A Game Theoretic Solution for the Optimal Selection of Software Components

Salah Merad and Rogério de Lemos

TECHNICAL REPORT SERIES

No. CS-TR-683

September, 1999

Contact: S Merad and R de Lemos
[Salah.Merad@newcastle.ac.uk, r.delemos@ukc.ac.uk]
http://www.cs.ncl.ac.uk/people/salah.merad/

A Game Theoretic Solution for the Optimal Selection of Software Components

Salah Merad

Department of Computing Science
University of Newcastle
Newcastle upon Tyne NE1 7RU, UK
salah.merad@newcastle.ac.uk

Rogério de Lemos

Computing Laboratory
University of Kent
Canterbury CT2 7NF, UK
r.delemos@ukc.ac.uk

Abstract. It is both difficult and expensive to design and implement a software system that excels in all its functional and non-functional properties. Instead, we could envisage a software system which is made up of several software components having complementary attributes. These components are obtained from a library of software components, each having a specific functional and non-functional profile, and they are selected by a decision maker to satisfy the system requirements and optimise the quality of service under cost constraints. This paper specifically considers the problem of the optimal selection of software components with respect to their non-functional attributes, and describes a game theoretic solution by formulating the problem as a bargaining game. In addition to the game theoretic solution, the paper also discusses other alternative approaches for the optimal selection of software components.

Keywords. Component-based software engineering, multi-attribute optimisation, game theory, value trade-offs, non-functional requirements, utility functions.

1. Introduction

A general trend in the design of complex software systems is to employ off-the-shelf software components aiming to achieve a degree of compositionality comparable with the one already attained for hardware. Component-based software engineering is regarded as an approach for developing large-scale software through component selection, evaluation and assembly where the components are obtained from different sources. Instead of building software systems that are complex in their nature, too costly to design and maintain, and difficult to validate, software systems can be built from simpler software components. Thus the wide range of services to be provided by such software system will depend on the complementary services provided by each of its individual components.

Software systems are normally characterised in terms of their functional and non-functional properties. Although this distinction is controversial, in general terms, the *functional* properties describe what the system does, while the *non-functional* properties impose restrictions on how the system does it; the latter are usually expressed in terms of some observable attributes, such as, dependability, usability and performance. During the design process, depending on the service to be delivered by the system, the designer has to select the components according with their functional capabilities. Once a set of alternative components which satisfy the functional requirements is identified, the next step is to optimally select components with respect to the non-functional requirements of the system, and the non-functional attributes of the identified components. Most of the work in component-based software engineering has focused on the functional aspects of these systems rather than optimising a measure of the utility of their non-functional aspects.

In this paper, a game theoretic approach is proposed for supporting the optimal selection of software components during design-time. Given a user’s non-functional requirements, we consider the problem of selecting a component, or a group of components, from a library of several components, which have the same functionality, but differ in their non-functional attributes (NF-attributes). In addition to the NF-attributes exemplified above, which together specify the quality of service provided by a component, we also consider the cost associated with the components. In the selection process, we seek to “optimise” the quality of service provided by the software system, subject to cost constraints.

The rest of the paper is organised as follows. In section 2, we briefly motivate the game theoretic solution for the selection of software components, in the context of alternative solutions, such the Combined Utility Function introduced by Keeney (Keeney 1976), and the Analytic Hierarchy Process which was used by Kontio (Kontio 1996) to solve a similar problem. In section 3, we present a general model of a system of software components, a solution in the form of a bargaining game, and a small case study to exemplify the proposed approach. Section 4 presents other alternative methods, and the criteria for evaluating the various solution procedures. Finally, section 5 concludes with a discussion evaluating our contribution and indicating directions for future work.

