STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 2, 2008

CONSTRAINT OPTIMIZATION-BASED COMPONENT SELECTION PROBLEM

ANDREEA VESCAN AND HORIA F. POP

Abstract. Component-Based Software Engineering (CBSE) is concerned with the assembly of pre-existing software components that leads to a software system that responds to client-specific requirements. Component selection and component assembly have become two of the key issues involved in this process.

We aim at a selection approach that guarantees the optimality of the generated component-based systems, an approach that considers at each step the cost of the selected component and the set of requirements remaining to be satisfied. The dependencies between requirements are also considered. We have modeled the Component Selection Problem as a Constraint Satisfaction Optimization Problem and applied the Branch and Bound algorithm. The experiments and comparisons with the Greedy algorithm show the effectiveness of the proposed approach.

1. Introduction

Since the late 90's Component Based Development (CBD) is a very active area of research and development. CBSE [5] covers both component development and system development with components. There is a slight difference in the requirements and business ideas in the two cases and different approaches are necessary. Of course, when developing components, other components may be (and often must be) incorporated and the main emphasis is on reusability. Components-based software development is focused on the identification of reusable entities and relations between them, starting from the system requirements.

Building software applications using components significantly reduces development and maintenance costs. Because existing components can often be

Received by the editors: October 5, 2008.

2000 *Mathematics Subject Classification.* 68W01, 68N01.

1998 *CR Categories and Descriptors.* D.2 [**Software Engineering**]: Subtopic – *Reusable Software*; D. 1 [**Programming Techniques**]: Subtopic – *Object-oriented Programming* .

Key words and phrases. Component selection problem, Constraint Satisfaction Optimization Problem, Automatic assembly.

reused to build new applications, it is less expensive to finance their development.

In this paper we address the problem of automatic component selection. Generally, different alternative components may be selected, each coming with their own set of offered functionalities (in terms of system requirements). We aim at a selection approach that guarantees the optimality of the generated component-based system, an approach that considers at each step the component with the maximum set of offered functionalities needed by the final system. In our previous research, disseminated in [14], the dependencies between requirements were not taken into account. The current paper considers also the requirements dependencies during the selection process. The compatibility of components is not discussed here, as it will be dealt with in a future development.

We discuss the proposed approach as follows. Related work on Component Selection Problem is discussed in Section 6. Section 2 introduces our approach for Component Selection Problem: Subsection 2.1 presents a formal statement of the Component Selection Problem (CSP), the necessity of normalization in Subsection 2.3 and the modeling of the CSP as Constraint Optimization Problems (COP) in Subsection 2.4. A Greedy and a Branch and Bound approaches are considered. Section 3 presents the elements of the Greedy algorithm and the chosen selection function. The Branch and Bound algorithm is presented in Section 4. Using the example in Section 5 we discuss the two proposed approaches: Greedy and Branch and Bound. We conclude our paper and discuss future work in Section 7.

2. Constructing component-based systems by automatic component selection

In Component-Based Software Engineering, the construction of cost-optimal component systems is a nontrivial task. It requires not only to optimally select the components but also to take their interplay into account.

We assume the following situation: Given a repository of components and a specification of the component-based system that we want to construct (set of final requirements), we need to choose components and to connect them such that the target component-based system fulfills the specification. Informally, our problem is to select a set of components from an available set which may satisfy a given set of requirements while minimizing the number of selected components and minimizing the sum of the costs of the selected components. To achieve this goal, we should assign to each component a set of requirements it satisfies.

2.1. **Formal Statement of the Component Selection Problem.**

Component Selection Problem (CSP) is the problem of choosing the minimum

number of components from an available set such that their composition satisfies a set of objectives (variation of CSP, the cost of each component is not considered). The notation used for formally defining CSP (as laid out in [6] with a few minor changes to improve appearance) is described in what follows.

