ISSN:0975-9646

Improved Web Service Self-Healing Connector

Khalid Kaabneh[1], Suha Afaneh[2], Heba Almalahmeh[3], Issam Alhadid[4]
[1]Associate Proffesor, Multimedia Information Systems, Isra University, Amman, Jordan.
[2]Assistant Proffessor, Computer Science Department, Isra University, Amman, Jordan.
[3]Assistant Proffessor, Management Information System, Isra University, Amman, Jordan.
[4]Assistant Proffessor, Computer Information System Isra University, Amman, Jordan.

***Abstract*— Web service-based application is an architectural style, where a collection of Web services communicates to each other to execute processes. With the popularity increase of developing Web service-based application and once Web services may change, in terms of functional and non-functional quality of service (QoS), we need mechanisms to monitor, diagnose, and repair Web services into a Web application. The goal of this paper is to build Web service that are reliable and adaptable without the need to be modified offline to meet the changing of the users' requirements and system resources. This can be achieved by using the Web Service Self Healing Connector that provides a mechanism to supervising the traffic between Web Service providers and requesters to Monitor, dynamic run-time reconfiguration and Quality of Service (QoS) management.**

***Keywords*— *Self Healing, Web Service, Service Oriented Architecture, Quality of Service, Service Agreement Level, Availability*.**

I. INTRODUCTION

Web services technology increasingly has been used to develop the new software systems' era, by moving from module implementation to unit composition which is the base of the Service Oriented Architecture (SOA). Web service technology can reduce the time to market, as well as the Quality of Service (QoS) according to the Service Agreement Level (SAL) must be provided by the service providers that can gain the clients' reputation and increase the market share. The new technologies era increased the functionality and the complexity of the software and systems in organizations, and as a result, a high system management costs and increased systems, sub system or component(s) failures. Accordingly, there is a growing interest in Self-Healing software as a solution to solve the problems of fault tolerance, reliability, security and availability of the systems. [1, 2, 3, 4]. Naccache, Gannod and Gary [3] stated that: "*Self-healing systems must be able to recover from the failure of underlying components and services. The system must be able to detect and isolate the failed component, fix or replace the component, and finally reintroduce the repaired or replaced component without any apparent application disruption*". Keromytis [4] claimed that Self-healing software systems have emerged as a research era in the recent years, motivated by the capabilities of monitoring, diagnosis, and repair anomalies as an exciting and potential solution to the existing problems of inability of traditional technologies to guarantee the software availability, robustness, and reliability.

Traditional technologies suffer from the problem of localized error handling which might be able to determine the real problem source to take the right action. Because they are included with the system's code; this is not well suited to recognize system anomalies such as performance, difficulties of changing adapted polices, and unexpected behavior. On the Other Hand, Self-Healing provides the QoS management in order to satisfy evolving process requirements and changing constraints. Self-Healing systems change its own behavior when evaluation indicates that the required QoS is not achieved, better performance or functionality is required, or an anomalies behavior is detected. The Web service Self-healing refers to the Web services ability to automatically monitor, fault detection and diagnosis, while repairing failures by executing an action to maintain an appropriate QoS. [5, 4]. According to Robertson & Williams [5] there are three main steps in the WS self- healing process which also called the lifecycle of self-healing, shown in Fig 1.

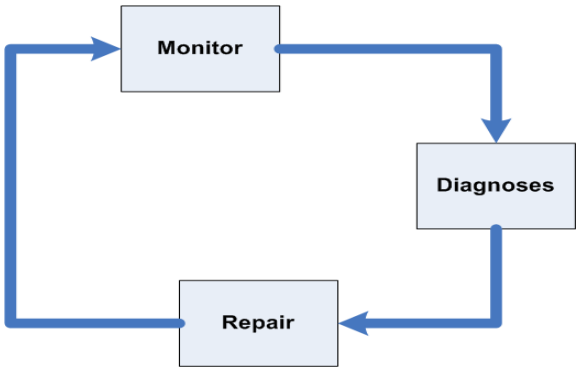

Fig 1: Self-Healing Process

Monitoring process collects and extracts information from managed service tasks accessed during the execution by track the tasks' behavior and the objects accessed by the tasks, Diagnosis process examine and analyze extracted information passed by the monitor step, anomalies occurred if service task does not execute as expected within specific time or return an error value, and Repair process heal the service by execution an action, specifies what action to be taken in order to recover from any malfunction. If accepted repair action is produced the system is updated accordingly. Ben Halima, Drira & Jmaiel [7] stated that there are three self-Healing levels; Service Level applied using extended interfaces for WS management, Flow Level applied using an extended manageable process execution engine used for orchestrated services on Business Process Execution Language (BPEL) level. And Communication level applied

using exchanged messages; by modifying the exchanged messaged (SOAP) by adding the QoS parameters values which will be applied in this paper using Web Service Connector to Configure the Self-Healing.

