

# IRatePL2C: Importance Rating-Based Approach for Product Lines Collaborative Configuration

Sihem Ben Sassi<sup>a</sup>

RIADI Lab., National School of Computer Science, Manouba University, Tunisia

**Keywords:** Product Lines, Collaborative Configuration, Conflict Resolution, Feature, Importance Scoring.


**Abstract:** The core of any proposed approach in the context of collaborative configuration of product lines focuses on how conflictual situations are resolved. Few works consider stakeholders preferences in their resolution strategy while allowing a free order configuration process. However, to generate a valid solution satisfying all constraints, they generally rely on a solution of exponential complexity. In this work, we propose the IRatePL2C approach, which resolution strategy relies on importance degrees assigned by the stakeholders to their initial configuration choices. IRatePL2C starts by merging stakeholders' configurations and then detecting and resolving the conflicts according to their type: explicit or implicit in sequential steps. Finally, domain constraints are propagated and the process is reiterated to reach a final valid configuration. An illustrative example is presented to evaluate the approach. The complexity of IRatePL2C is polynomial which an important advantage compared with previous works.

## 1 INTRODUCTION

Collaborative configuration of product lines refers to a coordinated process in which a set of stakeholders share the configuration activities based on their areas of expertise in order to decide about features that should be included in the final product (Edded et al., 2019). It has appealed the interest of several researchers, such as (Czarnecki et al., 2005), (Mendonca et al., 2007), (Pereira, 2017), (Edded et al., 2020), (Le et al., 2022) and (Ben Sassi et al., 2023). The mainspring of the proposed approaches is to manage conflictual situations that may arise when involved stakeholders in the configuration have contradictory choices regarding the final product. The prevailing means to represent product lines is feature model (Raatikainen et al., 2019), in which features are related with mandatory, optional, alternative (XOR) and OR relationships; and domain constraints are expressed through inclusion (require) and exclusion (exclude) relationships (Benavides et al., 2010). Product configuration consists in selecting a set of features that meet individual requirements toward the desired product. When the configuration is made by a single stakeholder, it is easy to ensure its conformance to the product line model with its domain con-

straints, thanks to constraints propagation (Czarnecki and Kim, 2005). However, when several stakeholders are involved in this process, their requirements may be contradictory or do not comply with the product model; such situations of inconsistency are referred to as conflicts (Osman et al., 2009). In this context, stakeholders requirements are better captured by allowing them to choose the undesired features ( $\neg F_j$ ) that should not be present in the final product as well as the desired ones ( $F_i$ ) (Stein et al., 2014), (Le et al., 2022), (Ben Sassi et al., 2023). According to (Edded et al., 2020), a conflict may be (i) explicit when the same feature is explicitly desired by a stakeholder and undesired by another; (ii) implicit when configuration choices of the stakeholders violate the product line feature model constraints. Two cases are distinguishable: (ii-1) two alternative features (related with XOR) are desired by different stakeholders; and (ii-2) domain constraint propagation makes the configuration not valid because it requires to include/exclude a feature that is already un/desired by another stakeholder.

Obtaining a valid configuration from the different stakeholders' configuration choices requires to resolve any detected conflict by eliminating one or more configuration choices involved in the conflict. Existing approaches tackle this issue by (1) supporting either workflow-based process imposing a predefined

<sup>a</sup>  <https://orcid.org/0000-0002-1925-4989>

configuration order, or a flexible one allowing a free order configuration; and (2) implementing a conflict resolution method adopted from social science like negotiation and predefined priority, or more technical one such as range fixes. The reader may refer to (Edded et al., 2019) for a comprehensive view characterizing collaborative configuration approaches. Actually, few works propose a flexible configuration process that takes into account stakeholders preferences in conflict resolution, namely (Stein et al., 2014), (Ochoa et al., 2015), (Le et al., 2022) and (Ben Sassi et al., 2023). However, either they do not generate a solution that takes into account the preferences of all stakeholders, and/or they implement an expensive computation process.

The remainder of this paper is structured as follows: Section 2 is dedicated to the proposed approach. Section 3 illustrates IRatePL2C with a full example. Section 4 discusses the solution. Section 5 concludes the paper.