Problem statement. *Denote by SR the set of final system requirements (target requirements) $SR = \{r_1, r_2, ..., r_n\}$, and by SC the set of components available for selection $SC = \{c_1, c_2, ..., c_m\}$. Each component c_i may satisfy a subset of the requirements from SR, $SR_{c_i} = \{r_{i_1}, r_{i_2}, ..., r_{i_k}\}$. In addition $cost(c_i)$ is the cost of component c_i. The goal is to find a set of components Sol in such a way that every requirement r_j $(j = \overline{1,n})$ from the set SR may have assigned a component c_i from Sol where r_j is in SR_{c_i}, while minimizing $\sum_{c_i \in SSol} cost(c_i)$ and having a minimum number of used components.*

2.2. **Requirement dependencies.** In [13] we have introduced the matrix for the requirements dependencies.

In Table 1 the dependencies between the requirements r_1, r_2, r_3 are specified: the second requirement depends on the third requirement, the third requirement depends on the first and the second requirement.

Dependencies	r_1	r_2	r_3
r_1	$\surd$	$\surd$	
r_2			$\surd$
r_3	$\surd$	$\surd$	

TABLE 1. Dependencies specification table

Some particular cases are required to be checked: no self dependency (the first requirement depends on itself), no reciprocal dependency (the second requirement depends on the third and the third depends on the second requirements) and no circular dependencies (the second requirement depends on the third, the first depends on the second and the third depends on the first). All the above situations are presented in Table 1.

2.3. **Data normalization.** Normalization is an essential procedure in the analysis to compare data having different domain values. It is necessary to make sure that the data being compared is actually comparable. Normalization will always make data look increasingly similar. An attribute is normalized by scaling its values so that they fall within a small-specified range, such as 0.0 to 1.0.

As we have stated above we would like to obtain a system by composing components, a system that will have a minimum final cost and all the requirements are satisfied. The cost of each available component is between 0 and 100. At each step of the construction the number of requirements not

yet satisfied is considered as a criterion to proceed with the search. We must normalize the cost of the components and also the number of requirements yet to be satisfied.

We have used two methods to normalize the data: decimal scaling for the cost of the components and min-max normalization for the requirements not yet satisfied.

Decimal scaling. The decimal scaling normalizes by moving the decimal point of values of feature X. The number of decimal points moved depends on the maximum absolute value of X. A modified value new_v corresponding to v is obtained using:

$$new\_v = \frac{v}{10^n},$$

where n is the smallest integer such that $max(|new\_v|) < 1$.

Min-max normalization. The min-max normalization performs a linear transformation on the original data values. Suppose that $minX$ and $maxX$ are the minimum and maximum of feature X. We would like to map interval $[minX, maxX]$ into a new interval $[new\_minX, new\_maxX]$. Consequently, every value v from the original interval will be mapped into value new_v using the following formula:

$$new\_v = \frac{v - minX}{maxX - minX}.$$

Min-max normalization preserves the relationships among the original data values.

2.4. **Constraint Optimization-based Component Selection Problem.** Constraint Satisfaction Problems (CSPs) are mathematical problems where one must find objects that satisfy a number of constraints or criteria. CSPs are the subject of intense research in both artificial intelligence and operations research. Many CSPs require a combination of heuristics and combinatorial search methods to be solved in a reasonable time.

In many real-life applications, we do not want to find any solution but a good solution. The quality of solution is usually measured by an application dependent function called objective function. The goal is to find such solution that satisfies all the constraints and minimize or maximize the objective function respectively. Such problems are referred to as Constraint Satisfaction Optimization Problems (CSOP).

A Constraint Optimization Problem can be defined as a regular Constraint Satisfaction Problem in which constraints are weighted and the goal is to find a solution maximizing the weight of satisfied constraints. A Constraint Satisfaction Optimization Problem consists [4] of a standard Constraint Satisfaction Problem and an optimization function that maps every solution to a numerical value. The most widely used algorithm for finding optimal solutions is Branch

and Bound and it can be applied to CSOP as well. The Branch and Bound algorithm was first proposed by A. H. Land and A. G. Doig in 1960 for linear programming. In Section 4 a more detail description is given.

3. Greedy Algorithm

Greedy techniques are used to find optimum components and use some heuristic or common sense knowledge to generate a sequence of sub-optimums that hopefully converge to the optimum value. Once a sub-optimum is picked, it is never changed nor is it re-examined.

