Conceptual Fit: A Criterion for COTS Selection

Antoni Olivé

Department of Service and Information System Engineering
Universitat Politècnica de Catalunya – Barcelona Tech
antoni.olive@upc.edu

Abstract. COTS systems selection consists in evaluating the user requirements with respect to characteristics of candidate systems, using a set of criteria. One criterion that has received little attention is what we call conceptual fit. The criterion assesses the fit between the conceptual structure of the user requirements and that of a system. 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 in defining a superschema, the mapping of the conceptual schemas of the candidate systems and of the user requirements to that superschema, and the automatic computation of the existing conceptual misfits. The method has been formalized in UML/OCL. We have conducted an exploratory experiment with the aim of evaluating the feasibility, difficulty and usefulness of the method, with positive results. We believe that the conceptual fit criterion could be taken into account by almost all existing COTS selection methods.

Keywords. COTS selection, Conceptual Fit, Conceptual Modeling

1. Introduction

Nowadays, many organizations build many of their information systems by customizing and/or integrating Commercial-off-the-shelf (COTS) systems [4]. 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 possibly large number of candidates, the large number of technical and non-technical characteristics that must be taken into account, and the possibly high impact of the decision on the future activities of the organization [6].

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 [7], and it is still an active research area. See [8], [15] 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 [7]. 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 OTSO, which refers to how well a system and its features map into the terminology and concepts of the domain [7]. 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 [2].

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.

Our notion of conceptual misfit has been inspired in the ontological expressiveness analysis [17], in the fitness relationship between a business and the system which supports it [5], and in CASSM, an analytical usability evaluation method of interactive systems that focuses on conceptual fit [3].

We propose conceptual fit as a criterion to be used for COTS selection. It can be taken into account in almost all existing selection methods. As an exploratory experiment, we have evaluated the conceptual fit of a potential new online shop with a set ecommerce platforms consisting of three content-management systems for ecommerce websites (osCommerce, Magento, CS-Cart) and Amazon webstore. Based on this experiment, we conjecture that conceptual fit analysis enables an early discrimination of candidate systems, which reduces the effort of the selection [10].

The structure of the paper is as follows. Next section formally identifies the different kind 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 describes an exploratory experiment in which we apply the proposed method in the above mentioned context of ecommerce platforms. 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. This can be easily extended, if desired [12].

Figure 1 shows the UML metamodel M of the schemas that we consider in this paper. Entity types have a name, may be abstract or concrete, may be a singleton or be

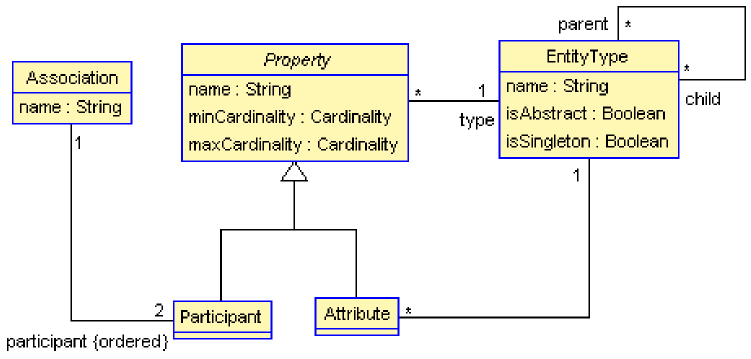

Fig.1. The metamodel of the schemas considered in this paper

unconstrained, and may have sub/supertype associations between them. 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.

Assume now that we have two instances of M that we call U_i (for user requirements) and S_j (for COTS system). We are interested in knowing how well U_i and S_j 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, called deficits, incompatibilities and excesses, which we define in the following. Of course, in a more complex metamodel, additional misfits could be identified. The idea is that the degree of fit of U_i and S_j is inversely proportional to the number of misfits, the maximum being the absence of them.

2.1 Entity Type Misfits

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

For example, if U_i includes the concrete entity type *Bundle* then there is an entity type deficit between U_i 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_i and S_j wrt E if E is a concrete entity type of U_i and an entity type of S_j, but E is unconstrained (not a singleton) in U_i and a singleton in S_j. Both U_i and S_j have the entity type E but, in U_i, E may have several instances while only one instance is allowed in S_j.

For example, if U_i includes the unconstrained concrete entity type *Store*, then there is an entity type cardinality incompatibility between U_i and osCommerce wrt to *Store* because *Store* is a singleton in osCommerce.