II. Related work

Self-Healing in the communication level is applied using middleware between the Web service and the service requester, it is used to control and enhance the messages traffic and QoS. In this section we will introduce the work that has been achieved in the communication level to provide the Self-Healing and guarantee the QoS. Halima, Drira & Jmaiel [7] proposed a healing framework which is based on service monitoring dynamic runtime reconfiguration, and architectural level repair actions; Their work is based on the communication level; implemented using a connector-based healing layer capable of intercepting, analyzing and enhancing the SOAP traffic and messages which contains the QoS data which will supports the QoS Monitoring and dynamic run-time reconfiguration to achieve the run-time QoS. They claimed that "A (QoS-centric) self-healing system inspects and changes its own behavior when the evaluation indicates that the intended QoS is not achieved, or when a better functionality or performance is required." Also, they suggested a service – level and healing messages to monitor, digenesis and repair the services. Shin & An [8] suggested an architecture that includes a healing layer in connectors. The self-healing connector contains two layers: a communication layer which manages exchanged messages, and a healing layer which reconfigures stub at connectors' level. Taher et al. [9], proposed a Web Service community which re-groups Web services having similar functionalities that addresses same users' needed into communities. This community is represented with an abstract Web service interface which is a common interface for all similar Web services. They used a mapping interface to map between the real Web Service Interface and the Abstract Web service Interface.

Vilas & Vilas [10], proposed a QoS features inside a virtual Web service called Wrapper to publish it as a standard Web service. Clients invoke this virtual Web service which is responsible for invoking real providers and mange the real Web service QoS. Naccache, Gannod & Gary [11] suggested a framework for developing an autonomic self-healing portal system that relies on the notion of differentiated services in order to survive unexpected traffic loads and slowdowns in underlying Web services. Zhou, Cai, & Godavari [12] proposed an architecture that define and assign requests into multiple user classes to differentiate the service level per request. Their approaches classify the user class of the request and assign it to the appropriate queue. If the server approaches overload, the lower class requests are dropped or delayed in order to allow the higher user class requests to go through. Almeida et al. [13] introduced a dynamic reconfiguration approach for distributed systems based on object- middleware to manipulate distributed entities replacements, migration, addition and removal Operations which provides distribution transparency and flexibility for application designers. Dabrowski and Mills [14] discussed the available Self-Healing strategies used by service-discovery systems, in addition to the effect of using a combination of strategies. This is to monitor the consistency of distributed components under varying network conditions, such as increasing network failures. The goal of this research is to provide a mechanism to supervising the traffic between Web service providers and requesters to Monitor, dynamic run-time reconfiguration and QoS management.

III. Proposed Architecture

The main goal of this paper is to provide a unified Architecture that expands the classical service oriented architecture (SOA) using Web Service Self-Healing Connector as Web service interface to guarantee the Quality of Service (QoS) and to provide the expected Web services' behavior and functionality and maintain both services integrity and availability. Fig 2 shows the proposed Web Service Self-Healing Connector Architecture.

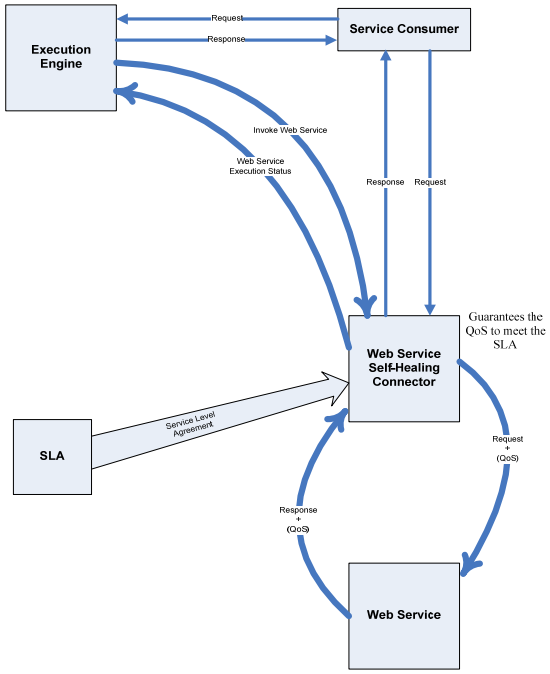

Fig 2: Web Service Self-Healing Architecture

A. *Web Service Self Healing Connector*

Web Service Self Healing Connector exposes Web services' interfaces to the clients, and allows the interaction between clients and Web services. Also, Web Service Self-Healing Connector Keeps an eye on the held requests that didn't responded, also, make sure that the response time out is not been exceeded, in addition, guarantee that the Web service will not be flooded with requests to provide the expected QoS; it will react if the Web service is hit with a higher than expected requests (flash crowed) that would slow or stop the Web service from responding to requests. As well, it will work to allocate Web service optimally and do not refuse any valid request. And finally it will map the consumer requester parameters with substituted Web

service WSDL input parameters. And as a result, it will guarantee the QoS as agreed in the Service Level Agreement (SLA) for the requested users. If the Web service executed successfully, it will send the response to the Web Service Self-Healing Connector which will send the request results to the service consumer who invoked the Web service and a notification to the Extended Execution Engine to invoked the related Web services participating in the composed service that fulfill a specific business process. The Web Service Self-Healing Connector as observer between the invoked Web service and the Service Consumer, and the invoked Web service and the Execution manager to guarantee the QoS according to the Service Level Agreement (SLA). Web Service Self-Healing Connector is responsible for the Web service QoS based on the SLA because we don't want to dump the network with redundant information. Fault detection notification will be enough.

Applying Web Service Self Healing Connector Algorithm will guarantee that Web services will not be flooded with requests to provide the expected QoS according to the Service Level Agreement (SLA).