## 2 THE IRatePL2C APPROACH

The proposed IRatePL2C approach is based on the following principles and hypotheses: **H1:** each stakeholder freely configures the product line; there is no predefined order between stakeholders during the configuration process. **H2:** each stakeholder makes a configuration decision by specifying the un/desired features to be absent/present in the product. **H3:** a prior agreement between stakeholders to express to which extent they desire a configuration choice i.e. a (un)desired feature (not) to be included in the final configuration. **H4:** each stakeholder assigns an importance degree (*ideg*) to each of his/her explicit configuration choices. **H5:** a five scale-based score is used to express the configuration choice *ideg*, 5 means that the choice is very important to be considered in the final configuration; 4: fairly important; 3: important; 2: slightly important; and 1: not at all important. **H6:** *idegs* are only assigned to the explicit initial choices made by the stakeholder. In other words, features that should be included in the configuration because of the constraints propagation to ensure its validity do not receive scores. **H7:** each stakeholder is aware that potential conflicting choices will be resolved based on the *idegs*, and assigns consciously scores to the configuration choices he/she makes.

Figure 1 shows the main steps of IRatePL2C approach: it starts by (1) merging the initial configuration choices made by each stakeholder. Second (2), the merged configuration is analyzed to identify explicit conflicts; these latter are resolved and the con-

figuration is accordingly updated. Third (3), the new configuration is analyzed to identify and resolve implicit conflicts resulted from XOR constraints; the configuration is updated accordingly. Fourth (4), the product line domain constraints are checked to propagate them on the configuration when applicable. The configuration is afterward checked. If it is valid, it will be returned as final configuration. When the configuration is not valid, it is question to compare it with the one of the previous iteration in order to decide to reiterate the process starting from Step -2- in the case the two configurations are different, or hand over to the product line manager in order to apply the rule he/she advocates to resolve remaining conflicts. This rule may be for example one of the substitution rules proposed in (Edded et al., 2020) and (Ben Sassi et al., 2023), that resolves conflicts, among others, by favoring "the most complete product", or by prioritizing "the explicit choices made by a given stakeholder".

Algorithm 1 summarizes the core process of IRatePL2C. It requires the feature model of the product line to configure as a tree (*PL\_FM*), the set of constraints related to that model including XOR, include and require ones (*PL\_ConstraintsList*), as well as the list of stakeholders involved in the collaborative configuration (*Stk\_List*). As result, it ensures a valid configuration as a list of features deduced from stakeholders' configuration choices (*FinalConfig*). Statements (*St.*) 6-8 allow to collect stakeholders' configurations. IRatePL2C's step -1- is reflected by *St.* 9 which invokes *MergeConfigs()* procedure. The *while* loop encompasses the three next steps implemented thanks to the *ResolveExplicitConflicts()* procedure (*St.* 13) for Step -2-, *ResolveXORConflicts()* procedure (*St.* 16) for Step -3-, and the *PropagateConstraints()* function (*St.* 19) for Step -4-. The two procedures have as output a list of configuration choices to remove from the current configuration *CurrentConfig* in order to resolve the related identified conflicts (*St.* 14 and 17). The function determines the features to add to the current configuration (*St.* 20) and updates the configuration choices *idegs* accordingly (*St.* 21). The *while* loop exits either with a valid configuration, in such case it represents the *FinalConfig* to return, or with a configuration that still has some remaining conflicts where *idegs* cannot help in their resolution; in such case, the product manager takes a decision and final configuration is returned to stakeholders.

### 2.1 Configurations Merging

As mentioned earlier, each stakeholder makes a set of explicit configuration choices expressing the features he/she desires to be included or not in the final con-

