Complex Systems Informatics and Modeling Quarterly
CSIMQ, Issue 5, December 2015/January 2016, Pages 14-25
Published online by RTU Press, https://csimq-journals.rtu.lv
http://dx.doi.org/10.7250/csimq.2015-5.02
ISSN: 2255-9922 online

A Formal Method for Conceptual Fit Analysis

Antoni Olivé

Department of Service and Information System Engineering
Universitat Politècnica de Catalunya – Barcelona Tech, Spain

antoni.olive@upc.edu

Abstract. One criterion that has received little attention in Commercial-off-the-shelf (COTS) systems selection is what we call conceptual fit. The criterion assesses the fit between the conceptual structure of the user requirements and that of a candidate system. In this paper, we evaluate the fit in terms of the existing misfits. We formally define the notion of conceptual misfit and we present a method that determines the conceptual misfits between the user requirements and a set of candidate systems. The method consists of defining a superschema, expressing the user requirements and the implementation of the candidate systems on the basis of that superschema, and the automatic computation of the existing conceptual misfits. The method has been formalized in UML/OCL. We show how our method improves on existing conceptual fit analysis methods for COTS selection.
Keywords: COTS selection, Conceptual Fit, Conceptual Modeling.

1 Introduction

Nowadays, organizations often build their information systems by customizing and/or integrating Commercial-off-the-shelf (COTS) systems [1]. In most cases, there are several alternative COTS systems that could be used to build an information system. Selecting the most convenient COTS system for a particular situation has become a critical activity in information systems engineering.

In general, COTS systems selection is a difficult decision for an organization due to the diversity of those systems, the possible large number of candidates, the large number of technical and non-technical characteristics that must be taken into account, and the possible high impact of the decision on the future activities of the organization [2].

The difficulty, frequency and practical significance of COTS systems selection justify the large volume of research work devoted to it and the large number of selection methods that have been proposed so far. Early published works date back at least to 1995 [3], and it is still an active research area. See [4], [5] for recent surveys on this topic.

COTS system selection essentially consists in evaluating user requirements with respect to characteristics of candidate systems. The evaluation is performed by defining a set of criteria, assessing the importance of each criterion for the users and the degree to which the criterion is satisfied by a system. Evaluation criteria must be customized for each selection situation [3]. The criteria taken into account usually include functionality, quality attributes, architecture, costs and risks.

One kind of criterion that has received little attention is what we call conceptual fit. It is similar to what is called domain compatibility in Systematic Process for Reusable Software Components Selection (OTSO), which refers to how well a system and its features map into the terminology and concepts of the domain [3]. It is also similar to what is called suitability of data in the GOThIC method, which evaluates how a particular system represents the data of a UML class or association of a common domain model [6].

This paper analyzes the conceptual fit between user requirements and COTS systems. We formally define the notion of *conceptual misfit* and we present a formal method that determines the existing conceptual misfits between a set of user requirements and a system. The absence of conceptual misfits indicates a perfect conceptual fit. We propose conceptual fit as a criterion to be used for COTS selection because it enables an early discrimination between candidate systems, which reduces the effort of the selection [7]. It can be taken into account in almost all existing selection methods.

Our notion of conceptual misfit has been inspired in the ontological expressiveness analysis [8], in the fitness relationship between a business and the system which supports it [9], and in CASSM, an analytical usability evaluation method of interactive systems that focuses on conceptual fit [10].

The structure of the paper is as follows. The next section formally identifies the different kinds of conceptual misfits that may exist between a set of user requirements and a COTS system. Section 3 formalizes the general problem of evaluating the conceptual fit of a set of user requirements and a set of COTS systems. In Section 4 we describe the method we propose for solving that problem. Section 5 analyzes how conceptual fit could be integrated into three existing COTS selection methods. Finally, Section 6 summarizes the conclusions and points out future work.

2 Conceptual Fit

By conceptual fit we mean the fit between two structural conceptual schemas. In our context, one conceptual schema is that of the user requirements and the other one is that of a particular COTS system. For the purposes of this paper, we will assume simple structural conceptual schemas consisting only of entity types, ISA hierarchies, attributes and binary associations. We assume that *n*-ary associations and association classes have been decomposed into their equivalent classes and binary associations [11].

Figure 1 shows the metamodel *M* in UML of the schemas that we consider in this paper. Entity types have a name, and may have sub/supertype associations between them. An abstract entity type is a derived entity type whose population is the union of that of its subtypes. An entity type is singleton if it has only one instance; otherwise it is assumed that its cardinality is unconstrained. Entity types may have attributes, which are properties. Properties have a minimum and a maximum cardinality, and a type. Cardinalities may be zero, one or unconstrained. Associations have two ordered participants, each of which is a property, as before.

