Optimized Mismatch Resolution for COTS Selection

Abdallah Mohamed[1], Guenther Ruhe[1], Armin Eberlein[2]
[1]*University of Calgary, 2500 University Drive, NW, Calgary, AB, T2N1N4, Canada*
[2]*American University of Sharjah, Sharjah, P.O. Box 26666, UAE*
{asamoham, ruhe, eberlein}@ucalgary.ca

Abstract

The use of Commercial Off-The-Shelf (COTS) products in the software development process requires the evaluation of existing COTS products, and then selecting the one that best fits system requirements. In this process, it is inevitable to encounter mismatches between COTS features and system requirements. Mismatches occur as a result of an excess or shortage of COTS capabilities. Many of these mismatches are resolved after selecting a COTS. Existing COTS selection approaches fail to properly consider these mismatches. This paper presents MiHOS (Mismatch Handling for COTS Selection), an approach that aims at addressing mismatches while considering limited resources. MiHOS can be integrated with existing COTS selection methods at two points: (1)When evaluating COTS candidates in order to estimate the anticipated fitness of the candidates if their mismatches are resolved. This helps to base our COTS selection decisions on the fitness that the COTS candidates will eventually have if selected. (2) After selecting a COTS product in order to plan the resolution of the most appropriate mismatches using suitable actions, such that the most important risk, technical, and resource constraints are met. A case study from the e-services domain is used to illustrate the method and to discuss its added value.

Keywords: COTS vs. requirements mismatches, MiHOS, COTS-based development.

1. Introduction and Motivation

Employing COTS (commercial off-the-shelf) products to develop software systems has the potential to reduce development time and effort. The term COTS refers to both commercially available and open source products [1] that are reused to build software systems [2,3]. A crucial factor in the success of the final system is to perform a good COTS selection [4], a process that aims at evaluating existing products and selecting the one that best fits the requirements.

In the so called buy-and-adapt approach [3], the selected COTS, although has the best fitness, would still have many mismatches with the requirements. These mismatches occur as a result of an excess or shortage of COTS features[1]. Such mismatches are inevitable as COTS products are made for broad use while system requirements are specific to their context [10,11]. As many of these mismatches as possible are resolved after the selection.

We argue that in the COTS selection process, the selection decision should be based on the anticipated fitness of COTS candidates if their mismatches are resolved. However, due to the fact that there are usually limited resources (i.e. effort and budget) for resolving these mismatches, we typically can only resolve a subset of all COTS mismatches. Resolving different subsets of mismatches will have different impacts on the COTS fitness. The question is, what is the 'right' subset of mismatches that should be addressed under the given resource constraints? This subset has to be identified when comparing the COTS candidates so as to estimate their anticipated-fitness values, and thus make appropriate COTS-selection decisions.

In addition, it is not enough to only identify the right mismatches, but also to apply the right resolution actions. Alternative resolution actions can be used with each mismatch. Different resolution actions require different amounts of resources, and impose different risks on the system. The question is, what is the right set of resolution actions that should be chosen such that the right mismatches are resolved, with the least risk, and within the given resource constraints?

---

[1] The term 'mismatch' as used in this paper should not be confused with other types of mismatches such as architectural mismatches, which are encountered between different parts integrated together in a software system [5-9].

This problem gets more and more complex as the number of mismatches and their alternative resolution actions increase. For some projects, inefficient decisions might not only affect the functionality and quality of the final system, but might also result in serious consequences such as losing market share. Therefore, we suggest the use of decision support techniques [12] to address this problem.

Decision support is a proven means to assist humans to make decisions in case of semi-structured or unstructured problems encountering incomplete or uncertain information. The paradigm of decision support suggests the pro-active evaluation of decision alternatives and aims at providing the best knowledge available to the decision maker to make more informed, transparent and effective decisions.

This paper tries to address the COTS mismatch problem by providing support to decision makers when selecting COTS products. The selection is done based on pro-active analysis of the impact of *mismatch handling*. Mismatch handling is the process in which mismatches and their resolution actions are analyzed; and a selected set of mismatches is resolved using certain resolution actions. For that, the paper presents MiHOS (**Mi**smatch **H**andling for C**O**TS **S**election), an approach that can be integrated into existing COTS selection methods to support handling mismatches. MiHOS has two objectives that may be realized during and after the selection process:

Objective_1 (During COTS selection): MiHOS is used to estimate the anticipated fitness of candidates if the right mismatches for each candidate are resolved.

Objective_2 (After COTS selection): MiHOS is used to help plan the resolution of the right mismatches for the selected COTS using the right set of resolution actions.

MiHOS follows an iterative and evolutionary decision-support framework called EVOLVE* [13]. Instead of providing only one solution to decision makers, MiHOS suggests a portfolio of qualified and structurally diverse solutions [14]. The decision makers are then invited to explore and analyze these solutions, and then they can either accept one solution, or refine the problem model and regenerate further refined solutions. This kind of interactive decision support allows decision makers to have more participation in and control over the decision making process.

This paper consists of six sections. In Section 2, a short overview of COTS selection and mismatch handling is given. Section 3 illustrates MiHOS, while section 4 describes a case study in the e-services domain. Section 5 gives the limitations of applicability. Finally, section 6 provides conclusions and suggestions for future research.

2. COTS Selection and Mismatch Detection

2.1. COTS-Based Development Process

Although the scope of this paper does not cover the full life cycle of the COTS-Based Development (CBD) process, it is necessary to understand the process of CBD in order to position the presented work. Overall, CBD includes five phases [15,16]:

1. *Requirements Engineering*, which defines the desired capabilities and constraints, and helps establishing the COTS evaluation criteria.
2. *COTS Selection*, which ensures selecting COTS products thatbest fits the system requirements. This activity is described in the next section (Section 2.2).
3. *COTS Tailoring*, which involves customizing the COTS product to address unsatisfied (or partially satisfied) requirements. This activity is described in the next section.
4. *COTS Integration*, which is the process of assembling a set of selected COTS products and components together to produce a system.
5. *System Evolution*, which includes maintenance issues such as updating the system with new COTS releases, adding new functionality to the system, and fixing errors.

