INCORPORATING NEW CONSIDERATIONS FOR PLATFORM SELECTION IN PRODUCT FAMILY DESIGN: CONNECTIVITY AND SECURITY

BY

ALVARO JOSE ROJAS ARCINIEGAS

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Systems and Entrepreneurial Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Adviser:

Assistant Professor Harrison M. Kim
Abstract

The research presented in this work entails the incorporation of connectivity and security as criteria to select the platform for a family of products (components to share) while optimizing the component arrangement of the individual products. The first part introduces a framework to find the optimal set of components to share across multiple products based on the connectivity between components. Simultaneously, each product architecture is defined and optimized with the consideration that the components to share should be excluded from the modules to facilitate sharing and should cause the least impact if it needs to change. This approach was validated using an illustrative example with three digital cameras in which the similarities between the products are exploited.

The second part is a logical progression from the first framework and introduces the consideration of the security of the products into the criteria to determine the components to share. In this case, the security of a product is understood as the ability to protect the sensitive information that product contains, or in other words, makes it as difficult as possible to extract the sensitive information from reverse engineering the product. This framework was applied to a set of inkjet printers and cartridges which exemplify three different architectures while providing the same basic functionality. A summary of the algorithms implemented in Matlab is presented in the appendix, providing the details of the calculation of the different metrics and the implementation of the optimization of the individual product architecture using Genetic Algorithms.
Acknowledgments

I would like to acknowledge my gratitude to Prof. Kim who opened the doors of UIUC for me. To my friends who made living in Chambana, in the middle of cornfields, fun. To my family for their relentless support and to my wife Vicky for her unconditional love.
To Vicky, my father and my sister who have always believed in me.
# Table of Contents

List of Tables ................................................................. vii
List of Figures ................................................................. viii

Chapter 1 Introduction ....................................................... 1
  1.1 Contributions ............................................................... 2  
    1.1.1 Optimal clustering and component sharing for multiple products ... 3  
    1.1.2 Securing multiple products using modularity ....................... 3  
  1.2 Applications ............................................................... 3

Chapter 2 Optimal Component Sharing in Product Family by Simultaneous Consideration of MDL and IM ....................................................... 5
  2.1 Introduction ............................................................... 5  
  2.2 Literature Review .......................................................... 6  
    2.2.1 Design Structure Matrix .............................................. 7  
    2.2.2 Minimal Description Length ....................................... 8  
    2.2.3 Impact Metric ....................................................... 9  
  2.3 Sharing Decision and Optimal Clustering Framework ..................... 11  
    2.3.1 Framework Overview ............................................... 11  
    2.3.2 Functional Matching ............................................... 12  
    2.3.3 Automated Selection of Shared Components ....................... 14  
    2.3.4 Iterative Approach ............................................... 15  
    2.3.5 A Genetic Algorithm Implementation ............................. 15  
  2.4 Illustrative Example ..................................................... 17  
    2.4.1 Digital Cameras .................................................... 17  
    2.4.2 Framework Application ............................................ 19  
    2.4.3 Threshold Level Selection ....................................... 21  
  2.5 Results and Discussion .................................................. 22  
  2.6 Conclusions ............................................................... 22

  3.1 Introduction ............................................................... 24  
  3.2 Literature Review .......................................................... 25  
    3.2.1 Security and Information Hiding .................................... 26  
    3.2.2 Optimal Clustering for Product Architecture ..................... 27  
    3.2.3 Enabling Manufacturing Techniques ................................ 28  
    3.2.4 Optimal Sharing Decision Making ................................ 29  
  3.3 Generating Security Constraints for Optimal Product Family Design ... 30  
    3.3.1 Defining Sensitive Components .................................... 30  
    3.3.2 Component Sharing ............................................... 33  
    3.3.3 Framework Overview .............................................. 34
List of Tables

2.1 CI rating system for sensitivity of specifications according to [22]. . . . . . . . . . . . . . . . 10

3.1 Example of the Security Matrix (SecM) for a product of 8 components. . . . . . . . . . . . . 31

3.2 Critical functions for printing subsystem. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
List of Figures

1.1 Overview of the organization of this thesis. ........................................... 2

2.1 Binary DSM for a fictitious product unordered and rearranged for clustering. .......... 7
2.2 Proposed framework for component sharing selection and optimum ......................... 12
2.3 Example of Function-Component Matrices for products A, B, and C with five .......... 13
2.4 Framework process diagram. ..................................................................... 16
2.5 Example of a chromosome for the DSM in Figure 2.1. ....................................... 17
2.6 Sony Digital Cameras (a) DSC-W100 [39], (b) DSC-S730 [40], ....................... 18
2.7 Impact Metric (IM) vectors and decision vectors (Y) for the Digital Cameras. .......... 20
2.8 Effect of the threshold variation on the component sharing. ............................. 21