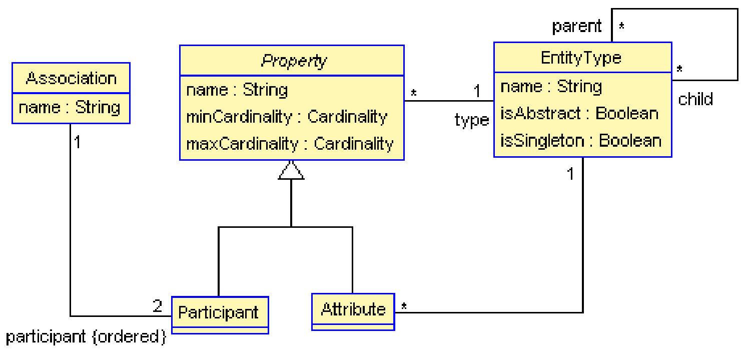

Figure 1. The metamodel of the schemas considered in this paper

Assume now that we have two schema instances of *M* that we call U^S (for user requirements) and S_j^S (for COTS system *j*). We are interested in knowing how well the schemas U^S and S_j^S fit each other. To this end, we try to see whether there are misfits between them. Based on the simple metamodel M we identify three kinds of misfits in the schema elements, which we call deficits, incompatibilities and excesses, and that we define in the following subsections. Of course, in a more complex metamodel, additional misfits could be identified. The idea is that the degree of fit of U^S and S_j^S is inversely proportional to the number of misfits, the maximum being the absence of them.

We illustrate the analysis by means of examples from the domain of e-commerce platforms that we have used in our experiments. The domain consists of three content-management systems (osCommerce [12], Magento [13], CS-Cart) and Amazon webstore.

2.1 Entity Type Misfits

We say that there is an *entity type deficit* between U^S and S_j^S with respect to (wrt) E if E is a concrete entity type of U^S but E is not an entity type of S_j^S. Note that we consider only the concrete entity types of U^S because these are the ones of interest to the users.

For example, if U^S includes the concrete entity type *Bundle* then there is an entity type deficit between U^S and osCommerce wrt to *Bundle* because that system does not include *Bundle*. It is not possible to define instances of bundles in that system.

There is an *entity type cardinality incompatibility* between U^S and S_j^S wrt E if E is a concrete entity type of U^S and an entity type of S_j^S, but E is unconstrained (not a singleton) in U^S and a singleton in S_j^S. Both U^S and S_j^S have the entity type E but, in U^S, E may have several instances while only one instance is allowed in S_j^S.

For example, if U^S requires an unconstrained concrete entity type *Store*, then there is an entity type cardinality incompatibility between U^S and osCommerce wrt to *Store,* because *Store* is a singleton in osCommerce. An e-shop may not have several stores in osCommerce.

We say that there is an *entity type excess* between U^S and S_j^S wrt E if E is a concrete entity type of S_j^S but E is not an entity type of U^S. In this case, S_j^S includes an entity type that is not of interest to U^S. For example, Magento includes the concrete entity type *GroupedProduct*. If this type is not required by U^S then there is an entity type excess.

2.2 Attribute Misfits

There is an *induced attribute deficit* between U^S and S_j^S wrt A if A is an attribute of the concrete entity type E in U^S and there is an entity type deficit between U^S and S_j^S wrt E. In this case, the deficit is induced by the entity type deficit. For example, if U^S includes the attribute *price* of *Bundle*, then there will be an induced attribute deficit with all systems whose schema does not include *Bundle*.

There is an *attribute deficit* between U^S and S_j^S wrt A if A is an attribute of the concrete entity type E in U^S, S_j^S includes E, but S_j^S does not include A.

There is an *attribute cardinality incompatibility* between U^S and S_j^S wrt A if A is an attribute of the concrete entity type E in U^S, S_j^S includes A, but the cardinalities are incompatible. An incompatibility arises when the minimum cardinality in U^S is zero and one in S_j^S, or when the maximum cardinality is unconstrained in U^S and one in S_j^S. An example of this misfit occurs when users require that *SaleableItem* may have several images (unconstrained attribute) and a system (such as osCommerce) allows at most one.

There is an *induced attribute excess* between U^S and S_j^S wrt A if A is an attribute of the concrete entity type E in S_j^S and there is an entity type excess between U^S and S_j^S wrt E. In this case, the excess is induced by the entity type excess.

There is an *attribute excess* between U^S and S_j^S wrt A if A is an attribute of the concrete entity type E in S_j^S, U^S includes E, but U^S does not include A. In this case, S_j^S includes an attribute that is not of interest to U^S.

