Component Selection Problem for Component-based Software

Jagdeep Kaur*
Pradeep Tomar**

Abstract

Component-Based Software Engineering (CBSE) is becoming popular due to the benefits of software reusability and the availability of many alternatives of reusable components. By reusing existing software components that have already been tested, software engineers may reduce errors and shorten the time and cost to market the system under development. However, software engineers still have a problem in selecting the existing reusable components as well as difficulties in determining the quality of the developed components for future reuse. Therefore, an appropriate component selection process is extremely needed to harvest the full benefits of software reuse. In this paper, the software component selection is reviewed in detail however the emphasize is given on the problems that occur in component selection. Firstly the component based software development is considered, followed by component selection problem in general (simple component selection problem) and (criteria based component selection).Subsequently, the authors analyzed about ten papers in terms of the major problem, proposed solution and critical analysis. In the end conclusion is drawn based on critical analysis the authors highlights the various factors(Considering non-functional factors, dynamic requirements, formal specification for component repositories, fuzzy theory for subjective judgments) that must be taken into consideration to improve the component selection process.

Keywords: Component Based Software Engineering (CBSE), Component-Based Software Development, (CBSD) Simple Component Selection Problem (SCSP), Criteria-Based Component Selection Problem (CCSP)

Introduction

In the field of software engineering, present day research organisation and software industry is facing big challenges due to increased maintenance cost and lack of intended software quality attributes viz. functionality, reliability, reusability, usability, efficiency, maintainability and probability. Software reusability has long been one of the major issues in Software engineering, where code reuse is seen as the key to many benefits, such as increased productivity, improved reliability, ease of maintenance, reduced time and cost. As a result, many software reuse technologies have been developed over the past few years. According to the (Cox, 2001) software reuse was first realised in the late 50's with the development of libraries of pre-compiled subroutines such as large numerical libraries for engineering and scientific computation. Reuse via subroutines has limited applicability, because of the difficulty of encapsulating high-level functionality in subroutines. To remove the demerits of sub-routines, a major step forward was made with the advent of Object-Oriented Programming (OOP), which provided inheritance as its primary reuse mechanism. As OOP research and practice has progressed, other reuse techniques have been devised, such as object composition. Here, simple data types and objects are combined to form the complex ones. Composition is also known as a part of or is a relationship because the member object is a part of the containing class and

**Department of Computer Science Engineering and Information Technology, ITM University, Gurgaon, Haryana, India.*
***School of Information and Communication Technology, Gautam Buddha University, Greater Noida, Uttar Pradesh, India.*

Authors are grateful to the anonymous referees for comments and suggestions for improving the text and contents of the paper. Authors alone are responsible for any errors/mistakes still remaining in the paper.

the member object cannot survive or exist outside the enclosing or containing class or doesn't have a meaning after the lifetime of the enclosing object (Wikipedia, 2012). More significantly, object-oriented application frameworks have emerged, providing the means to capture very large application design patterns in the form of "inverted libraries" which call the code supplied by the application developer, rather than the other way round as with traditional libraries. Another popular reuse technique in OOP involves design patterns, which provide guidance during the design phase of software by suggesting high-level organisations for the necessary abstractions but these techniques are not the complete solution to enhance the reusability.

