Asian Journal of Information Technology 15 (14): 2526-2530, 2016
ISSN: 1682-3915
© Medwell Journals, 2016

Prioritizing Components of Real-Time Dynamic Reconfigurable Software Systems for QoS Improvement

[1]P. Sudhakar, [2]Senthil Prakash and [1]G. Mohana Prabha
[1]Department of Information Technology, M.Kumarasamy College of Engineering, Thalavapalayam, 639113 Tamil Nadu, India
[2]Department of Computer Science, Shree Venkateshwara Hi-Tech Engineering College, Otthakkuthirai, 638455 Tamil Nadu, India

Abstract: The problem of service disruption in dynamic reconfiguration has not been studied well for the quality assurance. The dynamic reconfiguration must reduce the level of system transformation without disturb the service that much. We propose a new dynamic reconfiguration strategies for the components of the software system which will be prioritized based on the completeness measure. We use the state of cob for the selection of service for the user request and which helps the dynamic reconfiguration of components without disturbing the ongoing services. The completeness measure is computed based on the frequency of successful completion and failure completion of the service. We broadly explore the service reconfiguration strategies and we describe the model for reconfigurable components. The proposed component model is evaluated with the benchmark methods of reconfigurations. The notable inference of our examination is that the classified QoS physiognomies can be fully achieved under some acceptable restraints.

Key words: Dynamic reconfiguration, prioritized components, QoS assurance, system evolution, cob

INTRODUCTION

The component based model which is followed by the software systems has more impact in the area of information technology and in various area of the software computing. Whatever, the automation process employed in the real world industry could be viewed as a bundle of components. For example, an ATM machine of any bank consists of a set of interfaces for specified components to get access to the accounts of the user. The customer is able to withdraw the money from his/her account through the interface of the components available. What happens while withdrawing the money is the customer is requested to feed the card and then account pin number and other options. So, there is a set of functionality exist with the component of banking software which is available at the ATM.

The management of the banking system may think to adapt some other functionality to be adapted with the component available at the software system. The service disruption occurs at modification or updating the component which affects the quality of service parameters. The service availability which got affected at the time of component updating because the system will be shut down for reconfiguration. The ongoing transactions and services will get affected due to the service reconfiguration because they end up in an incomplete state. These factors have to be considered while reconfiguring services.

The dynamic reconfiguration is the process of reconfiguring the services without shutting down the whole software system. The thing is the software system has to maintain backup services to be provided under the minimum service requirement. The dynamic reconfiguration has to be performed at the back end without disturbing the ongoing services and service availability (Raskhe and Polze, 2008; Wurthinger *et al*., 2010a, b).

The components of the software system have to be prioritized in some way to provide service to the user in an efficient way. There may be thousands of services available but the successes of the software system depend on providing the correct service for the correct user so that the services has to be prioritized in providing access to the user and to support user goals.

Background: In quality of service assurance for dynamic reconfiguration of component-based software systems, a proof-of-concept model called the reconfigurale component model is employed to assist the illustration

Corresponding Author: P. Sudhakar, Department of Information Technology, M. Kumarasamy College of Engineering, Thalavapalayam, 639113 Tamil Nadu, India

and testing of the reconfiguration policies. Second, a reconfiguration benchmark is proposed to expose the whole spectrum of QoS problems. Third, each reconfiguration strategy is verified against the standard and the testing results are evaluated.

Host side dynamic reconfiguration with infiniBand (Guay *et al*., 2010) report an employment of flexible/runtime reconfiguration of such host side data-structures. The scheme jams the queue pairs and lets the application run with no disturbance. With this kind of operation, an inclusive result to fault tolerance in an InfiniBand network has been established, where dynamic network reconfiguration to a topology-agnostic routing function is used to evade and duck faulty components. This result is in source describes that let applications run repeatedly on the cluster as long as the topology is truly associated. With the help of dimension on test cluster, increased cost of proposed system in setup is slight and there is only a tiny concession in throughput during reconfiguration.

Dynamic reconfiguration for Java applications using AOP (Kim and Bohner, 2008) is delivered to reduce the efforts of software engineers which is to allow automated dynamic reconfiguration which is to ensure the integrity of software systems. The key area of the research is component-based application and addition, removal and replacement of components takes place.

Reliable dynamic reconfigurations in a Reflective component model (Leger *et al*., 2010), delivered a explanation for consistency configurations and reconfigurations in fractal element structural design with a design oriented on integrity constraints for example structural invariants. Consistency of reconfigurations is confirmed thanks to a transactional method which permits us both to promote with error recovery and to manage distributed and concurrent reconfigurations in Fractal applications.

