A Review on Dynamically Changing the Quality of Service Requirements for SOA based Applications in Cloud

C.Rajeev[1] and l.venkateswara reddy Reddy[2]

[1] Sree Vidyanikethan Engineering College,Tirupathi.

Received: 13 December 2013 Accepted: 5 January 2014 Published: 15 January 2014

Abstract

Service Oriented Applications have the ability to change their constituent services dynamically. This implies that they have the ability to change both, their functionality and their Quality of Service attributes dynamically. We present a Cloud-based-Multi-Agent System (Clobmas) that uses multiple double auctions, to enable applications to self-adapt, based on their Quality of Service requirements and cost restraints. Quality of Service attributes needed to provided, maintained, monitored at run time. A double auction is a two-sided auction, i.e., both the buyers and the sellers indicate the price that they?re willing to pay and accept, respectively. If any application uses self adaptation mechanism then it exhibits a high Quality of Service. Here we design a market mechanism that allows applications to select services, in a decentralized manner.

Index terms— self-adaptation, cloud-based-multi-agent system double-auction, decentralized.

1 Introduction

ver the past several years, interest has grown considerably in new techniques and technology for improving the task of creating and maintaining high-quality software. These efforts have arisen in response to a growing sense among application developers that traditional approaches are inadequate. Such new methods for improving software efficiency and predictability include intentional programming, evolutionary programming, model-based programming, and self-adaptive software. Some traditional approaches have not been worth-full in improving our ability to produce better code more affordably. Rather, the problem has been that one's reach always exceeds the grasp. As hardware capabilities improve and our understanding of how to apply computation to problems improves, we continually try to solve more difficult problems, driving up the complexity of solutions and overrunning the ability of our tools to manage the complexity.

Self-adaptive software [3] having its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible. This implies that the software has multiple ways of accomplishing its purpose and has enough knowledge of its construction to make effective changes dynamically. Such software should include functionality for evaluating its behavior and performance, as well as the ability to preplan and reconfigure its operations to improve its operation. Self-adaptive software should also include a set of components for each major function, along with descriptions of the components, so that system components can be selected and scheduled dynamically [2], in response to the evaluators. . Service-based applications will operate in a highly-dynamic world [4]. Systems will need to operate correctly despite of unexpected changes in factors such as environmental conditions, user requirements, technology, legal regulations, and market opportunities. They will have to operate in a constantly evolving environment that includes people, content, electronic devices, and legacy systems. They will thus need the ability to continuously adapt themselves in an automated manner to react to those changes. Adaptation must be achieved in an automatic fashion.

Service-based applications should exhibit selfhealing, self-optimizing, and self-protecting capabilities. In addition, they should be able to predict problems, such as potential degradation scenarios, future faulty behavior, and deviations from expected behavior, and move towards resolving those issues before they occur. This means that future service-based applications will need to become truly proactive.

Self-adaptive software uses a closed-loop mechanism. This loop, called the adaptation loop, [6] consists of several processes, as well as sensors and effectors. This loop is called the MAPE-K loop in the context of autonomic computing, and includes the Monitoring, Analyzing, planning and Executing functions.

We evaluate Cloud-based-Multi-Agent System (Clobmas) [5] in two stages. The first stage of evaluation is functional evaluation. This is to ensure that Clobmas meets the core objectives that it was set up to fulfill. The second stage of evaluation is to judge whether Clobmas Control theory-based paradigm gives a framework [1] for specifying and designing software that controls itself as it operates. Based on this paradigm, their self-controlling software model supports three levels of control: Feedback, Adaptation, and Reconfiguration. All the software subsystems interact with an environment that could be the external physical world or another layer of the computer system. Such environments can be characterized as dynamic systems. The essence of a dynamic system is that its output depends on the system's state. So, the system does not shift dramatically from one output to another (in response to changes in the input) but exhibits some form of inertia (because of the dependence on state).

Adaptation is possible in such a manner that the architected should be able to:

? Dynamically identify changed requirements, which will necessitate runtime adaptation.

? Initiate the search for new services, which better address the changed requirements.