- Req/Resp Connector Manager: Get Clients' Requests and insert QoS Parameters (service invocation time)
- WS Reconfiguration Manager: mapping of the input parameters (if necessary), and send request to the real Web service.
- Real Web service: execute and send response to Web Service Self-Healing Connector.
- WS Reconfiguration Manager: receive the Web service response and forward it to the Diagnosis Manager.
- Diagnosis Manager: analyze the Web service response:

  IF (Error, Fault, anomaly):
  - Send analyzed information to WS Healing & Reconfiguration Planner.

  Else
  - Remove the QoS parameters from the Web service response and send it to the Req/Resp Connector Manager.
  - Analyze statistically the QoS parameters and the Web service QoS History and Send results to WS Healing & Reconfiguration Planner.

  End IF
- WS Healing & Reconfiguration Planner: decide the action about the current Web service according to WS Diagnosis Manager results (anomalies detected, QoS parameters, and SLA), and send decision to WS Reconfiguration Manager.

  Actions:

  IF (Error, Fault, anomaly):
  - Send request to the Real Web service (B)
  - Send request to composed Web service.
  - Re-Invoke Real Web service (A)

  Else

  IF (SLA not satisfied, history of Web Service QoS):
  - Use Web service (B) for the future requests.
  - Keep using Web service.

  End IF
- Req/Resp Connector Manager: map the response results (if necessary), and send response to the Web service requester.

Repair Manager Healing Actions might be one of the following actions:

- Re-Invoke Web Service
- Invoke a substituted Web Service
- Invoke a composed Web service functionality equivalent
- The repair actions are generated automatically from the Web Service Definition language (WSDL) specification by substituting Web service by another functionality equivalent Web service.

Fig 3 shows the Web Service Self-Healing Connector architecture.

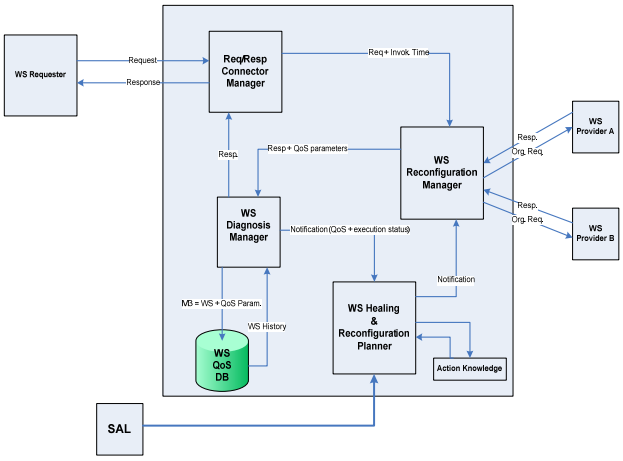

Fig 3: Service Self-Healing Connector architecture

Web Service Self-Healing Connector objects provide healing capabilities using monitor, diagnosis, and healing actions. The objects of the Web Service Self-Healing Connector are:

- Req/Resp Connector Manager: Used to Intercept the consumer request to check the sent request parameters number and types (Input faults or Type faults), also to insert the value of service invocation time QoS parameter, and forward it to WS Reconfiguration Manager. Also, send the response of the real Web service to the requesters and map results if necessary. In addition, prevent any unauthorized request to invoke the Web Service.
- WS Reconfiguration Manager: Responsible for sending requests and receiving responses of the real Web service which is selected according to WS Healing & Reconfiguration Planner, mapping process between the different Web service providers' WSDL input parameters. As well, react according to the WS Healing & Reconfiguration Planner notification to execute the healing action
- WS Diagnosis Manager: Analyze the real Web service response for any errors, in addition, analyze the QoS parameters' values and Web service QoS

history statistically and send the analyzed information to the WS Healing & Reconfiguration Planner. Also, store the QoS parameters values in the WS QoS DB. In case of no anomalies detected in the Web service response, it will send the real Web service response to the Req/Resp Connector Manager to forward it to the Web service requester.

- WS Healing & Reconfiguration Planner: Analyze the WS Diagnosis Manager results and decide the healing action in case Web service execution failure using the action knowledge. Also, compare the analyzed QoS results and the Service Level Agreement (SLA) to decide if the current Web service provides the expected service (might ask the WS Reconfiguration Manager to leave Web service "1" and bind the request to Web service "2"), and finally send the decision to WS Reconfiguration Manager.
- WS QoS Database: Used to store the Web service QoS parameters' values.
- Action Knowledge: Used to match the recommended healing action in order to heal the request.

Web Service Self-Healing Connector QoS parameters:

1. Invocation time of service by requester
2. Communication time to reach Web Service provider side taken by SOAP Message (depends on network latency)
3. Time response will take to reach the Web Service Self-Healing Connector side (network + execution).
4. Invocation time of the service by Web Service Self-Healing Connector.
5. Execution time associated with request by Web Service.