3.1 Framework overview flow diagram. ............................................................ 35
3.2 Printers and cartridge supplies for each printer: a. Printer HP Deskjet 1220C and cartridges HP45 (BK) and 78 (C, M, Y); b. Printer HP Photosmart Plus and cartridges HP564 (BK, Photo BK, C, M, Y); c. Printer HP Business Inkjet 2600 and cartridges HP10 (BK) and 11 (C, M, Y), Printheads HP11 (BK, C, M, Y). ......................................................... 36
3.3 Functional structure and definition of critical path for printing subsystem. .......... 38

A.1 DSM for Digital Camera Sony DSC-T30. ................................................... 68
A.2 DSM for Digital Camera Sony DSC-S730. .................................................. 69
A.3 DSM for Digital Camera Sony DSC-W100. .................................................. 70
A.4 FCM for Digital Camera Sony DSC-T30. .................................................... 71
A.5 FCM for Digital Camera Sony DSC-S730. .................................................. 72
A.6 FCM for Digital Camera Sony DSC-W100. .................................................. 73
A.7 Optimum DSM for Digital Camera Sony DSC-T30. ....................................... 74
A.8 Optimum DSM for Digital Camera Sony DSC-S730. ..................................... 75
A.9 Optimum DSM for Digital Camera Sony DSC-W100. .................................... 76

B.1 DSM for printing subsystem in printer HP Deskjet 1220C. ............................ 105
B.2 DSM for printing subsystem in printer HP Photosmart Plus. .......................... 106
B.3 DSM for printing subsystem in printer HP Business Inkjet 2600. ................. 107
B.4 FCM for printing subsystem in printer HP Deskjet 1220C. .......................... 108
B.5 FCM for printing subsystem in printer HP Photosmart Plus. .......................... 109
B.6 FCM for printing subsystem in printer HP Business Inkjet 2600. ................. 110
B.7 Results summary for printer HP Deskjet 1220C. ....................................... 111
B.8 Results summary for printer HP Photosmart Plus. ..................................... 112
B.9 Results summary for printer HP Business Inkjet 2600. ............................ 113
B.10 FCM, CrFun and SecM generation for printer HP Photosmart Plus. .............. 114
Chapter 1

Introduction

There have been many approaches to establish the platform in a family of products. Manufacturing, assembly, material compatibility, and functionality are just a few examples of the criteria used to define the common structure (platform) from which products (variants) can be developed [23]. In this thesis, two criteria are examined and incorporated into decision frameworks: connectivity and security.

The first aspect considers that the optimal arrangement of components in a product can be determined based on the relationship between them; if components are strongly related, they should be grouped together and they should form a module. This criterion has been used in various approaches (e.g. [51] and [53]) considering the optimal architecture of a single product. However, when multiple products are taken into consideration, sharing a set of components to provide common functions is desired in order to gain advantages in production. This implies that the architecture set for each product should facilitate sharing components minimizing the effort required to change an aspect (or component) of the product.

The second criterion explores the desire of an Original Equipment Manufacturer (OEM) to protect their Intellectual Property (IP) from third-party manufacturers, remanufacturers, or even customers. In the computer science domain, it is common practice to divide a system into modules to preserve certain information; in this way, only the intended user or part of the system gets access to the necessary information. Once the system is partitioned, the only visible information is the one being interchanged between the different modules or subprograms. This principle is known as information hiding (introduced by Parnas in [28]) and highlights the importance of the architecture decisions, since defining the module boundaries determines which information is visible and which information is hidden. These principles are not exclusive to computer science and indeed extend to the physical domain. Therefore, the question was how to define the module boundaries to protect sensitive information, while facilitating component sharing among multiple products to establish the optimal architecture of each product in the family.

This thesis is organized in the following manner: The remainder of this chapter gives an overview of the proposed methodological frameworks and highlights the major contributions in each work; chapter 2 explores the problem of defining the components to share among multiple products and finding the optimal
component arrangement to facilitate component sharing; chapter 3 builds upon the work described in chapter 2 to incorporate security considerations into the decisions for defining the product platform and the individual product architecture; chapter 4 summarizes the achievements of these frameworks and discusses some ideas for future work. The appendices include the numerical results of the illustrative examples of chapters 2 and 3 as well as the implementation codes in Matlab for both cases. Figure 1.1 summarizes the organization of the document.

1.1 Contributions

The main contribution of this work is proposing methodological frameworks to facilitate the component sharing decision and optimal architecture of multiple products. Each framework incorporates different attributes and requires different information from the products and the designers; nonetheless, they provide a guideline for the decisions to be made for the product family design. Both frameworks are meant to be applied in a
later stage of the design process when some details are known for each product, or as a means to improve the design of an initial set of products in a redesign process.

1.1.1 Optimal clustering and component sharing for multiple products

The first framework takes the concept of optimizing the architecture of a product by exploiting the connectivity between its components and develops the concept further to be able to consider multiple products simultaneously. The linking factor for the products is the functionality; therefore, the components that are easy to change and fulfill similar functions in each product are selected as candidates for sharing. The framework returns at the end of the process the list of components to share across the product family, as well as the optimal component arrangement for each product.