? Substitute old web-services for new web-services. Service-Oriented Architecture has brought about a paradigm shift in the way we think about creating an application. Instead of linking programs and libraries at statically, we are now able to specify the functionality that the component parts should have, and the application can be dynamically composed using web-services. A web-service is a self-describing computational entity that can be used to perform various kinds of functions. These can be composed together, in a specific order, to deliver some functionality.

A web service is so-called because it uses webbased standards like XML, SOAP, etc to achieve its communication and data exchange, thus allowing the application location and platform independence. This allows an application to search a service repository for service that it wants, and then bind to it. This dynamic binding allows for the notion of an application changing the QoS properties that it exhibits, at runtime. Depending on the task at hand, or the budgetary resources or any other Quality of Service restraints that the architect imposes, the application can potentially pick an appropriate service and achieve its functional and non-functional targets.

2 a) Buyer Agent

A trading agent that is responsible for fulfilling one Abstract Service. The Buyer Agent bids for, and buys a Concrete Service. The amount that the Buyer-Agent is prepared to pay is called the bid price and this is necessarily less than or equal to its budget. The combination of bid price and the QoS attributes demanded is called the Bid.

3 b) Seller Agen

A trading agent that sells a Concrete Service. A Seller Agent is responsible for only one Concrete-Service. Based on demand and supply, the Seller Agent adjusts the price at which it is willing to sell the Concrete Service. The amount that a Seller Agent is preparedto accept, is called the ask price. This is necessarily greater than or equal to its cost. The combination of ask price and the Qo S attributes offered, is called the Ask.

4 c) Market Agent

A trading agent that implements trading rounds for a market. It accepts Bids from Buyer Agents, and Asks from Seller Agents. It performs matching of Bids and Asks.

5 The Market Mechanism

We can view applications as either a buyer of web-services with certain quality-attributes, or a seller that is capable of delivering those quality attributes at a certain cost. The resource allocation problem can be set up as a optimization problem, where the buyers need to maximize their Quality Attribute, given that they have a limited budget while sellers have a limited capacity to sell. We treat the universe of web-services as an economy, consisting of several marketplaces, several buyers, Several sellers. All their actions are rational and will result in a non-negative utility for them. The marketplace operates a continuous double auction (CDA) [7] which brings buyers and sellers together, and decides when a transaction should take place and at what price.

6 a) The Buyer

This is the application that we are primarily concerned about. This is the application that reconfigures its architecture through the process of buying web-services. The application receives a relative weighting amongst the Quality Attributes that it is concerned about.

7 b) The Seller

This is the application that sells web services to the highest bidder. This application has a minimum 'ask' price, below which it is not economical for the seller to sell. This is so due to the fact that computation, storage and data transfer all have a cost in the cloud. These are all paid by the seller's web service.

8 c) The Marketplace

This is an application that resides in the cloud, and acts as the meeting point for buyers and sellers. Our condition of Individual Rationality (IR) means that this application does not exist for selfless purposes. That is, it gains some amount of money by virtue of bringing buyers and sellers together. The more the number of transactions that occur, the more it earns.

There are various challenges in ensuring Quality Attributes (QA) of applications hosted in the cloud and hence the perceived quality of service of the cloud as a whole. We advocate a self-management/optimization architecture driven approach to ensure that Quality Attributes are met. The approach uses Service Level Agreements (SLA) and Utility Theory to direct the selfoptimization. We will propose more accurate application of multi-attribute utility theory to SLA negotiation. This would enable simulations of a cloud with negotiating web-services, thus allowing us to test our idea of lowlevel self-optimization leading to an emergent higher level optimized application state in the cloud. If successful, this would lead to long-lived applications in the cloud being more bouncily to change, and successfully adapting to changing Quality attribute optimization needs.