A recent trend to enhance reusability is towards software development by using Component-Based Software Engineering (CBSE). CBSE is an approach which deals with design, selection and composition of components. This approach is becoming more popular with more number of commercially available software components. Software Developer select the components to satisfy a set of requirements while minimizing a set of various objectives (as cost, number of used components) and this task is becoming more difficult. Business organizations prefer to build their systems from previously built components which means they should concentrate more on selecting and composing components than to manually adopt software systems. In Component-Based Development (CBD), software industries reuse existing assets to build new systems. These components may be pre-existing intra organization components or commercially purchased ones. A component may be any reusable and independent unit of functionality, such as a class or a library. The process of development of a software system thus changes to combining components through selection of components, adaptation of components, composing them with each other and testing of the developed system. Since all activities in such process are influenced by the first step of selecting components, the process of component selection becomes an important aspect of CBD. Software development is becoming less and less associated with the development, ab initio, of a single software system and more and more an evolutionary process in which a system is incrementally developed over a series of releases. There is also an increase in the use of component-based approaches, in which the next release in the evolutionary process is defined in terms of a set of additional components that augment the existing system to meet a set of constraints. These components may be pre-existing or planned. Systems often contain an integrated mixture of pre-existing components and newly built components. CBSE methodology is based on the observation that hardware design and development has evolved well beyond the "build-from scratch" era. These days hardware is built from "off-the shelf" components which are customisable to a degree, cheap, well tested, reliable and have a well defined interface. The aim of CBSE is to attain reuse, economy, reliability and so forth by creating large catalogues of software components for assembling into software systems. This approach has some other desirable consequences as well, such as standardisation, certification and selection of components developed by different developers and these selected components work efficiently after integration. Software component selection is considered as an important solution to many of the problems in Component-Based Software Development (CBSD). In the early 1990's, it became apparent to both researchers and practitioners that Object-Oriented Technology (OOT) were not enough to cope with the rapidly changing requirements of real-world software systems. So researchers and practitioners decide to shift towards the component technology. If software developers had a collection of reusable software components, then software industry could build applications by simply plugging existing components together. In CBSD, a complex system is accomplished by assembling simpler pieces obtained in various manners. The research efforts have been made to make reusability process of CBS more effective, more predictable and less expensive in comparison to simple software reusability by using different approaches. CBSD and Component-Based Software Reusability (CBSR) approaches of software engineering is not similar to traditional engineering domain. CBSD and CBSR finally provide solution to all complex problems and not only reduce the time to market but also bring down the development cost heavily (Pressman, 2006). CBSE is an approach which is used to enhance the reusability from the pre-existing software components. But when reuse pre-existing software components, components selection factors play an important role to enhance the reusability, productivity and quality.

Component Based Software Development

In recent years CBSD has established itself as an extremely successful reuse technology, even in commercial terms. CBSD is based on the concept of composing or assembling systems out of independent but pluggable parts. Since it is rather unrealistic to make all components with similar interfaces, components have a configurable adapter by which they can be coupled. This capability for configuration or adaptation is the central characteristic of components. Beyond that, it is hardly possible to specify more common features or attributes of components, since different products employ different approaches. At run time, components have similarities with objects or object complexes. They also store a state in attributes which may (but need not) be encapsulated and provide operations for testing and manipulating the state. Furthermore, components define and trigger events of internal processes for an interested environment. Which event a component defines depends strongly on its functionality. Typically, events correspond to substantial state changes. Putting two components together means establishing a relation between component on the one side and an action on the other side. At run time, triggering an event causes the activation of all associated operations. Components can be described by means of one or more classes or by a conventional module. Thus OOP is no prerequisite for components because Component-Oriented Programming (COP) is altogether different for development of component and Component-Based Software (CBS). Usually, components have a static nature; they begin to exist at program start and live until program end. Assembling and configuring of components is often done in composition environments by means of scripting and direct linking. In addition to the already mentioned properties, component selection is difficult due to the incompatibility of component interface, component code and component database on different platform.

Component Selection Problems

According to Stritzinger (Stritzinger, 1996) composition environment provides functionality for arranging and directly manipulating user interface building blocks. But CBSD requires neither a composition environment nor a component library: a

regular programming environment and appropriate conventions would suffice. But a comfortable composition environment together with a powerful component library allows working at a very high abstraction level. Software systems are more frequently composed from pre-existing commercial and non-commercial components to save time and cost. These components provide complex and independent functionality and are engaged in complex interactions. A component is an independent executable entity that can be made up of one or more executable objects. For a component selection problem the system designer considers several candidate components for which the data available include estimates of the cost of acquisition (third party purchase or in house development), customer desirability, development time and expected revenue. The designer may also have information about the dependencies between components and may wish to include other factors in the decision making process, such as the priority given to each of the customers. From the set of all components, the designer must search for a subset that balances these competing concerns in the best way possible. The manager may also want to rank (or prioritize) the components in some way based upon these trade-offs. For systems with more than a few simple components the search space is unmanageably large and complex, with the consequence that no designer can be expected to find optimal choices that balance the constraints without some form of automated support. This is the 'Component Selection Problem'. The component selection problem helps the designer to decide which component combinations will make sensible choices for future evolution. Component selection is playing main role in CBSE. According to Vescan and Kaur, (Vescan, 2009), (Kaur, 2010) there are two types of component selection problem: Simple Component Selection Problem and Criteria Based Component Selection Problem.