The following is a generic QoS parameter which will be included inside the SOPA message between the Web Service Self-Healing Connector and the Web Service:

```
<SHQoS>
   <QoSparam>
        <ClintRequestTimeIntercept>VALUE</ClintRequestTimeIntercept >
        <WSRequestTimeIntercept>VALUE</WSRequestTimeIntercept >
        <ConnectorRequestTime>VALUE</ConnectorRequestTime>
        <WSResponseTimeIntercept>VALUE</WSResponseTimeIntercept>
        <WSFinishExecutionTime>VALUE</WSFinishExecutionTime>
   </QoSparam>
</SHQoS>
```

Diagnosis Manager will use Web service response QoS parameters and Web service QoS history to analyze the Web service behavior, the analyzed data will extract the following:

- Availability: availability of a service: availability rate, mean time to repair, mean time between failures.
- Throughput: The amount of requests that can be processed in a specified period of time
- Communication Time: The round trip time of a request and its response.
- Accuracy: The success rate produced by the service.
- Execution Time: The time for processing a request.
- Response Time: The time between sending a request and receiving its response.
- Accessibility: Ability of a service to process a given request.
- Reliability: the ability of a service to keep operating over time, characterized by availability/accessibility and successful execution rate
- Performance: productivity of a service: throughput, latency, response time.
- Security: check the client if authorized to invoke the Web service or access the Web service resources. As well as, an encryption technique might be used to increase the security features.

Fig 4 shows the Web Service Self-Healing Connector's components at work, the WS Reconfiguration Manager controls the incoming and outgoing invocations, in addition to executing the suitable healing actions recommended by WS Healing & Reconfiguration Planner. This is decided according to QoS parameters, SLA, and Web service's history analyzed by WS Diagnosis Manager.

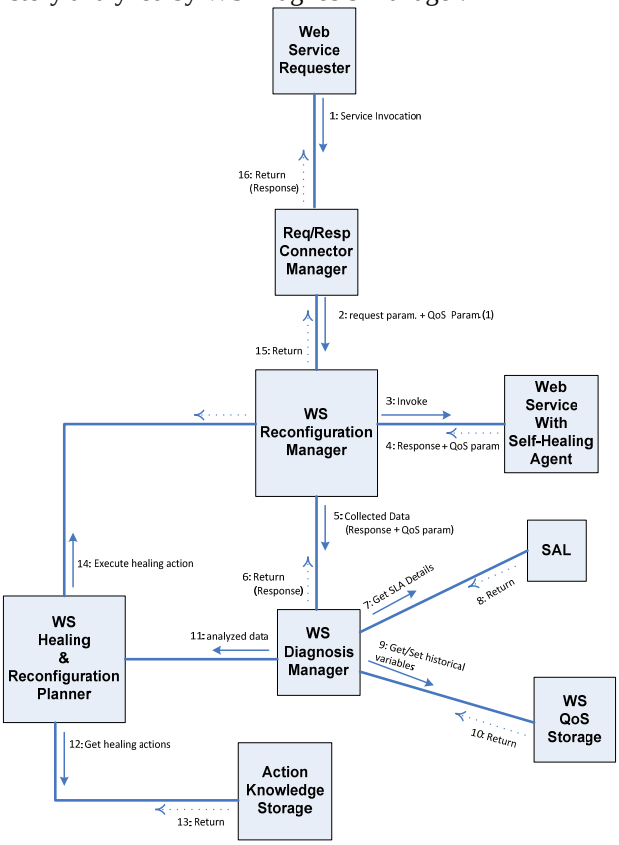

Fig 4: Web Service Connector's Components at Work

The request of the invoker will be intercepted by the Req/Resp Manager to add the first QoS parameter (received time). The request with the added QoS parameter will be forwarded to the WS Reconfiguration Manager to invoke the real Web service. When the response is received, it will

be sent to the Diagnosis Manager to analyze the result and compare the QoS parameters with the SLA contract. According to the analyzed data, and if an anomaly is detected, the WS Healing & Reconfiguration Planner will use the Action Knowledge to generate a healing strategy, and apply that action to repair the detected anomaly through the WS Reconfiguration Manager. If the healing action is successful, the Diagnosis Manager will send the result to the Req/Resp Manager to respond to the invocation's requester. If the healing action is not successful, then the next healing action will be applied and so on until no more actions are available. In this case, an error response will be sent to the invocation's requester that the request cannot be fulfilled.

B) Web Service Mapping

The substituted Web service must be functionality similar, fulfill the same users' needed at an abstract level, so it can take the place of the other Web service. Web service structure and behavior might be a problem in incompatibilities between Web services even if address the same functionality. Structure functionality focuses on parameters name, ordered, and type. On the other hand, behavior focuses on the execution order of the operations [9].

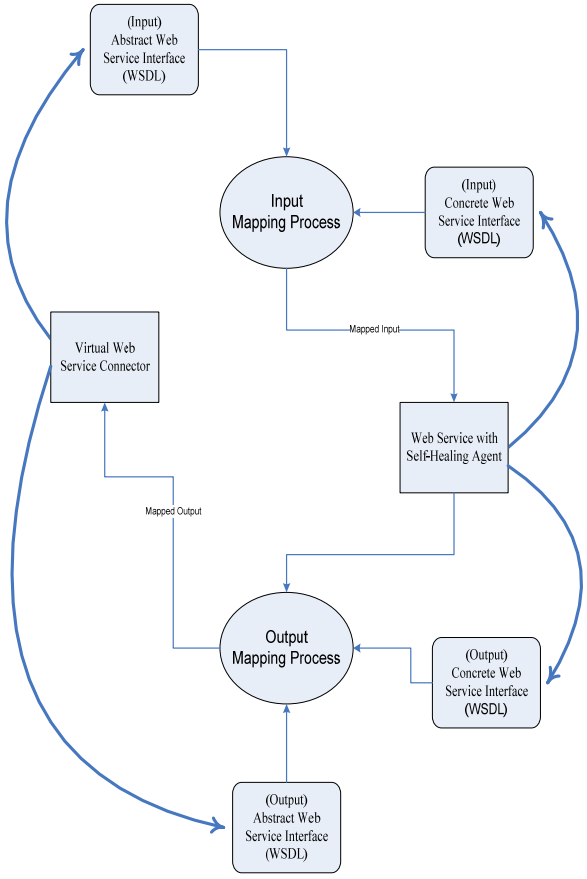

Fig 5 illustrates in general the mapping process.

