Reviewing Diagnosis Solutions for Valid Product Configurations in the Automated Analysis of Feature Models

Cristian L. Vidal-Silva1, José Rubio2, Erika Madariaga3, Franklin Johnson4, Luis Carter5 and Camilo Campos6
Ingeniería Civil Informática, Escuela de Ingeniería, Universidad Viña del Mar, Viña del Mar, Chile1
Área Académica de Informática y Telecomunicaciones, Universidad Tecnológica de Chile INACAP, Santiago, Chile2
Ingeniería Informática, Facultad de Ingeniería, Ciencia y Tecnología, Universidad Bernardo O’Higgins, Santiago, Chile3
Depto. Disciplinario de Computación e Informática, Facultad de Ingeniería, Universidad de Playa Ancha, Valparaíso, Chile4
Ingeniería Civil Industrial, Facultad de Ingeniería, Universidad Autónoma de Chile, Talca, Chile5,6

Abstract—A Feature Model (FM) is an information model to represent commonalities and variabilities for all the products of a Software Product Line (SPL). The complexity and large-scale of real feature models makes their manual analysis for determining the product configurations validity a tedious or even infeasible task. Efficient solutions for the diagnosis of errors in the Automated Analysis of Feature Models (AAFM) already exist such as FMDiag and FlexDiag. Thus, this work describes the fundamental basis for both diagnosis algorithms to apply the first of them on the validity of FM product configurations. The results highlight the applicability and efficiency of FMDiag and invite us to look for additional applications in the AAFM scenarios.

Keywords—AAFM; feature model; valid product; valid configuration; FMDiag; FlexDiag

I. INTRODUCTION

The main goal of Software Product Line Engineering (SPLE) is the reuse of assets (features) for the products definition looking for improving the quality, accelerating the production and time-to-market, and reducing the production costs of all the process [1]. Such as Bastos et al. [2] highlight, organizations emphasize the proactive reuse, interchangeable components, and multi-product planning cycles for the faster and cheaper construction of high-quality products.

The domain and application engineering processes constitute the SPL [3]. The first process looks for a development for reuse, that is, defining software products in terms of commonalities and variabilities, and also their constraints. The second process refers to development with reuse by the derivation of products from Feature Models (FMs) [4].

Different proposals of variability modeling techniques exist for the common and varying assets representation within a SPL, but FMs are one of the most widely used techniques for the SPL variability modeling [5]. Feature Model (FM) is a visual language for accomplishing the domain engineering in Software Product Line (SPL) [6], that is, FM is adequate for a SPL representation [7]. The application engineering process results in the set of valid features selection (products). Fig. 1 shows a FM example for the Debian Linux operating system adapted from [8].

The configuration of a FM refers to the process of choosing and unchoosing features in a feature model to reach a full configuration [9]. Paz [10] remarks that making decisions in a FM for deriving valid products usually is not a straightforward task mainly for the size and complexity of analyzed FM, and possible dependencies among features constraints in the selection process. Example of valid products for Fig. 1 FM are $P_1 = \{\text{Debian, texteditor, vi, bash, gui, kde}\}$ and $P_2 = \{\text{Debian, texteditor, vi, gedit, openoffice.org-gtk-gnome, openoffice.org-gnome 1:2:4.0-3 Ubuntu 6, bash, gui, gnome, games, gnuchess}\}$. The Automated Analysis of Feature Models (AAFM) is about computer-assisted operations for obtaining information from FMs. Those operations are mainly carried out in a two–step process such as Fig. 2 depicts [11].

The “valid product” or “valid configuration” is an AAFM operation that receives a FM and a product to return a value to know if the product is or not valid, that is, if the product belongs to the set of products that the FM represents [11].

For the dynamism and growing size of SPLs, defining and applying efficient solutions for an AAFM operation to determine “valid product” instances represents a relevant work. Large-scale variability models encoding hundreds or thousands of features nowadays exist which pushed beyond the manual computing capabilities and demand for efficient AAFM operations [12]. For example, more than 28.000 packages in a Debian distribution exist for their optionally installation. Thus, to validate new Debian derivative configurations such as Ubuntu or Linux mint results a tedious and error-prone manual process.