Simple Component Selection Problem (SCSP)
Simple Component Selection Problem (SCSP) is the problem of choosing a number of components from a set of components such that their composition satisfies a set of objectives. The notation used for formally defining SCSP, as laid out in with a few minor changes to improve appearance is described in the following.

Consider SR (set of requirements) the set of the final system requirements (target requirements) as
SR = {r1, r2, ..., rn},
and SC(set of components) the set of components available for selection as
SC = {c1, c2, ..., cm}.
Each component ci can satisfy a subset of the requirements from SR,
SRci = {ri1, ri2 , ..., rik}.

The goal is to find a set (subset) of components Sol (obtained solution) in such a way that to every requirement rj from the set SR can be assigned a component ci (considered cost) from Sol where rj is in SRci.

Criteria-Based Component Selection Problem (CCSP)
Criteria-Based Component Selection Problem (CCSP) is the problem of choosing a number of components from a set of components such that their composition satisfies a set of objectives using various criteria. In addition to the above description a cost of a component ci is considered cost (ci). The goal is to find a set of components Sol in such a way that to every requirement rj from the set SR can be assigned a component ci from Sol where rj is in SR ci , while minimizing the number of components in the solution Sol and/or while minimizing ci Sol cost (ci).

Analysis of Component Selection Problems

This study presents analysis of challenges faced during components selection to suggest how to select component due to which researchers and practitioners select optimal components from repository to fulfil the requirements of client. Components selection, - a number of software components selected from a subset of components or from components repository in such a way that their composition satisfies a set of objectives. So this study presents challenges like performance, time, components size, fault tolerance, reliability components functionality, components compatibility and available component subset for consideration during the software component selection.

According to Tang et al. (Tang et al., 2011) Software Component Selection plays an important role in component retrieval, adaptation and assembly. Here the management insights are provided for Component Selection purpose apart from the technical details. Here selection of components and deployment of component into applications is used to minimize the total cost, considering compatibility among components. Reusability and compatibility is considered to reduce the cost of software development by simultaneously using compatibility matrix to provide several management implications by experiments and sensitivity analysis.

Problem

The authors consider software component selection under multi application development at a time. The software component selection is to select the available components and deploy into applications, to minimize the total costs of adaptation cost of available components and procurement costs of available components on all undertaken applications, considering the compatibility among available components.

Proposed Solution
The problem is solved using the optimization model based on nonlinear binary integer programming. As it is associated with combinatorial explosion Genetic Algorithm is used to solve the problem.

Critical Analysis
- It is assumed that all the enterprise applications can be developed using COTS & in house developed components can be considered in future work.
- It is assumed that required components will be fulfilled by selecting a single component from the available component. In real scenarios required components may be fulfilled by composing available components.

Jha et al. (Jha P.C. et al., 2011) have presented a framework that will assist the system developers to decide whether to buy or build components. If the COTs are to be selected then different versions are available for each alternative of a module and only one version will be selected. If the component is an in-house built component, then the alternative of a module is selected.

Problem

The major problems faced by software organizations while developing or maintaining the software are how to deal with varying requirements of the users and whether to build the required software component in-house or to buy it. This paper discusses the issues related to reliability of the software systems and cost caused by integrating COTS or in-house built components.

Proposed Solution
The fuzzy approach can be used for component selection using "Build-or-Buy" strategy in designing software structure. The selection is done on the basis of cost and non functional factor reliability. Two fuzzy multi-objective optimization models have been proposed to solve the component selection problem. The first model maximizes the system reliability and minimizes the cost. The second optimization model considers compatibility between different alternatives of the module.

Critical Analysis
- In the optimization model proposed, the different alternatives are available but how to select the best among them is not known.
- The architecture, design or testability of modules can also be considered while selecting the components. The framework discussed by, is the details of the framework are not given.

A component selection framework for mobile pervasive computing is proposed by Dong et al. (Dong et al., 2009). It adopts the case-based reasoning technique to provide proactive component selection. Context awareness and personalization are embodied in the reasoning and selection process.

Problem

Now a days, due to extensive use of applications running on mobile devices like smart phones, tablets, PDAs etc there is requirement that applications should adapt their functionality according to the changing contexts. The present approaches are based on pre-defined rules or static mapping. So the pervasive computing environment needs to be context aware (adapt to available resources, situations and what user wants to do). Also, the degree of context awareness is to be taken into consideration, from physical context like battery available, CPU power, memory, bandwidth, connectivity to high level context like user's behavioral pattern, emotions etc. Recently, case based reasoning technique has gained interest in designing context-aware system.