Fig 6 shows a detail process for the Web Service Self-Healing Connector and the Repair Manager healing action taken to heal the anomalies detected during the process execution.

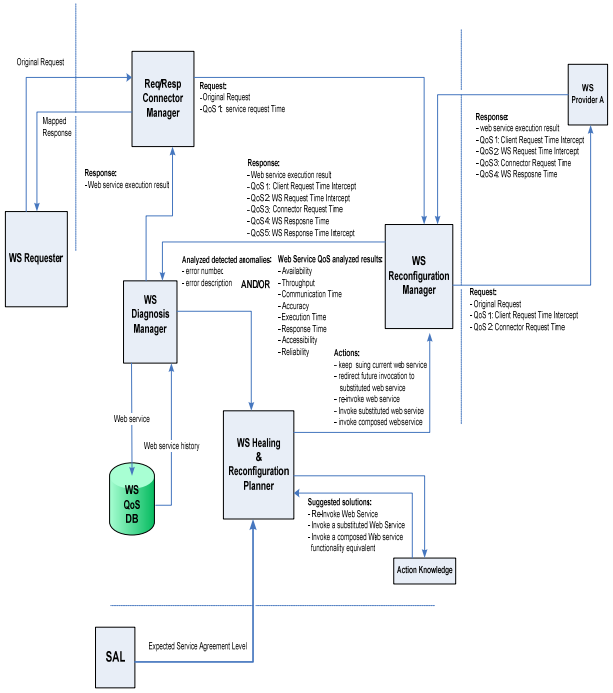

Fig 6: Web Service Self-Healing Connector Process

For example, FlightBooking is the web Method in airline reservations systems form company X, while company Y Web Method's name is FlightReservation for its airline reservations system. Also, the name and type of WSDL input parameters are different for each Web Method. To avoid such problem, Web Service Self-Healing Connector will provide a dynamic binding mechanism between the abstract and concrete Web service parameters and operations, to achieve the required functionality. Mapping process is shown in Table 1.

TABLE I
MAPPING PROCESS

Abstract Web Service Interface provided by Virtual Web Service Connector	**Concrete Web Service Interface provided by real Web Service**	**Mapping**
Operation: FlightBooking Input Parameters: - FName: String - LName: String - Gender: String - From: String - To: String - DateTime: String Output: - JDCost: Float	Operation: FlightReservation Input Parameters: - Name: String - Gender: integer - From: integer - Destination: integer - DateTime: DateTime Output: - DollarCost: Float	Input Mapping: - Name: MapName (FName, LName) - Gender: MapGender(Gender) - From: MapFrom(From) - Destination: MapDestination(To) - DateTime: MapDateTime (DateTime) Output Mapping: - JDCost= MapJDCost(DollarCost)

XQuery language can be used to implement the adaption on the XML SOAP message to map the input, output, and operations names and types. It is used to retrieve and interpret information from XML Data Sources [15].

The following is the Pseudo code for implementing the mapping process:

```
MapGender(Gender):
    If Gender = "Female"
        Return 1
    Else
        Return 2
MapName(FirstName, LastName):
    Return FirstName & " " & LastName
```

The implantation of Web service mapping adaption using Web Service Self-Healing Connector:

```
Receive:
    FlightBooking (FName, LName, Gender, From, To, DateTime)

Mapping:
    Name ← MapName (FName, LName)
    Gender ← MapGender (Gender)
    From ← MapFrom (From)
    Destination ← MapDestination (To)
    DateTime ← Map DateTime (DateTime)

Invoke FlightReservation [Input] (Name, Gender, From, Destination, DateTime) [Output] (DollarCost)
Mapping:
    JDCost ← MapJDCost (DollarCost)

Reply FlightBooking (JDCost)
```

Request sent to Real Web Service by Web Service Self-Healing Connector

```
<Soapenv: Envelope>
  <Soapenv: Header>
    </ SHQoS >
        {Output Parameters of the Web Service}
  <Soapenv: Body>
</Soapenv: Envelope>
```

III. IMPLEMENTATION

The process of E-Ticket booking will be implemented with and without the three levels self-healing, the results will show the effect of self-healing and if it's preferable to apply self-healing with all Web services' implementation. The process will start when a client sends the request to book a ticket, she/he will insert the required fields, such as; from, destination, departure date, return date, in addition to his/her credit card number which will be used for billing and invoicing operations. The process of booking, credit check, E-payment, and billing will be assigned to different Web services which will work as interfaces for the systems that will execute the processes. The Web Service Self-Healing Connector for each Web service will be used to guarantee the service availability. The E-Ticketing business process consists of the following set of participating Web service: Booking, Credit Check, E-Payment, and Billing, using the mentioned business process the clients can use, and the available composition service

```
  </Soapenv: Header>
  <Soapenv: Body>
     <SHQoS>
      <QoSparam>
        <ClintRequestTimeIntercept>20091117230510</
        ClintRequestTimeIntercept >
      </QoSparam>
      </ SHQoS >
        {Input Parameters for the Web Service: based on
        the mapped parameters above}
  <Soapenv: Body>
</Soapenv: Envelope>
```