How does MiHOS, the proposed approach, fit into the above CBD model? MiHOS is a method that supports the process of single '*COTS Selection*', and at the same time investigates the impact of '*COTS Tailoring*' on the selection process. MiHOS also helps human experts plan for '*COTS Tailoring*' after the selection process. On the other hand, MiHOS relies on existing methods such as PORE [4] for performing the '*Requirements Engineering*' activity. Lastly, MiHOS does not address '*COTS Integration*' or '*System Evolution*'.

2.2. The General COTS Selection Model

Several approaches have been proposed to model the COTS selection process; e.g. OTSO[17], CAP [18], and IQMC [19]. Although there is no generally accepted method for COTS selection [12], all methods share some key steps that might be iterative and overlapping. These steps formulate what we refer to by the general COTS selection model:

Step 1: Define evaluation criteria based on system requirements.
Step 2: Search for COTS products.
Step 3: Filter search results based on a set of must-have criteria & define a shortlist of products.
Step 4: Evaluate COTS candidates in the short list.
Step 5: Analyze evaluation data and select the best-fit COTS.

As described in the CBD model (Section 2.1), the selected COTS is tailored in order to resolve as much as possible of its mismatches with the requirements. However, it is not clear in existing COTS selection methods how the mismatches of COTS candidates influence the COTS selection decision at Step 5, especially when limited resources prevent the resolution of all mismatches. Resolving COTS mismatches, as stated by Vigder et al [3], can be realized by:

- *Add-ons*: to acquire additional add-ons that adds functionality to the COTS product.
- *Scripting*: to write custom code using a scripting language supported by the COTS product. Examples of such scripting languages include JavaScript, VisualBasic, and Perl.
- *API*: to develop a controlling program that calls COTS functions using its API interface.
- *Modifying source*, to modify the COTS source code if available. This is a risky action because of several maintenance issues [20-22]

To decide which mismatches are to be resolved using which actions, several factors should be considered: (a) the impact of different mismatches on the COTS-fitness, and (b) the effort, cost, and risk of alternative resolution actions for each mismatch. In addition, the application of the selected set of resolution actions should not exceed available project resources.

2.3. Detecting Mismatches in MiHOS

MiHOS suggests to hierarchically define the COTS-selection criteria using goals [23]. MiHOS defines two types of goals: strategic goals and technical goals. Strategic goals refer to high level requirements that cannot be directly measured in the COTS, i.e. by measuring the performance of one functional attribute of the COTS. Each strategic goal is decomposed into more refined goals (see Figure 1). The decomposition process continues until defining goals that are at the same level of granularity of COTS features. We refer to goals at this level as "technical goals" which represent measurable criteria. The result is a goal graph Γ that addresses stakeholders' strategic needs at the strategic levels, and maps to COTS features at the technical level.

COTS evaluation is done by comparing the COTS features with the technical goals in 1-to-1 relationships. The evaluation score is represented by a matching level (ML) [24] which indicates different levels of satisfaction of technical goals by corresponding COTS features. ML might be estimated on different scales, but eventually ML value is normalized to the range from 0 to 1; 1 indicates full satisfaction of the technical goal by the COTS feature, and 0 no satisfaction.

It is worth mentioning that COTS selection as performed in MiHOS is performed using a simple weighing and aggregation method. The relative weights of the technical goals are computed (as described in the next section). Then, the ML values are multiplied by the weights, and the weighted scores are summed.

Figure 1 goes here!

A mismatch m_i occurs between a COTS feature f_i and a technical goal g_i when $f_i \neq g_i$. A mismatches m_i can be classified into one of five types [24]:

1. *ZeroMatch*: a COTS product fails to satisfy a technical goal.
2. *PartialMatch*: a COTS feature partially satisfies a technical goal.
3. *Surplus*: a COTS has extra feature that is not required.
4. *OverMatch*: a COTS feature exhibits more capability than required.
5. *Equivalence*: a COTS feature contributes to achieving the strategic goal behind a required technical goal, but does not match the technical goal itself.

3. MiHOS: A Proposed Approach to Handle COTS Mismatches

MiHOS is a three-phase approach that allows decision makers to interactively participate in and have more control over the decision making process. The kernel of MiHOS is a formal model of the mismatch handling problem. This formal model is described in Section 3.1; and then MiHOS is described in details in Section 3.2.2.

3.1. Mismatch Handling: A Formal Model

Suppose a COTS has a set of mismatches $\{m_i : i=1 \text{ to } \mu\}$ with the technical goals $\{g_1, g_2, \ldots, g_\mu\}$. The key aspects of the formal model are described in the next two subsections (see Table 1).

3.1.1. Problem Characteristics

a) Relative Importance of Technical-Goals

Consider a mismatch m_i that exists between a COTS feature and a technical-goal g_i. This technical-goal is at the leaf level of a hierarchical graph which consists of Y levels of strategic-goals $G_{i,y}$: $y=1$ *to* Y. Now, assume g_i is assigned a weight of importance υ_i; and $G_{i,y}$ is assigned a weight of importance equals to ω_{iy}. Then for a uniquely defined path $\rho[g_i , G_{i,Y}]$ between the nodes g_i and $G_{i,Y}$, the relative importance Ω_i of the node g_i with respect to $G_{i,Y}$ is obtained by multiplying all weights of the arcs along ρ (see Figure 2):

$$\Omega_i = \upsilon_i \cdot \prod_{y=1\ldots(Y-1)} \omega_{i\,y}$$

In case there is more than one path from a technical-goal to the highest-level strategic-goal, then the path resulting in the maximum relative importance is considered.

Figure 2 goes here!

b) Mismatch Amount

For a mismatch m_i between a COTS feature f_i and a technical goal g_i, the amount of mismatch, $Amount_i$, can be estimated by measuring how much f_i fails to satisfy g_i; $Amount_i = 1 - ML_i$, where ML_i is the matching level between f_i and g_i.