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

2.2 Attribute Misfits

There is an *induced attribute deficit* between U_i and S_j wrt A if A is an attribute of the concrete entity type E in U_i and there is an entity type deficit between U_i and S_j wrt E. In this case, the deficit is induced by the entity type deficit. For example, if U_i 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_i and S_j wrt A if A is an attribute of the concrete entity type E in U_i, S_j includes E, but S_j does not include A.

There is an *attribute cardinality incompatibility* between U_i and S_j wrt A if A is an attribute of the concrete entity type E in U_i, S_j includes A, but the cardinalities are incompatible. An incompatibility arises when the minimum cardinality in U_i is zero and one in S_j, or when the maximum cardinality is unconstrained in U_i and one in S_j. 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_i and S_j wrt A if A is an attribute of the concrete entity type E in S_j and there is an entity type excess between U_i and S_j wrt E. In this case, the excess is induced by the entity type excess.

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

2.3 Association Misfits

There is an *induced association deficit* between U_i and S_j wrt R if R is an association between the concrete entity types E_1 and E_2 in U_i, and there is an entity type deficit between U_i and S_j 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_i and S_j wrt R if R is an association between the concrete entity types E_1 and E_2 in U_i, S_j includes E_1 and E_2, but S_j does not include R.

There is an *association cardinality incompatibility* between U_i and S_j wrt R if R is an association between the concrete entity types E_1 and E_2 in U_i, S_j 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_i is zero and one in S_j, or when the maximum cardinality is unconstrained in U_i and one in S_j. For example, consider the association *SaleableItem – Category*. If U_i 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_i and S_j wrt R if R is an association between the concrete entity types E_1 and E_2 in S_j, and there is an entity type excess between U_i and S_j 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_i and S_j wrt R if R is an association of the concrete entity types E_1 and E_2 in S_j, U_i includes E_1 and E_2, but U_i does not require R.

3. Evaluating the Conceptual Fit Criterion for COTS Selection

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

Given:

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

Determine:

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

Conceptual fit analysis can be performed considering the complete set of user requirements U_i and of the candidate systems $S_1,\dots,S_n$, 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_i and S_j, then there is a perfect fit between them.

If there are one or more deficits or incompatibilities between U_i and S_j, then the selection of S_j would require either the change of the user requirements U_i (changing their intended way-of-working) or 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_i and S_j, then the selection of S_j would imply dealing 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 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 must 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 ($j = 1,\dots,n$) separately, and determine the conceptual misfits between U_i and S_j as indicated in Sect. 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_i 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 $S_1,\dots,S_n$, with different user requirements U_i, then a better solution would be to build an intermediate superschema S. That superschema S should integrate $S_1,\dots,S_n$ in a way such that U_i and each of the $S_1,\dots,S_n$ could be mapped to S, and such that the conceptual misfits of U_i and each of the $S_1,\dots,S_n$ could then be computed automatically.

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."[1] One of the most prominent examples of reference model is the HL7 RIM[2].

A similar idea was proposed in the "Domain-based COTS product selection method" (DBCS) [9] 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 [1] 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:

1. A superschema S that is a union of all schemas $S_1,\dots,S_n$ and all possible user requirements $U_1,\dots,U_m$ in a given domain.
2. The definition of the schemas $S_1,\dots,S_n$ in terms of S.
3. The definition of user requirements U_i in terms of S.
4. The (automatic) computation of the misfits between U_i and $S_1,\dots,S_n$.

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 Fig. 1 for a domain D such that:

- S includes the schemas of all possible COTS systems $S_1,\dots,S_n$ in D.
- S includes all possible conceptual user requirements $U_1,\dots,U_m$ in D.

By inclusion of schemas here we mean that:

- S comprises all concrete entity types, attributes and associations that may be required by $U_1,\dots,U_m$. On the other hand, the cardinalities of the attributes and associations in S must not be incompatible with those that may be required by $U_1,\dots,U_m$.
- S comprises all concrete entity types, attributes and associations that are implemented in $S_1,\dots,S_n$. On the other hand, the cardinalities of the attributes and associations in S must not be incompatible with those that are implemented in $S_1,\dots,S_n$.