2. Motivation

One of the stages in the lifecycle of component-based software engineering is the selection of components among several alternatives (Brown 1996). Once the functional and non-functional attributes of these components are well characterised at their interfaces (Franch 1998, Frolund 1998), the selection of the “best” component or group of components becomes a multi-attribute decision problem. There exists already a large literature outside the field of software engineering which deals with this type of problem, see, for example, (Fishburn 1970, Keeney 1976) where the authors seek to represent the decision maker’s preferences with a simple function. One of these approaches shows that under some independence assumptions between the attributes, it is possible to represent the user’s preferences and value trade-offs with a combined utility function $U(x_1, x_2, ..., x_N)$ in terms of a simple expression. For example, under the assumption of preference independence, the utility function has the following additive form

$$U(x_1, x_2, \ldots, x_N) = \sum_{n=1}^{N} k_n u_n(x_n),$$

where $u_n(x_n)$, $1 \le n \le N$, is the utility function for attribute X_n, and $k_1, k_2, \ldots, k_N$ are scaling coefficients through which the user’s value trade-offs are evaluated empirically. Hence, the combined utility function (CUF) requires assumptions about the independence between attributes and the empirical evaluation of scaling coefficients. This evaluation process is subjective and often yields inconsistencies, which are difficult to eliminate even after repeating the process many times, especially when the number of attributes is large. For a more detailed account of this method and its applicability for the selection of components, see (Merad 1999).

In the context of software engineering, the complexity of component selection has already been recognised, and a framework that supports multi-variable component selection has been developed (Kontio 1996). In this framework two solution methods are considered: the Weighted Scoring Method (WSM) and the Analytic Hierarchy Process (AHP). The latter was recommended because it is theoretically sound, it constructs the user's utility function for each attribute, and it has a strong empirical validation. It is true that the AHP is superior to the WSM, but it nevertheless has some shortcomings. The AHP is a heuristic method, which yields an additive aggregate utility function, which has the same form as the combined utility function under the assumption of preference independence. In this aggregate utility function, the utilities for each attribute are separately evaluated by pair-wise comparisons of the alternatives; they are in fact positive numbers which add up to 1, and which represent ratio scale preferences between the alternatives. But these utilities are not as accurate as the ones computed using the method of von Neumann when representing the user's strength of preferences and attitude to risk (von Neumann 1947). Moreover, the inconsistencies in the pair-wise comparisons may be difficult to eliminate when the number of components is large. Also in this aggregate utility function the coefficients of the individual utilities are the weights of the attributes, hence not incorporating the user's value trade-offs between the attributes.

For these reasons, we need to define another solution concept less prone to inconsistencies, which is scaleable for a large number of components and attributes, and which has an optimal solution in a well defined sense. Moreover, in addition to the above features, we would like the method to be able to support the selection of components during run-time, in order to adapt to changes that might occur in the operating environment of the software system. In the case of CUF and AHP methods, all the evaluations have to be carried out during design time for a specific set of requirements.

3. A Game Theoretic Solution

In the following, we propose a solution concept, which yields the selection of software components which is the optimal in some well defined sense, without relying on the decision maker's (DM) subjective value trade-offs. For every attribute there is a certain preference pattern over the available alternatives. If we try to optimise simultaneously all the attributes, then the solution will be a compromise between the preference patterns of all the attributes.

This solution is obtained when the DM delegates the decision process to self-interested rational agents, each representing an attribute. The agent for attribute X_n has utility function $u_n(x_n)$. The agents will then have to bargain with each other to reach an agreement on which alternative should be selected. This solution will yield the alternative that is as satisfactory as possible for each attribute. In this formulation, the DM's value trade-offs between the attributes are not incorporated into the structure of the game.

3.1 General Model of a System of Software Components

Let $\mathrm{A} = \{A_1, A_2, ..., A_M\}$ be the set of components, which satisfy the user's requirements. Let $X_1, X_2, ..., X_N$ be N NF-attributes that specify the quality of service of a component. Let a_{mn} be the value of attribute X_n in component A_m, $1 \le m \le M$ and $1 \le n \le N$. Then, vector $\mathbf{a}_m = (a_{m1}, a_{m2}, ..., a_{mN})$ represents the profile of component A_m.

Let rv_n be a required value for attribute X_n. For NF-attributes in which the higher the value the better, the value of attribute X_n in a component must be equal to or greater than rv_n. We denote the user's NF-requirements by the vector $\mathbf{rv} = (rv_1, rv_2, ..., rv_N)$.