FMDiag [13] and FlexDiag [14] are two efficient diagnosis solution on FMs, that is, to determine minimal sets of constraints in the configuration knowledge base (FM) to delete or adapt for making the remaining constraints a consistent set. This article look to answer: How can we apply existing diagnosis solutions for the “valid product” AAFM operation? This paper describes the theory of FMDiag an FlexDiag for applying them into the “valid product” AAFM operation to potentially get efficient solutions. For the FMDiag and FlexDiag algorithmic simplicity and their previously known efficiency on diagnosis, their application represent a significant advance for the SPL community.
We use the FAMA tool [15] and its framework [16]. FAMA includes implementations of the FMDiag and FlexDiag for different reasoner tools. Specifically, we use the FAMA working on the Choco reasoner to adapt and apply FMDiag and FlexDiag for the “valid product” analysis to evaluate and validate their performance and results.

In the next, this paper structures as follows: Section II describes related works for the “valid product” AAFM operation; Section III describes main FMs ideas and discusses the main structure and functioning of the diagnosis algorithms FMDiag and FlexDiag; Section IV presents main concepts of the Valid product AAFM operation; Section V shows and highlights the results of applying FMDiag for the “valid product” AAFM operation, and Section VI concludes and gives feature work ideas.

II. RELATED WORK

AAFM constitutes an active research and application area. Next, we mentioned only a few of related work:

- Galindo et al. [17] explore the use of Big Data technique to look for more efficient AAFM solutions. Even though, improved operation exist for the valid configuration, FMDiag and FlexDiag solutions continue being more appropriate mainly for their efficiency and simple algorithm pattern.
- White et al. [18] uses a CSP solver for debugging basic feature model configurations and automating the evolving product configurations on basic feature models.
- The work of Ross-Frantz et al. [19] uses the Orthogonal Variability Model (OVM) instances for their mapping on a CSP for the developing of FaMa-OVM to identify void models, dead and false optional features.
- Hu et al. [20] proposes an approach for evolving basic feature model and analyze their products evolution. They present refactoring strategies and semi-automated support for the commonality extraction and feature refactoring.
- Mauro et al. [21] present a framework for the modeling and evolving reconfiguration of context-aware SPL instances. They consider the environmental impact on updated features and new contextual information on the SPL evolution. That research work defines a meta-model for Hybrid Feature Model (HyFM) for the attributes in features and represent contextual information. According to their findings, being possible of proposing configuration on FMs of up 10,000 features in less than a minute is more than enough for the majority of the daily use cases.

III. FEATURE MODELS & AAFM DIAGNOSIS

A Feature Model (FM) is a information modeling tool useful for the representation all the products of a SPL along with their features and relations [11]. Different kind of FMs already exist such as cardinality-based FMs, extended-FMs which support the attributes definition, and basic FMs. For their simplicity and usability, in this work we use basic FMs.

In a FM we distinguish among the following relationships and features:

- Unary relations. These are between a father and a child feature, and we can distinguish between mandatory (black circle in top of the child feature) and optional (white circle in top of the child feature) child features. In Fig. 1, (Debian, texteditor) and (Debian, games) are examples of mandatory and optional relations.
• Set relations. These relations define a father feature of a set of children features, and we can distinguish between optional and alternative sets, that is, for the optional set we can select more than one feature if their father is selected, and for the alternative set only one child feature from the options set of features. In both cases of set features, we need to select at least one feature. In Fig. 1, (texteditor, {vi, gedit, openoffice.org-gtk-gnome} and (games, {gnuches, glchess}) are examples of optional and alternative set relations of features, respectively.