[1] OASIS SOA Reference Model (SOA-RM) TC (https://www.oasis-open.org/committees/soa-rm/faq.php)
[2] HL7 Reference information model. (http://www.hl7.org/implement/standards/rim.cfm)

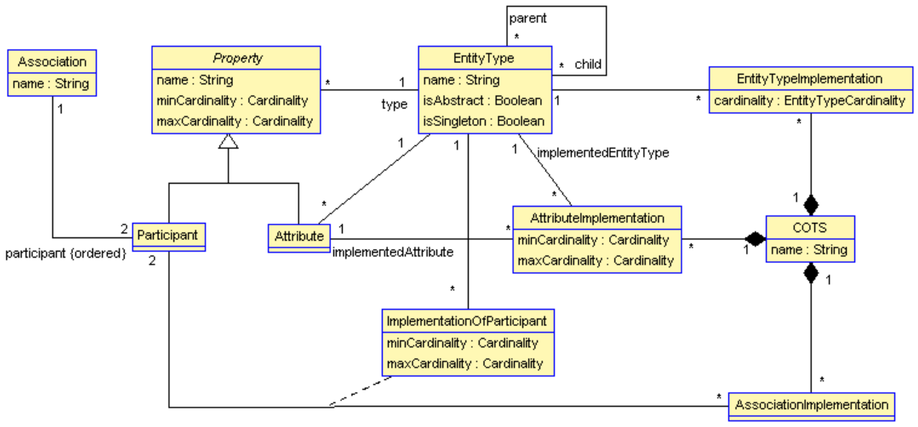

Fig.2. Extension of the metamodel of Fig. 1 with COTS implementation of a superschema

4.2 Mapping Conceptual Schemas of COTS Systems to the Superschema

For the purposes of conceptual fit analysis we need to know for each S_j ($j = 1,\dots,n$) in D:

- 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 Fig. 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 that may be Singleton or Unconstrained), a set of attributes and a set of associations.

Note that if S includes an abstract entity type E with subtypes $E_1,\dots, E_m$ 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 Fig. 2 takes this possibility into consideration by indicating in *AttributeImplementation* the implemented entity type. A similar reasoning applies to the association participants.

The mapping process 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. To use this process, the conceptual schema of S_j needs not to be explicit; what is needed to know 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.

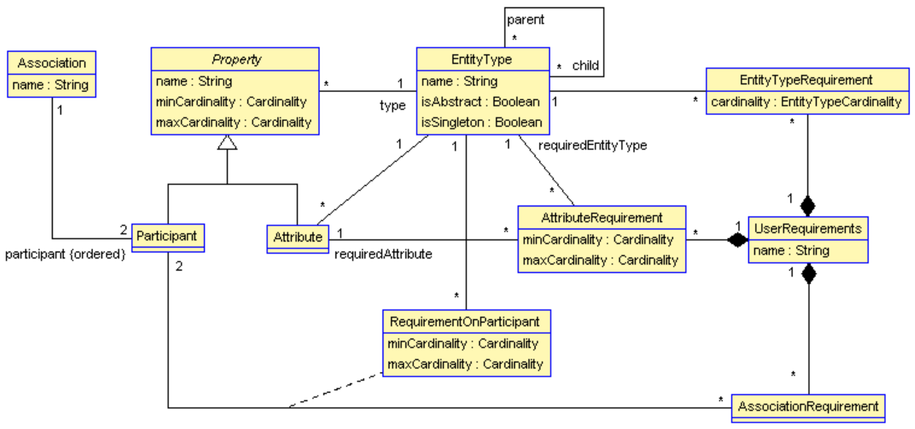

Fig.3. Extension of the metamodel of Fig. 1 with user requirements

4.3 Defining Conceptual User Requirements

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

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

Figure 3 shows the extension of the metamodel defined in Fig. 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 Fig. 2. User requirements are assumed to consist of concrete entity types (with a cardinality that may be Singleton or Unconstrained), a set of attributes and a set of associations.

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

As in the mapping of systems, the definition of user requirements can be superschema-driven or requirements-driven.