1.1.2 Securing multiple products using modularity

The second framework proposed in this thesis builds upon the first framework and introduces the security considerations into the decision criteria. The intention is to protect intellectual property or sensitive components or information by means of the product architecture, carefully defining the module boundaries to enclose or encapsulate those sensitive items.

This analysis is conducted while considering component sharing among multiple products, thence, at the end of the process, the framework returns the set of components or modules to share among the product family, as well as the optimal component arrangement for each product.

1.2 Applications

In this thesis, consumer electronic applications are considered since this is an area where the products are being developed from common platforms, there is a lot of competition, and the product lifecycle is typically short (e.g. 12-18 months for cell phones, 3-4 years for laptops). These characteristics make them ideal choices for the implementation of the methodological frameworks proposed.

Specifically, the first framework was applied to a set of digital cameras from Sony from three different lines of products which are targeted for very similar markets, sharing functionalities since all are basic point-and-shoot cameras. Even with these similarities, the cameras did not share many components and the application of the framework showed potential areas for improvement. The case of not sharing components across a portfolio of digital cameras is not exclusive to Sony and it has been seen in other brands like Kodak in which the cameras may look extremely similar from the outside but the components are completely different. This
observation suggests the lack of an established methodology that specifically considers component sharing, and that is precisely the intention of the proposed framework.

The second framework was applied to a set of printers and cartridges from HP. The three printers are intended for different users and present very distinct architectures in their printing subsystems. The key aspect to consider was how the ink was handled and which were the consumable of each printer. In this way, the entry-level printer offers a cartridge that integrates the ink with the printhead, the photography-oriented printer offers individual ink cartridges for each color with the printheads fixed to the printer, and the professional-level printer offers not only ink cartridges for each color but also individual printheads.

This distribution conveys with the results found by applying the framework with the intention to protect both the ink and the printheads which are the most important elements in a printing subsystem. It also showed room for improvement highlighting elements to secure that have not been taken into account before and indicating which elements should be shared among the different printers.

Other areas that may benefit from these frameworks include automotive, aerospace, and military applications in which the products are traditionally being developed in families and there are concerns on how to share more components and how to protect key aspects of the products.
Chapter 2

Optimal Component Sharing in Product Family by Simultaneous Consideration of MDL and IM

2.1 Introduction

In the field of product design, product architecture has been defined as “the scheme by which the function of a product is allocated to physical components” - [47]. Similarly, when a set of products (also known as variants) are developed from a common set of subsystems, modules, and/or components (which constitute the product platform), the products are considered a product family [23, 17]. When designing a family of products, defining the architecture is a key decision to make since it affects not only the products being developed at that time, but also the variants to develop down the road from the given platform. Therefore, the decisions of which components should form a module or which should be shared across some or all the products in the family requires careful consideration.

In the case of single products, many of the studies of product architecture have used the Design Structure Matrix (DSM) to model the product, as pointed out by [3]. The DSM was introduced by [43] and it has been widely used as a way to represent the interconnections of a product, whether it means geometrical joints, electrical connections, material flow, etc. This representation of the product has allowed researchers and designers to find the optimal module definition by clustering the cells with strong relationship among components through manipulation of the rows and columns of the matrix (e.g. [53]). However, when multiple products are considered, trying to capture the relationships, similarities, or differences among the various products through the use of multiple DSMs is not an easy task. Every product has its own set of components; therefore, each DSM is different in size and content, making it very difficult to relate to other DSMs.

Other approaches rely on a commonality metric which indicates the number of unique components of a product relative to the total number of components (e.g. [21, 24]). Even when this metric involves only counting the number of components, a very important aspect is captured in a product family: Commonality. [36] highlights that the main objective when creating a platform is to facilitate the generation of new products by having a common structure from which the new variants can be developed. Therefore, the question is not how many components should be common or standard, but rather, which components should be shared.
across the family. The difference is subtle, but of great significance: it is more important to find the set of components to have common, rather than finding a number of components to share.

Different criteria have been used in order to determine the components to share: cost considerations (e.g. [4, 54, 25]), Bill of Materials (BOM) (e.g. [42]), product attributes (e.g. [46]), environmental concerns (e.g. [18, 5, 27]), product design variables (e.g. [16, 17]), etc. However, little attention has been paid to the way in which the components are arranged into the products. The attempt of this paper is to provide guidance to the designer/engineer in which components should be shared across the products of a given family based on the product architecture.

The framework proposed in this paper goes further than previous approaches which only consider optimal clustering for a single product DSM (e.g. [52, 50, 53]), and considers multiple DSMs simultaneously in order to find the set of components to share across the product family. The decision is based on the connectivity between components, along with the functional decomposition from each one of the products.