Such as Benavides et al. [11] highlight, defining and optimizing AAFM operations represent a current and active research area. Diagnosis on a FM means to discover and inform mistakes on the analyzed FMs. FMDiag [13] and FastDiag [22] are diagnosis solution applicable on FMs. Such as Felfernig et al. [22] argue, FastDiag directly determines a minimal diagnosis for a given ranking of preferences.

Structurally, FMDiag defines a base function (algorithm 1) and a recursive function (Algorithm 2). The base function receives a set of customer requirements \( S \) to analyze its consistency, and a configuration knowledge base \( AC \) that includes \( S \), that is, \( AC \) without \( S \) should be consistent. FMDiag returns an empty diagnosis if \( S \) is not empty and \( (AC - S) \) is non-consistent. Otherwise, the base function calls the recursive function \( Diag \) for \( D = \emptyset, S \) and \( AC \).

The function \( Diag \) receives \( D \) that represents a subset of the main set to analyze previously removed from the base of knowledge \( AC \) (initially \( D \) is empty), \( S \) (the current set in analysis), and \( AC \) (the current base of knowledge that contains \( S \)). \( Diag \) presents two base cases:

• If \( D \) is not \( \emptyset \) and the current \( AC \) is consistent then \( Diag \) returns an empty diagnosis, that means the diagnosis was not in \( S \), that is, \( D \) contains the diagnosis. This situation can occur after the second recursive call (in the first call \( D \) is empty and \( AC \) is non-consistent).

• If the size of \( S \), that is, the number of constraints in \( S \) were either \( <= \) \( m \) for FlexDiag or \( = 1 \) for FMDiag then \( Diag \) returns the current set \( S \) as a diagnosis. That base case can occur if the previous described one is not true, that is, \( S \) contains a diagnosis.

If no base-case were valid, we know that \( S \) contains the diagnosis and the size of \( S \) is not minimal. Hence, the function \( Diag \) partitions \( S \) in \( S_1 \) (from the 1st until the constraint in the middle) and \( S_2 \) (from the middle + 1 to the last constraint) to go to the 1st recursive call for the arguments \( D = S_2, S = S_1, AC = AC - S_2 \), that is, to take off \( S_2 \) from \( AC \) to evaluate the consistency of \( AC \) (the second half of the current \( S \)). If \( AC \) were consistent, because current \( D \) is not empty, \( Diag \) returns \( \emptyset \) (the diagnosis is in \( D \)). If \( AC \) were non-consistent, the current set \( S \) contains inconsistencies and repeat the division process on the current \( S \), and new recursive calls proceed again. The \( Diag \) function receives \( D = S_2, S = S_1, AC = AC - S_2 \) from the 1st recursive call. The 1st recursive call returns \( \Delta_1 \). The 2nd recursive call receives \( D = \Delta_1, S = S_2, AC = AC - \Delta_1 \) (the 2nd recursive call needs the result of the 1st one to proceed). The 2nd recursive call returns \( \Delta_2 \). Thus, the function \( Diag \) returns \( \Delta_1 \cup \Delta_2 \) as a diagnosis.

Applying FMDiag for diagnosis on product configurations seems direct, that is, \( S \) contains of the constraints for the features selection of the product in analysis, and \( AC \) contains the set of constraints for the consistent FM plus \( S \).

**Algorithm 1** FMDiag(\( D, S = \{s_1, ... , s_r\}, AC \)): \( \Delta \)

```plaintext
if isEmpty(S) or inconsistent(AC-S) then
    return \( \emptyset \);
else
    return Diag(\( D, S, AC \));
end if
```

**Algorithm 2** Diag(\( D, S = \{s_1, ..., s_r\}, AC \)): \( \Delta \)

```plaintext
if \( D \neq \emptyset \) and consistent(\( AC \)) then
    return \( \emptyset \);
else if
    if FlexActive then
        if size(\( S \)) <= \( m \) then
            return \( S \);
        end if
    else
        if size(\( S \)) = 1 then
            return \( S \);
        end if
    end if
    \( k = \lceil \frac{r}{2} \rceil \);
    \( S_1 = \{s_1, ..., s_k\}; S_2 = \{s_k+1, ..., s_r\} \);
    \( \Delta_1 = Diag(S_2, S_1, AC - S_2) \);
    \( \Delta_2 = Diag(\Delta_1, S_2, AC - \Delta_1) \);
    return(\( \Delta_1 \cup \Delta_2 \));
```