Let $u_{mn}, m = 1,..., M$ be the utilities of attribute X_n of a component. The utilities are computed using the method of von Neumann (von Neumann 1947, Keeney 1976), and are scaled so as the utility is 0 at the required value and 1 at the best value (more details on how to compute the utilities, see (Merad 1999)). The utility profile of component A_m is denoted by $\mathbf{u}_m = (u_{m1}, u_{m2}, ..., u_{mN})$.

Let ω_n represent the weight or importance of attribute X_n. We have, $0 \le \omega_n \le 1$, $n = 1,2,.., N$, and $\sum_{n=1}^{N} \omega_n = 1$.

Let c_m be the cost of component A_m, and c^* the budget available for the software system. Without loss of generality, we assume that $c_m \le c^*$ for all $1 \le m \le M$.

Table 1 gives a summary of the components' profiles and their weights together with the user's requirements.

Table 1.components' profiles and user's requirements

Attributes	X_1	X_2	$\dots$	X_N	Cost
Weights	ω_1	ω_2		ω_N	
Required Values Components	rv_1	rv_2	$\dots$	rv_N	c^*
A_1	a_{11}	a_{12}	$\dots$	a_{1N}	c_1
A_2	a_{21}	a_{22}	$\dots$	a_{2N}	c_2
.	.	.			.
.	.	.			.
.	.	.			.
A_M	a_{M1}	a_{M2}	$\dots$	a_{MN}	c_M

Given a user's non-functional requirements (NF-requirements) and a limited budget, the problem is to select the "best" component, or group of components, from a library of components with respect to their NF-attributes and their cost.

3.2 The Nash Solution

Nash proposed a solution to the bargaining game in which, to avoid the prospect of not reaching agreement, the players/agents are willing to submit their conflict to a "fair" arbitrar (Nash 1950): an impartial outsider who will resolve the conflict by suggesting a solution. The arbitration scheme devised by Nash is defined by a function, which associates to each conflict a unique payoff to the players. The arbitration solution should give each player at least as much as the player could get under the worst case, and there should not be any other feasible payoff preferred by all players. In the context of components selection, the worse case corresponds to the required value for an attribute at which the utility is set at 0 for simplicity. For the mathematical formulation of the subjective intuition of fairness, Nash defined a number of "fairness" axioms that can be verbally expressed as follows (Luce 1957):

1. The arbitration solution should not depend upon the particular utility units used by the players.
2. The arbitration scheme should be egalitarian in the sense that it is independent of the names or labels attached to the players.

3. The solution must be robust, i.e. slight perturbations or errors of measurements should not alter drastically the arbitrated solution.

Before we describe the method to find the Nash solution, we will define some terms:

1. A randomised strategy δ is an M-tuple $(\delta_1, \delta_2, \ldots, \delta_M)$, where $0 \le \delta_m \le 1$, $m = 1,2,\ldots, M$ and $\sum_{m=1}^{M} \delta_m = 1$. δ_m is the probability of selecting component A_m.
2. The expected utility for attribute X_n under the randomised strategy δ is given by

$$EU_n(\delta) = \sum_{m=1}^{M} \delta_m u_m(a_{mn}) .$$

3. A strategy δ is dominated if there exists a strategy β such that $EU_n(\delta) \le EU_n(\beta)$, for all $n \in \{1,2,\ldots, N\}$ and at least one inequality is strict.
4. A strategy δ is said to be Pareto optimal if it is not dominated.
5. Let Δ be the set of randomised strategies that are Pareto optimal and such that the expected utility (payoff) for every attribute is positive. Then the set of Pareto-optimal payoffs $V(\Delta)$ is defined as

$$V(\Delta) = \left\{ v = (v_1, v_2, \ldots., v_N) \middle| v_n = EU_n(\delta), \delta \in \Delta \right\}.$$