Table1 goes here!

c) Decision Variables

Assume that for each mismatch m_i we identified a set A_i of J possible resolution actions,

$$A_i = \{a_{i,1}, a_{i,2}, \ldots, a_{i,J}\}$$

For instance, if the mismatch m_i indicates that a required functionality is not supported by a COTS, then A_i will include possible ways to tailor this COTS so as to support this functionality; e.g. A_i={*"tailor by custom-code", "tailor by add-ons"*, …}. In MiHOS, the goal is to select one resolution action $a_{i,j}$ for each mismatch m_i. This is described by the set of decision variables

$$X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,J}\}$$

where $x_{i,j}$=1 if the resolution action $a_{i,j}$ is chosen to resolve the mismatch m_i, and $x_{i,j}$=*0* otherwise. This means if a mismatch m_i is to be resolved using action $a_{i,1}$, then X_i={1,0,0,…,0}; and if $a_{i,2}$ is to be used, then X_i = {0,1,0,0,…,0}, and so on.

Note that for the set A_i, we define each resolution action $a_{ij} \in A_i$ to be sufficient to completely resolve m_i. Therefore, we are constrained by choosing only one resolution action to resolve each m_i, This constraint is represented by, $\sum_j x_{ij} \leq 1$. In case a mismatch m_i is tolerated, no resolution action is chosen, and thus X_i = {0**,** 0**,** … **,** 0}.

d) Constraints

There are two types of constraints considered by MiHOS: resource constraints and pre-assignments:

Resource Constraints: Applying resolution actions requires different resources. Each resource has a maximum capacity that should not be exceeded. Assume we have T resource types, and a capacity Cap_t for each resource type t. Every action a_{ij} uses an amount $Usage_{ijt}$ of resource t. Thus, the total amount of resources consumed by the selected set of resolution actions (represented by decision variables x_{ij}) must be less than the total capacity of these resources,

$$(\forall t): \sum_{i,j} x_{ij} \cdot Usage_{ijt} \leq Cap_t$$

For this paper, we consider two types of resources: *available_budget* and *available_effort*. The method remains applicable also if more constraints are defined. That is, we assume each resolution action $a_{i,j}$ requires an effort equal to $effort_{i,j}$ and costs an amount of $cost_{i,j}$. The total effort and budget consumed by the selected resolution actions must be less than the available resources. Thus, the above equation can be broken into the following two equations:

$$\sum_{i,j} x_{i,j} \cdot effort_{i,j} \leq available\_effort$$
$$\sum_{i,j} x_{i,j} \cdot cost_{i,j} \leq available\_budget$$

Pre-assignments: The user has the option to pre-assign a certain mismatch to be either "resolved" or "tolerated" before running the model. If m_i is pre-assigned to be resolved, then the model must select exactly one resolution action to apply. In this case, the sum of decision variables related to all resolution actions A_i applicable to m_i must equal to 1 (i.e., $\sum_j x_{ij} = 1$). On the other hand, if a mismatch m_i is to be tolerated, then no resolution actions should be suggested for it (i.e., $\sum_j x_{ij} = 0$).

e) Technical Risk

We assume applying each resolution action a_{ij} imposes a technical risk equal to r_{ij}. The technical risk is estimated based on: (i) *Failure risk*: the risk that the developers might fail to apply the action $a_{i,j}$, and (ii) *Instability risk*: the risk that an action $a_{i,j}$ would cause instability of the target system. Both (i) and (ii) may be estimated on a 9-point ordinal scale {*1,3,5,7,9*}; this indicates very low, low, average, high, and very high risk. Even numbers can be interpreted as intermediate values. Assuming equal distance between the 9-point scale, the technical risk can be estimated from the formula, $r_{ij} = \alpha \cdot FailureRisk_{ij} + \beta \cdot InstabilityRisk_{ij}$, where: $\alpha + \beta = 1$; α and β indicate the relative importance of the two types of risks, and are defined by experts. In our work, both types are of equal importance, and thus we use $\alpha = \beta = 0.5$.

f) Stakeholders

In many cases, several stakeholders are involved in the estimation process of "*the goals' relative importance*" and "*the technical risk of resolution actions*". Assume we have a set S of K stakeholders abbreviated by $S = \{s_k: k=1 \text{ to } K\}$. The degree of importance of each stakeholder s_k is determined by the relative importance $\lambda_k \in \{1, 3, 5, 7, 9\}$; this indicates very low, low, medium, high, and very high importance respectively. Even numbers (e.g. λ_k =2, 4, 6, and 8) indicate a refinement of values above and below. We use a 9-point ordinal scale to allow sufficient differentiation between stakeholders' degrees of importance.

For aggregating different stakeholders' estimations, we use a weighted average function: (i) If each stakeholder s_k estimated a relative importance Ω_{ik} for a goal g_i, then the overall relative importance is $\Omega_i = \sum_k \lambda_k \cdot \Omega_{i,k}$. (ii) If s_k estimated a technical risk r_{ijk} for a resolution action a_{ij}, then the resultant technical risk is determined by $r_{i,j} = \sum_k \lambda_k \cdot r_{i,j,k}$.

3.1.2. Objective Function

The objective function brings together the problem facets described so far in Section 3. Typically, planning for mismatch handling aims at: (i) Maximizing COTS fitness, (ii) Minimizing risk for resolution of mismatches, and (iii) Fulfillment of resource constraints.

Point (i) is influenced by the goals' relative importance Ω_i and the mismatch amounts $Amount_i$, point (ii) is influenced by the technical risk r_{ij} of selected resolution actions, and point (iii) is relevant to the resource consumptions and constraints. The proposed objective function brings these aspects together in a balanced way. The objective is to maximize function $F(x)$ subject to the satisfaction of the above constraints. $F(x)$ is given as:

$$F(x) = \sum_k \left(Amount_i \cdot \Omega_i \cdot \sum_j (x_{i,j} \cdot \Delta r_{i,j}) \right)$$

Where $\Delta r_{i,j} = 10 - r_{i,j}$ indicates how safe an action a_{ij} is. We use Δr_{ij} instead of r_{ij} because maximizing $F(x)$ should yield the minimum risk (i.e. the maximum safety) of selected actions.

