AUSTRALIAN JOURNAL OF BASIC AND APPLIED SCIENCES

ISSN:1991-8178 EISSN: 2309-8414
Journal home page: www.ajbasweb.com

Software Component as a Service (SCaaS)

Mahmoud M. Elkhouly

Head of Information Technolohy Department, Faculty of Computers and Information, Helwan University, Cairo, Egypt.

Address For Correspondence:
Mahmoud M. Elkhouly, Head of Information Technolohy Department, Faculty of Computers and Information, Helwan University, Cairo, Egypt.

ARTICLE INFO

Article history:
Received 11 September 2016
Accepted 10 November 2016
Published 28 November 2016

Keywords:
Cloud, software reuse, software component, automatic programming.

ABSTRACT

Background: Cloud computing is a challenging task for many software engineering projects, especially for those projects which need development with reusability. Cloud computing is a style of computing in which virtualized resources are provided as a service over the Internet. Software as a Service (SaaS) is one layer of cloud computing, and it can be used for providing different types of business services. **Objective**: In this paper, we present a new service named Software Component as a Service "SCaaS" to be available in cloud computing environment. **Results**: This new layer will be used for reuse software components, as reusable and modular software components are expected to play a vital role in improving software construction processes and in reducing software building time to market. **Conclusion**: The core of this service had been implemented, and had been used to automatically generate code for several programming languages. Experimental results showed that using SCaaS reduced cost and improved time to market.

INTRODUCTION

Currently, the software reuse potential has not yet adequately met the expectations to satisfy the growing demands on software (Iyapparaja, 2015) (Humphrey and Watts 2001) (Mili *et al*., 1999a) (Mili *et al*., 1999b) (Mili *et al*., 1995) (Shiva, 2007). Therefore, a considerable amount of effort is still required for the adaptation and customization of existing software code elements to fit particular application contexts and requirements. The challenges of growing and ever changing software requirements, increased software complexity, are fueling the expectations for reuse especially in the context of heterogeneous computation platforms (Brooks, 1987) (Fraser *et al*., 2008) (Hughes, 1990) (Mcdirmid, 2001). Component technology and component-based software construction provide a promising remedy for those challenges. In this respect, components are perceived to be modular, loosely-coupled and compositional (Coker, 1997).

Software reuse is the process of creating new software systems using existing software components. These software components contain some code and an interface that provides access to the component. The code explains the functions that the component will perform, while the interface informs the component-user needs about the component (Frank *et al*., 2000). Ideally, components should be black boxes. Black-box components may no longer be internally modified; rather its environment should be changed to support the component. In other words, the interface of a component should provide all the information needed by its users. The components can be considered as white-box reuse if the code can be modified to some extend in which the component is applied (Chuang, 1996) (Dash, 2009). Previous research showed that with black -box reuse higher reuse levels can be achieved than with white-box reuse (Tomer *et al*. 2004) (Fingar, 2009).

In particular, a component must:

• Have interfaces (connection points) and interfaces' implementations for interaction;

• Provide a nearly independent behavior;
• Have a standard model to which it complies;
• Have an infrastructure necessary to permit its composition, deployment and proper functioning;
• Provide or easily create its executable format so that it can be dealt with as a black box.

Advantages of software components

1. Increase software reuse.
2. Reduce development efforts and time to market costs.
3. Promote and encourage the use of standards.
4. Provide better interoperability and distribution with the emergence of mediators such as Object Request Brokers (ORB).

Background:

Recent Research Trends:

Reusable software components have been promoted recently. Software reuse is the process of using existing software components to develop a new software product. Reusability of components requires some repository where a component can be stored and efficient retrieval mechanism to retrieve and modified software components.

Some of the existing techniques are going to be reviewed here (Ebtessam , 2015) as follows:

Amato *et al*. (2015), proposed solution adopts two collaborative modules. The semantic engine, whose aim is to create an agnostic description of resource based on users' service requirements and a brokering system, the cloud agency, whose aim is to acquire resources from providers for finding the most suitable cloud provider that satisfy users' requirements.

Liu *et al*. (2015), Proposed an adaptive fuzzy clustering technique based on clustering algorithms to address the issue of single metric. The proposed technique considers distance, density, and the trend of density change of component instances in the membership degree calculation. The post-clustering separation of clustered components based on the predefined thresholds and regrouping of the separate component points result in higher cohesive clustering.