The Pseudocode of the Greedy algorithm is illustrated in Algorithm 1.

Algorithm 1 Greedy algorithm

Require: SR; {set of requirements}
SC. { set of components }
Ensure: Sol. { obtained solution }

```
1: Sol := ∅; RSR := SR;{RSR=Remaining Set of Requirements}
2: while (RSR <> ∅) do
3:    Choose a c_i from SC, not yet processed;
4:    @ Mark c_i as processed.
5:    if Sol ⋃ { c_i } is feasible then
6:       Sol := Sol ⋃ { c_i };
7:       RSR := RSR - SRc_i;
8:    end if
9: end while
```

The selection function is usually based on the objective function. Our selection function considers the sum of number of requirements to be satisfied (function f) and the cost of the already selected components plus the cost of the new selected component (function g) to be minimal ($(g + h)$ *is minimal*) and all the dependencies are satisfied.

4. Branch and Bound Algorithm

Branch and Bound algorithms are backtracking algorithms storing the cost of the best solution found during execution and use it for avoiding part of the search. More precisely, whenever the algorithm encounters a partial solution that cannot be extended to form a solution of better cost than the stored best cost, the algorithm backtracks, instead of trying to extend this solution.

The term Branch and Bound refers to search methods which have two characteristics that makes them different from other searching techniques:

(1) The method expands nodes from the search tree (this expansion is called *branching*) in a particular manner, trying to optimize the search.

(2) The search technique uses a *bounding* mechanism in order to eliminate (not expand) certain branches (paths) that does not bring any improvements.

The problem solving using $B\&B$ technique is based on the idea of building a search tree during the problem solving process. By a *successor* of a node n we mean a configuration that can be reached from n by applying one of the allowed operations. By *expansion* of a node we mean to determine all the possible successors of the node.

The selection of the successors of a node must also take into consideration the dependencies between requirements. The list of successors of a node is thus reduced.

Because by expanding the initial configuration some configurations can be repeatedly generated, and because the number of nodes can be large, we will not store the entire tree, but only a list with the nodes (configurations) that have to be processed (denoted $SOLUTION\_LIST$). At a given time a node from $SOLUTION\_LIST$ can have one of the following states: *expanded* or *unexpanded*.

The main problem is what node for the list should be selected at a given moment in order to obtain the shortest solution of the problem. Each node n from the list has an associated value (cost function),

$$f(n) = g(n) + h(n),$$

where:

- $g(n)$ represents the cost of the components that were used until now (from the root node to node n) to construct the solution;
- $h(n)$ represents the number of remaining requirements that need to be satisfied (to reach the final solution starting from the current node n). The function h is called heuristic function.

The $B\&B$ [7] algorithm is described using Pseudocode in Algorithm 2.

5. Case study

In order to validate our approach the following case study is used.

Starting for a set of six requirements and having a set of ten available components, the dependencies between the requirements of the components, the goal is to find a subset of the given components such that all the requirements are satisfied.

The set of requirements $SR = \{r_0, r_1, r_2, r_3, r_4, r_5\}$ and the set of components $SC = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9\}$ are given.

In Table 2 the cost of each component from the set of components SC is presented. We have used decimal scaling to normalize the cost of the components.

Algorithm 2 Branch and Bound algorithm

Require: SR; {set of requirements}
SC. { set of components }
Ensure: Sol. { obtained solution }