DynaQoS-RDF (Li, 2011, 2012) delivered various QoS plans and offers a standard for valuing QoS-assurance plans for the dynamic reconfiguration of dataflow process. This benchmark is implemented using the DynaQoS©-RDF v1.0 Software platform. Several plans, containing those from the research works are benchmarked, and the best efforts for QoS-assurance are recognized.

Emergence of component based software engineering (Mandal and Pal, 2012) is to gain attention towards this new component based software development paradigm and to highlight the benefits and impact of the approach for making it a successful software development approach to the concerned community and industry. Researchers presents various readings of reusability and presents a approach of reconfiguring the victim workings (Basha and Moiz, 2012). The CBO measure helps in identifying the component to be reconfigured. The proposed strategy is simulated using HR portal domain specific component system.

A strategy to identify components using clustering approach for component reusability, presents a strategy has been proposed for the identification of a business component using clustering methodology. This approach will be useful in identifying the reusable components for different domains. The proposed approach has identified the reconfigured component using the CBO measure to reduce the coupling between the objects. By considering this proposed strategy, the productivity can be increased in the organization.

MATERIALS AND METHODS

Component model: The component is a collection of software interfaces through which we can access some software module or set of services. From Fig. 1 the component has interfaces to access the services named message and encryption. The dynamic configuration of this component may be addition of some services or removal or updating a service. In a dynamic reconfiguration of the components are handled in such a way that the service provider does not bother about the assurance of quality of services. We focused on providing quality of service assurance with state of the art (Truyen *et al*., 2008).

Dynamic reconfiguration model: The proposed dynamic reconfiguration model has three different working modules which help addition, deletion and updating of components without disturbing the ongoing services. The proposed method uses the active and passive states of the components for each of the process to perform (Fig. 2).

Addition: The addition of an interface and service is performed on any component using two strategic approaches. One is new versioning technique with the new service added to the component and another one is activate and passive state of the component. Initially a new version is generated and the source component of the new version is compiled and added to the component base. The component base maintains details of components deployed to the system and its version details.

The new component added will not disturb the ongoing services of the existing components. The newly added component will be taken for service only at the next service request which is handled by the service prioritizer (Alogrithm A).

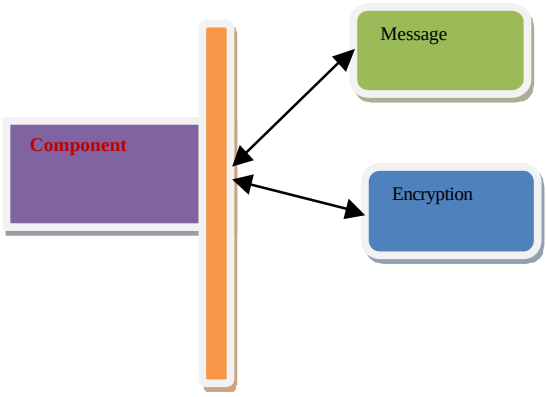

Fig. 1: Example component model

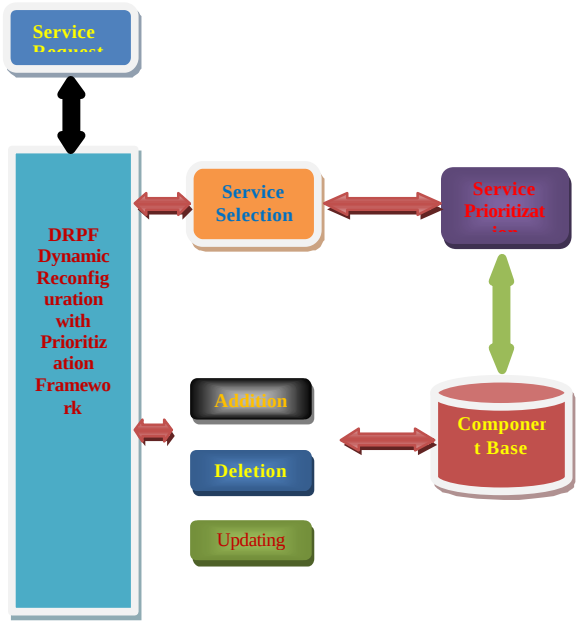

Fig. 2: Proposed architecture

Algorithm A deletion of service:

Step1: start
Step2: read component base CB.
Step3: read component details CD.
Step4: if Component Addition Request then
Create new version CBV.
CBV= {Parameters, Services,
Interfaces, NewService};
Compile Component.
Deploy New Component version.
End.
Step5: set component state as active.
Ste6: look for previous request to end
Set previous version state as Passive.
Step7: stop.

Deletion: The deletion of a component or service occurs when it is outdated. The deletion of a service from a component can be performed dynamically without disturbing the nature and ongoing request and service handling. The deleted service from the component will be getting effective at the new service request. The deletion of service will be performed using the state of art like active and passive state of the component. Initially the component state will be set as passive which will not be available for the next new request and then will be removed or deleted and the newly compiled component will be deployed (Alogrithm B).