Syntactically, the main difference between FMDiag and FlexDiag are the size of the minimal set to return, but semantically their differences are highly relevant. Such as the works of Felfernig et al. [14] [23] remark, FlexDiag is more adequate for anytime diagnosis scenarios, that is, diagnosis within time limits even though trade-offs between diagnosis quality and performance of the diagnostic search exist.

IV. VALID PRODUCT AAFM OPERATION

Products of an SPL are the combination and configuration of features through the assembly of corresponding and reusable artifacts [24]. Such as Benavides et al. [11] describe, “valid product” is an AAFM operation example that receives as input a FM and a product (a set of features), and returns a boolean result that determines either the product belongs to the set of products that the feature model represents or not. The products \( P_1 \) and \( P_2 \) are examples of valid products of the FM of Fig. 1, whereas \( P_3 = \{Debian, texteditor, openoffice.org-gtk-gnome, openoffice.org2-gnome 1:2.4.0-3 Ubuntu 6, bash, gui, kde\} \) and \( P_4 = \{Debian, bash, gui, gnome, games\} \) are example of non-valid products: \( P_3 \) does not respect the only defined cross-tree constraint in the model whereas \( P_4 \) does not consider mandatory features and features from a set of options. Fig. 3
and 4 show the features selection (green features) for valid products of $P_1$ and $P_2$, respectively. Fig. 5 and 6 show the features selection (green features) for non-valid products $P_3$ and $P_4$, respectively.

Applying FMDiag and FlexDiag for the validity of FM product would be direct: we need to define the constraints for the product to analyze, and the constraints for the FM definition to review in. We can repeat that process for analyzing multiple products.

## V. Application Results & Discussion

For space reasons we only present the FMDiag testing results on the diagnosis of “valid product”. FlexDiag follows the same functioning idea and by applying it we can get more efficient and lesser precise solutions. Tables 1 and 2 present the steps of applying FMDiag for the diagnosis of product $P_3$ and $P_4$ for the SPL of Fig. 1 to appreciate the algorithmic functioning for the diagnosis on the “valid product” AAFM operation. FMDiag can directly determine the validity of a product, and return involved constraints for a non-valid configuration.

Tables 3 and 4 show the diagnosis results for the “valid product” operations on small size FMs of the SPL [25] and large-scale FMs generated by the Betty tool [26], respectively. FMDiag reaches a fast performance for small FMs, and for large-scale feature models its performance seems also adequate, even though FMDiag diagnosis usually contains only one inconsistency cause. Felfernig et al. [13] detail how to obtain all the diagnosis of a FM applying FMDiag. FMDiag and FlexDiag are appropriate for interactive scenarios which demand for a fast-time for diagnosis even for the diagnosis on large-scale feature models.

The works of Felfernig et al. [13] and [14] compared the performance of FMDiag and FlexDiag to Constraint Satisfaction Problem (CSP) solutions to remark the FMDiag and FlexDiag efficiency advantages for preferred diagnosis, even though for getting all diagnosis the results are almost equivalent.

Such as Felfernig et al. [27] remark, variability management is essential for the product configurations of SPL looking for extensive customization to attend different clients’ needs. The AAFM solutions such as FMDiag [13] and FlexDiag [14] [23] efficiently achieve that goal. Undoubtedly, FMDiag and FlexDiag represent and advance concerning the HSDAG solution [28].