2.3 Association Misfits

There is an *induced association deficit* between U^S and S_j^S wrt R if R is an association between the concrete entity types E_1 and E_2 in U^S, and there is an entity type deficit between U^S and S_j^S wrt E_1 or E_2. In this case, the deficit is induced by the entity type deficits.

There is an *association deficit* between U^S and S_j^S wrt R if R is an association between the concrete entity types E_1 and E_2 in U^S, S_j^S includes E_1 and E_2, but S_j^S does not include R.

There is an *association cardinality incompatibility* between U^S and S_j^S wrt R if R is an association between the concrete entity types E_1 and E_2 in U^S, S_j^S includes E_1 and E_2, but the cardinalities of one of its participants are incompatible. An incompatibility arises when the minimum cardinality in U^S is zero and one in S_j^S, or when the maximum cardinality is unconstrained in U^S and one in S_j^S.

For example, consider the association *SaleableItem – Category*. If U^S requires that an item may have several categories, then there will be an *association cardinality incompatibility* with Amazon webstore, because it only allows one.

There is an *induced association excess* between U^S and S_j^S wrt R if R is an association between the concrete entity types E_1 and E_2 in S_j^S, and there is an entity type excess between U^S and S_j^S wrt E_1 or E_2. In this case, the excess is induced by the entity type excess.

There is an *association excess* between U^S and S_j^S wrt R if R is an association of the concrete entity types E_1 and E_2 in S_j^S, U^S includes E_1 and E_2, but U^S does not require R.

3 Determining the Conceptual Fit for COTS Selection

The general problem of determining the conceptual fit can be defined as follows:

Given:

- The user requirements U of a system in some domain and
- A set $S_1,\ldots,S_n$ of n candidate COTS systems in that domain.

Determine:

- The conceptual misfits (deficits, misfits and excesses as defined in the previous section) between U and each of the $S_1,\ldots,S_n$.

Conceptual fit analysis can be performed considering the complete set of user requirements U and of the candidate systems $S_1,\ldots,S_n$, or considering only a fragment of them. The latter possibility is likely to be of much more practical interest in most cases.

The set of conceptual misfits found can be used as a basis for selection. If there are no misfits between U and S_j, then there is a perfect fit between them.

If there are one or more deficits or incompatibilities between U and S_j, then the selection of S_j would require either the change of the user requirements U (changing their intended way-of-working) or, if possible, a customization of S_j for the user (customizing existing systems to accommodate users' requirements) [14].

If there are one or more excesses between U and S_j, then the selection of S_j would imply both a potential value and cost. The value is the set of additional features that are not needed now, but that may be of interest in the future. The cost is the need of dealing now with the unneeded features related to those excesses, and the need of the corresponding resources.

If all misfits had the same cost, measured by the cost of changing requirements, the cost of customization or the cost of the unneeded features then, ignoring the potential value of excesses, the preferred system according to the conceptual fit criterion would be the one with a minimum number of such conceptual misfits. In practice, however, it is likely that users find some misfits costlier than others and therefore some weighting and judgment will be required.

4 A Method for Determining the Conceptual Fit

A straightforward approach to the solution of the general problem of determining the conceptual fit would be to consider each S_j (j = 1,…,n) separately, and determine the conceptual misfits between U and S_j as indicated in Section 2. This may be the only available solution in some contexts, but it is very costly. It requires knowing the n conceptual schemas and evaluating U wrt each of those schemas. When the number n is large and/or the conceptual schemas are large, the evaluation effort may be large too.

However, in a context where the selection process must be performed several times with the same set of candidate systems $S_1,…,S_n$, with different user requirements U, then a better solution would be to build an intermediate superschema S. That superschema S should integrate the schemas of $S_1,…,S_n$ in a way such that U and each of the $S_1,…,S_n$ could be formally defined in terms of S. When this is possible, we will show that then the conceptual misfits of U and each of the $S_1,…,S_n$ could be computed automatically. Note that S and the definition of $S_1,…,S_n$ in terms of S must be done only once per domain and that they are reused in all selection processes in that domain.

A similar idea was proposed in the Domain-based COTS product selection method (DBCS) [15] where a "domain model" is the common reference for the system to be developed and the existing COTS systems.

In the context of schema translation, a similar idea was proposed in MIDST [16] where there is a supermodel, such that each model is a specialization of the supermodel and a schema in any model is also a schema in the supermodel.

Based on the above idea, the method we propose consists of four parts:

- A superschema S that is a union of all schemas $S_1,…,S_n$ in a given domain.
- The definition of the schemas $S_1,…,S_n$ in terms of S.
- The definition of conceptual user requirements U in terms of S.
- The (automatic) computation of the misfits between U and $S_1,…,S_n$.

We describe these parts in the following.