Algorithm B deletion service:

Step 1: start
Step2: read component base CB.
Step3: read component details CD.
Step4: if Component Deletion Request then
Pcb=Check for previous version of component.
If Pcb exist then
Set state of Pcb active.
End
Set CB state as Passive.
Read component CB..
CB= {Parameters, Services, Interfaces };
Delete service CB(s) from CB.
CB = CB×CB(Ser).
Compile Component CB.
Deploy CB.
Set CB state as Active.
Set Pcb state as Passive.
End.
Step5: stop

Service updating: The dynamic updating of object configuration is performed as like the deletion and addition of software components and will not be disturb the ongoing service requests using the active and passive states of software components. The prioritization of components is the key here which helps the all features of the dynamic real time reconfiguration of the software systems.

Component prioritization: The service selection is performed on receiving the service request from the user side. Initially the set of available services is retrieved from the component based and from the retrieved service set we identify the services and components with active and passive state. For the requested component and service we identify sets of available active components.

For the active component set we compute the service completeness measure which shows the reliability of the service on user access. Based on the service completeness measure, we select the top valued component to serve the user request. The service reconfiguration procedure does not affect the service handling and prioritization because it uses the state of art as the key for everything (Fig. 3).

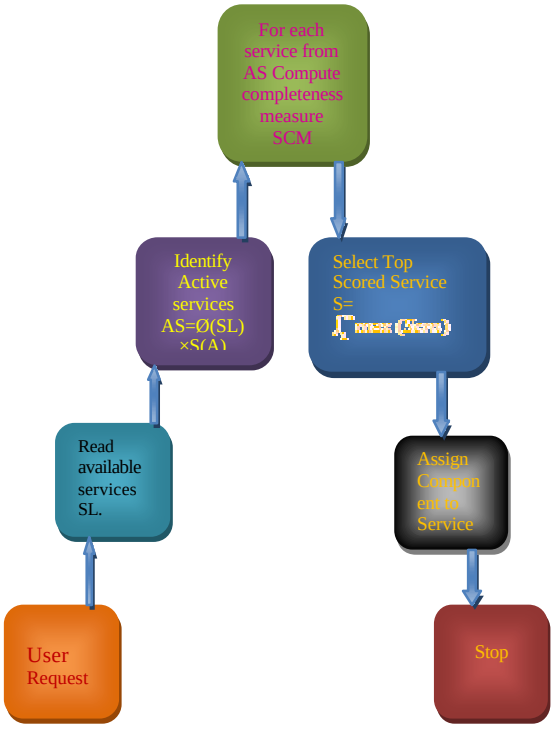

Fig. 3: Flow diagram of component prioritization

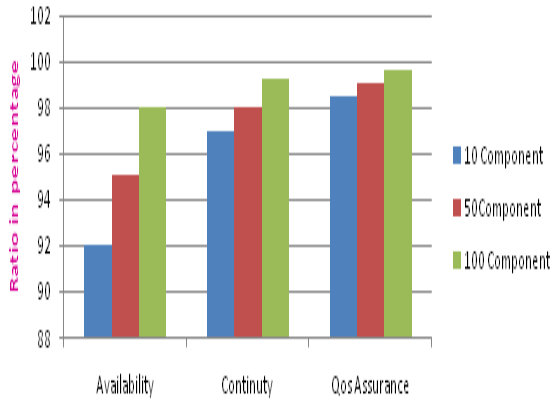

Fig. 4: Shows the quality of service assurance parameters

RESULTS AND DISCUSSION

Experimental analysis: Testing process has been proposed between or on two systems, one is called single processor system and the other is called as two-processor system. Both systems were designed with the same version of Microsoft Windows 7 operating system and the same Java SE 6 runtime. The variant platforms were selected to exhibit the individuality of RCM

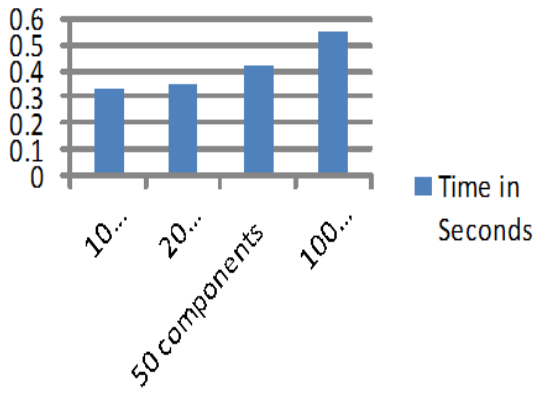

Fig. 5: Shows the response time of the proposed approach

Table 1: Comparison of reconfiguration overhead

Method	Number of components	Time taken seconds	Reconfiguration overhead
Tranquility	10	1.8	21
REDAC	10	1.5	18
Dynamic Reconfiguration	10	0.85	9
Prioritized	10	0.35	3.6

reconfiguration plans and the reproducibility of outcome. On system-1 (Processor from Intel (Core 2 Duo) and Memory of 2 GB RAM), the testing platform was produced for CPU saturation, while on machine-2 (P4 from, 3.00 GHz Processing speed and limited memory of 1 GB RAM), the platform was produced for CPU non saturation (Fig. 4).