Proposed Solution
Here, a context-aware personal communicator (CAPC) application using adaptive component selection, is developed and evaluated with a synthesized execution trace obtained from real-life E-mail software ported to CAPC. This framework is built on sparkle software.

Critical Analysis
- In evaluating the reasoning engine, the network latency is not considered.
- While evaluating response time of context filtering, the filtering mechanism should be further considered for more reliable results.
- The reasoning efficiency can be improved by using index within context.
- Using different combination of components and unpredictable execution order leads to large case size. So, clustering the similar cases can reduce this.
- User's intention model can also be incorporated in reasoning process.

Component selection problem as stated by Vescan et. al (Vescan et al., 2011) is a multiobjective optimization problem involving four objectives: the number of components used, the number of new requirements, the number of provided interfaces and number of initial requirements that are not in solution.

Problem

The component selection problem is concerned with gathering of pre-existing software components form software that is according to the client's requirements. It depends on changing or dynamic environment and multiple criteria are to be taken into consideration. The system requirements are changing over time.

Proposed Solution
An evolutionary algorithm with four objectives is used with a repair mechanism which helps the algorithm to continue from the solution already found in the last step instead of restarting the process from scratch.

Critical Analysis
- The component composition is considered as static and component selection is considered in a dynamic or changing environment with two different cases: the requirements of the problem change over time and the components available at a certain time step change.
- On analysis it was found that the functionality of components is considered in terms of required interfaces only so selection process in not dependent on discrete components.
- When component selection includes non-functional properties like performance, reliability etc such dependency will arise. This problem is ignored here. This approach can be extended for multilevel component composition.

As believed by Serban (Serban et al., 2009) component selection process is to select a subset of components satisfying the system requirements but as the components share common functionality, an algorithm for decision process is required.

Problem

This paper discuses the simple problem of component selection that is selecting subset of components satisfying the system requirements. The difficulty is that each component has a related set of components that share similar functionalities and due to this a decision making process is required for it.

Proposed Solution
Initially components are selected based on some quality attributes of the components. Metrics are also defined in order to quantify the considered attributes for component selection. The metrics used are Cost, Provided Service Utilization, Required Services Utilization and Functionality. The components are

considered as objects to be clustered and the fuzzy clustering algorithm (Fuzzy n-means algorithm) is used to determine the fuzzy partition.

Critical Analysis
- Need to validate the approach on a real case study.
- The number of clusters can be increased if other clustering algorithms are used.
- The metrics used for selection can be further investigated.

As reported by Nakkrasae et al. (Nakkrasae et al., 2004) software component repository is classified into similarity component cluster group which can be used for retrieval purpose.

Problem

Formal approaches to software reuse depends on specification matching criterion, where a search query using formal specifications is used to search a library of components indexed by specifications. Active research is going on the use of formal methods and component libraries to support software reuse and construction of software based on component specifications. The desired technique must be able to support component storage, retrieval, adaptation and verification.

Proposed Solution
The components are represented using formal specification for describing structure, functional and behavioural properties. Components in software component repository are classified into similar component cluster group with the help of Fuzzy Subtractive Clustering algorithm. The classification index is proposed based on the centre of each cluster and it is compared with the required software component. The component giving the closest match is further used for component selection process. The classification correctness is measured by recall and precision techniques.

Critical Analysis
- In the component repository the number of elements in the set of equivalent classes for a given software component be finite and number of each equivalent class in each property of the components be known.
- This work can be tested by using some other clustering techniques viz. hierarchical clustering, Gaussian clustering, kernel k means, density based clustering so that number of iterations are reduced to generate the cluster centres, components whose functionalities are overlapping are also considered, outliers can be detected easily.

According to Becker et al. (Becker et al., 2010) there is a need for a reliable, repeatable and reproducible software component selection method and tool that varies from component based software development to web service selection and composition.

Problem

Initially software component selection was considered to be a one time process carried out in an organization, but as new paradigms are coming like Service – oriented Architecture where each individual service is partitioned into operational software unit, the number of candidate components are increasing & the trust-worthiness of the components become questionable. The subjective human judgment & declared capabilities of software components cannot replace objective measurements obtained in a controlled environment as the foundation of decision making.