6. Let v_n, $1 \le n \le N$, be the expected utility for attribute X_n under strategy δ. Then the (generalised) Nash product (Binmore 1992), which gives a measure of the quality of service, is defined as

$$F(v_1, v_2, \ldots, v_N) = \prod_{n=1}^{N} v_n^{\omega_n} .$$

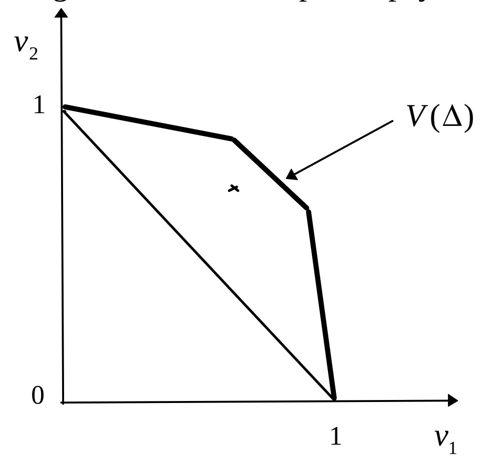
The Nash solution is achieved by maximising the Nash product in the set of Pareto-optimal payoffs. The Nash product not only satisfies the above four “fairness” axioms, it can be shown that it is the only function which does so (Nash 1950). Hence, these fairness conditions implicitly define a unique arbitration scheme for bargaining games.

In terms of the problem of selecting a component, or a group of components, from a library, the set of Pareto-optimal payoffs is a portion of the boundary of the convex hull generated by the utility profiles of the alternative components. This convex hull, which we denote by $H(\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_M\})$, can then be represented as a convex combination of these profiles (Bazaraa 1993), that is

$$H(\{\mathbf{u}_1, ..., \mathbf{u}_M\}) = \left\{ \mathbf{v} = (v_1, ..., v_N) : v_n = \sum_{m=1}^{M} \delta_m u_{mn},\ 1 \le n \le N,\ \delta_m \ge 0 \text{ and } \sum_{m=1}^{M} \delta_m = 1 \right\}.$$

In the context of our problem, this means that the optimal solution will be the combination of at most N components from the pool. Figure 1 illustrates an example with a library of five components and two attributes. The extreme points of the polytope represent four of the components, and the internal point represents the fifth component. The set of Pareto-optimal payoffs is drawn in bold.

Fig.1. Set of Pareto-optimal payoffs



To find the Nash solution, we need to maximise the Nash product in the set of Pareto-optimal payoffs. This is achieved by maximising the Nash product over the region defined by the convex hull $H(\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_M\})$; that is, by solving the non-linear constrained optimisation problem

$$P1: \text{ maximise } F(v_1, v_2, ..., v_N) = \prod_{n=1}^{N} v_n^{\omega_n}$$

subject to:

$$\mathbf{v} \in H(\{\mathbf{u}_1, \mathbf{u}_2, ... \mathbf{u}_M\}).$$

If we substitute v_n by $\sum_{m=1}^{N} \delta_m u_{mn}$ in the expression of $F(v_1, v_2, ..., v_N)$, problem $P1$ is then equivalent to problem

$$P2: \text{ maximise } G(\delta_1, \delta_2, ..., \delta_M) = \sum_{n=1}^{N} \omega_n \ln\left(\sum_{m=1}^{M} \delta_m u_{mn} \right)$$

subject to:

$$\sum_{m=1}^{M} \delta_m = 1,$$
$$\delta_m \geq 0, \text{ for } 1 \leq m \leq M.$$

Problem $P2$ is a non-linear optimisation problem with one linear equality constraint and M linear inequality constraints. There are numerous approximate numerical methods to solve problem $P2$ (Bazaraa 1993), for instance, the Zoutendijk method (Zoutendijk 1960), the successive linear programming approach (Griffith 1961) and the generalised reduced gradient method (Abadie 1969). The latter method is the basis of the solver used in numerous software packages such as LINGO (Schrage 1991) and GRG2 (Lasdon 1978).