The remaining sections of the paper proceed as follows: a review of the relevant literature in this topic is presented in section 2.2; the sharing decision framework is introduced in section 2.3; which is followed by its application to a case study in section 2.4; the results of the case study and its relevance are discussed in section 2.5; and the paper concludes with section 2.6, which summarizes the proposed framework and the implications of the observed results.

2.2 Literature Review

Finding a way to represent a product, capture its relationships, and to be able to analyze similarities, dependencies, component clustering, and modularity, have been some of the areas related to product architecture that have been greatly explored. In terms of product representations, three tendencies tend to dominate the approaches: a graph/network representation (e.g. [51, 41]), matrices (e.g. [43, 3]), and commonality indices (e.g. [45]). The first two are intrinsically related since it has been shown that graphs can be represented by matrices and vice versa [3, 50]. The commonality approaches tend to focus on counting the number of components that are standard or unique, compared to the total number of components (e.g. [21, 24]). This representation does not capture the strength in the relationships inside a product and is not intended to capture the component arrangement in the product; therefore, this approach is not considered in this work.

The Design Structure Matrix (DSM) is a widely accepted approach for a matrix representation and will be reviewed briefly in section 2.2.1.

The modularity of an architecture is understood as the degree to which the product functions are imple-
mented by the physical elements of the product \[48, 8\]. Multiple metrics have been developed to quantify this concept in a particular manner, most of them reflecting that a ‘good’ module has strong internal connections and weak external connections \[9\]. This idea is reflected by the Minimal Description principle, which has been used to find the optimal clustering for the product, and the Impact Metric, which provides a description of the easiness to change a component in a given architecture. These two methods will be reviewed in sections 2.2.2 and 2.2.3 respectively, since these are key aspects to consider when trying to decide which components should be shared across a family of products.

### 2.2.1 Design Structure Matrix

The Design Structure Matrix (DSM) introduced by \[43\] has been well established as a tool for system analysis since it captures the relationship between elements. “A DSM provides a simple, compact, and visual representation of a complex system” \[3\], and it has been applied to model products, processes, tasks, and organizations \(e.g.\ [6, 3, 4]\). In the case of product design the DSM arranges the components of the product in both rows and columns of a matrix, and each cell contains the relationship between the corresponding components.

The most basic DSM is binary (see Figure 2.1) in which the Xs or ones represent a relationship or connection between the corresponding components. The DSM is also able to capture directionality in the relationships. Usually the components in the columns are the suppliers and those in the rows are the receivers, and in that case, the DSM is not necessarily symmetric. Given the flexibility of this representation, there are many different adaptations depending on the information to be collected from the product. \[29\], and \[37\] use the DSM to specify whether the nature of the relationship between components is a physical connection,
energy flow, mass flow, or information flow. Similarly, [22], and [20] introduced the Coupling Indices, which used a DSM that contains the sensitivity analysis for specification changes in a product.

Overall, the DSM is a very effective tool to represent the relationships inside a product. However, when multiple products are considered, there is no representation established yet, and it is difficult to extract similarities or differences between products from multiple DSMs.

### 2.2.2 Minimal Description Length

A different approach to model a product structure is through the use of the minimal description length (MDL). According to [53] the goal of the minimal description principle is to represent a system uniquely, in the simplest possible way.

In the case of product architecture, a binary DSM can be represented by the number of clusters it has and the elements that belong to each cluster. In figure 2.1, the product of nine components can be described by stating that it has two clusters, one with components \{A,E,G\}, and other with components \{B,C,F,H\}. However, that description is not accurate since the cells outside the clusters are not all empty (mismatch Type I), and the cells inside the clusters are not fully populated (mismatch Type II); therefore, the representation is only complete when these mismatches have been taken into account.

Nevertheless, there would be many possible ways to represent a given DSM depending on the number of clusters or the selected components for each cluster. Each representation would be a valid one, but the length of the descriptions would be different. A more complicated model would require more details and thus a longer description. Therefore, the MDL principle can be interpreted as trying to find the shortest of those valid descriptions for the data [52, 51].

[53] used this approach to represent a DSM and find the optimal clustering strategy. In their implementation, the length of the model is captured according to equation 2.1 and the length of the mismatches is captured by equation 2.2.

\[
S_1(2\log_2 n_c + 1) + S_2(2\log_2 n_c + 1)
\]

In these equations, \(n_{cl}\) represents the number of clusters, \(n_c\) is the total number of components of the product, and \(cl_i\) is the number of elements in cluster \(i\). In equation 2.2 the term \(S_1\) corresponds to the number of non-empty cells outside the clusters defined by the model (mismatch Type I), and the term \(S_2\) corresponds to the number of empty cells inside the clusters (mismatch Type II) excluding the elements of the main diagonal which are not considered as mismatches. This definition works only when the DSM is
binary; when the elements in the DSM are not binary entries, the mismatches (according to [53]) should be redefined as follow:

\[
S_1 = \sum_{i,j} (1 - p_{ij}); \quad S_2 = \sum_{i,j} (p_{ij});
\]

\[
p_{ij} = \frac{d_{ij} - d'_{ij}}{d_{\text{max}} - d_{\text{min}}}; \quad d_{\text{max}} = \max_{ij} d_{ij}; \quad d_{\text{min}} = \min_{ij} d_{ij};
\]

where \(d_{ij}\) is the entry of the \(i^{th}\) row and \(j^{th}\) column of the DSM, \(p_{ij}\) is the same entry normalized, and \(d'_{ij}\) is the entry for the ideal binary DSM described by the model without considering any of the mismatches.