3.2. MiHOS in Action

3.2.1. Generating Alternative Solutions

Qualified Solutions: The above formal model constitutes an optimization problem that could be solved by optimization packages. We used a package called LINGO [25]. As most optimization packages, LINGO is designed to provide one optimal solution at a time for each instance of the problem. However, for decision-making under uncertainty, having a set of near-optimal (or "qualified") solutions is more relevant than having just one [14]. As discussed below, MiHOS provides a portfolio of five qualified solutions. Each solution represents a plan that suggests a strategy for resolving a subset of COTS mismatches. The motivation of offering five plans to the decision maker instead of only one is that it allows addressing implicit and additional concerns not formally described in the original problem statement. In fact, the quality of the five solutions is very similar, but they may differ in terms of their ability to accommodate additional concerns.

MiHOS generates the qualified solutions as follows: Assume that LINGO[2] identifies an optimum solution X_0 with the maximum possible objective function value equal to $F(X_0)$. We define a qualified solution as any near-optimum solution X^* that possesses two characteristics: (i) X^* lies in the feasible space delimited by the problem constraints, and (ii) $F(X^*) \geq \alpha \cdot F(X_0)$, where α is a predefined quality level, and $\alpha \in (0,1]$. In MiHOS, we choose the quality level

[2] Although in this section we describe our diversification algorithm as applied in MiHOS using LINGO, the algorithm could be implemented using any other optimization packag.

α=0.95 in order to address the uncertainty involved when defining the values of problem parameters. Requiring higher accuracy would make little sense with such uncertainty.

Diversified Solutions: The generated alternative solutions should also be significantly different; otherwise they are not really alternatives. A strategy that tries to achieve diversification was introduced in [14]. This strategy tries to simplify the problem by searching only a subset of the feasible space for the most diversified solutions. However, this strategy does not guarantee the identification of the solutions with the maximum diversification among the whole feasible space.

In our approach, we developed another diversification algorithm that tries to increase the level of diversification incrementally:

1. Apply LINGO to the original problem formulation to obtain an optimal solution X_0
2. Set η = 1. η represents the number of structural differences that any two plans in the final solution should have.
3. The algorithm then tries to find four qualified solutions having η structural differences by adding more and more constraints to the problem formulation to ensure diversification:
   a. LINGO is asked to search for a solution X_1 that is qualified (i.e. α=0.95) and satisfies the constraint: "X_1 must have exactly a number of η structural differences with X_0".
   b. Then LINGO is asked to search for another qualified solution X_2 such that it satisfies two constraints:"X_2 must have η differences with X_0," and "X_2 must have η differences with X_1."
   c. Similarly, the algorithm repeats (i) and (ii) to get X_3 and X_4.
4. In steps (i) through (iii), if LINGO could find all four solutions, then these solutions as well as X_0 are stored in the final solution set $SOL = \{X_0, X_1, X_2, X_3, X_4\}$.
5. η is incremented, and step 3 is repeated with the new value of η. The algorithm keeps incrementing η until LINGO fails to solve the problem. When this happens, then the last successfully saved set *SOL* becomes the required portfolio of solutions.

Our algorithm has the advantage of searching the whole set of qualified solutions for maximally diversified solutions. However, our strategy assumes equal distances between the diversified solutions, which still does not guarantee maximum diversification because in reality the distance between the diversified solutions might vary. Yet the diversification obtained from our strategy is very close to the maximum because LINGO searches the whole set of qualified solutions, and keeps trying to find solutions with larger diversification until failing to do so.

In addition, our strategy takes more time than the strategy proposed in [13] – about three to six times longer. Nevertheless, in the case of generating only one set of five alternative solutions, the increase in time would only be in terms of seconds or few minutes, depending on the speed of the computer used in solving the problem and the size of the problem.

Note that our diversification strategy might fail in very rare (unrealistic) cases. For example: if the resource constraints are very low and can only resolve one mismatch, and in the qualified solution space there are no five mismatches that the strategy can pick under the given resource constraints. This is obviously an unrealistic case because MiHOS should only be used when we have many mismatches and the resource constraints, although limited, are sufficient to resolve more than one mismatch.

3.2.2. A Hybrid Approach for Mismatch-Handling during COTS Selection

The above process of solutions generation is applied in MiHOS in a 3-phase process. Before describing the details of this process, we will give the big picture of how MiHOS is integrated into the general COTS selection (GCS) process described in Section 2.2. MiHOS generates a set of plans which are used to generate two outputs (Figure 3):

Output1, the anticipated fitness of COTS candidates if these plans are used to resolve the COTS mismatches. This output is fed to Step 5 of the GCS model to help selecting the best-fit COTS based. This output is used to realizing Objective_1 of MiHOS as discussed earlier.

Output2, a portfolio of mismatch-resolution plans. This output is fed to the COTS tailoring step of the GCS model to help resolving the mismatches of the selected COTS. This output is used to achieve Objective_2 of MiHOS.

Figure 3 goes here!

In order to generate the above two outputs, MiHOS follows a framework called EVOLVE* [13] that provides decision support using hybrid intelligence that brings computational and human intelligence together. The computational part of MiHOS is used to solve the formal model of the problem and to generate a set of qualified solutions with maximum diversification as described earlier. The decision makers are then invited to analyze these solutions and either accept one of them, or refine the formal model based on their knowledge which evolves through successive iterations. Such iterative and evolutionary approach allows addressing the wicked and uncertain character of the problem. This process is realized in three phases: Modeling, Exploration, and Consolidation (see Figure 4).

Figure 4 goes here!