1: Select a component (node) from the set of available components SC. The component (node) is added into the list $SOLUTION\_LIST$, initially assumed empty (hereby called "the list"). This component has the cost as the value of the function g and the total number of requirements in the set SR, yet to be satisfied, as the value of the function h.
2: **while** (unexpanded nodes still exist in the list) **do**
3: Select from the list the unexpanded node n having the minimum value for the function $f = g + h$.
4: Expand node n and generate a list of successors $SUCC$.
5: **for** (each successor *succ* from $SUCC$) **do**
6: Compute the function g associated to *succ*.
7: Compute the function h associated to *succ*, i.e. the number of remaining requirements from the set SR that need to be satisfied to reach the final solution (with all the requirements satisfied) starting from the node *succ*.
8: **if** (the value of h is 0 (a solution is found)) **then**
9: *Sol* will memorize the best solution between the previously obtained solution (if exists) and the current obtained solution.
10: **else**
11: **if** (component *succ* does not appear in the list) **then**
12: Add *succ* into the list with its corresponding cost value $f(succ) = g(succ) + h(succ)$ and mark as unexpanded;
13: **else**
14: **if** (the value $g(succ)$ is $<$ the g value of the node found in the list) **then**
15: The node found in the list is directed to the actual parent of *succ* (i.e. n) and is associated with the new value of g. If the node was marked as unexpanded, its mark is changed.
16: **end if**
17: **end if**
18: **end if**
19: **end for**
20: **end while**

Table 3 contains for each component the provided services (in terms of requirements of the final system).

Table 4 contains the dependencies between each requirement from the set of requirements.

Component	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
Cost	12	7	3	9	6	14	8	14	7	6
Cost Normalization	0.12	0.07	0.03	0.09	0.06	0.14	0.08	0.14	0.07	0.06

TABLE 2. Cost values for each component in the SC

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
r_0	√		√	√						√
r_1					√				√	
r_2		√				√			√	
r_3	√						√			
r_4						√	√	√		√
r_5		√					√	√		√

TABLE 3. Requirements elements of the components in SC

Dependencies	r_0	r_1	r_2	r_3	r_4	r_5
r_0		√				
r_1						
r_2	√				√	
r_3		√				
r_4	√					
r_5		√				

TABLE 4. Specification Table of the Requirements Dependencies

Table 5 contains the normalization of the number of remain requirements to be satisfied.

5.1. **Results obtained by Greedy algorithm.** In the current section we discuss the application of the Greedy algorithm (presented in Subsection 3) to our problem instance.

The first step of the selection function is the computation of the functions g and h: g is the cost of the used components and h is the number of requirements yet to be satisfied. The component with the minimum value of the function $f = g + h$ is chosen to be a part of the solution. The ties are broken randomly. The dependencies must be also satisfied.

In the first iteration of the algorithm the c_4 component has the minimum value for the function f, i.e. 0.89 and has no dependencies. The set of requirements that are satisfied by choosing the c_4 component is: $\{r_1\}$. Next, only

No. of requirements to be satisified	Normalization	Value
0	0/6	0
1	1/6	0.16
2	2/6	0.33
3	3/6	0.50
4	4/6	0.66
5	5/6	0.83
6	6/6	1

TABLE 5. Normalization of the number of requirements to be satisfied.

the components that may improve the solution (by satisfying new requirements) are considered: $\{c_0, c_1, c_2, c_3, c_5, c_6, c_7, c_8, c_9\}$ but only three of them have all the dependencies satisfied, i. e. $\{c_0, c_2, c_3\}$. The c_0 component has the smallest value of the f function (0.68) and this component is selected to be considered into the solution.

The set of requirements that must still be fulfilled is $\{r_2, r_4, r_5\}$. Only three components may provide some of the remaining requirements and at the same time having all the dependencies satisfied: $\{c_6, c_7, c_9\}$. The c_9 component has the smallest value of the f function (0.40) and this component is the next to be considered for selection.

There is only one requirement to be satisfied, i. e. $\{r_2\}$. Only three components may provide this functionality and all of them have the dependencies satisfied: $\{c_1, c_5, c_8\}$. The component with the minimum value for the g (0.31) function is the c_8 component.

The set of the requirements to be satisfied RSR is empty and we have reached a solution with all the requirements satisfied by the selected components: c_4, c_0, c_9 and c_8. The cost of the final solution 0.31 is the sum of the cost of the selected components. Still, we will see in the next Section 5.2 that there are better solutions with the final cost 0.24: $\{c_4, c_2, c_6, c_1\}$ or $\{c_4, c_2, c_6, c_8\}$.