Zafar *et al*. (2015), developed an algorithm for assigning clusters to the reusable components. Their framework consists of three processes: threshold checking, clustering and retrieval process. In threshold, checking process threshold value is applied to the category-based cluster to lemmatize the maximum number of clusters in the category-based cluster.

Srinivas *et al*. (2015), used the k-means clustering algorithm to cluster the software components. They used the distance measure, which is designed to find the similarity between the software components to find the pairwise project distance matrix and apply the k-means algorithm on that distance matrix. The main idea is to use more than one distance measure, to explore consensus-based technique, to cluster software components, instead of using only one measure to cluster the components.

Jagli *et al*. (2013), proposed the clustering model for evaluating SaaS to evaluate potential software services on the cloud computing by using data mining clustering algorithms. It helps service provider to increase the availability of software services in the cloud computing environment that needed upon cloud users demand and requirement. It also helps cloud users to evaluate potential software services available on the cloud-computing environment.

Kaur *et al*. (2015), presented that using Genetic Algorithm (GA) and Ant Colony Optimization (ACO) improved the efficiency of component retrieval.

Singh *et al*. (2012), proposed cloud computing reusability model. The model has been validated by CloudSim, and their experimental work shows that reusability based cloud computing approach is effective in minimizing time and cost.

Jatain *et al*. (2013), presented a clustering approach for grouping components of similar reusability using an already worked out fuzzy data set to cluster components of similar reusability together for the purpose of minimizing the efforts of the developer using agglomerative hierarchical clustering. Components attribute affecting the reusability are classified into rules using a fuzzy system and are then taken as the inputs to the proposed clustering model.

Al-Saiyd *et al*. (2014), presented the design and implementation of ontology-based multi-agent software component retrieval system using semantic and structural formalism Ontologies are usually organized into a taxonomy tree where each node represents a concept with its attributes.

Software Component Models and Technologies:

To facilitate the reuse of a software component, dedicated software tools and infrastructures are often implemented, e.g.:

a) COM, DCOM, COM+ and COM3: Microsoft's Component Object Model (COM) (Microsoft, 1996). Microsoft used this model internally, in its Windows operating systems as well as in applications available on that platform.

b) ActiveX: A particular type of COM components is ActiveX controls (Chappell, 1996).

c) DCOM is an extension of COM, which allows component-based applications to be distributed across memory spaces or physical machines.

d) EJB: providing similar services to Sun's Enterprise JavaBeans (EJB) (Sun, 2001).

e) CCM: CORBA Component Model (Object Management Group, 2002). A CCM application is an assembly of CCM and possibly EJB components, whose configuration is described in XML.

f) .NET: The name.NET is used by Microsoft to denote a comprehensive set of new technologies. That includes a new component model, intended to replace COM/DCOM.

Cloud Computing:

Cloud computing moves most of computing and data away from PCs into large distrubted data centers that offer on-demand services through the Web on a "pay as you go" basis. Within SaaS vendors, IBM and Microsoft are two of the major service providers although Google and Amazon also fall under this category (Amit, 2014). The applications can run on the user's computing systems or the provider's web servers. SaaS provides for efficient management and promotes collaboration (Ronald, 2000).

Proposed Service:

Software Components as a Service (SCaaS) solutions deliver software components over the Web. A SCaaS provider deploys software components to the user on demand, commonly through a licensing model. The provider may host the software components on its server infrastructure or use another vendor's hardware. The searched cloud component will be checked whether the component support required architecture, functionality and interfaces. If it passes then it will be reused as a black box otherwise, it will be reused as a white box. This white box reuse will take place via the modification then; the component will be linked with up to date cloud application and send back to cloud component repository for future utilize.

Location-Independent Resource Pooling:

The SCaaS must have an extensive and flexible resource pool to meet the consumer's needs, worth economies of scale. Applications need resources for their execution, and these resources must be allocated efficiently for optimum performance (Ronald, 2000).

Service Consolidation:

This capability depended on advances in, optimization, and virtualization. The decreasing cost of cloud computing hardware has made shared; consolidated services are even more desirable today (Ronald, 2000).

How ScaaS works?:

ScaaS has many modules, among them *Add module* and *Retrieve module* will be presented as follows.

Add Module:

The component repository in *SCaaS* is created by its indexing subsystems: concepts indexing and signature indexing. SCaaS extracts and indexes functional descriptions (concepts) and signatures (constraints) from the new code added to the system as shown in figure 1.