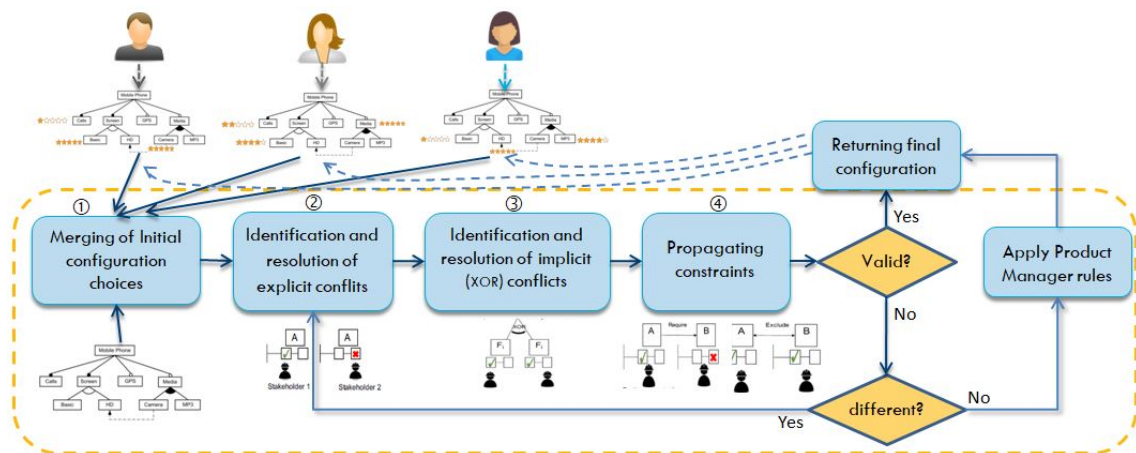


Figure 1: The proposed approach process.

figuration; each explicit choice is accompanied by an *ideg* ranging from 1 to 5. Only explicit choices made by the stakeholder are subject to *ideg* assignment. Algorithm 2 describes the process of configuration merging, which consists in obtaining, from the stakeholders configuration choices  $LE\_ConfigStk_i$ , two lists: (1) the set of all explicit configuration choices including the desired features and non desired features without redundancy  $List\_EFs$  (St. 7), and (2) the set of all explicit configuration choices, where each one of them is accompanied with an ordered list of its related *idegs* expressed by stakeholders who made that choice  $List\_EFsImportance$  (St. 8 and 10).

## 2.2 Explicit Conflicts Resolution

This step consists in detecting explicit conflicts, occurring when the same feature is, at the same time, desired by a stakeholder and undesired by another one, in the merged configuration choices; then resolving them using *idegs* comparison of involved configuration choices. If the configuration choice is present only one time, the same *ideg* expressed by that stakeholder is reported. In case that several stakeholders desired the same state regarding a given feature, the corresponding *ideg* is based on the ordered importance given by the stakeholders related to that feature. The rationale is to keep what each stakeholder believes essential to include in the configuration. We resolve conflicts by keeping the configuration choice with the highest *ideg*, and eliminating from the configuration the other one involved in the conflict.

Algorithm 3 describes the explicit conflicts resolution steps. It requires as input a configuration in an intermediate state  $CurrentConfig$  as well as  $List\_EFsImportance$  list. It returns two lists; the first one is the list of configuration choices to delete from

the configuration  $CurrentConfig$  to resolve the identified explicit conflicts ( $toRemoveList$ ); the second one contains any eventual non resolved explicit conflict ( $RemainedExplicitConflictsList$ ). St. 10 compares the *idegs* of a configuration choice  $f$  with those of its negation  $\neg f$  extracted from  $List\_EFsImportance$  in St. 8 and 9. The comparison consists in walking through the two lists degrees; since they are ordered, the first who has an element of greater value than the corresponding one in the other list, or who has a greater number of elements in case of equality is the winner. The loser is added to  $toRemoveList$  (St. 11-15). In case of equality in values and lists length, the conflict is added to  $RemainedConflictsList$  (St. 17). This process is applied to each configuration choice in the form of  $\neg f$  in the  $CurrentConfig$  in order to detect all conflicts (St. 4-5). The current configuration is afterward updated by removing  $toRemoveList$  from  $CurrentConfig$  (St. 14 in Algorithm 1).

## 2.3 Implicit Conflicts Resolution

with XOR) are desired by different stakeholders. The resolution principle of the XOR implicit conflicts, is similar to the used one for the explicit conflicts, but applied to features involved in violated XOR constraints. As illustrated in St. 4-7 of Algorithm 4, it is first question to identify any two features participating in a XOR constraint; then, extract the *idegs* associated to each one (St. 9-10) to determine which one to remove (St. 11-16), or add the conflict to  $RemainedXORConflictsList$  if necessary. The configuration choices in the returned  $toRemoveList$  are removed from the configuration  $CurrentConfig$  and the non resolved conflicts are added to  $RemainedConflictsList$  (St. 17 and 18 in Algorithm 1).

Algorithm 1: Core Process of IRatePL2C Approach.