(1) MODELING

The modeling phase aims at describing the settings of the mismatch problem to be suitable for the format given in Section 3. This includes three main tasks:

1. *Identify mismatches and resolution actions*: For each COTS candidate, identify: the set of mismatches m_i and the applicable resolution actions A_i, the technical risk r_{ij} associated with each resolution action a_{ij}, and the resources (i.e. $effort_{ij}$ and $cost_{ij}$) required for each resolution action a_{ij}. Estimating the effort and cost using combination of expert judgment with formal models (e.g. [26]) has proven promising [27].
2. *Identify relative goal importance*: This task includes estimating the relative importance Ω_i of each technical-goal g_i that has a mismatch m_i with a COTS feature.
3. *Estimate constraints*: This task includes estimating both available resource constraints as well as performing any pre-assignments for mismatches resolution.

(2) EXPLORATION

In this phase, the solution space is explored, and a set of alternative qualified solutions (plans) with maximal diversification is generated as discussed earlier (Section 3.2.1). To explain the structure of the resolution plans, consider a COTS having a set of mismatches $\{m_1, m_2, \ldots, m_\mu\}$. A mismatch-resolution plan Y is represented by the set $\{y_1, y_2, \ldots, y_\mu\}$, where for each mismatch m_i, a suggestion y_i may take one of the following values:

- y_i = "*Do not resolve* m_i". All decision variables $x_{i,j}$ are set to zero; no resolution action is used.
- y_i = "*Resolve* m_i *using action* $a_{i,1}$". Only $x_{i,1}$ =1 while $x_{i,j}$ = 0 otherwise; *i.e.* $X_i = \{1, 0, 0, \ldots, 0\}$.
- y_i = "*Resolve* m_i *using action* $a_{i,2}$". Only $x_{i,2}$ =1 while $x_{i,j}$ = 0 otherwise; *i.e.* $X_i = \{0, 1, 0, \ldots, 0\}$.
- …
- y_i = "*Resolve* m_i *using action* $a_{i,J}$". Only $x_{i,J}$ =1 while $x_{i,j}$ = 0 otherwise; *i.e.* $X_i = \{0, 0, 0, \ldots, 1\}$.

Based on the five generated plans, the two outputs of MiHOS are generated as follows:

For Output1: The anticipated fitness of a COTS candidate is estimated in three steps:

1. MiHOS assumes that the first mismatch-resolution plan is applied, and the mismatches are resolved as it suggests.

2. Having the assumption in (1), MiHOS recalculates the fitness of COTS candidates using the weighing and aggregation method described in Section 2.3.
3. MiHOS repeats (1) and (2) for the remaining four mismatch-resolution plans. The overall anticipated fitness is the average of the five fitness values obtained in these three steps:

$$Anticipated\_Fitness = \frac{\sum_{n=1...5} Anticipated\ \ fitness\ assuming\ plan\ \ Y_n\ is\ applied}{5}$$

Where Y_n is a mismatch-resolution plan generated by MiHOS, $Y_n = \{ y_1, y_2, \ldots, y_\mu \}$

For Output2: The second output of MiHOS is the same five mismatch-resolution plans originally generated with the aid of LINGO [25]. These plans are given to decision makers so that they select the one that best suit their interests.

(3) CONSOLIDATION

In this phase decision makers review the output of the exploration phase. They might then refine the problem settings as necessary and go to the next iteration.

For Output1: after evaluating COTS products, MiHOS' consolidation phase enables decision makers to perform *what-if* analysis to examine the impact of 'changing the problem settings' on the 'anticipated fitness' of different COTS candidates. For example, decision makers might want to compare COTS candidates if all mismatches related to security are resolved. This is done using the "pre-assignment" feature of our model. Decision makers can then analyze COTS candidates based on their 'anticipated security scores' as well as their 'anticipated overall scores'.

For Output2: when trying to resolve mismatches of the selected COTS product, MiHOS' consolidation phase enables decision makers to extensively review and analyze the alternative plans suggested in the exploration phase. The decision makers then have the choice either to accept one of the plans, or refine the problem settings and go to the next iteration. During each iteration, decision makers' knowledge evolves and more refinement of the underlying model occurs. The iterations continue until a desirable solution alternative is found.

4. Case Study

To illustrate the proposed method, we use a real-world case study from the e-services domain. The aim of the case study was to acquire a content management system (CMS) for creating an e-business solution. CMS are used to facilitate the creation of news portals using managed contents (i.e. text, images, videos, etc). Due to the page limitation, we briefly describe the main results from the case study in this paper, while full details are given in [24,28].

Initially, a set of 275 goals were defined to represent system needs at different levels of abstraction. From these, 153 goals were defined at the technical-goals level. The COTS market was searched, and a subset of 30 must-have technical-goals was used to define a shortlist of five COTS candidates using progressive filtering [4]. All candidates had mismatches with different mismatch amounts. On the other hand, the system resources for resolving selected COTS mismatches were estimated as: available_effort=60 person-hours, available_budget= $2000.

4.1. Results for Objective_1: Applying MiHOS during COTS Selection

MiHOS was applied to estimate the average anticipated fitness for COTS candidates after resolving the right mismatches. The average is based on the five qualified plans suggested by MiHOS. The differences between the anticipated fitness in these plans did not exceed ±1%. On the other hand, some mismatches were pre-assigned to be resolved due to their criticality while others were pre-assigned to be ignored because no applicable resolution action was found.

Table 2 compares COTS selection with and without the use of MiHOS. The fitness-values in *column2* (i.e. without using MiHOS) was calculated using the weighing-and-aggregation method

discussed in Section 2.2, and the values in *columns 3 to 5* (i.e. after using MiHOS) were calculated using the method discussed in the exploration phase in Section 3.2.22.