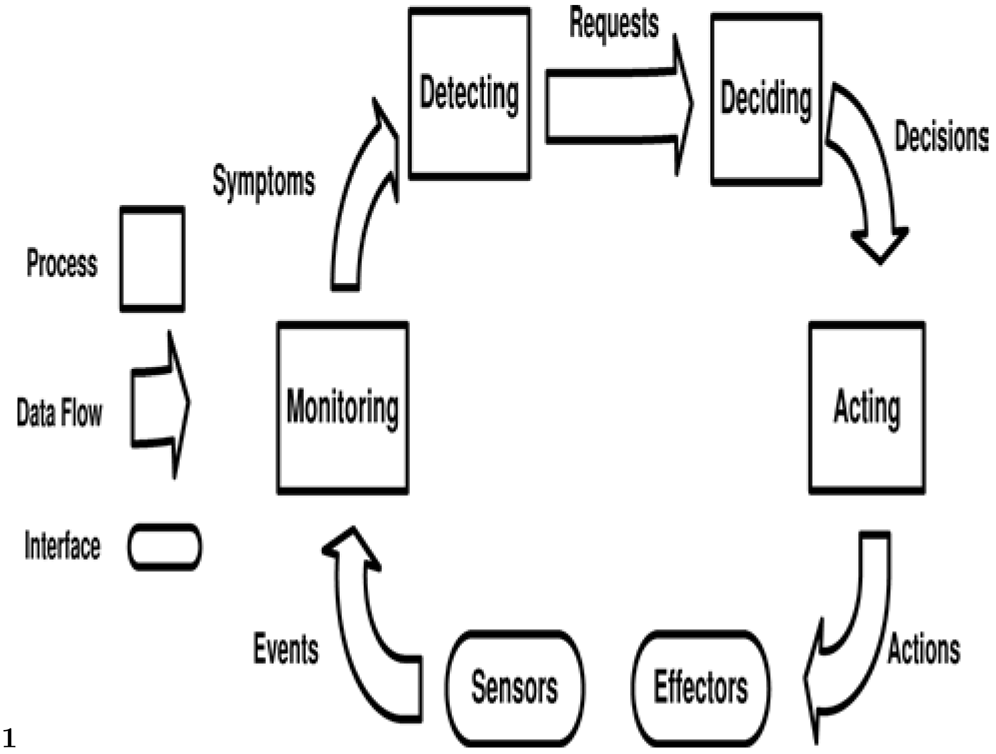
9 III.

10 Conclusions

Cloud-based service-oriented applications have the potential to self-adapt their Qo S, depending on demand. Using a market-based mechanism maps nicely to the real-world situation of unpredictable change of Quality of Service requirements, costs involved in adaptation and adaptation by competing applications. Service-based applications will thus have to continuously adapt themselves to react to changes in their context and to address changing user requirements. Adaptation must be achieved in an automatic fashion. Service-based applications should exhibit self-healing, self-optimizing, and self-protecting capabilities. Services in the cloud a are moving from a fixed-price package to a more flexible, auction-based approach. [1]