---

**Data:** *PL\_FM*: the tree of features representing the product line to configure  
*PL\_ConstraintsList*: the list of constraints related to the product line  
*Stk\_List*: the list of stakeholders involved in the configuration  
**Result:** *FinalConfig*: a valid configuration as a list of features deduced from stakeholders' choices

```

1 begin
2   valid  $\leftarrow$  False ;
3   different  $\leftarrow$  True ;
4   toRemoveList  $\leftarrow$   $\emptyset$  ;
5   RemainedConflictsList  $\leftarrow$   $\emptyset$  ;
6   foreach (Stki  $\in$  Stk_List) do
7      $\lfloor$  LE_ConfigStki  $\leftarrow$  Configure(PL_FM) ;
8   MergeConfigs( $\{LE\_ConfigStk_i\}_{i=1..Nb\_Stks} \rightarrow List\_EFs, List\_EFsImportance$ ) ;
9   CurrentConfig  $\leftarrow$  List_EFs ;
10  while (not valid and different) do
11    PreviousConfig  $\leftarrow$  CurrentConfig;
12    ResolveExplicitConflicts(CurrentConfig, List_EFsImportance  $\rightarrow$ 
13      toRemoveList, RemainedExplicitConflictsList);
14    CurrentConfig  $\leftarrow$  RemoveFromList(CurrentConfig, toRemoveList);
15    RemainedConflictsList  $\leftarrow$  RemainedConflictsList  $\cup$  RemainedExplicitConflictsList ;
16    ResolveXORConflicts(CurrentConfig, List_EFsImportance, PL_ConstraintsList  $\rightarrow$ 
17      toRemoveList, RemainedXORConflictsList);
18    CurrentConfig  $\leftarrow$  RemoveFromList(CurrentConfig, toRemoveList);
19    RemainedConflictsList  $\leftarrow$  RemainedConflictsList  $\cup$  RemainedXORConflictsList ;
20    toAddList  $\leftarrow$  PropagateConstraints(CurrentConfig, PL_ConstraintsList);
21    CurrentConfig  $\leftarrow$  AddToList(CurrentConfig, toAddList);
22    List_EFsImportance  $\leftarrow$  UpdateFImportance(List_EFsImportance, toAddList);
23    different  $\leftarrow$  CompareConfigLists(PreviousConfig, CurrentConfig);
24    valid  $\leftarrow$  CheckValidity(CurrentConfig);
25  if valid then
26    FinalConfig  $\leftarrow$  CurrentConfig;
27  else /* not valid and not different */
28    RemainedConflictsList  $\leftarrow$  CheckRemainedConflicts(CurrentConfig, RemainedConflictsList) ;
29    FinalConfig  $\leftarrow$  ApplyProductManagerRules(CurrentConfig, RemainedConflictsList);
30  return FinalConfig;

```

---

## 2.4 Constraints Propagation

Resolving implicit conflicts resulting from *require* resp. *exclude* constraints violation amounts to resolving explicit conflicts as they imply the inclusion of the second part of the constraint ( $f_i$  resp.  $\neg f_j$ ) to the configuration, which may lead to an explicit conflict in case that  $\neg f_i$  resp.  $f_j$  already part of the configuration. Constraints propagation is already dealt with in the literature (Czarnecki and Kim, 2005), (Benavides et al., 2010); it is simply done by walking through domain (*require* and *exclude*) constraints and updating the configuration by adding a feature or its negation accordingly. For the purpose of this work, we need to consider the *ideg* of any (un)desired feature added to the configuration in order to keep be-

ing able to resolve conflicts based on *idegs*. As the source of any addition is a configuration choice belonging to the configuration and causing the *require* or *exclude* constraint to be triggered, we assign to the added (un)desired feature the highest *ideg* associated with the source configuration choice. *St.* 19 in Algorithm 1 returns the list of (un)desired features to add to the current configuration (*St.* 20). This same list is also used to update the *idegs* associated with the configuration choices already made by stakeholders and/or add non existing (un)desired features with their related *idegs* (*St.* 21).

On the other hand, any newly added feature to the configuration may violate a XOR constraint; this explains the iterative processing for the conflicts resolution in IRatePL2C approach. Furthermore, it is ob-



Algorithm 2: MergeConfigs: Stakeholders initial Configurations Merging.