5.2. **Results obtained by Branch and Bound algorithm.** The Branch and Bound algorithm initialize the first used component in the solution list with the component c_4 (the only component with no dependencies). The set of satisfied requirements is: $\{r_1\}$. The first iteration of the Algorithm 2 adds the $\{c_0, c_2, c_3\}$ components (ordered by the value of the function f) to the list $SOLUTION\_LIST$ (n represents *not expanded node* and e represents *expanded node*).

$$SOLUTION\_LIST = \left\langle \frac{c_0 \;\; c_2 \;\; c_3 \;\; c_4}{n \quad n \quad n \quad e} \right\rangle .$$

The next step of the algorithm expands the first unexpanded node from the list, i.e. c_0. The components that may provide some functionalities from the set of requirements to be satisfied are: $\{c_1, c_5, c_6, c_7, c_8, c_9\}$. Only three components have the dependencies satisfied. The list of successors is reduced to: $\{c_6, c_7, c_9\}$. The new list of nodes is: $\{c_9, c_6, c_7, c_0, c_2, c_3, c_4\}$ with two expanded nodes, components c_4 and c_0.

The next node to be expanded is c_9. Three solutions are found but only the best one is memorized: $\{c_4, c_0, c_9, c_8\}$ with cost 0.31. Next expanded nodes are c_6 and c_7 but the obtained solutions have the cost greater that the previously best obtained solution.

The expansion of the c_2 node modifies the list of nodes. From the list of components that may provide new needed functionalities only four of seven components have the dependencies satisfied: $\{c_0, c_6, c_7, c_9\}$. All the successor are already part of the list but, except the c_0 node, the value of the g function is smaller than the value from the list. The list of nodes is updated according to the new values of f functions.

$$SOLUTION\_LIST = \left\langle \frac{c_6\ c_9\ c_7\ c_0\ c_2\ c_3\ c_4}{n\ \ n\ \ n\ \ e\ \ e\ \ n\ \ e} \right\rangle.$$

The next node that is expanded is the node c_6. The successors are: $\{c_1, c_5, c_8\}$. The new obtained solution considering the c_1 component is better then the current best solution: the cost is $0.24 < 0.31$. The other two obtained solutions (with components c_5 and c_8) have the cost greater or equal that the cost of the new solution, i.e. 0.31 and 0.24.

By expanding next the node c_9 four components may provide the needed functionalities (r_2 or r_3) and all have the dependencies satisfied. For the components c_0 and c_6 the new values for g are greater than those from the list. Therefore the values for the stated components is not going to be changed. The other components will be included into the final list:

$$SOLUTION\_LIST = \left\langle \frac{c_6\ c_1\ c_9\ c_5\ c_7\ c_0\ c_2\ c_3\ c_4}{e\ \ n\ \ e\ \ n\ \ n\ \ e\ \ e\ \ n\ \ e} \right\rangle.$$

With the next expanded node two solutions are found but with the cost greater than the best found solution, i.e. 0.34 and 0.30. By expanding the other nodes no new solution may be found and no new nodes mat be added to the solution list.

The solution obtained with the Branch and Bound algorithm (considering the g function as the sum of the cost of the used components and the h function as the number of requirements to be satisfied) consists of the components: $\{c_4, c_2, c_6, c_1\}$ having the cost 0.24.

6. Related work

Component selection methods are traditionally done in an architecture-centric manner. An approach was proposed in [12], where the authors present a method for simultaneously defining software architecture and selecting off-the-shelf components. They have identified three architectural decisions: object abstraction, object communication and presentation format. Three type of matrix are used when computing feasible implementation approaches. Existing methods include OTSO [10] and BAREMO [11].

Another type of component selection approaches is built around the relationship between requirements and components available for use. In [8] the authors have presented a framework for the construction of optimal component systems based on term rewriting strategies. By taking these techniques from compiler construction they have developed an algorithm that builds a cost-optimal component-based system. In PORE [2] and CRE [1] the same relation between requirements and available components is used. The goal here is to recognize the mutual influence between requirements and components in order to obtain a set of requirements that is consistent with what the market has to offer. The [6] approach considers selecting the component with the maximal number of provided operations. The algorithm in [3] considers all the components to be previously sorted according to their weight value. Then all components with the highest weight are included in the solution until the budget bound has been reached.