Let $\delta^* = \left(\delta_1^*, \delta_2^*, ..., \delta_M^*\right)$ be the optimal solution of problem $P2$. All the software components for which there is a positive probability of being selected will compose the optimal group. As was noted above, there will be up to N components in the optimal group. The total cost C_{δ^*} of the group, which corresponds to the Nash solution δ_m^*, is given by

$$C_{\delta^*} = \sum_{m=1}^{M} \theta_m c_m,$$

where

$$\theta_m = \begin{cases} 1 & \text{if } \delta_m^* > 0 \\ 0 & \text{if } \delta_m^* = 0 \end{cases}.$$

The budget available for expenditure on the system is c^*, and if the total cost C_{δ^*} does not exceed c^*, we say that the solution δ^* is admissible. To avoid having non admissible solutions, we could express the budget requirement as a constraint which we add to the optimisation problem P_2. Unfortunately, we can only approximate this constraint by the nonlinear inequality

$$\sum_{m=1}^{M} \frac{\delta_m c_m}{\delta_m - \varepsilon} \leq c^*,$$

where ε is a small positive number, and the resulting solution turns out to be very sensitive to the choice of ε. Moreover, when there are nonlinear constraints, the standard algorithms do not guarantee convergence to the optimal solution. We hence adopt an iterative process to find a feasible solution.

When the optimal solution is non-admissible, components of the optimal group have to be removed from the pool, and the new Nash solution for the *reduced pool* has to be found. A tree diagram can represent this removal process: the nodes are the pools together with their optimal solutions. The root node is the original library and its solution, and every node has a number of offspring equal to the number of components in the optimal group associated with a non-admissible solution; that is, up to N offspring. If a node yields either an admissible solution or a non-admissible solution, whose Nash product is lower than the highest Nash product of the existing admissible solutions, then the removal process is discontinued on this node. On the other hand, if the Nash product is higher than the one of all the admissible solutions, then the removal process continues.

At the end of the removal process, if there is more than one reduced pool for which the Nash solution is admissible, then the DM will choose the solution with the highest Nash product. But the optimal admissible solution may be only marginally better than some of the other solutions, whereas its cost may be much higher. In this case, the DM may prefer to trade-off the Nash product of the groups against their total costs. For instance, suppose that there are K reduced pools whose solutions are $\delta^{(1)}, \delta^{(2)}, \ldots \delta^{(K)}$, with Nash products $F^{(1)}, F^{(2)}, \ldots F^{(K)}$, respectively. For each solution, we define the value trade-off index $T^{(k)}$ by

$$T^{(k)} = \frac{F^{(k)}}{C_{\delta^{(k)}}},$$

and the solution with the highest index is chosen. But this exhaustive search may require iterating the removal of components up to $M-1$ times. In the worst case scenario, at iteration k, $1 \le k \le M-1$, there are N^{k-1} pools, and hence a total of $\frac{1-N^{M-1}}{1-N}$ pools are examined, which is very large. But if c^* is large enough, say comparable to $N\bar{c}$, where $\bar{c}$ is the average cost of the components in the pool, then the search is very likely to terminate quickly.

When c^* is much smaller than $N\bar{c}$, it is more efficient to consider all the groups of size equal to or smaller than N and whose total cost does not exceed c^*, find the Nash solution for every group, and select the group with the highest Nash product or trade-off index.

3.3 Example

In the following example, the aim is to optimally select a component, or group of components from a pool of four software components which are characterised by the three attributes:

1. X_1 represents the reliability of a component, and it is given by the probability of operating without a fault during a given length of time under a given set of operating conditions;

2. X_2 represents performance, and it is measured in the number of operations executed per unit time;

3. X_3 represents the availability of the system, and it is given by the probability that the system is functioning correctly at any given time.

Table 2 gives the profiles of the four components, together with the user's weights and required values.

Table 2. Data for the example

Attributes	X_1	X_2	X_3	Cost(£)
Weights	0.5	0.25	0.25	
Required Values / Components	0.95	100	0.90	500
A_1	0.97	130	0.92	100
A_2	0.99	110	0.96	170
A_3	0.98	120	0.93	160
A_4	0.96	140	0.94	150

Assuming linear utility functions for all attributes, and setting utilities at 0 at the required values, and at 1 at the maximum values, the utilities are given by

$$u_{mn} = \frac{a_{mn} - rv_n}{a_n^* - rv_n},$$

for $1 \le m \le 4,\ 1 \le n \le 3$, where $a_n^* = \max\limits_{1 \le m \le 4} a_{mn}$. The computed utilities are presented in Table 3.