Where the value 20091117230510 stands for: Year, Month, Day, Hour, Minute, and seconds of the client request interception time. The response that will be sent by the Web Service to the Web Service Self-Healing Connector holding the QoS Service parameters will be as follows:

```
<Soapenv: Envelope>
  <Soapenv: Header>
  </Soapenv: Header>
  <Soapenv: Body>
    <SHQoS>
      <QoSparam>
        <ClintRequestTimeIntercept>20091117230510</
        ClintRequestTimeIntercept>
        <ConnectorRequestTime>20091117230511</Con
        nectorRequestTime>
        <WSRequestTimeIntercept>20091117230513</W
        SRequestTimeIntercept>
        <WSFinishExecutionTime>20091117230518</W
        SFinishExecutionTime>
        <WSResponseTimeIntercept>20091117230520</
        WSResponseTimeIntercept>
      </QoSparam>
```

for ticket booking. Also, the Airline companies can trace all booking, credit check, payment, and billing data. Fig 7 shows booking E-ticket process.

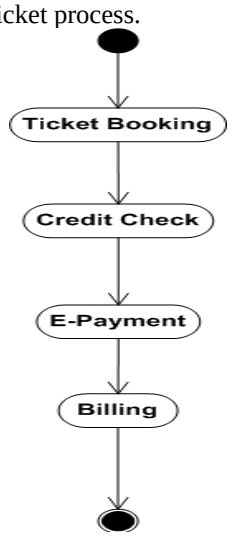

Fig 7: E-ticket Booking Process

Late we ran the E-Ticket business process with and without Web Service Self-Healing Connector. The business process had been invoked 100 times with random faults, and the program's execution was forced to invoke the participant

Web services. There were random errors with 10% of the invocations. Fig 8 shows the total number of invocations, number of invocations without errors, and number of invocations with errors.

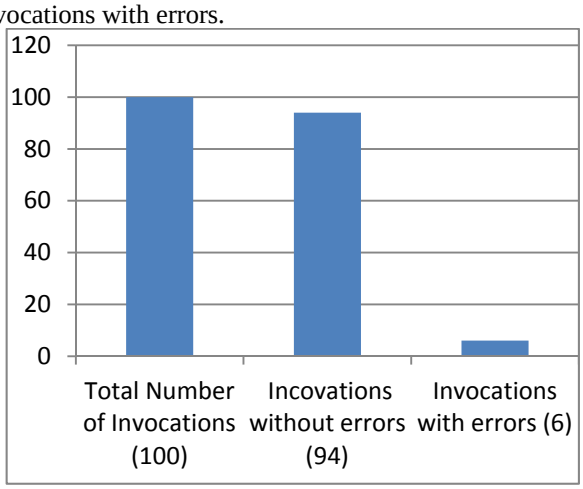

Fig 8: Execution Engine Invocations without Web Service Self-Healing Connector

In addition to the injected errors in the first run, which was without the Self-Healing capabilities, the results show there are unexpected errors in the program execution caused by the service unavailability, and unmapped input and results. Fig 9 shows the number of failed and successful invocations.

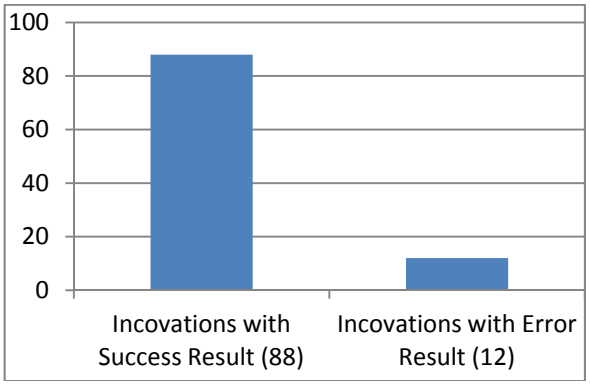

Fig 9: Invocations' Results without Web Service Self-Healing Connector

Also Fig 10 shows the execution time of the whole business process without the Self-Healing. The invocation with errors are shown with long time delay in the execution time, in addition, it returned an error responses.

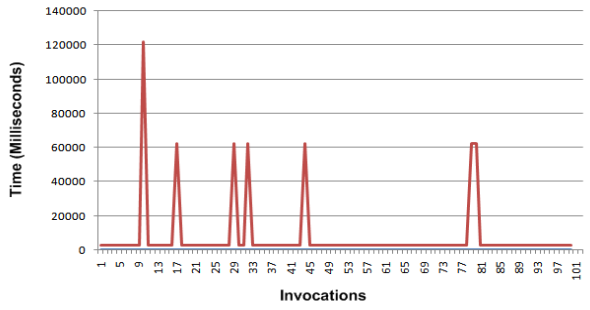

Fig 10: Invocations' Execution Time without Web Service Self-Healing Connector

On the other hand, the program execution with Self-Healing capabilities showed the program's behavior remained the same even in the presence of the randomly injected errors. Fig 11 shows the total number of invocations, number of invocations without errors, and the number of invocations with errors.