Australian Journal of Basic and Applied Sciences, 10(16) November 2016, Pages: 45-51

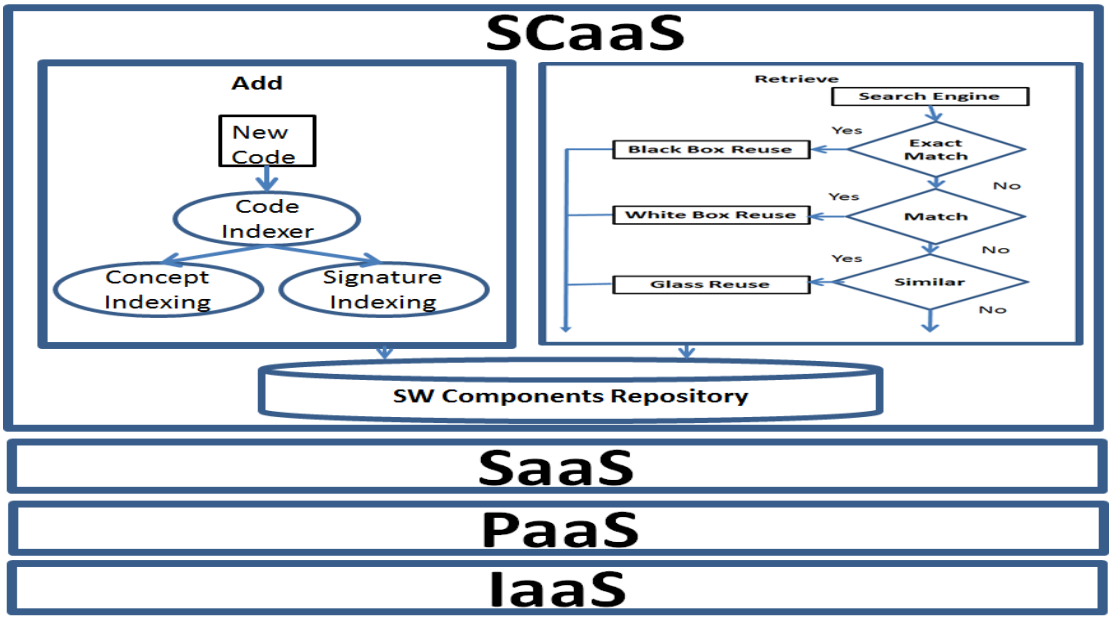

Fig. 1: SCaaS within Cloud Environment

Retrieval System:

Data item:

Suppose we have a first order definition of data item L with signature

$$D=<L,C,N,I,R,S,M> \quad (1)$$

Where L is a programming language used, C is a super class. N is a name of the software component; I is a set of instance variables. R is a formula of the software component; S is a set of similar software omponents' names of this software component, and M is a semantic of this software component, as shown in figure 2.

We are interested in some formal criteria for obtaining s software component F, with signature

$$F=<N,I,R,M> \quad (2)$$

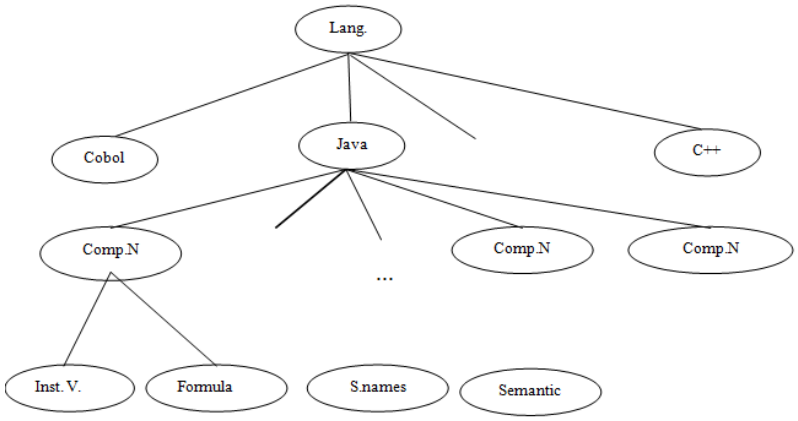

Fig. 2: Class hierarchy

i.e., a software component F should retrieve with its name, instant variables, formula and semantic. We use equation (1) in building our repository, while we use equation (2) as a result from the retrieval process.

Find & Similar:

Def.1 A software component F is *Exact match* with D iff

$$((D(N)=F(N)) \vee (F(N) \epsilon D(S)) \wedge ((F(M) \subseteq D(M)) \vee (D(M) \subseteq F(M))) \quad (3)$$

It means that a software component F is *exact match* with data item in the repository if and only if the following two conditions are true: (a) they have the same name, or the name of the required software component exists in the list of similar names of that data item. (b) the semantic of the required software component is a subset or equal to the semantic of that data item; or vice versa.