Table 3. Utility profiles of the components

Attributes / Components	X_1	X_2	X_3
A_1	0.5	0.75	1/3
A_2	1	0.25	1
A_3	0.75	0.5	0.5
A_4	0.25	1	2/3

Using the package LINGO (Schrage 1991), we find that the optimal solution is to select the group $\{A_2, A_4\}$, where components A_2 and A_4 are selected with probabilities 0.84 and 0.16, respectively. Its Nash product is 0.72 and its total cost is 320. This solution is hence admissible. Its trade-off index is $\frac{0.72}{320} = 0.002$.

If the available budget $c^* = 270$, then the above solution is not admissible because the total cost of the group is 320. We need to remove one of the components composing the group.

1. Component A_2 is removed from the pool. The optimal solution in the resulting reduced pool is to select component A_3 only. The Nash product is equal to 0.61 and its cost is 160. This solution is hence admissible. Its trade-off index is $\frac{0.61}{160} = 0.004$.

2. Component A_4 is removed from the pool. The optimal solution in the resulting reduced pool is to select the group $\left\{A_1, A_2\right\}$, and the components are selected with probabilities 0.07 and 0.93, respectively. The Nash product of this solution is 0.71 and its total cost is 260. This solution is hence admissible. Its trade-off index is $\frac{0.71}{260} = 0.003$.

Both solutions are admissible, but group $\left\{A_1, A_2\right\}$ has the higher Nash product, whereas $\left\{A_3\right\}$ has the higher trade-off index.

Note that when the available budget is $c^* = 500$, the optimal group $\left\{A_2, A_4\right\}$ is admissible, but it has a lower trade-off index than the groups $\left\{A_3\right\}$ and $\left\{A_1, A_2\right\}$obtained if the removal process was carried out on the original pool. The Nash product of group $\left\{A_2, A_4\right\}$ is only slightly higher than that of group $\left\{A_1, A_2\right\}$, but the total cost of the latter is significantly lower. However, the search procedure cannot prevent such situations as, once an admissible solution is found at a node, the search is terminated at that node.

4. Alternative Methods and Evaluation Criteria

In the methods considered so far, there is a need to construct utility functions, make assumptions, and perform computations. These tasks can be inaccurate, tedious, and time consuming. However, depending on the problem, simpler methods can be adopted for the selection of components. For instance, if one of the attributes is clearly much more important than the others, than we can make the selection with respect to only that attribute. However, in most cases there is a need for methods that are able to handle the selection of components based on more than one attribute. The first part of this section introduces two alternative methods, and the second part discusses criteria that can be used for evaluating the methods considered in this paper.

4.1 Alternative Methods

The two alternative methods we present below are computationally simpler, at the cost of disregarding the information about the DM's strength of preferences. The solutions obtained using these methods are all Pareto-optimal. Without loss of generality, it is assumed that for all the attributes that characterise a component the high values are the most desirable.

- **Minimum Weighted Sum of Ranks (MWSR)**

  In this method, components are ranked from the best to the worst for every attribute, with the best having rank 1, and the worst rank M.

  Let r_{mn}, with $1 \le r_{mn} \le M$, be the rank of component A_m with respect to attribute X_n. Taking into account all attributes and their importance, the overall rank of component A_m be given by its weighted sum of ranks defined as

$$R_m = \sum_{n=1}^{N} \omega_n r_{mn} .$$

  The optimal component is the one with the lowest overall rank.

  This measure has the same form as the aggregate utility function of the AHP method, but with ordinal individual utility functions. The optimal component is obviously the one with the minimum weighted sum of ranks.

- **Maximum Weighted Product (MWP)**

  In this method, an index is evaluated for every component, and the component with the highest index is selected. The index I_m of component A_m is defined as

$$I_m = a_{m1}^{\omega_1} a_{m2}^{\omega_2} \ldots a_{mN}^{\omega_N} .$$

  This index is a measure of utility first introduced by Bridgeman (Bridgeman 1922) and later used by Miller and Star (Miller 1960) for goal programming problems. But the rationale behind this solution, the implicit assumptions, and how the solution is related to the decision maker’s preference structure were not well understood (Johnsen 1978). This index has in fact the same form as the Nash product when restricted to non-randomised strategies, with the individual utilities being identical to the values taken by the components for the attributes. The individual utility functions are linear and represent ratio scale preferences, however, they are not of the von Neumann type. This method is hence not suitable if the decision maker’s attitude to risk is not neutral, because this gives rise to a nonlinear utility function.