If MiHOS was not used, COTS2 might have been selected due to its high fitness value (*column2*). However, MiHOS was applied in several iterations, during which we explored various scenarios of applying small adjustments to the problem constraints. Eventually, it was found that both COTS4 and COTS5 have an anticipated-fitness value (*column3*) higher than COTS2. Yet, it was difficult to choose between COTS4 and COTS5 as we could not rely on the accuracy of such a small difference between their anticipated-fitness values. It was then decided to re-apply MiHOS and investigate the scenario if the mismatches related to the *security* goal of the COTS candidates are resolved – this was done using the pre-assignment constraint (refer to MiHOS' formal model in Sect. 3.1.1(d)). The new results showed that COTS4 is the best choice since it showed the best anticipated overall anticipated fitness (*column4*) with the maximum anticipated security (*column5*). Consequently, we decided to select COTS4 as the best-fit one.

Table2 goes here!

4.2. Results for Objective_2: Applying MiHOS after COTS Selection

After selecting COTS4, we wanted to choose the best plan to solve its mismatches. The same estimates used before for MiHOS' Objective_1 are reused here again (i.e. mismatch amount, effort, etc). The plans suggested by MiHOS for resolving COTS4's mismatches are analyzed more extensively to choose the best one. In more details, COTS4 had an overall of 64 mismatches between its features and the technical-goals. The set of 95% qualified solutions were explored and the most diversified five solutions were identified (see Table 3)

In Table 3, the rows show alternative plans, while the columns refer to different mismatches. The cells containing 'x' suggest tolerating relevant mismatches, while other cells suggest solving relevant mismatches m_i using actions a_{ij}. The last four columns show the evaluation of the suggested plans. As can be seen, these five suggested plans have a consensus on the majority of resolution-actions. Therefore, we felt more confident to use those actions. One the other hand, we focused our analysis on the differences between the suggested plans. All alternative plans were reasonable, and eventually *Alt1* was adopted as it required the least amount of resources.

Table3 goes here!

5. Limitations

As in the case with any approach, the quality of MiHOS' results relies on the quality of the input data. In order to reduce the uncertainty in the input data, we use an evolutionary and iterative approach so as to refine the input values based on expert's feedback after analyzing the output. MiHOS generates a set of near-optimal solutions being structurally diversified instead of only one 'best' solution. Having such tolerance when accepting qualified solutions minimizes the effect of inaccuracies in input values. A similar approach that uses comparable techniques has proven promising in the area of software release planning [13,14].

Also, MiHOS requires extra effort to apply. The total effort required in the case study was 256 person-hours: 96 person-hours for running MiHOS activities (37.7%), and 160 person-hours (62.3 %) for running other activities in the general COTS selection model (i.e. criteria definition, searching for COTS, etc). However, we argue that the extra effort needed for applying MiHOS during and after the selection process is more helpful and less risky than acting reactively if problems occur. We suggest using MiHOS in big projects in which inefficient decisions would have critical consequences, e.g. related to finances or human safety. We also recommend applying MiHOS during the selection process only after having a small set of the most promising COTS candidates so as to reduce the effort required for defining the problem settings. However,

once the problem settings are defined, it is very easy and almost effortless to re-run MiHOS several times so as to explore different scenarios of the problem. For example, in our case study the scenario "*evaluate COTS candidates if their security-score is maximized*" was done in no time, and yet yielded important information that significantly affected our decision.

In addition, MiHOS is currently dedicated for single COTS selection. Also, MiHOS is not designed to handle mismatches related to non-functional requirements (NFR). This problem arises from the difficulty of estimating the effort/cost for achieving a specific target level by solving such mismatches. Although in our case study, MiHOS was able to deal with several NFRs (e.g. security), applying it to some other NFRs (e.g. reliability) would be difficult.

6. Summary and Outlook

We proposed MiHOS, a decision-support approach that can be integrated with most existing COTS selection methods at two points: (1) when evaluating the most promising COTS products in order to estimate the anticipated fitness if the right mismatches are resolved for each COTS; and (2) after selecting a COTS in order to help planning the resolution of the right mismatches using appropriate resolution-actions. MiHOS takes into consideration factors such as the impact of a mismatch on COTS fitness, and the cost, effort and risk of resolution actions. The case study showed the significance of using MiHOS in qualifying the COTS-selection decisions.

Despite the limitations in Section 5, the structure of MiHOS is of high *extensibility* to address more complex problems. For example, instead of considering the overall-effort for applying resolution-actions, the formal model can be easily modified to address more refined types of effort; e.g. development, testing, and documentation efforts. This differentiation between effort types is necessary when having different teams in the project with different effort constraints.

Also, instead of assuming that each mismatch is resolved by only one resolution-action, the formal model can be modified to consider several resolution actions that together can solve one or more mismatches. This can be done by adding some technological constraints that define the dependencies between resolution-actions.

We are also working on exploring the use of *'linear-combination'* for formulating the objective function, and comparing the results to the current objective function. Linear combination might be useful to conduct more complex *'what-if'* analysis. This increases the customizability of the objective function towards experts' preferences.

7. Acknowledgement

Our thanks go the Natural Sciences and Engineering Council of Canada (NSERC) and to Alberta Informatics Circle of Research Excellence (iCORE) for their financial support to this research.

8. References

1. Torchiano M, Morisio M. Overlooked Aspects of COTS-Based Development. *IEEE Softw.* 2004;21(2): pp.88-93.
2. Firesmith DG. Achieving Quality Requirements with Reused Software Components: Challenges to Successful Reuse. MPEC'05 Workshop in conjunction with the ICSE'05; May 2005; St. Louis, Missouri.
3. Vigder MR, Gentleman WM, Dean J. COTS Software Integration: State of the art. Report 39198: National Research Council Canada (NRC); Jan 1996.
4. Maiden NA, Ncube C. Acquiring COTS Software Selection Requirements. *IEEE Software* 1998;15(2): pp.46-56.
5. Garlan D, Allen R, Ockerbloom J. Architectural Mismatch: Why Reuse Is So Hard. *IEEE Software* 1995;12(6): pp.17-26.
6. Gacek C, Abd-Allah A, Clark B, Boehm B. On the Definition of Software System Architecture. ICSE First International Software Architecture Workshop; 1995; Seattle, WA. p 85-95.
7. Shaw M. Architectural Issues in Software Reuse: It's Not Just the Functionality, It's the Packaging. ACM SIGSOFT Symposium on Software Reusability; 1995. p 3-6.