---

**Data:**  $LE\_ConfigStk_i$   $i = 1..Nb\_Stks$ : list of initial configuration choices made by each stakeholder  
**Result:**  $List\_EFs$ : a list of features representing the merged stakeholders configuration choices  
 $List\_EFsImportance$ : a list of the merged configuration choices along with their *idegs*

```

1 begin
2    $List\_EFs \leftarrow \emptyset$ ;
3    $List\_EFsImportance \leftarrow \emptyset$ ;
4   foreach  $LE\_ConfigStk_i$   $i = 1..Nb\_Stks$  do
5     foreach  $e \in LE\_ConfigStk_i$  do
6       if not  $(e.F \in List\_EFs)$  then
7          $List\_EFs \leftarrow List\_EFs \cup e.F$ ;
8          $List\_EFsImportance \leftarrow List\_EFsImportance \cup (e.F, e.I)$ ;
9       else
10         $List\_EFs \leftarrow insertOrderedImportance(List\_EFs, e.F, e.I)$ ;

```

---

Algorithm 3: ResolveExplicitConflicts: Determine explicit conflicts and which feature state to remove.

---

**Data:**  $CurrentConfig$ : a list of features reflecting an intermediate state of a configuration  
 $List\_EFsImportance$ : a list of the merged configuration choices with their related importance  
**Result:**  $toRemoveList$ : a list of features to remove from  $CurrentConfig$  to resolve detected conflicts  
 $RemainedExplicitConflictsList$ : a list of explicit conflicts that has not been resolved

```

1 begin
2    $toRemoveList \leftarrow \emptyset$ ;
3    $RemainedExplicitConflictsList \leftarrow \emptyset$ ;
4   foreach  $nf \in CurrentConfig$  do
5     if  $nf.charAt[0]='-'$  then
6        $f \leftarrow nf.subString(1)$ ;
7       if  $(f \in CurrentConfig)$  then
8          $f\_ImpList \leftarrow extract(List\_EFsImportance, f)$ ;
9          $nf\_ImpList \leftarrow extract(List\_EFsImportance, nf)$ ;
10         $res \leftarrow isMoreImportant(f\_ImpList, nf\_ImpList)$ ;
11        if  $(v=1)$  then
12           $toRemoveList \leftarrow toRemoveList \cup nf$ ;
13        else
14          if  $(v=2)$  then
15             $toRemoveList \leftarrow toRemoveList \cup f$ ;
16          else
17             $RemainedExplicitConflictsList \leftarrow RemainedExplicitConflictsList \cup (f, nf)$ ;

```

---

vious that the resolution of a given conflict may result in the resolution of another conflict. This also explains the rationale behind the stepped resolution of conflicts. In each step, conflicts of a certain type are detected and resolved before dealing with another type. This also explains why remained non resolved conflicts need to be checked before passing them to the product manager in *St.* 26-27 of Algorithm 1.

### 3 ILLUSTRATIVE EXAMPLE

We illustrate IRatePL2C through the example of the Web portal feature model shown in Figure 2, used in previous works (e.g. (Mendonca et al., 2008) and (Ben Sassi et al., 2023)). The model contains 3 XOR relationships, 5 domain constraints of inclusion requirement and 1 exclusion requirement. The configuration scenario is summarized in Table 1, where the *ideg* is noted as superscript of the related explicit con-

Algorithm 4: ResolveXORConflicts: Determine XOR conflicts and which feature to remove.

---

**Data:** *CurrentConfig*: a list of features reflecting an intermediate state of a configuration  
*List\_EFsImportance*: a list of the merged configuration choices with their related importance  
*PLConstraintsList*: the list of constraints related to the product line  
**Result:** *toRemoveList*: a list of features to remove from *CurrentConfig* to resolve detected conflicts  
*RemainedXORConflictsList*: a list of implicit XOR conflicts that has not been resolved

```

1 begin
2   toRemoveList ← ∅ ;
3   RemainedXORConflictsList ← ∅ ;
4   foreach Constrainti ∈ PLConstraintsList do
5     if (TypeOf(Constrainti)=XOR ) then
6       for j ← 1 to nbFeatures(Constrainti)-1 do
7         for m ← j + 1 to nbFeatures(Constrainti) do
8           if ((Fj ∈ CurrentConfig) and (Fm ∈ CurrentConfig)) then
9             fj_ImpList ← extract(List_EFsImportances, Fj);
10            fm_ImpList ← extract(List_EFsImportances, Fm);
11            v ← isMoreImportant(fj_ImpList, fm_ImpList);
12            if (v=1) then
13              toRemoveList ← toRemoveList ∪ Fm;
14            else
15              if (v=2) then
16                toRemoveList ← toRemoveList ∪ Fj;
17              else
18                RemainedXORConflictsList ← RemainedXORConflictsList ∪ (Fm, Fj);

```

---

figuration choice made by each of the five involved stakeholders. Throughout the example description, and for the sake of clarity and space, we mainly focus on features susceptible to cause conflicts.