4.1 The Superschema

In our method, the superschema S is an instance of the metamodel shown in Figure 1 for a domain D such that S includes the schemas of all existing COTS systems $S_1,…,S_n$ in D.

By inclusion of schemas we mean that S comprises all concrete entity types, attributes and associations that are totally or partially implemented in $S_1,…,S_n$. On the other hand, the cardinalities of the attributes and associations in S must not be incompatible with those that are implemented in $S_1,…,S_n$.

Note that the superschema we propose is similar to the "reference models" used in professional organizations as "an abstract framework for understanding significant relationships among the entities of some environment, and for the development of consistent standards or specifications supporting that environment." One of the most prominent examples of a reference model is the HL7 RIM

4.2 Defining Conceptual Schemas of COTS Systems in Terms of the Superschema

For the purposes of conceptual fit analysis we need to know for each S_j (j = 1,…,n):

- The entity types of S implemented in S_j and their corresponding cardinalities. We are interested only in the entity types that are concrete in S_j. If S_j implements all subtypes of an abstract entity type E in S, then S_j also implements E.

- The attributes and associations of S implemented in S_j and their corresponding cardinalities.

Figure 2 shows the extension of the metamodel defined in Figure 1 needed to represent the part of S that is implemented by S_j. A COTS system is assumed to implement a set of concrete entity types (with a cardinality of type *EntityTypeCardinality*, which may be Singleton or Unconstrained), a set of attributes and a set of associations.

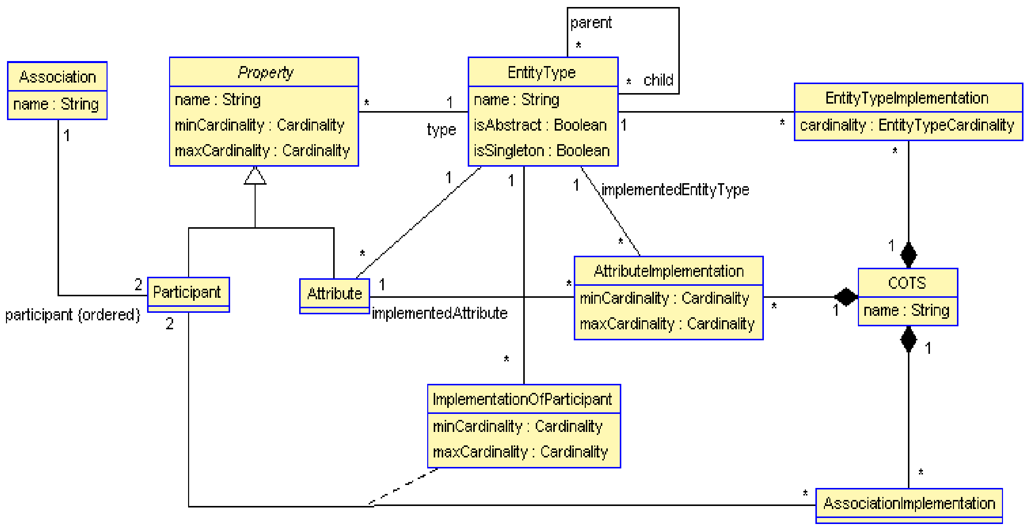

Figure 2. Extension of the metamodel of Figure 1 with COTS implementation of a superschema

Note that if S includes an abstract entity type E with subtypes $E_1,\ldots, E_m$ and E has an attribute A, then a system S_j that implements two or more of those subtypes could implement A differently in each case. Our metamodel of Figure 2 takes this possibility into consideration by indicating in *AttributeImplementation* the implemented entity type. A similar reasoning applies to the association participants.

The definition of a COTS implementation can be superschema-driven or system-driven. In the former, the elements of S are taken in some convenient order, and for each of them it is checked whether or not it is implemented by the system. If the element is a concrete entity type that is not implemented by S_j then there is no need to check the implementation of its attributes and associations. Note that in order to use this process the conceptual schema of S_j needs not to be explicit; what is needed to be known is what entity types, attributes and associations of S are implemented in S_j.

In the system-driven process, the elements of the conceptual schema of S_j are taken in some convenient order, and each of them is mapped to S. To use this process the conceptual schema of S_j must be explicit.

4.3 Defining Conceptual User Requirements

For the purposes of conceptual fit analysis of U we need to know:

- The entity types of S required by U and their corresponding cardinalities. We need to know only the entity types that are concrete in U. If U requires all subtypes of an abstract entity type E in S, then U also requires E.
- The attributes and associations of S required by U and their corresponding cardinalities.