Proposed Solution
A framework for automated component evaluation and selection is given which consists of defining requirements, evaluating alternatives, analyzing results, recommendation, Integrating the product and monitoring conditions. It differs from the general component selection, As monitoring and re-evaluation are integrated in the final stage of the process. The software tool Plato is used for forming tree hierarchy. The complete evaluation of alternatives, transformation and aggregation is used as an evidence base for supporting the decision for recommending the candidate components. The advantage of automated measurements is the degree of evidence and documentation that is produced along with the evaluation.

Critical Analysis
- It is assumed that the components are providing homogeneous functions, that is, fine grained components are used.
- Large number of components are available for selection and ranking controlled environments and automated measurement can be used.
- The controlled experimentation cannot be applied to coarse grained components so it can be extended for that.
- The framework is validated on a specific application domain that is digital preservation & it can be extended for wider context.

Sen et al. (Sen et al., 2010) views software component selection problem is mostly based on functional requirements and non-functional requirements are ignored.

Problem

In most of the software selection methods functional requirements are taken into consideration and non-functional requirements are ignored. Many issues like representation and prioritization of requirements, relation between non-functional and functional requirements are to be tackled.

Proposed Solution
Here, the fuzzy quality function deployment approach for determining which of the non-functional requirements reported by earlier studies are important to a company's software selection decision and integrated with its functional requirements. The solution provided in this study, not only assists decision makers in acquiring software requirements and defining selection criteria, but also supports determining the relative importance of these criteria.

Critical Analysis
It yields that the Quality Function Deployment approach involves four phases 1) Product planning 2) product design 3) Process planning 4) Process control. This study focuses on only first phase; it can be extended for other phases. Some other important points that came into light are:
- The approach needs to be modified for large number and changing requirements.
- The trade-offs and interdependencies among the requirements should be considered and requirement

prioritization techniques can be used.
- Further verification is required to ensure validation of input data.
- For preparing the non-functional criteria template software experts can be involved to identify important and unimportant factors.

Jhadav et.al (Jhadav et al., 2011) have clearly identified three major techniques used so far in software component selection process along with its demerits. Firstly in Analytical Hierarchy Process (AHP) where the decision making process is divided into a hierarchy which on one hand simplifies the problem but on the other hand proves to be a time consuming process due to large number of mathematical calculations and comparisons involved. Secondly, the Weighted Scoring Method (WSM) is easy to use and understand but as the weights is assigned arbitrarily and becomes very difficult when increases number of criteria. The authors also highlights the need decision making framework for software selection.

Problem

The task of selecting software packages is becoming difficult due to many reasons like i) complexity involved for mapping components to business requirements ii) lack of interoperatability among various components iii) lack of experience of decision makers. Thus, the task of selecting software package is complicated and time consuming process. This motivates the researchers to find better ways for software package selection.

Proposed Solution
A decision making framework for evaluation and selection of software packages is proposed which consists methodology for software selection, criteria for evaluating software packages & Hybrid knowledge based system approach.

The methodology for software selection consists of following steps:
a) Requirement Definition
b) Preliminary investigation of availability of software packages
c) Short Listing packages
d) Establishing criteria for evaluation
e) Evaluating software packages
f) Selecting software packages

Common criteria for evaluating software packages consists of Functional, Technical, quality, Vendor, Output, Cost and benefit and Opinion. Hybrid Knowledge Based System (HKBS) approach is used for evaluation and selection of the software packages.

The HKBS approach that employs an integrated rule based and case based reasoning technique for evaluation and selection of software. Rule based reasoning, is a deductive reasoning approach in which reasoning system mimics problem solving behavior of human experts but the rule based reasoning is appropriate for problem solving tasks that are well constructed and case based reasoning, is an inductive reasoning approach in which problem solving approach solve problem by adapting solution of more similar case that has been solved in past.

Critical analysis
The similarity measure used here is based on bottom up approach. The following points can be added in the similarity measure:
- Some machine learning algorithm can be used to calculate the similarity. Then an error function can be computed based on the difference between case order determined by the algorithm and that computed by the similarity measure. So, we need to optimize the similarity measure by minimizing error.
- The adaptability of cases can be considered that is under some situations it is better to select a particular case with a solution that is less useful than the solutions contained in other cases

According to Upadhyay et.al (Upadhyay et. al, 2011), non functional properties plays important role in software selection.

Problem

Most of the existing component selection techniques are concentrated on functional properties and even the non functional properties are for end user point of view only. A major problem in CBSE is the quality of the component used from the system designer, component selector, component acquirer and system integrator point of view and very less research has been done.