Main limitations of this study are the dependency of

<table>
<thead>
<tr>
<th>Step</th>
<th>D</th>
<th>S</th>
<th>AC</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>return</th>
<th>prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>(Debian, texteditor, openoffice.org – gtk – gnome, openoffice.org2 – gnome1 : 2.4.0 – 3Ubuntub, bash, gui, kde)</td>
<td>(Debian, openoffice.org – gtnk – gnome, openoffice.org2 – gnome1 : 2.4.0 – 3Ubuntub, texteditor, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9, kde)</td>
<td>(Debian, texteditor, openoffice.org – gtk – gnome)</td>
<td>openoffice.org – gnome1 : 2.4.0 – 3Ubuntub, bash, gui, kde</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>(Debian, openoffice.org – gnome1 : 2.4.0 – 3Ubuntub, bash, gui, kde)</td>
<td>(Debian, openoffice.org – gtnk – gnome, openoffice.org2 – gnome1 : 2.4.0 – 3Ubuntub, texteditor, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9)</td>
<td>(Debian, openoffice.org – gnome1 : 2.4.0 – 3Ubuntub, bash)</td>
<td>(gui, kde)</td>
<td>(kde)</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td>(Debian, openoffice.org – gnome1 : 2.4.0 – 3Ubuntub, bash)</td>
<td>(Debian, openoffice.org – gtnk – gnome, openoffice.org2 – gnome1 : 2.4.0 – 3Ubuntub, texteditor, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9)</td>
<td>(Debian, openoffice.org – gnome1 : 2.4.0 – 3Ubuntub, texteditor, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9, kde)</td>
<td>(gui)</td>
<td>(kde)</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td>(Debian, openoffice.org – gnome1 : 2.4.0 – 3Ubuntub, bash)</td>
<td>(Debian, openoffice.org – gtnk – gnome, openoffice.org2 – gnome1 : 2.4.0 – 3Ubuntub, texteditor, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9)</td>
<td>(Debian, openoffice.org – gnome1 : 2.4.0 – 3Ubuntub, texteditor, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9, kde)</td>
<td>(gui)</td>
<td>(kde)</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td>(Debian, openoffice.org – gnome1 : 2.4.0 – 3Ubuntub, bash)</td>
<td>(Debian, openoffice.org – gtnk – gnome, openoffice.org2 – gnome1 : 2.4.0 – 3Ubuntub, texteditor, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9)</td>
<td>(Debian, openoffice.org – gnome1 : 2.4.0 – 3Ubuntub, texteditor, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, vi, c9, kde)</td>
<td>(gui)</td>
<td>(kde)</td>
</tr>
</tbody>
</table>

Fig. 3. $P_1$: A valid product example for the Debian Linux distribution.

Fig. 4. $P_2$: A valid product example for the Debian Linux distribution.

Fig. 5. $P_3$: A non-valid product example for the Debian Linux distribution.

Fig. 6. $P_4$: A non-valid product example for the Debian Linux distribution.
TABLE II. FMDiag application example for the diagnosis of inconsistencies of $P_3$. 

<table>
<thead>
<tr>
<th>Step</th>
<th>D</th>
<th>S</th>
<th>AC</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>return</th>
<th>previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>{ Debian, bash, gui, kde }</td>
<td>( Debian, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9, kde )</td>
<td>( Debian, bash )</td>
<td>( gus, kde )</td>
<td>kde</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>{ gui, kde }</td>
<td>{ Debian, bash }</td>
<td>( Debian, bash, c1, c2, c3, c4, c5, c6, c7, c8, c9, kde )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>{ gui, kde }</td>
<td>( Debian, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9, kde )</td>
<td>( gui )</td>
<td>( kde )</td>
<td>kde</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>{ kde }</td>
<td>{ gui }</td>
<td>( Debian, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9, kde )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>{ kde }</td>
<td>( Debian, bash, gui, c1, c2, c3, c4, c5, c6, c7, c8, c9, kde )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

TABLE III. FMDiag application example for “VALID PRODUCT” operations of SPLIT FMs. 