Paper [9] proposes a comparison between a Greedy algorithm and a Genetic Algorithm. The discussed problem considers a realistic case in which cost of components may be different.

7. Conclusion and future work

CBSE is the emerging discipline of the development of software components and the development of systems incorporating such components. A challenge in component-based software development is how to assemble components effectively and efficiently.

A proposal for the Component Selection Problem as a Constraint Optimization Problem is given. Two considered approaches are: Greedy and Branch and Bound. Further work will investigate different criteria for component selection: dependencies, different non-functional qualities. A real world system application will be considered next to (better) validate our approach. We have discussed the case when only the dependencies between the requirements from the set of requirements SR. A more real case should be also considered: a component could have other requirements that need to be satisfied before some of its provided services are used.

8. Acknowledgement

This material is based upon work supported by the Romanian National University Research Council under award PN-II no. ID_550/2007.

References

[1] C. Alves, J. Castro, *Cre: A systematic method for cots component selection*, Proceedings of the Brazilian Symposium on Software Engineering, IEEE Press, 2001, pp. 193–207.

[2] C. Alves, J. Castro, *Pore: Procurement-oriented requirements engineering method for the component based systems engineering development paradigm*, Proceedings of the Int'l Conf. Software Eng. CBSE Workshop, IEEE Press, 1999, pp. 1–12.

[3] P. Baker, M. Harman, K. Steinhofel, A. Skaliotis, *Search Based Approaches to Component Selection and Prioritization for the Next Release Problem*, Proceedings of the 22^{nd} IEEE International Conference on Software Maintenance, IEEE Press, 2006, pp. 176–185.

[4] R. Bartk, *Constraint Programming, In Pursuit of the Holy Grail*, Proceedings of the Week of Doctoral Students, Part IV, MatFyzPress, 1999, pp. 555–564.

[5] I. Crnkovic, M. Larsson, *Building Reliable Component-Based Software Systems*, Norwood: Artech House publisher, 2002.

[6] M. R. Fox, D. C. G. Brogan, P. F. Reynolds, *Approximating component selection*, Proceedings of the 36^{th} conference on Winter simulation, 2004, pp. 429–434.

[7] M. Frentiu, H. F. Pop, G. Serban, P*rogramming Fundamentals*, Cluj University Press, 2006.

[8] L. Gesellensetter, S., Glesner, *Only the Best Can Make It: Optimal Component Selection*, Electron. Notes Theor. Comput. Sci. 176 (2007), pp. 105–124.

[9] N. Haghpanah, S. Moaven, J., Habibi, M., Kargar, S. H., Yeganeh, *Approximation Algorithms for Software Component Selection Problem*, Proceedings of the 14^{th} Asia-Pacific Software Engineering Conference, IEEE Press, 2007, pp. 159–166.

[10] J. Kontio, *OTSO: A Systematic Process for Reusable Software Component Selection*, Technical report, University of Maryland, 1995.

[11] A. Lozano-Tello, A. Gómez-Pérez, *BAREMO: how to choose the appropriate software component using the analytic hierarchy process*, Proceedings of the 14^{th} International Conference on Software engineering and knowledge engineering, ACM, 2002, pp. 781–788.

[12] E. Mancebo, A. Andrews, *A strategy for selecting multiple components*, Proceedings of the Symposium on Applied computing, ACM, 2005, pp. 1505–1510.

[13] A. Vescan, *Dependencies in the Component Selection Problem*, Proceedings of the International Conference of Applied Mathematics, Baia–Mare, Romania, 2008 (accepted).

[14] A. Vescan, H. F. Pop, *The Component Selection Problem as a Constraint Optimization Problem*, Proceedings of the Work In Progress Session of the 3^{rd} IFIP TC2 Central and East European Conference on Software Engineering Techniques (Software Engineering Techniques in Progress), Wroclaw University of Technology, Wroclaw, Poland, 2008, pp. 203-211.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: {avescan,hfpop}@cs.ubbcluj.ro