8. Shaw M, Clements P. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. Proceedings of COMPSAC'97 conference; 1997 Aug. p 6-13.
9. Yakimovich D, Bieman JM, Basili VR. *Software architecture classification for estimating the cost of COTS integration*. IEEE Computer Society Press: Los Angeles, California, United States; 1999. 296-302 p.
10. Alves C. COTS-Based Requirements Engineering. In: *Component-Based Software Quality – Methods and Techniques*. Volume 2693, Lecture Notes in Computer Science: Springer; 2003. p 21-39.
11. Carney D, Hissam SA, Plakosh D. Complex COTS-based software systems: practical steps for their maintenance. *Journal of Software Maintenance* 2000;12(6): pp.357-376.
12. Ruhe G. Intelligent Support for Selection of COTS Products. *Web, Web-Services, and Database Systems, Lecture Notes in Computer Science (Springer)* 2003;2593: pp.34-45.
13. Ruhe G, Ngo-The A. Hybrid Intelligence in Software Release Planning. *International Journal of Hybrid Intelligent Systems* 2004;1(2): pp.99-110.
14. Ngo-The A, Ruhe G. A Systematic Approach for Solving the Wicked Problem of Software Release Planning. *Appears in: Soft Computing* 2008;12(1).
15. Abts C, Boehm BW, Clark EB. COCOTS: a COTS software integration cost model: model overview and preliminary data findings. The 11th ESCOM conference; 2000 April; Munich, Germany. p 325-333.
16. Boehm BW, Port D, Yang Y, Bhuta J. *Not All CBS Are Created Equally: COTS-Intensive Project Types*. Springer-Verlag: Berlin Heidelberg; 2003. 36-50 p.
17. Kontio J. OTSO: A Systematic Process for Reusable Software Component Selection. Report CS-TR-3478. Maryland: University of Maryland; 1995 December.
18. Ochs M, Pfahl D, Chrobok-Diening G, Nothhelfer-Kolb B. A COTS Acquisition Process: Definition and Application Experience. ESCOM-SCOPE 2000; 2000; Munich, Germany. p 335-343.
19. Franch X, Carvallo JP. Using Quality Models in Software Package Selection *IEEE Softw.* 2003;20(1): pp.34-41.
20. Brownsword L, Carney D, Oberndorf T. The opportunities and complexities of applying commercial off-the-shelf components. *CrossTalk* 1998;11(4): pp.25-30.
21. Morisio M, Seaman CB, Parra AT, Basilli VR, Kraft SE, Condon SE. Investigating and Improving a COTS-Based Software Development Process. ICSE'00; 2000; Limmerick, Ireland. p 32-41.
22. CeBASE. Center for Empirically Based Software Engineering, at www.cebase.org.
23. Kontio J. A Case Study in Applying a Systematic Method for COTS Selection. 18th International Conference on Software Engineering (ICSE'96); 1996 March; Berlin, Germany. p 201-209.
24. Mohamed A. Decision Support for Selecting COTS Software Products based on Comprehensive Mismatch Handling [PhD]. Canada: University of Caglary; 2007.
25. LINDO_Systems. http://www.lindo.com.
26. Li J, Ruhe G, Al-Emran A, Richter M. A Flexible Method for Effort Estimation by Analogy. *Empirical Software Engineering* 2006;12(1): pp.65-106.
27. Briand LC, Wieczorek I. Resource Estimation in Software Engineering. In: *Encyclopedia of Software Engineering (2nd edition)*, Marciniak JJ (Eds.). New York: John Wiley; 2001. p 1160-1196.
28. Mohamed A, Ruhe G, Eberlein A. A Case Study in Handling Mismatches During OTS Selection. Report 057/2006. Calgary: SEDS Lab, UofC; 2006 July.

Table 1 Input parameters of MiHOS

Technical goals		COTS1														COTS2								…
		Mismatches		Resolution actions												Mism-atches		Resolution Actions						…
				$a_{i,1}$					…	$a_{i,J}$						…	…	…		…				…
ID	Relative weight	ID	*Amount*	ID	Decision Variables	*effort*	*cost*	*risk*	…	ID	Decision Variables	*effort*	*cost*	*Risk*	…	…	…	..	..	..	..	..	…	
g_1	Ω_1	m_1	*Amount* $_1$	$a_{1,1}$	$x_{1,1}$	$effort_{1,1}$	$cost_{1,1}$	$r_{1,1}$	…	$a_{1,J}$	$x_{1,J}$	$effort_{1,J}$	$cost_{1,J}$	$r_{1,J}$	…	…	…	..	..	..	..	..	…	
g_2	Ω_2	m_2	*Amount* $_2$	$a_{2,1}$	$x_{2,1}$	$effort_{2,1}$	$cost_{2,1}$	$r_{2,1}$	…	$a_{2,J}$	$x_{2,J}$	$effort_{2,J}$	$cost_{2,J}$	$r_{2,J}$	…	…	…	..	..	..	..	..	…	
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	..	..	..	..	..	…	
g_μ	Ω_μ	m_μ	*Amount* $_\mu$	$a_{\mu,1}$	$x_{\mu,1}$	$effort_{\mu,1}$	$cost_{\mu,1}$	$r_{\mu,1}$	…	$a_{\mu,J}$	$x_{\mu,J}$	$effort_{\mu,J}$	$cost_{\mu,J}$	$r_{\mu,J}$	…	…	…	..	..	..	..	..	…	

Table 2 Analyzing COTS candidates before and after applying MiHOS

	Fitness without resolving mismatches*	After resolving the 'right-mismatches' identified by MiHOS		
		Anticipated fitness	Anticipated fitness with max security	Security (Before→After)
COTS1	63 %	78 %	68 %	84% → 100%
COTS2	75 %	82 %	81 %	86% → 92 %
COTS3	61 %	73 %	70 %	70% → 77.9%
COTS4	68 %	92 %	91 %	72% → 100%
COTS5	53 %	90 %	69 %	58% → 95%
Column1	*Column2*	*Column3*	*Column4*	*Column5*