4.4 Computing Misfits

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

Entity type deficit. Let *E* be an entity type required by U_i. 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*, $E_1,\dots, E_p$ ($p > 0$) and $E_1,\dots, E_p$ are not required by U_i. The exclusion of $E_1,\dots, E_p$ by U_i implies that the population of *E* and *E*' will always be the same, and therefore *E*' can implement *E* in S_j. In OCL:

```
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_i. 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_i. 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_i requires an attribute of entity type E and there is an entity type deficit between U_i 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_i 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_i 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_i. 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_i 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[3]. There is misfit of this kind when U_i requires an association R between the concrete entity types E_1 and E_2 and there is an entity type deficit between U_i and S_j wrt E_1 or E_2.

Association deficit. There is misfit of this kind when U_i 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_i 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_i.

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_i but R is not.

5. Application to the Selection of an eCommerce Platform

In what follows, we describe an exploratory experiment we performed to evaluate the feasibility, difficulty and usefulness of the application of the conceptual fit criterion in COTS selection. We assumed the requirements of a potential online shop and considered four existing ecommerce platforms: osCommerce[4], Magento[5], CS-Cart[6], and Amazon webstore[7].

5.1 The Superschema

Online shop platforms have large conceptual schemas, and their complete integration into one (even larger) superschema would not be easy. However, for the purpose of COTS selection, in our method such complete integration is not necessary. It suffices to only consider those concepts (entity types, attributes and associations) that enable an effective discrimination between systems [10]. This means that the superschema should include the concepts that are important or critical for the users and that are not implemented by all candidate systems.
In the experiment, we had available the complete schemas of osCommerce [16] and

[3] Due to space constraints the OCL formalization of the association misfits is not shown, although it is similar to the one of the attribute misfits.

[4] www.oscommerce.com

[5] magento.com

[6] www.cs-cart.com

[7] services.amazon.com

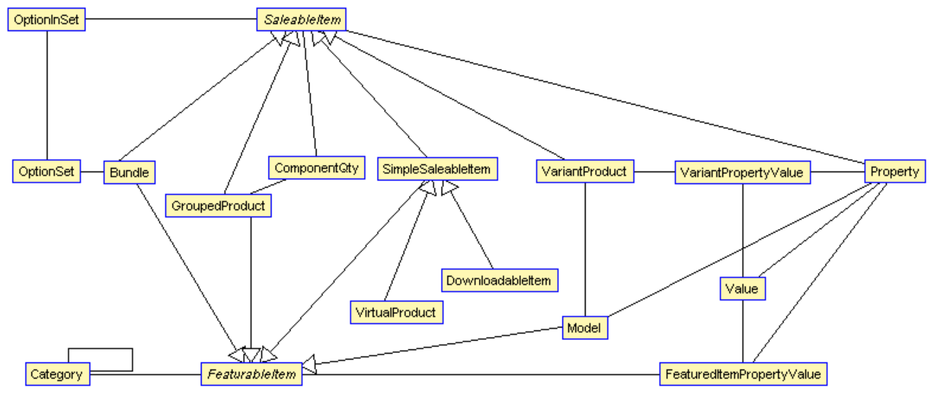

Fig.4. A fragment of the superschema of the online shops domain

Magento [13] and we studied in depth the relevant parts of the other two. Based on this, we developed a superchema consisting of 35 entity types, 82 attributes and 40 binary associations. Figure 4 shows a fragment of that superschema. The fragment shows the kinds of products that are sold in on-line shops. Those kinds are *SimpleSaleableItem* (with subtypes *VirtualProduct* and *DownloadableItem*), *VariantProduct* (a materialization of a *Model*), *GroupedProduct* (a set of products) and *Bundle* (a product with options).

5.2 Mapping Conceptual Schemas of COTS Systems to the Superschema

The mapping of a candidate system to the superschema is easy for those who know (or, better, have developed) that system and are familiar with its domain. In the experiment, due to our knowledge of the superschema and the candidate systems, we used a combination of the superschema and system-driven processes.

5.3 Defining Conceptual User Requirements

In the experiment, we assumed that the user requirements were those of an arbitrary existing commercial online shop. We studied and experimented (read-only) the relevant parts of that shop. We then used a superschema-driven process to define its conceptual requirements, as indicated in Section 4.3. In total, the requirements of that shop consist of a subset of the superschema consisting of 22 entity types, 29 attributes and 18 associations.

5.4 Computing Misfits

Table 1 summarizes the number of misfits per each type found in each system, computed as indicated in Sect. 4.4. In this experiment, no entity type deficits have

Table 1. Misfits found per type in each system

	osCommerce	Cs-cart	Magento	Amazon
entity type deficit	0	0	0	0
entity type card. incompatibility	0	0	0	1
entity type excess	2	6	12	3
induced attribute deficit	0	0	0	0