In the end, the length of a representation for a given DSM can be obtained applying:

\[
f_{\text{MDL}} = (1 - \alpha - \beta)(n_{cl} \log_2 n_c + \log_2 n_c \sum_{i=1}^{n_{cl}} c_i) + \alpha(S_1(2 \log_2 n_c + 1)) + \beta(S_2(2 \log_2 n_c + 1))
\]

where \(\alpha\) and \(\beta\) are weights between 0 and 1, making the function a linear combination of three terms, the length of the model and the length of the mismatches type I and II.

In the work by [53], this becomes the objective function (\(f_{\text{MDL}}\)) for the optimization in which the optimal clustering is found. The representation of the entire DSM observed in equation 2.5 is intended for product-level analysis; thence, there is no consideration for multiple products, component description, or component sharing, which are key aspects of product family design.

### 2.2.3 Impact Metric

[32, 34] proposed a metric to capture the impact to change a component in a given platform, combining the MDL representation for each component and the Coupling Index (CI) score. The MDL formulation used in the IM is different given that the objective is to find a representation for the components rather than the entire DSM. The MDL formulation was derived from [52, 51], and [50], in which each component is represented by the interconnections it has with other components/clusters, and it is compared to the total number of connections in the system at the same level. Level in this context is understood in the following manner: inside a module, a component is at the same level with other components or submodules that are in it. When there exists a connection from a component inside a module to a component outside the module, the component is considered an interface and it is compared against other interfaces at the same level.

This means that for a unit \(j\) (either component, module, or interface) the description length would be given by:
Table 2.1: CI rating system for sensitivity of specifications according to [22].

<table>
<thead>
<tr>
<th>Rating</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Small change in specification impacts the receiving component</td>
</tr>
<tr>
<td></td>
<td>(High sensitivity)</td>
</tr>
<tr>
<td>6</td>
<td>Medium High sensitivity</td>
</tr>
<tr>
<td>3</td>
<td>Medium Low sensitivity</td>
</tr>
<tr>
<td>1</td>
<td>Large change in specification impacts the receiving component</td>
</tr>
<tr>
<td></td>
<td>(Low sensitivity)</td>
</tr>
<tr>
<td>0</td>
<td>No specification affects the receiving component</td>
</tr>
</tbody>
</table>

\[
MDL_{j}^{(u)} = -\log_2 \left( \frac{N_j^{(u)}}{\sum_{k=1}^{N^{(u)}} N_k} \right)  \tag{2.6}
\]

in which \(N_j^{(u)}\) is the number of units connected to unit \(j\), and \(N^{(u)}\) is the number of units at the level in which the unit \(j\) is. Since a unit can be seen as a component if it is considered inside a module, but also as an interface if it has connections outside the module, the total representation of unit \(j\) in product \(k\) would be:

\[
MDL_{j}^{k} = MDL_{j}^{(c)} + MDL_{j}^{(o)} = -\sum_{u \in \{c,o\}} \log_2 \left( \frac{N_j^{(u)}}{\sum_{k=1}^{N^{(u)}} N_k} \right)  \tag{2.7}
\]

where \(c\) represents components and \(o\) interfaces.

In order to obtain the IM for each component, it is necessary to obtain the Coupling Index (CI). Therefore, instead of using a binary DSM (which would be enough to calculate the MDL for the components), the DSM contains the sensitivity analysis for the impact of a change in the specifications of the components (according to Table 2.1 from [22]). This assumes that the components in the columns supply the specification and the components in the rows receive the specifications.

Each row and column of the DSM is added giving the CI-Receiving (CI-R) for the rows, and the CI-Supplying (CI-S) for the columns. The total CI for each component is the sum of the CI-S and the CI-R (see equation 2.8 for the CI of component \(j\) in product \(k\)).

\[
CI_{j}^{k} = CI - S_{j}^{k} + CI - R_{j}^{k} = \sum_{i=1}^{n_c} DSM_{i,j}^{k} + \sum_{i=1}^{n_o} DSM_{j,i}^{k}  \tag{2.8}
\]

The last step in the calculation of the IM for each component is to multiply the MDL representation by the CI (see equation 2.9). This scaling effect serves well to leverage the architectural description of each component based on the number of interconnections as well as the strength of those connections or coupling of the components. It also highlights those components which would be more difficult to make a change (a
higher IM rate indicates that the component is much more connected with the rest of the system, and/or the connections are strong; thence, other components would get affected by a change in that particular component).

\[
IM_j^k = MDL_j^k \cdot CI_j^k
\]  

(2.9)

This score is a good way to distinguish between components which are easy to change and those which are difficult. However, there is no standard way to compare the scores for components of different products since the application of the IM to analyze multiple products has not been considered before.