Def.2 A software component F is Match with D iff

$$(D(N) = F(N)) \vee (F(N) \epsilon D(S)) \quad (4)$$

It means that the software component F is matched with the data item in the repository if and only if they have the same name, or the name of the required software component exists in the list of similar names of the data item.

Def.3 A software component F is similar to D iff

$$(D(C) = F(C)) \wedge (F(N) \epsilon D(J)) \quad (5)$$

i.e., the software component F is similar to the data item in the repository if and only if their super class has the same name and the name of F exists in the list of similar functions of that data item.

We use the above definitions to construct two important commands in our retrieval system. Find which uses definitions 1 and 2 to retrieve *exact match* or *match* software components, and *similar* which applies definition 3 to find similar software components.

Query Process:

The interaction between the broker and the repository of reusable software components must follow a well-defined process. This process will control both the visual layer and the access to the repository. The element of the retrieval mechanism responsible for the automatic modification of queries is called the *Query Process* (El-Khouly, 1999). The query process permits the retrieval module to check if the software component retrieved qualified for black-box reuse; white-box reuse; or glass-box reuse, as shown in figure 1, as follows (Yunwen, 2001):

Black-Box Reuse:

In black-box reuse, a component is reused without modification.

White-Box Reuse:

In white-box reuse, the component can be reused after the programmers modify to their needs. White-box reuse does not contribute as much as Black-box reuse because of maintenance, and evolution required. However, it can reduce development time.

Glass-Box Reuse:

In glass-box reuse, we do not directly reuse the component; instead, we can use it as an example for our new software development, this can reduce the cognitive load of programmers (Neal, 1966).

Conclusion:

Recently, reusability has a great importance in software engineering. However, the problem is that there is no proper way to extract reusable components from repositories. The effectiveness of the reuse-based approach in software development is strongly dependent on the underlying classification scheme and retrieval process. In this paper, both issues had been covered, by enhancing the structure of the software components repository, and in the retrieving system used. A new service named "SCaaS" had been presented with its two modules *add module* and *retrieve module*. This new cloud's service offers software components as a service in a cloud environment to increase the potential of software reuse to satisfy the growing demands on software. Some of the SCaaS benefits are:

- Increase the reliability and quality of the new software systems.
- Identify independent components are having a low coupling and high cohesion.
- Accelerate cloud-based development
- Improve time to market
- Reduce cost
- Increase generality

We implemented the core of SCaaS using PROLOG. The software components stored as clauses, while *retrieve module* added at the beginning as predicate rules. Experimental results showed that we saved 56.1% to

71.1% of original time in generating 138 lines of C code, and 79.6% to 83.7% of original time in generating 74 lines of C code. Fully automated programming code generation is our target for future work.

REFERENCES

Al-Saiyd, N.A., M.F. Al-Samarae and I.A. Al-Sayed 2014. A multi-agent systems engineering for semantic search of reuse software components, IJCS, ISSN (Print): 1694-0814.

Amato, A., G. Cretella, B. Di Martino, L. Tasquier and S. Venticinque, 2015. Semantic Engine and Cloud Agency for Vendor Agnostic Retrieval, Discovery, and Brokering of Cloud Services , Springer, LNCS 8993, pp: 825.

Amit, 2014. "SaaS (Software as a Service) Based Business Model: Cost Analysis", IJMCI, Vol. 1, Issue 1, pp: 6-10.

Brooks, F.P., Jr. 1987. "No Silver Bullet Essence and Accidents of Software Engineering". *Computer*, 20(n° 4): 10-19.

Chappell, D., 1996. Understanding ActiveX and OLE, ISBN 1-572-31216-5, Microsoft Press.

Chuang, J.H., 1996. Potential-Based Approach for Shape Matching and Recognition. Pattern Recognition, 29: 463-470.

Coker, L. and R.R. Hayes, 1997. Monterey, California. "Services First, Components Second!". In *6th International OMG-DARPA Workshop on Compositional Software Architectures.*

Dash, D., V. Kantere, A. Ailamaki, 2009. An economic model for self-tuned cloud caching. In: Proceedings of the IEEE International Conference on Data Engineering.

Ebtessam Desouky, and Mahmoud El-Khouly, 2015. A Survey on Clustering Software Components for efficient component retrieval", Journal of Information Society, ISSN 2356-9328, 1(1): 6-12.