Table 1: Collaborative configuration scenario.

Stakeholder	Explicit configuration choices
Stk1	KeyWordSupport <sup>(2)</sup> , DB <sup>(4)</sup> , ¬Active <sup>(3)</sup> , https <sup>(5)</sup>
Stk2	XML <sup>(4)</sup> , ¬Text <sup>(4)</sup> , ¬Active <sup>(5)</sup> , ms <sup>(3)</sup>
Stk3	Active <sup>(5)</sup> , Php <sup>(2)</sup> , XML <sup>(1)</sup> , DataTransfer <sup>(4)</sup>
Stk4	Text <sup>(2)</sup> , Dynamic <sup>(5)</sup> , KeyWordSupport <sup>(4)</sup> , DB <sup>(3)</sup> , ¬https <sup>(1)</sup> , ¬Sec <sup>(3)</sup>
Stk5	Text <sup>(4)</sup> , Database <sup>(5)</sup> , Active <sup>(4)</sup> , DataTransfer <sup>(3)</sup>

**Step-1: Merging Stakeholders' Configuration Choices.** As described in the previous section, merging stakeholders' configuration choices leads to two results: (*List\_EFs*) and (*List\_EFsImportance*) shown in Table 2. For example, both Stk1 and Stk4 desire *KeyWordSupport* with an *ideg* equals to 2 and 4 respectively. The list (4, 2) is therefore associated as ordered *idegs* to the configuration choice *KeyWordSupport*.

**Step-2: Resolving Explicit Conflicts.** Three explicit conflicts are identified, as mentioned in Table 3. In the case of the first conflict, *https* has one *ideg* which is 5, while *¬https* has 1 as sole *ideg*; this means that

Table 2: Merged configuration choices results.

Merged configuration choices			
KeyWordSupport, DB, https, XML, ms, Active, Php, ¬https, DataTransfer, Text, Dynamic, Database, ¬Text, ¬Active, ¬Sec			
Configuration choices with importance degrees			
Feature	degree	Feature	degree
https	5	¬https	1
Active	5, 4	¬Active	5, 3
Text	4, 2	¬Text	4
ms	3	¬Sec	3
XML	4, 1	PHP	2
DataTransfer	4, 3	Dynamic	5
KeyWordSupport	4, 2	Database	5
DB	4, 3		

*https* is retained and *¬https* has to be removed. As to *Active* and *¬Active*, both of them have 5 as first *ideg*; the following degree should therefore be considered. The second *ideg* is respectively 4 and 3, which means to keep *Active* in the configuration and remove *¬Active*. Regarding *Text* and *¬Text*, both have 4 as first *ideg*; however *¬Text* has only one *ideg* while *Text* has two ones. This means that *Text* is retained and *¬Text* should be removed. The configuration is updated accordingly as reported in Table 3.

**Step-3: Resolving Implicit XOR Conflicts.** The XOR constraint between *XML* and *Database* is not

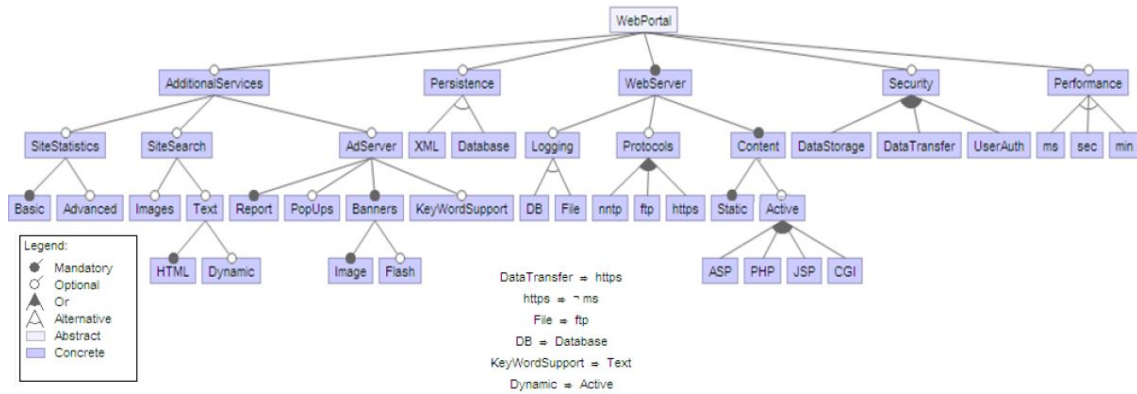


Figure 2: Web Portal Feature Model.