2.3 Sharing Decision and Optimal Clustering Framework

This section presents an approach to alleviate the problem of determining the optimal set of components to share across the products in a family while achieving an optimal clustering of their components. The goal is to identify the components to be shared across multiple products of the family in an automated manner, based on the architectural information of the products and the individual components.

2.3.1 Framework Overview

The proposed framework is an iterative process and it identifies candidates for component sharing based on the functional description of each component and their IM score, which represents the ease of changing the component under the current architecture.

The decision in component sharing is marked in binary vectors (one for each product) of as many elements as the number of components for the product, one (1) for the component to be shared and zero (0) for the component that will be kept unique for the product. If, for example, a product with six components shared the second and fourth components with other products, the corresponding decision vector would be [0 1 0 1 0 0].

Once the decision vectors have been established, an optimization problem is solved for each individual product of the family with a dual objective: on one hand, the product MDL representation of equation 2.5 finds a compact arrangement of the components in clusters; meanwhile, the decision vector is multiplied by the IM vector from equation 2.9, thus selecting the components to include in the objective function and allowing those components to be easily changed in the product. At each iteration a new component arrangement is achieved for each product, progressively establishing the final list of components to share.
The process is repeated until the change in the decision vectors for all the products being considered is below a given tolerance $\epsilon$ (see equation 2.10). An overall view of the framework can be seen in Figure 2.2.

$$\sum_{i=1}^{m} \left\| \hat{Y}_i(k) - \hat{Y}_i(k-1) \right\|^2_{2} \leq \epsilon$$

(2.10)

The major contribution of this framework is the automation of the optimal sharing decision process. However, one of the major concerns in achieving this goal is to identify the components of different products that can be considered as candidates for sharing. When considering multiple products, the name of the component and its material or dimensions are not always helpful to identify similar components; thence, the proposed attempt is to look for components that fulfill identical functions.

### 2.3.2 Functional Matching

Since the products in the family may share many functions they fulfill, it is necessary to construct a functional description for each product, in which the components can be related to the functions they fulfill. Each product has its own set of functions and components. For the purpose of the analysis it is necessary to construct an enumerative list of functions across all the products, which serves as a reference list to determine which function is common or unique.

A Function-Component Matrix (FCM) relates functions and components in a similar manner to a binary...
Figure 2.3: Example of Function-Component Matrices for products A, B, and C with five functions, and functional matching.

DSM [44]; each marked cell indicates that the corresponding function is fulfilled partially or completely by the corresponding component. Each component (columns in the FCM) must have a relation with at least one function (rows in the FCM), since every component has a reason to be a part of a product. Similarly, each function should have at least one relation with a component in the entire family, since there may be features that are unique to a particular product in the family.

In Figure 2.3 three products with a list of five functions are examined. First, looking at products A and B only, it is clear that component 1 in product A, as well as component 2 in product B, are fulfilling function 2. Therefore, they are a match (one-to-one components for a single function). Similarly, components 3 and 4 of product A match component 1 of product B that corresponds to function 3 (one-to-many components for a single function). Meanwhile, component 2 of product A must match with components 3 and 4 of product B since they are fulfilling both functions 1 and 4 (one-to-many components for multiple functions). Component 5 of product A could not be matched with any component from product B since function 5 is unique for product A.

When matching components for products A and C based on the functional description, component 1 of product A is a match for components 2 and 3 of product C since they fulfill function 2. Component 5 of product A cannot be matched with any component of product C since function 5 is unique for product A. Nevertheless, when the remaining components are examined there is no way to match them individually since the functions they fulfill make them interconnected. Thence, the result is that components 2, 3, and 4 of product A match components 1, 4, 5, and 6 of product C since they fulfill functions 1, 3, and 4 (many-to-many components for multiple functions).
2.3.3 Automated Selection of Shared Components

Once all the cases are identified, the strategy to match the components based on the functional description is developed for pairwise comparisons in which:

- A master function-vector (f-v) keeps track of the functions evaluated for the pair,
- A local function-vector marks the functions fulfilled by the matching components,
- Two component-vectors (c-vs) mark the components in each product that are necessary to fulfill the functions in the local function-vector, and
- Two decision-vectors are returned after the evaluation and indicate the components to share for each product.