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Model Features #</th>
<th>Model Dependencies #</th>
<th>Product Size</th>
<th>Diagnosis Size</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aircraft_fm.xml</td>
<td>13</td>
<td></td>
<td>13</td>
<td>1</td>
<td>452</td>
</tr>
<tr>
<td>car_fm.xml</td>
<td>14</td>
<td></td>
<td>14</td>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>connector_fm.xml</td>
<td>20</td>
<td></td>
<td>20</td>
<td>1</td>
<td>85</td>
</tr>
<tr>
<td>DELL-LAPTOP-NOTEBOOK-FM.xml</td>
<td>47</td>
<td>105</td>
<td>21</td>
<td>21</td>
<td>59</td>
</tr>
<tr>
<td>fame_doms_fm.xml</td>
<td>21</td>
<td></td>
<td>21</td>
<td>1</td>
<td>26</td>
</tr>
</tbody>
</table>

TABLE IV. FMDiag application example for “VALID PRODUCT” operations of large-scale FMs. 

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Model Features #</th>
<th>Model Dependencies #</th>
<th>Product Size</th>
<th>Diagnosis Size</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>model-50-10-1.xml</td>
<td>50</td>
<td>5</td>
<td>49</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>model-50-100-1.xml</td>
<td>50</td>
<td>50</td>
<td>45</td>
<td>1</td>
<td>41</td>
</tr>
<tr>
<td>model-100-10-1.xml</td>
<td>100</td>
<td>10</td>
<td>98</td>
<td>1</td>
<td>805</td>
</tr>
<tr>
<td>model-100-100-1.xml</td>
<td>100</td>
<td>100</td>
<td>95</td>
<td>1</td>
<td>186</td>
</tr>
<tr>
<td>model-1000-10-1.xml</td>
<td>1000</td>
<td>100</td>
<td>997</td>
<td>1</td>
<td>1149</td>
</tr>
<tr>
<td>model-1000-100-1.xml</td>
<td>1000</td>
<td>1000</td>
<td>994</td>
<td>1</td>
<td>993</td>
</tr>
<tr>
<td>model-1000-30-1.xml</td>
<td>1000</td>
<td>300</td>
<td>996</td>
<td>1</td>
<td>464</td>
</tr>
<tr>
<td>model-2000-10-1.xml</td>
<td>2000</td>
<td>200</td>
<td>1996</td>
<td>1</td>
<td>896</td>
</tr>
<tr>
<td>model-2000-100-1.xml</td>
<td>2000</td>
<td>2000</td>
<td>1994</td>
<td>1</td>
<td>1403</td>
</tr>
<tr>
<td>model-2000-30-1.xml</td>
<td>2000</td>
<td>600</td>
<td>1996</td>
<td>1</td>
<td>903</td>
</tr>
<tr>
<td>model-2000-30-1.xml</td>
<td>2000</td>
<td>600</td>
<td>1996</td>
<td>1</td>
<td>865</td>
</tr>
</tbody>
</table>

VI. Conclusion

Defining a “valid product” configuration is a current and demanding activity in product lines such as component-based software and SPL applications. It is highly relevant overall for large-scale configurations. Time demanding and complexity add complexity to that process. We show that existing diagnosis solution are applicable for efficiently determining the validity of a product configuration which can be partial or simply considering non-valid constraints. Thus, our defined research question is effectively answered. Even though we show FMDiag application results only, FlexDiag is usually more efficient, but less precise.

For the efficient obtained results of applying FMDiag and FlexDiag solutions and their algorithmic simplicity, we can look for new applications on the AAFM area since diagnosis algorithms are able for error detection in general. As a future work, we plan to show new applications for the AAFM support of both solutions on traditional and large-scale FMs and SPLs as well as extensions to work with on even more complex AAFM and SPL scenarios.

ACKNOWLEDGMENT

Franklin Johnson Parejas is supported by Grant “Dirección General de Investigación de la Universidad de Playa Ancha, Concurso Regular 2017, Clave ING 04-181”.

This work was partially supported by CONICYT-CCV/Doctorado Nacional/2018-21181055.

REFERENCES