attribute deficit	9	0	0	5
attribute card. incompatibility	0	0	0	0
induced attribute excess	1	12	21	4
attribute excess	8	28	33	4
induced association deficit	0	0	0	0
association deficit	2	0	0	0
association card. incompatibility	0	0	0	2
induced association excess	1	8	14	5
association excess	4	5	5	3

been found, and therefore there are not induced deficits. The analysis has detected a number of deficits and incompatibilities whose importance should be assessed in the selection process. Some of them may be critical. The high number of excesses of Cs-cart and Magento may indicate that they are “excessive” for the user’s needs.

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 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.

In principle, the conceptual fit criterion could be taken into account by almost all existing selection methods. In particular, it is likely to be useful in methods such as PORE [11] that propose an iterative selection approach. Conceptual fit could be taken into account in the early stages of product selection, because it enables an early discrimination between candidate products.

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 mapping 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. We have applied the method in an exploratory experiment of COTS selection in the domain of online shops to evaluate its feasibility, difficulty and usefulness, with positive results.

The main effort required by our method is the development of the superschema and the mapping of the candidate systems to 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.

The work reported here can be extended in several directions. We mention three of them here. The first is to take into account more conceptual constructs than those considered in the metamodel of Fig.1, such as association classes, data types or enumerations, or behavioural constructs [14]. Second, the method should be tested in a real-world project of COTS selection in order to experimentally confirm its cost effectiveness in practice. Ideally, the project 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. Third, it could be useful to develop recommendations for the integration of the conceptual fit criterion into existing selection methods.

References

1. Atzeni, P., Cappellari, P., Torlone, R., Bernstein, P.A., Gianforme, G.: Model-independent schema translation. VLDB J. 17(6): 1347-1370 (2008)
2. Ayala, C.P., Franch, X.: 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)
3. Blandford, A., Green, T. R. G., Fursniss, D., Makri, S.: Evaluating system utility and conceptual fit using CASSM. *Int. Journal of Human–Computer Studies.* 66. 393-409 (2008)
4. Brownsword, L., Oberndorf, P.A., Sledge, C.A.: Developing New Processes for COTS-Based Systems. IEEE Software 17(4): 48-55 (2000)
5. Etien, A., Rolland, C.: Measuring the fitness relationship. Reqs Eng. 10(3): 184-197 (2005)
6. Feblowitz, M., Greenspan, S.J.: Scenario-Based Analysis of COTS Acquisition Impacts. Requirements Eng. 3(3/4): 182-201 (1998)
7. Kontio, J.: 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)
8. Land, R., Blankers, L., Chaudron, M.R.V., Crnkovic, I: COTS Selection Best Practices in Literature and in Industry. In: Hong Mei (Ed.) ICSR 2008. LNCS 5030, pp. 100-111. Springer, Heidelberg (2008)
9. Leung, K.R.P.H., Leung, H.K.N.: On the efficiency of domain-based COTS product selection method. Information & Software Technology 44(12): 703-715 (2002)
10. Maiden, N.A.M., Ncube, C., Moore, A.: Lessons Learned During Requirements Acquisition for COTS Systems. Comm. ACM December Vol. 40, No. 12, pp. 21-25, (1997)
11. Maiden, N.A.M. Ncube, C.: Acquiring COTS Software Selection Requirements. IEEE Software 15(2): 46-56 (1998)
12. Olive, A.: Conceptual Modeling of Information Systems. Springer, Berlin (2007)
13. Ramirez, A.: Esquema conceptual de Magento, un sistema de comerç electrònic. Master thesis. http://hdl.handle.net/2099.1/12294 (2011)
14. Reinhartz-Berger, I., Sturm, A., Wand, Y.: Comparing functionality of software systems: An ontological approach. Data Knowl. Eng. 87: 320-338 (2013)
15. Tarawneh, F., Baharom, F., Jamaiah Hj. Yahaya; J.Hj., Ahmad, F.: Evaluation and Selection COTS Software Process: The State of the Art. International Journal on New Computer Architectures and Their Applications 1(2): 344-357 (2011)
16. Tort, A.: Esquema conceptual de l'osCommerce. Master thesis. http://upcommons.upc.edu/pfc/handle/2099.1/5301?locale=en (2007)
17. Wand, Y.: Ontology as a foundation for meta-modelling and method engineering. Information & Software Technology 38(4): 281-287 (1996)