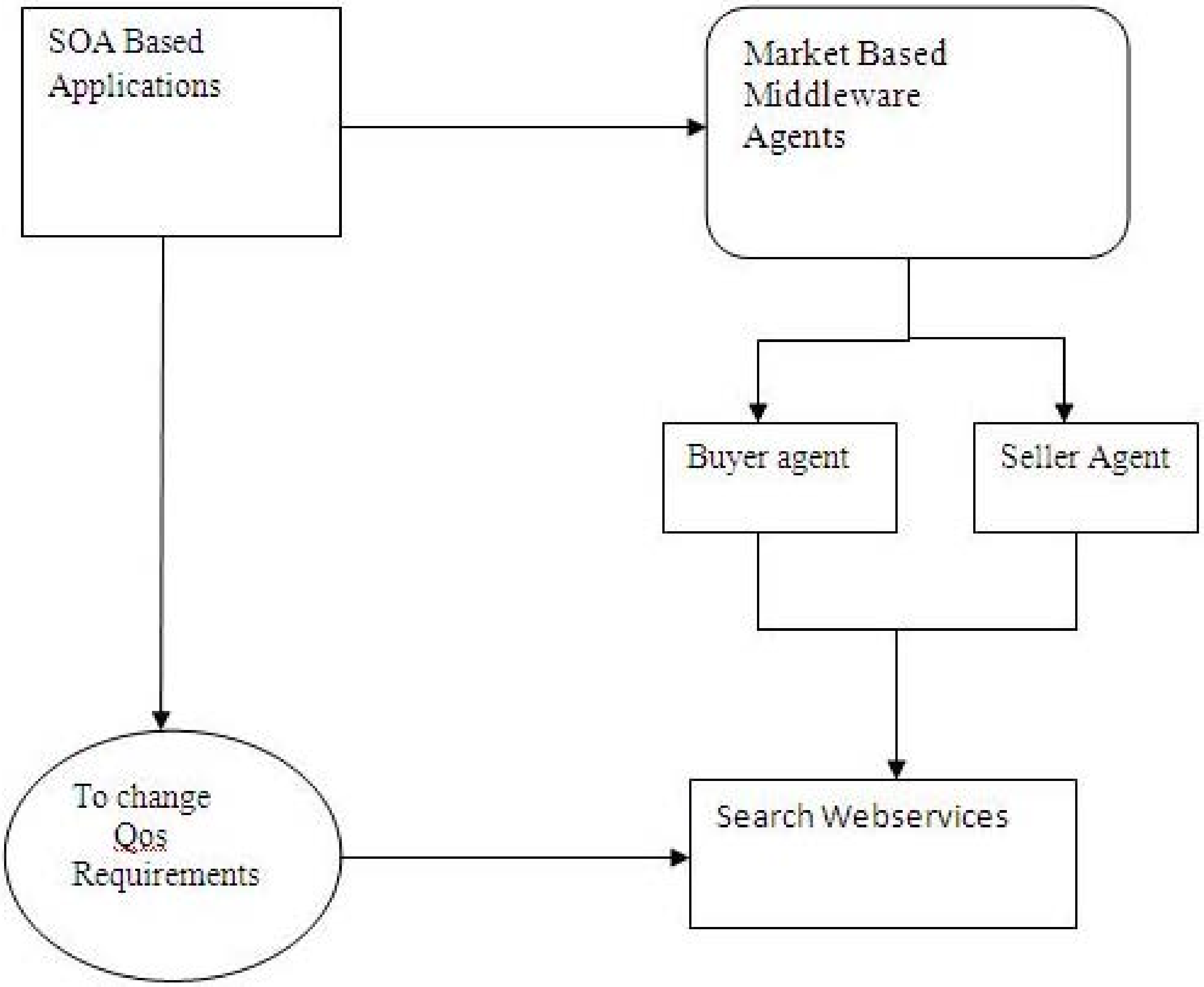
Figure 1:

[1]© 2014 Global Journals Inc. (US)



1

Figure 2: Figure 1 :



2

Figure 3: Figure 2 :

.1 Global Journals Inc. (US) Guidelines Handbook 2014

www.GlobalJournals.org

[Viveknallur and Bahsoon ()] 'A Decentralized Self-Adaptation Mechanism for Service-Based Applications in the Cloud'. Rami Viveknallur , Bahsoon . *IEEE Transactions on Software Engineering* 2013. 39 p. . (5 Year)

[Di Nitto et al. (2008)] 'A journey to highly dynamic, self-adaptive service based applications'. Elisabetta Di Nitto , Carlo Ghezzi , Andreas Metzger , Mike Papazoglou , Klaus Pohl . *Automated Software Engineering* September 2008. 15 (3-4) p. .

[Oreizy et al. ()] 'Architecture-based runtime software evolution'. P Oreizy , N Medvidovic , R N Taylor . *Proceedings of the 20th International Conference on Software Engineering*, (the 20th International Conference on Software Engineering) 1998. p. .

[Kokar and Baclawski ()] 'Control theory-based foundations of self-controlling software, Self-Adaptive Software and their Applications'. M M Kokar , Baclawski . *IEEE Intelligent Systems* 1999.

[Laddaga (1999)] 'Creating robust software through self-adaptation'. Robert Laddaga . *IEEE Intelligent Systems* May 1999. 14 p. .

[Nallur and Bahsoon] 'Design of a Market-Based Mechanism for Quality Attribute Tradeoff of Services in the Cloud'. Vivek Nallur , Rami Bahsoon . *Proceedings of the 25th Symposium of Applied Computing*, (the 25th Symposium of Applied Computing) ACM SAC.

[Salehie and Tahvildari (2009)] 'Self-adaptive software'. Mazeiar Salehie , Ladan Tahvildari . *ACM Transactions on Autonomous and Adaptive Systems* May 2009. 4 (2) p. .