4.2 Evaluation of Methods

In the previous sections, a number of solution concepts were presented for selecting components from a library. The solution concept that should be adopted depends on the decision maker’s interests, the information he/she can provide, the practicality of the computation, and most importantly, the trade-offs between the computational effort and the gain in utility over simpler but “cruder” methods. Below, the methods presented will be evaluated qualitatively with respect to the *information* required, the *type of solution* obtained, and the *computational effort* involved.

4.2.1 Information

To be able to help a DM make a rational choice between the available alternatives, we need to know the profile of every component for the various attributes, the individual utility functions, and how he/she may trade-off between the attributes.

- Profiles: In all the methods, the components need to be measured with respect to every attribute in some chosen scale. In the MWSR method, the components need only to be ranked from the worst to the best for each attribute, so the precision in the measurement is less important than in the other methods so long as the difference between the components is obvious.

- Individual Utility Functions: All the methods use individual utility functions, but the functions used can be different and some may contain more information than others about the DM’s preference patterns. For instance, in the combined utility and the Nash solution, the utility functions are constructed using the method of von Neumann, whereas in the AHP and the MWP they represent ratio scale preferences. In the

MWSR method, the utility functions are ordinal and hence they contain the least information about the DM’s strength of preferences.

- Value Trade-offs: Only the combined utility method incorporates the DM’s value trade-offs explicitly through the scaling constants evaluated in indifference experiments that can only be carried out off-line. As noted above, the method of evaluation of the constants is subjective and can lead to inconsistencies that can be difficult to eliminate completely. Other methods such as the MWSR, the MWP and the Nash solution use the importance coefficients as partial measures of the value trade-offs. A simpler method that bases the component selection on the most crucial attribute, obviously, does not incorporate any measure of value trade-offs.

4.2.2 Type of Solution

All the methods suggested yield Pareto-optimal solutions. This is a minimal requirement for any method. The combined utility function represents the DM’s true preference structure when some independence assumptions between the attributes hold. The AHP and the MWSR methods yield an aggregate utility which approximates the true preferences of the DM, but it is not clear what assumptions are implicit and how good is the approximation. The Nash solution yields a solution which optimises simultaneously all the attributes taking into account their importance and the DM’s strength of preferences for every attribute. The MWP yields a solution of the same type as the Nash solution, but without including the DM’s true structure of preferences for the attributes. The game theoretic solution can yield randomised strategies, and hence to the selection of more than one component. This leads to higher costs, but also to a solution that is robust to small changes in the components’ profiles or the utilities

4.2.3 Computational Effort

The amount of computation needed varies greatly between the methods. In the Nash solution, the main computational effort is the optimisation of a separable function over the boundary of a convex hull for which efficient methods exist. In the combined utility, the main computation is to solve a system of linear equations, as part of the evaluation process of the scaling coefficients. The other methods require very simple arithmetic and sorting operations.

5. Conclusions

As the discipline of component-based software engineering (CBSE) gains momentum as a means to build large and complex systems from components originated from different sources, new methods have to be devised to efficiently build these systems. Within the context of CBSE lifecycle, the activity of selecting components has not yet received its due attention, if compared with the other activities that define this lifecycle (Brown 1996). In this paper, we have described a game theoretic approach for the optimal selection of components from a pool of software components, which can be viewed as a library of components dispersed on the Internet. It is assumed that the components of this pool are functionally identical, but have different, perhaps complementary, non-functional properties. Also it is assumed, when composing components, that the non-functional properties of the individual components are not affected. The problem that we undertake to solve is the optimal selection of software components with respect to their non-functional attributes and their associated costs, and the non-functional requirements of the system to be built and its allocated budget.