Table 3: Explicit conflicts resolution results.

Detected conflict	Retained choice	To Remove choice
(https, ~https)	https	~https
(Active, ~Active)	Active	~Active
(Text, ~Text)	Text	~Text
Updated configuration		
KeyWordSupport, DB, https, XML, ms, Active, Php, DataTransfer, Text, Dynamic, Database, ~Sec		

respected. Importance degrees determined in Step-1 and shown in Table 2 are used to decide which feature to remove. The *idegs* associated with *XML* are 4 and 1. Even though *Database* has only one *ideg*, it wins as its *ideg*, which is 5, is greater than 4. *XML* should therefore be eliminated from the configuration. The resulted configuration is therefore:

(KeyWordSupport, DB, https, ms, Php, Text, Dynamic, ~Sec, Database, Active, DataTransfer)

**Step-4: Propagating Constraints.** The full list of added (un)desired features to the configuration as consequence of constraint propagation are clarified in Table 4. The *idegs* associated with the configuration choices and the newly (un)desired added features is updated accordingly. The new resulted configuration is as follows: (KeyWordSupport, DB, https, ~ms, ms, Php, Text, Dynamic, ~Sec, Database, Active, DataTransfer)

Table 4: Constraints propagation results.

Constraint	(Un)desired feature	ideg
(KeyWordSupport => Text)	Text	4
(DB => Database)	Database	4
(https => ~ms)	~ms	5
(Dynamic => Active)	Active	5
(DataTransfer => https)	https	4

In the second iteration, (i) explicit conflicts resolution reveals that the conflict (*ms*, ~*ms*) is resolved by keeping ~*ms* since the latter has 5 as *ideg* vs. 3 for *ms*; (ii) all XOR constraints are respected; (iii) constraints propagation does not alter

the configuration obtained from the previous step; (iv) the validity is checked and verified. The process stops and returns the following final configuration. (KeyWordSupport, DB, https, ~ms, Php, Text, Dynamic, ~Sec, Database, Active, DataTransfer)

## 4 DISCUSSION

Satisfaction results for the scenario example (c.f. Table 5) that 4 configuration choices among 5 that were assigned 5 as *ideg* and 5 among 7 with 4 as *ideg* were retained in the final configuration. Besides, no configuration choice with 1 as *ideg* was retained, which makes sense. The weighted satisfaction shows that the final configuration globally satisfied 72% of the stakeholders configuration choices. When each stakeholder is taken separately, we find that *Stk5* is fully satisfied (100%), while *Stk2* is not satisfied at all (0%) even though he/she assigned the highest *idegs* to his/her configuration choices (5, 4 and 3). The scenario example was deliberately chosen to show that the retained configuration choices depend not only on what the stakeholder assigns, but also on what the other stakeholders chose and what *idegs* they assigned to their choices. After all, the IRatePL2C approach is based on the awareness of stakeholders about how conflicts are resolved.

Regarding the complexity, the proposed solution is polynomial. Each of its sub-processes relies on walking through lists and constructing new lists. It depends on the number of involved stakeholders, the number of features of the product model, the number of related constraints and the number of features per constraint. Previous works that may be compared to ours, i.e. focusing on collaborative configuration of product lines and allowing a free-order process and taking into account stakeholders preferences in the conflicts resolution, namely (Stein et al., 2014),

Table 5: Satisfaction results for the scenario example.

