on Testing, analysis, and verification of web services and applications, pp.10-16, ACM, 2006.

[8] Alonso, G., Casati, F., Kuno, H., Machiraju, V., "*Web Services: Concepts, Architectures and Applications"*, Springer Publishing Company, Incorporated, 2010.

[9] Sireteanu, N. A., "*A Survey of web ontology languages and semantic web services*". Scientific Annals of the Alexandru Ioan Cuza University of Iaşi Economic Sciences, 2013

[10] W3C, Web Ontology Language (OWL) - Reference Version 1.0, 2002. Available at http://www.w3.org/TR/2002/WD-owl-ref-20021112/.

[11] W3c, The OWL-S Coalition. OWL-S 1.0 (Beta)

Draft Release, 2003.
http://www.daml.org/services/owl-s/1.0/.

[12] Khadka, R., Sapkota, B., "*An evaluation of dynamic web service composition approaches*", pp. 67-79, 2010.

[13] Shen, L., Li, F., Ren, S., Mu, Y., "*Dynamic composition of web service based on coordination model*". In Advances in Web and Network Technologies, and Information Management, Springer Berlin Heidelberg, pp. 317-327, 2007.

[14] Dustdar, S., Schreiner, W., "*A survey on web services composition*", International Journal of Web and Grid Services, *1*(1), pp.1-30, 2005.

[15] Tosic, V., Mennie, D., & Pagurek, B., "*Dynamic Service Composition and Its Applicability to E-Business Software Systems*", The ICARIS Experience.Advances in Business Solutions, pp,93-104, 2002.

[16] M. Solanki, A. Cau, and H. Zedan, "*Augmenting Semantic Web Service Descriptions with Compositional Specification*", In Proceedings of the 13th international conference on World Wide Web (WWW'04), ACM Press, pp.544–552, 2004.