Although the scenario of component selection that we considered might appear unrealistic in today's context of the commerce of software components, it is nevertheless plausible in the near future if the current practices of software development change. This will increase the availability of a wide range of software components, thus making it possible for software houses to start commercialising their products. However, before we are able to exercise in practice the optimal selection of software components from the perspective of their non-functional attributes, there is the foremost need of providing the required support for the composition of software components from the functional perspective.

As future work, study is being conducted to identify simpler and more efficient approaches targeted for systems that have to be easily reconfigured during run-time to adapt to changes that occur in their evolving environment. The optimal selection of components should be done without any human intervention, and within the context of plug-and-play architectures (Lowry 1998).

Another issue which needs attention is how to evaluate the integration cost of a group of components to be used together. This cost has to be estimated in advance, and some experience may be needed to reduce the uncertainty in the estimation.

References

1. Abadie J., and Carpentier J. 1969. Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints. in Optimization, Ed. R. Flecher.

2. Bazaraa M. S., Sherali H. D., and Shetty C. M. 1993, Nonlinear Programming: Theory and Algorithms, John Wiley & Sons, Inc., New York, NY.

3. Binmore K. 1992, Fun and Games: A Text on Game Theory, D. C. Heath and Company.

4. Bridgeman P. W. 1922, Dimensional Analysis, Yale University Press, New Haven.

5. Brown A. W. , and Wallnau K. C. 1996. Engineering of Component-Based Systems. in Component-Based Software Engineering, Ed. A. C. Brown, IEEE Computer Society Press, Los Alamitos, CA, pp. 7-15.

6. Chankong V., and Haines Y. Y. 1983, Multiobjective Decision Making. Theory and Method, North-Holland.

7. Fishburn P. C. 1970, Utility Theory for Decision Making, Wiley, New York.

8. Franch X., and Botella P. 1996. Putting Non-Functional Requirements into Software Architecture. Proceedings of the 9th International Workshop on Software Specification and Design, Ise-Shima, Japan, IEEE Computer Society, Los Alamitos, CA, pp. 60-67.

9. Frolund S., and Koistinen J. 1998. Quality-of-Service Specification in Distributed Object Systems. Hewlett-Packard Laboratories, Technical Report 98-158.

10. Griffith R. E., and Stewart A. 1961. A Nonlinear Programming Technique for the Optimization of Continuous Processing Systems, Management Science, 7: 379-392.

11. Johnsen E. 1968, Studies in Multiobjective Decision Models, Studentlitteratur, Lund.

12. Keeney R. L., and Raiffa H. 1976, Decisions with Multiple Objectives: Preferences and Value Tradeoffs, Wiley, New York, NY.

13. Kontio J. 1996. A Case Study in Applying a Systematic Method for COTS Selection. Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany, Los Alamitos, CA, IEEE Computer Society Press, pp. 201-209.

14. Lasdon L. S., Warren A. D., Jain A., and Ratner M. 1978. Design and Testing of a GRG Code for Nonlinear Optimization, ACM Trans. Mathematical Software, 4: 34-50.

15. Lowry M. R. 1988. Component-based reconfigurable systems, Computer, 31: 44-46.

16. Luce R. D., and Raiffa H. 1957, Games and Decisions, Wiley, New York, NY.

17. Merad S., de Lemos R., and Anderson T. 1999. Dynamic Selection of Software Components in the Face of Changing Requirements. Department of Computing Science, University of Newcastle upon Tyne, UK, Technical Report No 664.

18. Miller D. W., and Starr M. K. 1960, Executive Decisions and Operations Research, Prentice-Hall, Englewood Cliffs, NJ.

19. Nash J. F. 1950. The Bargaining Game, Econometrica, 18: 155-162.

20. Saaty T. L. 1990, The Analytic Hierarchy Process, McGraw-Hill, New York, NY.

21. Shrage L. 1991, User’s Manual for LINGO, LINDO Systems Inc, Chicago, IL.

22. von Neumann J., and Morgenstern O. 1947, Theory of Games and Economic Behaviour, Wiley.

23. Zoutendijk G. 1960, Methods of Feasible Directions, Elsevier, Amsterdam, D. Van Nostrand, Princeton, N.J.