Figure 3 shows the extension of the metamodel defined in Figure 1 needed to represent the user requirements in terms of S. It is nice to see that the extension has the same structure as that of Figure 2. An instance of *UserRequirements* consists of a set of concrete entity types (with a cardinality that may be Singleton or Unconstrained), a set of attributes and a set of associations.

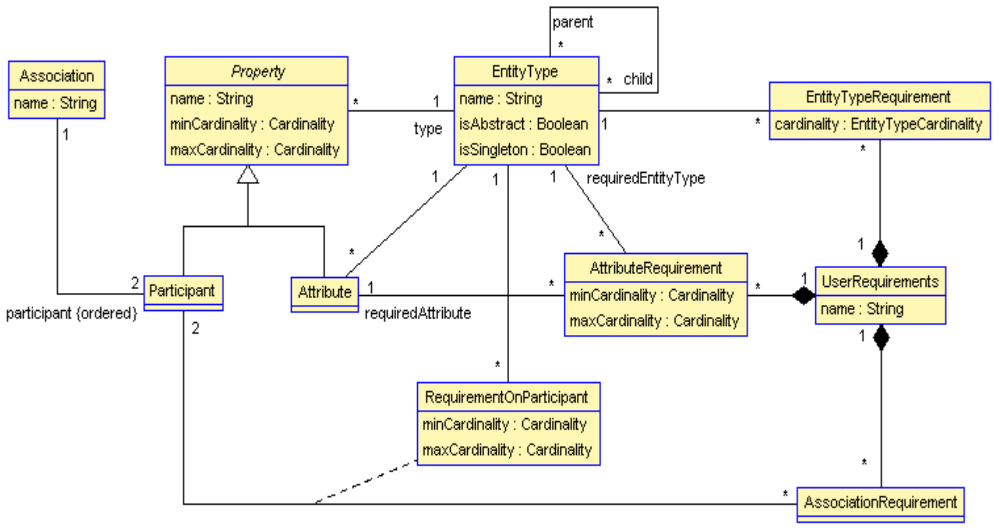

Figure 3. Extension of the metamodel of Figure 1 with user requirements

Note that similarly to the previous case, if S includes an abstract entity type E with subtypes $E_1, \ldots, E_m$ and E has an attribute A, then if U requires two or more of those subtypes, it could require A differently in each case. The same applies to association participants.

As in the definition of the implementation in COTS systems, the definition of user requirements can be superschema-driven or requirements-driven. In the former, the elements of the superschema are taken in some convenient order, and whether or not it is required by the users is checked for each of them. If the element is a concrete entity type that is not required then there is no need to check the requirement of its attributes and associations. Note that in order to use this process the conceptual schema of the user requirements needs not to be explicit; what is needed to be known is what entity types, attributes and associations of S are required.

In the system-driven process, the elements of the conceptual schema of U are taken in some convenient order, and each of them is mapped to S. To use this process the conceptual schema of U must be explicit.

4.4 Computing Misfits

In our method, once we have defined the instance of M (Figure 1) corresponding to the superschema S for a domain D, the instances of the candidate COTS systems $S_1, \ldots, S_n$ in D and their definitions in terms of S (Figure 2), and the instance of the user requirements U and its mapping to S (Figure 3) we can then automatically compute the misfits between U and $S_1, \ldots, S_n$. In what follows we explain the details of the computation in terms of the UML schemas shown in Figure 2 and Figure 3 and we give the formal definition of the misfits in OCL.

Entity type deficit. Let E be an entity type required by U. There is a deficit of E in S_j if E is not implemented in S_j. E can be implemented in S_j directly or by exclusion. There is a direct implementation when E is also an entity type of S_j.

There is an implementation by exclusion when there is an entity type E' implemented by S_j such that E' is a supertype of E, $E_1, \ldots, E_p$ ($p > 0$) and $E_1, \ldots, E_p$ are not required by U. The exclusion of $E_1, \ldots, E_p$ by U implies that the population of E and E' will always be the same, and therefore E' can implement E in S_j. For example, assume that S includes the entity type *Vehicle* with subtypes *Motorcycle* and *Car*, that U requires only *Motorcycle*, and that S_j implements *Vehicle*, but none of its subtypes. In this case, *Motorcycle* can be implemented by *Vehicle* in S_j.