Importance degree	Stk1			Stk2			Stk3			Stk4			Stk5			Final config.		
	d	r	s	d	r	s	d	r	s	d	r	s	d	r	s	d	r	s
5	1	1	100%	1	0	0%	1	1	100%	1	1	100%	1	1	100%	5	4	80%
4	1	1	100%	2	0	0%	1	1	100%	1	1	100%	2	2	100%	7	5	71%
3	1	0	0%	1	0	0%	0	-	-	2	2	100%	1	1	100%	5	3	60%
2	1	1	100%	0	-	-	1	1	100%	1	1	100%	0	-	-	3	3	100%
1	0	-	-	0	-	-	1	0	0%	1	0	0%	0	-	-	2	0	0%
weighted satisfaction			78.5%			0%			91.6%			94.4%			100%			72%

d: # of made configuration choices      r: # of retained choices in the final config.      s: satisfaction rate (r/d)

(Ochoa et al., 2015), (Le et al., 2022) and (Ben Sassi et al., 2023) have exponential complexity since they build a set of combinations based on a given set of features (e.g. SAT solvers, MCS computing); such problems are known to be NP-complete.

## 5 CONCLUSION

In this paper, we presented a new approach to resolve conflicts in the context of collaborative configuration of product lines. Its process allows stakeholders to freely configure the product line model and takes into account their preferences to resolve conflicts expressed through an importance degree assigned to each explicit configuration choice. To reach its aim, IRatePL2C proceeds in steps; in each step, conflicts are detected, resolved and the intermediate configuration is updated accordingly before starting the following step. This allows to reduce the number of conflicts and reach easier to a valid solution. The approach does not prevent to have a completely dissatisfied stakeholder (i.e. all his/her configuration choices are not included in the final configuration), as the solution depends on the choices of all stakeholders regarding the features to include exclude and the importance degrees they assign. Further empirical investigation is needed to reveal some “tips” to avoid such situations. This is the subject of our future work.

## REFERENCES

- Ben Sassi, S., Edded, S., Mazo, R., Ben Ghezala, H., and Salinesi, C. (2023). Colla-config: A stakeholders preferences-based approach for product lines collaborative configuration. *Journal of Syst. and Soft.*, 197:111586.
- Benavides, D., Segura, S., and Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35:615–636.
- Czarnecki, K., Helsen, S., and Eisenecker, U. (2005). Staged configuration through specialization and multilevel configuration of feature models. In *Software Process: Improvement and Practice*.
- Czarnecki, K. and Kim, C. H. P. (2005). Cardinality-based feature modeling and constraints: A progress report. In *International Workshop on Software Factories*, pages 16–20. ACM San Diego, California, USA.
- Edded, S., BenSassi, S., Mazo, R., Salinesi, C., and BenGhezala, H. (2019). Collaborative configuration approaches in software product lines engineering: A systematic mapping study. *Journal of Syst. and Soft.*, 158:110422.
- Edded, S., BenSassi, S., Mazo, R., Salinesi, C., and BenGhezala, H. (2020). Preference-based conflict resolution for collaborative configuration of product lines. In *15th Int. Conf. on Evaluation of Novel Approaches to Software Engineering*, pages 297–304.
- Le, V.-M., Tran, T. N. T., and Felfernig, A. (2022). Consistency-based integration of multi-stakeholder recommender systems with feature model configuration. In *26th ACM Int. Systems and Software Product Line Conference*, page 178–182.
- Mendonca, M., Bartolomei, T., and Cowan, D. (2008). Decision-making coordination in collaborative product configuration. In *ACM symposium on applied computing*, pages 108–113.
- Mendonca, M., Cowan, D., and Oliveira, T. (2007). Process-centric approach for coordinating product configuration decisions. In *40th Hawaii Int. Conf. on System Sciences*, pages 1–10.
- Ochoa, L., González-Rojas, O., and Thüm, T. (2015). Using decision rules for solving conflicts in extended feature models. In *ACM SIGPLAN Int. Conf. on Software Language Engineering*, pages 149–160.
- Osman, A., Phon-Amnuaisuk, S., and Ho, C. K. (2009). *Investigating Inconsistency Detection as a Validation Operation in Software Product Line*, pages 159–168. Springer Berlin Heidelberg.
- Pereira, J. A. (2017). Runtime collaborative-based configuration of software product lines. In *39th Int. Conf. on Software Engineering Companion*, pages 94–96.
- Raatikainen, M., Tiuhonen, J., and Männistö, T. (2019). Software product lines and variability modeling: A tertiary study. *Journal of Syst. and Soft.*, 149:485–510.
- Stein, J., Nunes, I., and Cirilo, E. (2014). Preference-based feature model configuration with multiple stakeholders. In *18th Int. Software Product Line Conf.*, pages 132–141.