The pairwise selection of candidates for sharing is conducted as follows:

1. Start by evaluating the first function;
2. mark the master f-v with the corresponding function;
3. initialize the local f-v and c-vs,
4. follow the row of the Function-Component Matrix (FCM) corresponding to the function being evaluated until a 1 is found;
5. mark the corresponding component in the c-v;
6. follow the column of the corresponding component and mark the functions that it fulfills in the local f-v;
7. return to step 4 until all the components in both products have been evaluated;
8. mark the functions marked in the local f-v into the master f-v;
9. evaluate if the summation of the IM of the components marked in the c-v for each product is below a threshold previously defined, if so, mark the components in the c-vs into the decision-vectors; and
10. continue the evaluation of the next function unmarked in the master f-v and return to step 2.

Once a pairwise evaluation has been conducted the process is repeated with all the possible pairs of products. The result of this stage is the decision vectors for all the products in the family containing the components to share from each product.
2.3.4 Iterative Approach

The decision vectors are passed to the individual optimizations of the product architecture, in which the optimum clustering is found for all the products (considering the set of components to share among the family, which were selected in each decision vector). However, the new arrangement of components alters the IM score from which the candidates for sharing were selected. Therefore, the new architecture becomes the initial step for the process and the evaluation is conducted again until a stopping criterion is met (see equation 2.10).

One important factor in the automated selection process is the threshold value for the IM set by a decision maker; a higher value implies that more components will become candidates for sharing if the functional matching is achieved. However, this may be an important strategy for the company either to allow greater commonality among the family or decide to share only few components that will not implicate much effort to change according to the architectural information.

The overall framework proceeds as follows:

1. Construct the product DSM for each variant of the family;

2. Construct the functional relationship matrix for each product and include all the functions in the family;

3. Calculate the IM for all the components in all the products;

4. Run the component sharing selection algorithm to obtain the decision vectors for each product in the analysis;

5. Run the optimization for each product separately including both the MDL representation of the product and the IM of the selected components for sharing to obtain a new arrangement of the components in the products considering component sharing; and

6. Verify if a stopping criteria is met (i.e. no further change in all the decision vectors), if not return to step 3.

The process is summarized in Figure 2.4.

2.3.5 A Genetic Algorithm Implementation

The optimization is implemented by the use of Genetic Algorithms similar to the one proposed by [53], in which the chromosome represents the clusters and its elements out of the components of the product. The
Figure 2.4: Framework process diagram.
Figure 2.5: Example of a chromosome for the DSM in Figure 2.1.

<table>
<thead>
<tr>
<th>Component:</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 1:</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cluster 2:</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Chromosome: 1 0 0 0 1 0 1 0 0 0 1 1 0 0 1 0 1 0

Since the number of components is specified in the DSM, the only additional parameter to specify is the maximum number of clusters. That implies that there may be some empty clusters, and the actual number of clusters (non-empty) is automatically determined by the GA.

The objective function is a dual objective: First, the MDL representation of the product is obtained following equation 2.5, which provides a global score in how compact the structure is; Second, the decision vector is multiplied by the IM vector adding the IM score of the selected components for sharing into the objective function. The formulation is as follows:

\[
f(DSM^i, X^i) = f_{MDL}^i(DSM^i, X^i) + (Y^i(k)^T \ast IM^i(DSM^i, X^i))
\]  

(2.11)

in which \(X^i\) is the chromosome that is being evaluated. This formulation preserves the tight clustering achieved by the MDL and allows the consideration of sharing components by including its IM score.

The details of the algorithms can be seen in the Appendix A.1. The Simple Genetic Algorithm used is based on the class notes and exercises by Prof. Goldberg at UIUC.

### 2.4 Illustrative Example

In order to illustrate the application of the proposed framework, an example was developed for three digital cameras from the Sony DSC (Digital Still Capture) portfolio.

#### 2.4.1 Digital Cameras

In the market of digital cameras most of the brands offer an entry level family of compact point-and-shoot cameras, a set of ultra-slim point-and-shoot cameras, higher end high-zoom cameras, and the professional
level Digital Single-Lens Reflex (DSLR) cameras. The first two classes offer many similarities both in size and features as well as the customer groups and their uses. Therefore, conducting an analysis for component sharing among the point-and-shoot cameras was a logical choice.

Sony in particular has had two lines of cameras in the first category (compact point-and-shoot), the S family and the W family. The cameras from the S family were the entry level to the cyber-shot digital cameras (the line was recently discontinued) and featured the basic functions of a point-and-shoot digital camera: easy to use, auto focus, several exposure modes, extendable optical zoom, flash, video capture with sound, among others. The cameras from the W family had been very similar but usually included slightly better features than the cameras from the S family, such as optical stabilization or a viewfinder in order to attract a more demanding customer with very similar exterior design and size. The set of ultra-slim point-and-shoot cameras has been covered by the T family, which featured a lens cover that slides, a non-extendable zoom, and a very sleek design, along with most of the features from the other families. This camera is intended for a different customer and has a slightly higher price compared to a camera from the other two families.

Given the similarities and unique features in these types of cameras, an illustrative study was developed with three cameras: DSC-W100, DSC-S730, and DSC-T30 (one from each line) from the available service manuals at [39, 40, 38]. Figure 2.6 shows an image of the selected cameras for this case study.