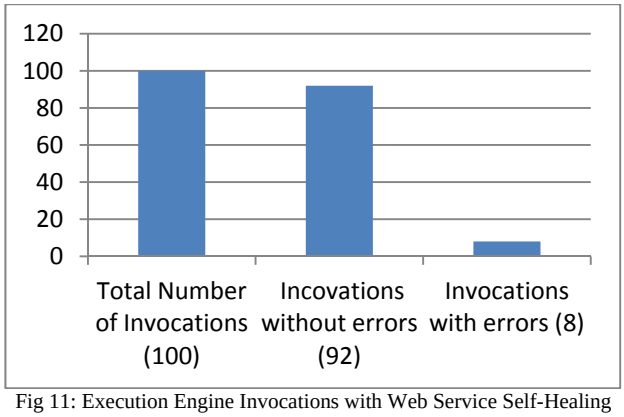

Fig 11: Execution Engine Invocations with Web Service Self-Healing Connector.

Fig 12 shows the number of failed and successful invocations in the presence of the Self-Healing. The Figure shows that there were no failures during the business process execution.

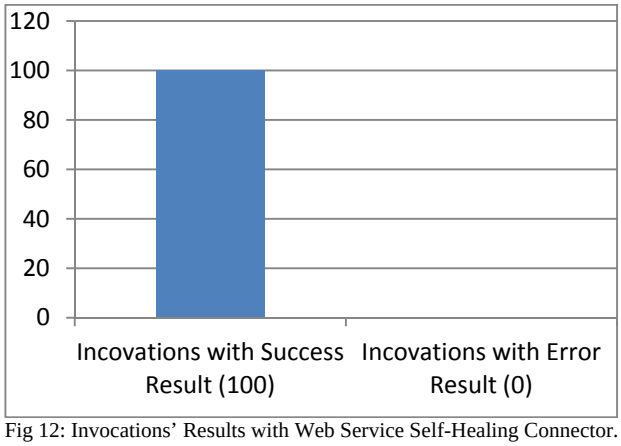

Fig 12: Invocations' Results with Web Service Self-Healing Connector.

Also, Fig 13 shows the execution time of the whole business process execution with the Self-Healing capabilities. The Figure shows that even in the presence of injected errors in the invocations. The maximum executions' time is less than the same invocations with errors without Self-Healing. In addition, all the results returned a valid response without errors.

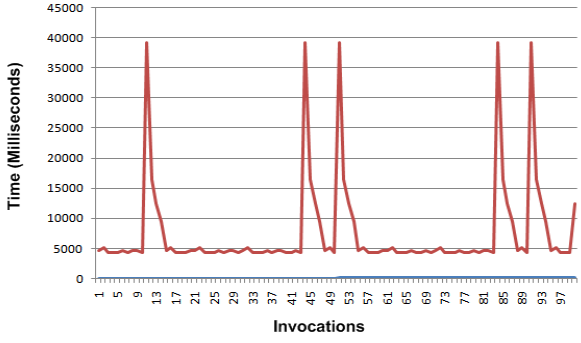

Fig 13: Invocations' Execution Time with Web Service Self-Healing Connector

According to the run results shown in Fig 10 and Fig 13, the minimum execution time in the first run without Self-healing is 2890.625 milliseconds and the maximum is 122000 milliseconds. The average execution time is 7660.15625 milliseconds with 12 failures. On the other hand, the second run with Self-healing minimum execution time is 4343.75 milliseconds, the maximum is 39156.25 milliseconds. The average execution time is 7607.80875 milliseconds with no failures. Implementations' Execution time with and without Self-Healing is shown in Fig 14.

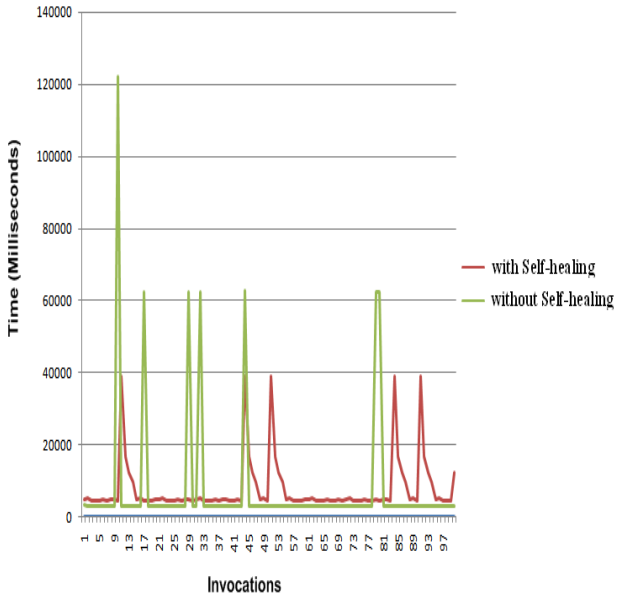

Fig 14: Implementations' Execution Time With and Without Web Service Self-Healing Connector

The proposed architecture with Self-Healing implementation increased the availability of the participant Web services, as well as the reliability. All invocations were successfully executed even with presence of the errors, in addition to the average execution time is better than the one without self-healing. Also, the maximum execution time, which indicates that an error occurred during the execution, has been healed successfully. Table 2 shows a comparison between the runs with and without self-Healing capabilities.