Proposed Solution
The methodology consists of digraph and matrix approach for interaction of sub-characteristics. An index has been proposed to rank the components. Based on this user can evaluate and rank potential candidates in component design and development.

Critical Analysis
- The current methodology can be improved by adding cause-effect relationship. The component designer can modify the interaction among sub characteristics to achieve the desired results.
- The values associated with attributes and their interactions need to be specified accurately but as human judgments can be imprecise the fuzzy logic can be added in the methodology

Conclusion

From the different component selection processes discussed it is evident that the problems are more emphasis on functional properties rather than non-functional properties like reliability, performance, security etc, multi-criteria decision for build vs. buy. There is a need to develop context aware applications for mobile devices, how to deal with dynamic requirements. Also there is a need for formal specifications for searching component repositories. The subjective judgments need to be replaced by objective measurements under controlled experiments.

References

1. *Becker, C.; and Rauber, A. (2010), Improving Component Selection and Monitoring with Controlled Experimentation and Automated Measurements, Information and Software Technology, Vol.52, No.6, pp. 641-655.*
2. *Cox, P. T. (2001), A Formal Model for Component-Based Software, in proceedings of Symposia on Human Centric Computing Languages and Environments, pp. 304-311.*
3. *Dong, F.; Zhang, L.; Hu, D. H. and Wang, C. L. 2009, A Case-Based Component Selection Framework for Mobile Context-Aware Applications, in proceedings of International Symposium on Parallel and Distributed Processing with Applications, pp.366-373.*
4. *Jha, P. C.; Arora, R. and Kumar, U.D. (2011), A Fuzzy Approach for Components Selection Amongst Different Versions of Alternatives for a Fault Tolerant Modular Software System under Recovery Block Scheme Incorporating Build-or-Buy Strategy, American Journal of Operation Research, pp. 249-258.*
5. *Jadhav, A. S., and Sonar, R. M. (2011), Framework for Evaluation and Selection of the Software Packages: A Hybrid Knowledge Based System Approach, Journal of Systems and Software, Vol. 84, No.8, pp.1394-1407.*
6. *Kaur, A. and Mann K. S., (2010), Component Selection for Component-Based Software Engineering, International Journal of Computer Applications, Vol.2-No.1, pp. 109-114.*
7. *Nakkrasae S.; Sophatsathit P. and Edwards W. R. (2004), Fuzzy Subtractive Clustering Based Indexing Approach for Software Components Classification, International journal of Computer and Information Science, Vol. 5, No. 1, pp 63-72.*
8. *Pressman R. S. 2006, Software Engineering: A Practitioners Approach, 7th edition McGraw Hill,.*
9. *Serban C.; Vescan A. and Pop H. F. (2009), A New Component Selection Algorithm Based on Metrics and Fuzzy Clustering Analysis in proceedings of the 4th International Conference on Hybrid Artificial Intelligence Systems, pp. 621-628.*
10. *Sen, C. G., and Baracl, H. (2010), Fuzzy Quality Function Deployment Based Methodology for Acquiring Enterprise Software Selection Requirements, Expert Systems with Applications, Vol.-37, Issue-4, pp.3415-3426.*
11. *Stritzinger, A. (1996), A Component-Based Modeling Approach in proceedings of the WOON ' 96, St. Petersburg, Russia, pp. 1-12.*
12. *Tang, J. F.; Mu, L. F.; Kwong, C. K. and Luo, X. G. (2011), An Optimization Model for Software Component Selection under Multiple Applications Development, European Journal of Operation Research, Vol. 212, pp. 301-311.*
13. *Upadhyay, N.; Deshpande, B. M. and Agrawal, V. P. (2011), Concurrent Usability Evaluation and Design of Software Component: A Digraph and Matrix Approach, IET Software,Vol. 5 Issue 2, pp.188-200.*
14. *Vescan A., Grosan Cans Yang S. (2011), A Hybrid Evolutionary Multi objective Approach for the Dynamic Component Selection Problem in proceedings of 11th conference on Hybrid Intelligent Systems, pp. 714-721.*
15. *Serban C.; Vescan, A. and Pop H. (2009), A New Component Selection Algorithm Based on Metrics and Fuzzy Clustering Analysis in proceedings of Hybrid Artificial Intelligent System, pp. 621-628.*