The formalization in OCL is:

```
context EntityTypeRequirement::isDeficit(c:COTS):Boolean
body isImplementedBy(c).isUndefined
```

where *isImplementedBy*(*c*) is defined in the same context by:

```
isImplementedBy(c:COTS):EntityTypeImplementation
body
if directImplementation(c)->notEmpty then
       directImplementation(c)->any(true)
else
       if implementationByExclusion(c)->notEmpty then
        implementationByExclusion(c)->any(true)
       else oclUndefined(EntityTypeImplementation)
       endif
endif
```

and such that *directImplementation* and *implementationByExclusion* are:

```
directImplementation(c:COTS):Set(EntityTypeImplementation)
body c.entityTypeImplementation ->
       select(ei|ei.implementedEntityType = self.requiredEntityType)

implementationByExclusion(c:COTS):Set(EntityTypeImplementation)
body self.requiredEntityType.parent.entityTypeImplementation->
select(ei|ei.cOTS = c and  ei.implementedEntityType.child->
forAll(e|e.entityTypeRequirement->
select(er|er.userRequirements = self.userRequirements)->isEmpty))->
asSet()
```

Entity type incompatibility. Let *E* be an unconstrained entity type required by *U*. There is an incompatibility when *E* is implemented by a singleton entity type in S_j. The OCL formalization is:

```
context EntityTypeRequirement:: isIncompatible(c:COTS):Boolean
body cardinality = EntityTypeCardinality ::Unconstrained and
isImplementedBy(c).cardinality = EntityTypeCardinality::Singleton
```

Entity type excess. Let *E* be an entity type in S_j. There is a misfit of this kind when *E* does not implement any entity type in *U*. In OCL:

```
context EntityTypeImplementation::isExcess(u:UserRequirements):Boolean
body not u.entityTypeRequirement ->
       exists(er|er.isImplementedBy(self.cOTS) = self)
```

Induced attribute deficit. This happens when *U* requires an attribute of entity type *E* and there is an entity type deficit between *U* and S_j wrt *E*. In OCL:

```
context AttributeRequirement::isInducedDeficit(c:COTS):Boolean
body requiredEntityType.entityTypeRequirement->
       exists(er|er.userRequirements = self.userRequirements and
              er.isDeficit(c))
```

Attribute deficit. This happens when *U* requires an attribute *A* of an entity type *E* that is implemented in S_j, but that implementation does not include *A*. In OCL:

```
context AttributeRequirement::isDeficit(c:COTS):Boolean
body requiredEntityType.entityTypeRequirement->
       exists(er|er.userRequirements = self.userRequirements and
              er.isImplementedBy(c).isDefined)
and self.isImplementedBy(c).isUndefined
```

where *isImplementedBy*(*c*) is defined in the same context by:

```
isImplementedBy(c:COTS):AttributeImplementation
body
let ai:Set(AttributeImplementation) =
       c.attributeImplementation->
       select(ai|ai.implementedEntityType = self.requiredEntityType)
in
if ai -> notEmpty then ai->any(true)
else oclUndefined(AttributeImplementation)
```

```
endif
```

Attribute cardinality incompatibility. This happens when the cardinalities of an attribute required by *U* are incompatible with those of its implementation in S_j.

```
context AttributeRequirement:: isIncompatible(c:COTS):Boolean
body (minCardinality = Cardinality ::isZero and
isImplementedBy(c).minCardinality = Cardinality::isOne) or
(maxCardinality = Cardinality ::Unconstrained and
isImplementedBy(c).maxCardinality = Cardinality::isOne)
```

Induced attribute excess. Let *A* be an attribute of a concrete entity type *E* in S_j. There is a misfit of this kind when *E* is an entity type excess for *U*. In OCL:

```
context AttributeImplementation::isInducedExcess(u:UserRequirements):Boolean
body implementedEntityType.entityTypeImplementation->
exists(ei|ei.cOTS = self.cOTS and ei.isExcess(u))
```

Attribute excess. Let *A* be an attribute of a concrete entity type *E* in S_j. There is a misfit of this kind when *E* is an implementation of an entity type required by *U* but *A* is not implemented.

```
context AttributeImplementation::isExcess(u:UserRequirements):Boolean
body implementedEntityType.entityTypeRequirement->
       exists(er|er.userRequirements = u and
              er.isImplementedBy(self.cOTS).isDefined)
and not
u.attributeRequirement->exists(ar|ar.isImplementedBy(self.cOTS) = self)
```

Induced association deficit. There is misfit of this kind when *U* requires an association *R* between the concrete entity types E_1 and E_2 and there is an entity type deficit between *U* and S_j wrt E_1 or E_2.

Association deficit. There is misfit of this kind when *U* requires an association *R* between the concrete entity types E_1 and E_2 that are implemented in S_j, but S_j does not include *R*.

Association cardinality incompatibility. This happens when the cardinalities of an association required by *U* are incompatible with those of the implemented association in S_j.

Induced association excess. Let *R* be an association between the concrete entity types E_1 and E_2 in S_j. There is a misfit of this kind when E_1 and E_2 are an entity type excess for *U*.