Figure 4 shows the quality of service parameters achieved by the proposed method at different number of components deployed. We evaluate the proposed approach by deploying a number of components and tested by generating huge number of requests generated from different machines. The results show that the proposed approach has produced a higher rate of quality of service than other methods (Fig. 5).

Table 1 shows the overhead introduced by different algorithms by performing reconfiguration of components. It is clear that the proposed prioritized dynamic reconfiguration approach has produced less overhead than other methods.

CONCLUSION

We proposed a new real time dynamic reconfiguration strategy where the services or components are prioritized. We have implemented a new service prioritization algorithm which sorts the service based on service completeness measure. The service completeness

measure has been computed using the service history of components and services about the state of finishing. This helps us to reconfigure the components and we used state of art which shows the state of components as whether it is active or passive. The proposed method has increased the efficiency of the reconfiguration and produced good results.

REFERENCES

Basha, N.M. and S.A. Moiz, 2012. A methodology to manage victim components using CBO measure. Intl. J. Software Eng. Appl., Vol.3,

Guay, W.L., S.A. Reinemo, O. Lysne, T. Skeie and B.D. Johnsen *et al*., 2010. Host side dynamic reconfiguration with infiniband. Proceedings of the 2010 IEEE International Conference on Cluster Computing, September 20-24, 2010, IEEE, New York, USA., ISBN: 978-1-4244-8373-0, pp: 126-135.

Kim, D.K. and S. Bohner, 2008. Dynamic reconfiguration for Java applications using AOP. Proceedings of the IEEE Conference on SoutheastCon, April 3-6, 2008, IEEE, New York, USA., ISBN: 978-1-4244-1883-1, pp: 210-215.

Leger, M., T. Ledoux and T. Coupaye, 2010. Reliable Dynamic Reconfigurations in a Reflective Component Model. In: International Symposium on Component-Based Software Engineering. Grunske, L., R. Reussner and P. Frantisek (Eds.). Springer Berlin Heidelberg, Heidelberg, Germany, ISBN: 978-3-642-13238-4, pp: 74-92.

Li, W., 2009. DynaQoS-RDF: A best effort for QoS assurance of dynamic reconfiguration of dataflow systems. J. Software Maintenance Evol. Res. Pract., 21: 19-48.

Li, W., 2011. Evaluating the impacts of dynamic reconfiguration on the QoS of running systems. J. Syst. Software, 84: 2123-2138.

Li, W., 2012. QoS assurance for dynamic reconfiguration of component-based software systems. IEEE. Transac. Software Eng., 38: 658-676.

Mandal, A. and S.C. Pal, 2012. Emergence of component based software engineering. Int. J. Adv. Res. Comput. Sci. Software Eng., 2: 311-315.

Mandal, A., 2009. BRIDGE: A model for modern software development process to cater the present software crisis. Proceedings of the IEEE International Conference on Advance Computing, March 6-7, 2009, IEEE, New York, USA., ISBN: 978-1-4244-2927-1, pp: 1617-1623.

Rasche, A. and A. Polze, 2008. ReDAC-Dynamic reconfiguration of distributed component-based applications with cyclic dependencies. Proceedings of the 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), May 5-7, 2008, IEEE, New York, USA., ISBN: 978-0-7695-3132-8, pp: 322-330.

Truyen, E., N. Janssens, F. Sanen and W. Joosen, 2008. Support for distributed adaptations in aspect-oriented middleware. Proceedings of the 7th International Conference on Aspect-Oriented Software Development, March 21-April 4, 2008, ACM, New York, USA., ISBN: 978-1-60558-044-9, pp: 120-131.

Wurthinger, T., C. Wimmer and L. Stadler, 2010. Dynamic code evolution for Java. Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java, September 15-17, 2010, ACM, New York, USA., ISBN: 978-1-4503-0269-2, pp: 10-19.

Wurthinger, T., W. Binder, D. Ansaloni, P. Moret and H. Mossenbock, 2010. Improving aspect-oriented programming with dynamic code evolution in an enhanced Java virtual machine. Proceedings of the 7th Workshop on Reflection, AOP and Meta-Data for Software Evolution, June 21-25, 2010, ACM, New York, USA., ISBN: 978-1-4503-0536-5, pp: 1-5.