Table 3 Suggested resolution plans for COTS4

Mismatches	m (1)	m (2)	m (3)	m (4)	m (5)	m (6)	m (7)	m (8)	m (9)	m (10)	m (11)	m (12)	m (13)	m (15)	m (16)	m (17)	m (18)	m (19)	m (20)	m (23)	m (24)	m (25)	m (26)	m (27)	m (30)	m (31)	m (32)	m (41)	m (44)	m (45)	m (46)	m (47)	m (48)	m (49)	m (50)	m (51)	m (52)	m (53)	m (54)	m (55)	m (56)	m (57)	m (58)	m (59)	m (60)	m (61)	m (62)	m (63)	m (64)	Obj. Funct. Value	Effort (person-hours)	Cost ($)	Resultant COTS Fitness (%)
alternative resolution plans Alt 1	$a_{1,1}$	$a_{2,1}$	$a_{3,1}$	$a_{4,1}$	$a_{5,1}$	$a_{6,1}$	$a_{7,1}$	$a_{8,1}$	$a_{9,2}$	x	x	$a_{12,1}$	$a_{13,1}$	$a_{15,1}$	x	$a_{17,1}$	x	x	$a_{20,1}$	$a_{23,1}$	$a_{24,1}$	x	$a_{26,1}$	$a_{27,1}$	$a_{30,1}$	x	x	$a_{41,1}$	$a_{44,1}$	$a_{45,1}$	x	x	$a_{48,1}$	x	$a_{50,1}$	$a_{51,1}$	$a_{52,1}$	$a_{53,1}$	$a_{54,1}$	$a_{55,1}$	x	x	$a_{58,1}$	$a_{59,1}$	$a_{60,1}$	$a_{61,1}$	$a_{62,1}$	x	$a_{64,1}$	1535	59	1180	91
Alt 2	$a_{1,1}$	$a_{2,1}$	$a_{3,1}$	$a_{4,1}$	$a_{5,1}$	$a_{6,1}$	$a_{7,1}$	$a_{8,1}$	$a_{9,2}$	x	x	$a_{12,1}$	$a_{13,1}$	$a_{15,1}$	x	x	$a_{18,1}$	x	x	$a_{23,1}$	$a_{24,1}$	x	$a_{26,1}$	$a_{27,1}$	x	x	x	$a_{41,1}$	$a_{44,1}$	$a_{45,1}$	$a_{46,1}$	$a_{47,1}$	x	x	$a_{50,1}$	$a_{51,1}$	$a_{52,1}$	$a_{53,1}$	$a_{54,1}$	$a_{55,1}$	$a_{56,1}$	x	$a_{58,1}$	$a_{59,1}$	x	$a_{61,1}$	$a_{62,1}$	$a_{63,1}$	$a_{64,1}$	1520	60	1200	92
Alt 3	$a_{1,1}$	$a_{2,1}$	$a_{3,1}$	$a_{4,1}$	$a_{5,1}$	$a_{6,1}$	$a_{7,1}$	$a_{8,1}$	$a_{9,1}$	x	x	$a_{12,1}$	$a_{13,1}$	$a_{15,1}$	x	$a_{17,1}$	x	$a_{19,1}$	$a_{20,1}$	$a_{23,1}$	$a_{24,1}$	$a_{25,1}$	$a_{26,1}$	$a_{27,1}$	x	x	x	$a_{41,1}$	$a_{44,1}$	$a_{45,1}$	$a_{46,1}$	$a_{47,1}$	x	x	$a_{50,1}$	$a_{51,1}$	$a_{52,1}$	$a_{53,1}$	$a_{54,1}$	$a_{55,1}$	x	x	x	$a_{59,1}$	$a_{60,1}$	$a_{61,1}$	$a_{62,1}$	$a_{63,1}$	$a_{64,1}$	1514	60	1200	91
Alt 4	$a_{1,1}$	$a_{2,1}$	$a_{3,1}$	$a_{4,1}$	$a_{5,1}$	$a_{6,1}$	$a_{7,1}$	$a_{8,1}$	$a_{9,1}$	x	x	$a_{12,1}$	$a_{13,1}$	$a_{15,1}$	x	$a_{17,1}$	$a_{18,1}$	$a_{19,1}$	x	$a_{23,1}$	x	$a_{25,1}$	$a_{26,1}$	$a_{27,1}$	$a_{30,1}$	x	x	$a_{41,1}$	$a_{44,1}$	$a_{45,1}$	$a_{46,1}$	x	$a_{48,1}$	x	$a_{50,1}$	$a_{51,1}$	$a_{52,1}$	$a_{53,1}$	$a_{54,1}$	$a_{55,1}$	$a_{56,1}$	x	$a_{58,1}$	$a_{59,1}$	x	$a_{61,1}$	$a_{62,1}$	$a_{63,1}$	$a_{64,1}$	1502	60	1200	92
Alt 5	$a_{1,1}$	$a_{2,1}$	$a_{3,1}$	$a_{4,1}$	$a_{5,1}$	$a_{6,1}$	$a_{7,1}$	$a_{8,1}$	$a_{9,2}$	x	x	$a_{12,1}$	$a_{13,1}$	$a_{15,1}$	$a_{16,1}$	x	x	$a_{19,1}$	x	$a_{23,1}$	$a_{24,1}$	$a_{25,1}$	$a_{26,1}$	$a_{27,1}$	x	x	x	$a_{41,1}$	$a_{44,1}$	$a_{45,1}$	x	x	x	$a_{49,1}$	$a_{50,1}$	$a_{51,1}$	$a_{52,1}$	$a_{53,1}$	$a_{54,1}$	$a_{55,1}$	x	x	x	$a_{59,1}$	x	$a_{61,1}$	$a_{62,1}$	x	$a_{64,1}$	1496	60	1200	90