Association excess. Let *R* be an association between the concrete entity types E_1 and E_2 in S_j. There is a misfit of this kind when E_1 and E_2 are implementations of entity types in *U* but *R* is not.

5 Integration into Existing COTS Selection Methods

In principle, conceptual fit analysis could be integrated into (almost) all existing COTS selection methods. In what follows, we indicate how this could be done in three methods that explicitly consider something similar to our conceptual fit: OTSO [3], CAP [17], and GOThIC [6].

OTSO. This method classifies the selection evaluation criteria into four main areas: functional requirements, product quality characteristics, strategic concerns, and domain and architecture compatibility. Domain compatibility refers to how well the candidate system and its features map into domain terminology and concepts. OTSO provides a template for the definition of criteria although no specific criteria are suggested. Each criterion must indicate how to measure the degree to which a system satisfies the criterion, and a baseline, which is the minimum value that must be achieved.

Conceptual fit analysis could provide two concrete evaluation criteria to OTSO, which may be called Conceptual deficits and incompatibilities and Conceptual excesses. Both correspond to the OTSO evaluation criteria area of domain compatibility. The first would be defined by the list of the conceptual deficits and incompatibilities that are not allowed by the user requirements. The baseline for this criterion would then be the absence of all misfits in the list. The Conceptual excesses criterion would be implicitly defined by the list of all possible conceptual excesses, as defined in Section 2. The baseline for this criterion would also be the absence of excesses.

CAP. The core part of the measurement and decision-making procedure of CAP is its evaluation taxonomy, which comprises a set of more than 100 pre-defined quality metrics. The definition of each quality metric indicates how to measure the degree, to which a system satisfies the metric, and the scale of the measure. The taxonomy is organized in a four-level tree. The first level is identical to the four areas of OTSO. The second level includes Domain compatibility, but there are no specific refinements on this level of the taxonomy.

As in the case of OTSO, conceptual fit analysis could provide the same two concrete evaluation criteria to CAP: Conceptual deficits and incompatibilities and Conceptual excesses. The measure could be the number of (possibly weighted) misfits.

GOThIC. The basis of this method is a large domain model which includes all relevant attributes for the selection of COTS in that domain. In particular, the domain model assumes the existence of a structural conceptual schema (UML class diagram), which is the equivalent to our superschema. From this schema the method derives, for each class or association appearing in it, a quality attribute with an ordinal metric which can take three values: Satisfactory, Acceptable and Poor. For a given class or association and system, the value corresponds to the degree with which the system satisfies the user requirements with respect to that class or association.

Conceptual fit could be easily integrated into GOThIC. The quality attributes derived from classes and associations would remain the same, but the values of their ordinal metric would now be Absence and Presence of misfits (deficit and/or incompatibilities), instead of the rather subjective current values. These values would be automatically computed from the definition of the superschema implementation (Figure 2) and of the user requirements (Figure 3). Moreover, the evaluation of the user requirements wrt those quality attributes for each system would now be objective.

6 Conclusions

We have proposed a new criterion for COTS systems selection, which we call conceptual fit. The criterion assesses the fit between the conceptual structure of a given system and that of the user requirements. We have identified three kinds of misfits in the schema elements, called deficits, incompatibilities and excesses. The idea is that the degree of conceptual fit is inversely proportional to the number of misfits, the maximum being the absence of them.

We have formally defined the general problem of evaluating the conceptual fit between the user requirements and a set of COTS systems in some domain, and we have proposed a new method for its solution. The method consists in defining a superschema, the definition of the conceptual schemas of the candidate systems and of the user requirements to that superschema, and the automatic computation of the conceptual misfits. We have formalized the method in UML and OCL.

The main effort required by our method is the development of the superschema and the definition of the candidate systems in terms of it. However, this must be done only once per domain (such as online shops) and the result could be reused in all COTS selections of a domain. This fact opens the possibility for professional organizations, consulting companies, and so on to make that effort and make the results available to all interested information systems developers.

In principle, the conceptual fit criterion could be taken into account by almost all existing selection methods. We have shown its integration into three existing selection methods. Although it has not been shown in the paper, we conjecture that conceptual fit could be taken into account in the early stages of product selection, because it enables an early discrimination between candidate products. In particular, it is likely to be useful in methods such as PORE [18] that propose an iterative selection approach.

The work reported here can be extended in several directions. We mention two of them here. The first is to take into account more conceptual constructs than those considered in the metamodel of Figure 1, such as data types or enumerations, or behavioural constructs [14]. Second, the method should be tested in real-world projects of COTS selection in order to experimentally confirm its cost effectiveness in practice. Ideally, the projects could be developed in one of the domains for which there is already a superschema, such as the reference model HL7 RIM in the health care domain.