Table 2 shows that with Self-Healing the invocations were all executed successfully, while without Self-Healing, returned 12 failures. On the other hand, the minimum execution time without using Self-Healing is better; around 1453.125 milliseconds less than the minimum execution time using Self-Healing; because we used the Web Service Self-Healing Connector in the Self-Healing architecture. The maximum execution time using the Self-Healing is three times better three than the maximum execution time using the architecture without Self-Healing, which at the same time returns a valid value, without Self-Healing return errors. Finally, the average execution time using Self-Healing for all invocations is better than the average execution time without Self-Healing.

TABLE II
COMPARISON BETWEEN RUNS WITH AND WITHOUT SELF-HEALING CAPABILITIES

Category	Without Self-Healing	With Self-Healing
Number of all invocations	100	100
Number of invocations without errors	94	92
Number of invocations with errors	6	8
Number of success invocations	88	100
Number of failure invocations	12	0
minimum execution time	2890.625 millisecond	4343.75 millisecond
Maximum execution time	122000 millisecond	39156.25 millisecond
average execution time	7660.15625 millisecond	7607.80875 millisecond

IV. CONCLUSION

In this paper we argued that SOA architectures require more monitoring and controlling during the run-time. In fact, since the components of the Service Oriented applications and their interconnections may change after deployment, the traditional error handling is not enough to guarantee that the application will satisfy the required quality requirements. We have proposed a SOA with new Self-Healing Architecture with healing capabilities that provides Services' interface which monitors the invocations and make sure that the required QoS has been achieved. In addition to increases the Web service's availability, reliability, accuracy, and guarantee the expected behavior.

REFERENCES

[1] Andrzejak, A., Geihs, K., Shehory, O., and Wilkes, J., "Self-Healing and Self-Adaptive Systems", *In Proceedings 09201 Combinatorial Scientific Computing*, 2009.

[2] Horn P., "autonomic computing : IBM's Perspective on the State of Information Technology", 2001, IBM Research. Retrieved August 25, 2009 from http://researchweb.watson.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[3] Shehory, O., "A Self-healing Approach to Designing and Deploying Complex, Distributed and Concurrent Software Systems", in *Programming Multi-Agent Systems*, Vol. 4411/2007, R.H. Bordini et al. (Eds.). Berlin: Springer, 2007, pp. 3–13.

[4] Keromytis, D. A., "Characterizing Software Self-healing Systems", in *Computer Network Security*,Vol 1, V. Gorodetsky, I. Kotenko, and V.A. Skormin (Eds.). Berlin: Springer, 2007, pp. 22–33.

[5] Al-Hadid, I. (2012), Improved Airport Enterprise Service Bus with Self-Healing Architecture. In A. El-Sheikh, M. Jafari, E. Abu Taeeh, *Technology Engineering and Management in Aviation: Advancements and Discoveries*, Pennsylvania: IGI.

[6] Robertson, P., and Williams. B., "Automatic recovery from software failure", *Communications of the ACM*, Vol. 49, No. 3, pp.41– 47, 2006.

[7] Ben Halima, R., Drira, K., and Jmaiel, M., "A QoS-driven reconfiguration management system extending Web services with self-healing properties", in *16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 07,)* 2007, pp. 339-344.

[8] Shin, M., and An, J., "Self-reconfiguration in self-healing systems", *In Proceedings of the Third IEEE International Workshop on*

Engineering of Autonomic & Autonomous Systems (EASE'06), 2006, pp. 89–98.

[9] Taher, Y., Benslimane, D., Fauvet, M.C., and Maamar, Z., "Towards an Approach for Web services Substitution", *In Proceedings 10th IEEE International Database Engineering and Applications Symposium (IEEE IDEAS 200*6), 2006, pp. 166-173.

[10] Vilas, J., Arias, j., and Vilas, A., "An architecture for building Web services with quality-of-service features", *In Proceedings of the 5th International Conference on Web-Age Information Management (WAIM 2004)*, 2004.

[11] Naccache, H., Gannod, G., and Gary, K., "A Self-healing Web Server Using Differentiated Services", *In Proceedings of the 4th International Conference on Service Oriented Computing (ICSOC 2006)*, 2006.

[12] Zhou, X., Cai, Y., and Godavari, G, "An adaptive process allocation strategy for proportional responsiveness differentiation on web servers", *In IEEE International Conference on Web Services (ICWS 2004)*, 2004, pages 142–149.

[13] Almeida, J.P.A., Wegdam, M., Pires, L.F., and Sinceren, M.V., "An approach to dynamic reconfiguration of distributed systems based on object-middleware", *In Proceedings of 19 Brazilian Symposium on Computer Networks (SBRC'2001*), 2001, pp. 1589–1621.

[14] Dabrowski, C., and Mills, K., "Understanding Self-healing in Service-Discovery Systems", *In Proceedings of the first workshop on Self-healing systems*, 2002, pp. 15 - 20.

[15] XQuery, "An XML Query Language", Retrieved October 25, 2009 from http://www.w3.org/TR/xquery/