El-Khouly, M., B. Far, Z. Koono, 1999. "Software-Agent for reuse Software Components", proceedings of 7th Int. Conf. on A.I. Application, 3-7. Egypt.

Fingar, P., 2009. Extreme Competition: Cloud Oriented Business Architecture. Business Process Trends.

Frank Lüders, Kung-Kiu Lau, and Shui-Ming Ho 2000, "Specification of Software Components", In Ivica Crnkovic and Magnus Larsson (Editors), Building Reliable Component-Based Software Systems, ISBN 1-58053-327-2, Artech House Books.

Fraser, S., and D. Mancl, 2008. "No Silver Bullet: Software Engineering Reloaded". *Software, IEEE,* 25(n°1): 91-94.

Hughes, J., 1990. "Why Functional Programming Matters". *The Computer Journal,* 32: 2.

Humphrey, Watts S., 2001. "The Future of Software Engineering: II". Online.
<http://www.sei.cmu.edu/library/abstracts/news-at-sei/wattsnew2q01.cfm>.

Iyapparaja, M and S. Dr.Sureshkumar, 2015. Normal Distribution and Questionnaire Based Assessment of Risk in Software Reusability. Aust. J. Basic & Appl. Sci., 9(11): 160-166.

Jagli, D. and A. Gupta, 2013. Clustering Model for Evaluating SaaS on the Cloud ,IJAIEM, ISSN 2319 – 4847.

Jatain, A., A. Nagpal and D. Gaur, 2013. Agglomerative Hierarchical Approach for Clustering Components of Similar Reusability, International Journal of Computer Applications, 0975-8887.

Kaur, I. and S. Kaur, 2015. Software Component Retrieval Using GA and ACO ,IJARCSSE, ISSN: 2277 128X.

Liu, D., C. Lung, Samuel and S.A. Ajila, 2015. Adaptive Clustering techniques for Software Components and Architecture", IEEE, 0730-3157/15.

Mcdirmid, S., M. Flatt and C.H. Wilson, 2001. "Jiazzi: New-Age Components for Old- Fashioned Java.". *In International Conference on Object Oriented Programming, Systems, Languages and Applications.*

Microsoft Corporation 1996, The Component Object Model Specification, v0.99. 130 Use of Component-Based Software Architectures in Industrial Control Systems

Mili, H., F. Mili and A. Mili, 1995. "Reusing software: issues and research directions". *IEEE Transactions on Software Engineering*, 21(n° 6): 528-562.

Mili, A., S. Yacoub, E. Addy and H. Mili, 1999a. "Toward an engineering discipline of Software reuse". *Software, IEEE,* 16(n° 5): 22-31.

Mili, H., J. Dargham, A. Mili, O. Cherkaoui and R. Godin, 1999b. "View programming for decentralized development of OO programs". In *Proceedings on Technology of Object-Oriented Languages and Systems, 1999.* pp: 210-221.

Neal, L., 1966. Support for Software Design, Development and Reuse through an Example-Based Environment, in G. Szwillus & L. Neal (eds.), *Structure-Based Editors and Environments,* Academic Press, San Diego, CA, pp: 185-192.

Object Management Group, 2002. CORBA Components, Version 3.0, report formal/02-06-65.

Ronald L. Krutz, Russell Dean Vines, 2000. “Cloud Security: A Comprehensive Guide to Secure Cloud Computing”, Wiley Publishing.

Shiva, Sajjan G., and Lubna Abou Shala, 2007. "Software Reuse: Research and Practice". In *Fourth International Conference on Information Technology, ITNG '07*. pp: 603-609.

Singh, S. and R. Singh, 2012. Reusability Framework for Cloud Computing, ijcer, 2: 6.

Srinivas, C. and C.V. Guru Rao, 2015. A Feature Vector Based Approach for Software Component Clustering and Reuse Using K-means, ACM. ISBN 978-1-4503-3418-1/15/09

Sun Microsystems, 2001. Enterprise JavaBeans Specification, Version 2.0.

Tomer, *et al*. 2004. "Evaluating software reuse alternatives: A model and its application to an industrial case study," IEEE Transactions on Software Engineering, 30: 601-612.

Yunwen Ye, 2001. “Supporting Component-Based Software Development with Active Component Repository Systems”, PhD thesis, Faculty of the Graduate School of the University of Colorado.

Zafar, M.H., R. Aslam and M. Ilyas, 2015. Classification of Reusable Components Based on Clustering, .J. Intelligent Systems and Applications, 10: 55-62.