Acknowledgments

This work has been partly supported by the Ministerio de Economía y Competitividad and FEDER under project TIN2008-00444, Grupo Consolidado, and by the Secretaria d'Universitats i Recerca de la Generalitat de Catalunya.

References

[1] L. Brownsword, P. A. Oberndorf, and C. A. Sledge, "Developing New Processes for COTS-Based Systems," IEEE Software, vol. 17(4), pp. 48-55, 2000. Available: http://dx.doi.org/10.1109/52.854068

[2] M. Feblowitz and S. J. Greenspan, "Scenario-Based Analysis of COTS Acquisition Impacts," Requirements Eng., vol. 3(3/4), pp. 182-201, 1998. Available: http://dx.doi.org/10.1007/s007660050004

[3] J. Kontio, "OTSO: A Systematic Process for Reusable Software Component Selection," University of Maryland Technical Reports. College Park, University of Maryland. CS-TR-3478, UMIACS-TR-95-63, 1995.

[4] R. Land, L. Blankers, M. R. V. Chaudron, and I. Crnkovic, "COTS Selection Best Practices in Literature and in Industry," in Hong Mei, Ed., ICSR 2008. LNCS 5030, pp. 100-111, Springer, Heidelberg, 2008. Available: http://dx.doi.org/10.1007/978-3-540-68073-4_9

[5] F. Tarawneh, F. Baharom, J.Hj. Yahaya, and F. Ahmad, "Evaluation and Selection COTS Software Process: The State of the Art," International Journal on New Computer Architectures and Their Applications, vol. 1(2), pp. 344-357, 2011.

[6] C. P. Ayala and X. Franch, "Domain Analysis for Supporting Commercial Off-the-Shelf Components Selection," in D.W. Embley, A. Olivé, and S. Ram, Eds., ER 2006, LNCS 4215, pp. 354–370, 2006. Available: http://dx.doi.org/10.1007/11901181_27

[7] N. A. M. Maiden, C. Ncube, and A. Moore, "Lessons Learned During Requirements Acquisition for COTS Systems," Comm. ACM December vol. 40, no. 12, pp. 21-25, 1997. Available: http://dx.doi.org/10.1145/265563.265567

[8] Y. Wand, "Ontology as a foundation for meta-modelling and method engineering," Information & Software Technology, vol. 38(4), pp. 281-287, 1996. Available: http://dx.doi.org/10.1016/0950-5849(95)01052-1

[9] A. Etien, C. Rolland, "Measuring the fitness relationship," Reqs Eng, vol. 10(3), pp. 184-197, 2005. Available: http://dx.doi.org/10.1007/s00766-005-0003-8

[10] A. Blandford, T. R. G. Green, D. Fursniss, and S. Makri, "Evaluating system utility and conceptual fit using CASSM," Int. Journal of Human–Computer Studies. vol. 66. pp. 393-409, 2008. Available: http://dx.doi.org/10.1016/j.ijhcs.2007.11.005

[11] A. Olive, “Conceptual Modeling of Information Systems,” Springer, Berlin 2007. Available: http://dx.doi.org/10.1007/978-3-540-39390-0

[12] A. Tort, “Esquema conceptual de l’osCommerce,” Master thesis. [Online]. Available: http://upcommons.upc.edu/ pfc/handle/2099.1/5301?locale=en, 2007

[13] A. Ramirez, “Esquema conceptual de Magento, un sistema de comerç electronic,” Master thesis. [Online]. Available: http://hdl.handle.net/2099.1/12294, 2011

[14] I. Reinhartz-Berger, A. Sturm, and Y. Wand, “Comparing functionality of software systems: An ontological approach,” Data Knowl. Eng., vol. 87, pp. 320-338, 2013. Available: http://dx.doi.org/10.1016/j.datak.2012.09.005

[15] K. R. P. H. Leung, and H. K. N. Leung, “On the efficiency of domain-based COTS product selection method,” Information & Software Technology, vol. 44(12), pp. 703-715, 2002. Available: http://dx.doi.org/10.1016/S0950-5849(02)00118-0

[16] P. Atzeni, P. Cappellari, R. Torlone, P. A. Bernstein, and G. Gianforme, “Model-independent schema translation,” VLDB J., vol. 17(6), pp. 1347-1370, 2008.

[17] M. Ochs, D. Pfahl, G. Chrobok-Diening, and B. Nothhelfer-Kolb, “A COTS Acquisition Process: Definition and Application Experience,” 11th ESCOM Conference, pp. 335-343, 2000.

[18] N. A. M. Maiden and C. Ncube, “Acquiring COTS Software Selection Requirements,” IEEE Software, vol. 15(2), pp. 46-56, 1998. Available: http://dx.doi.org/10.1109/52.663784