The disassembly instructions along with the information in the manuals were analyzed in order to construct a DSM for each product as well as the Function-Component Matrix (FCM) and can be seen in the Appendix (see section A.2 for the DSMs and A.3 for the FCMs). The cells in the DSMs were populated according to the CI specification sensitivity analysis in order to calculate the CI and IM afterwards.
2.4.2 Framework Application

Since the DSMs and FCM were already developed (steps 1 and 2 of the framework), the framework continues by calculating the IM for all the components in all the products included in the analysis. The IM calculation implies a component arrangement, thence an initial chromosome was generated in which no cluster was defined (a 1-by-$n_i$ zero vector was used). $IM(k)$ would correspond to the IM vector at the $k^{th}$ iteration for each camera (see Figure 2.7).

The next step is to run the component sharing selection algorithm which performs pairwise comparisons among all the products to determine component matching (according to the functional description of the FCM) and evaluates if the sum of the IM score of the matching elements is below the threshold, defined at 50% for this case (see section 2.4.3 for more details). The decision vectors are returned with the components to share from the cameras marked as ones (1), and the components to maintain unique for each product as zeros (0) (see Figure 2.7).

The decision vector for each product is then fed to the optimization algorithm (individual for each product) in which the dual objective function is used to find the optimum arrangement of components in clusters for the product. The chromosome for the optimum objective function value is returned and contains the information of the component arrangement in clusters.

The chromosome contains the clustering strategy which includes clusters of components that are tightly related, and single element clusters which are related to the components to be shared, since it is a premise of the analysis that a component would be easier to share if it is not part of any cluster unless the entire module is going to be shared. The chromosome may contain empty clusters indicating that a fewer number than the maximum of clusters is needed in order to achieve the optimum objective function value.

The results from the optimization (new component arrangement) alter the values of the IM. Therefore, there is the need to go back, calculate the new IM, and rerun the process until there is no change in the decision vector in any of the products considered in the analysis. For the case of the three digital cameras hereby considered, only two additional iterations were necessary and Figure 2.7 shows the summary of the IM and Y (decision) vectors for the three cameras.

At the end of the process, the optimization algorithm returns the optimal chromosomes for the three cameras, representing the optimum component arrangement that not only preserves clusters in which the components are tightly coupled, but also facilitates the component sharing of the selected elements. The rearranged DSMs can be seen in the appendix A.4.
<table>
<thead>
<tr>
<th>DSC-W100 Component</th>
<th>IM(3)</th>
<th>Y(3)</th>
<th>DSC-S730 Component</th>
<th>IM(3)</th>
<th>Y(3)</th>
<th>DSC-T30 Component</th>
<th>IM(3)</th>
<th>Y(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ornamental Ring</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>Lens Cover</td>
<td>1</td>
<td>112</td>
</tr>
<tr>
<td>Cabinet Front</td>
<td>2</td>
<td>511</td>
<td>0</td>
<td></td>
<td></td>
<td>Cabinet Front</td>
<td>2</td>
<td>660</td>
</tr>
<tr>
<td>LCD Window</td>
<td>3</td>
<td>140</td>
<td>0</td>
<td></td>
<td></td>
<td>LCD Window</td>
<td>3</td>
<td>79</td>
</tr>
<tr>
<td>Buttons</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>Buttons</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Side Lid</td>
<td>5</td>
<td>123</td>
<td>0</td>
<td></td>
<td></td>
<td>Control Switch</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Cabinet Rear</td>
<td>6</td>
<td>532</td>
<td>0</td>
<td></td>
<td></td>
<td>Cabinet Rear</td>
<td>6</td>
<td>633</td>
</tr>
<tr>
<td>Side Cover</td>
<td>7</td>
<td>70</td>
<td>0</td>
<td></td>
<td></td>
<td>LCD</td>
<td>7</td>
<td>486</td>
</tr>
<tr>
<td>Control Switch</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>LCD Plate</td>
<td>8</td>
<td>156</td>
</tr>
<tr>
<td>LCD</td>
<td>9</td>
<td>233</td>
<td>0</td>
<td></td>
<td></td>
<td>BT Lid Shaft</td>
<td>9</td>
<td>239</td>
</tr>
<tr>
<td>LCD Plate</td>
<td>10</td>
<td>95</td>
<td>0</td>
<td></td>
<td></td>
<td>LCD Plate</td>
<td>10</td>
<td>267</td>
</tr>
<tr>
<td>LCD PCB</td>
<td>11</td>
<td>155</td>
<td>0</td>
<td></td>
<td></td>
<td>Battery Lid</td>
<td>11</td>
<td>328</td>
</tr>
<tr>
<td>Side Frame 1</td>
<td>12</td>
<td>153</td>
<td>0</td>
<td></td>
<td></td>
<td>Shooter and Power</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Shoot Button</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>Strap shaft</td>
<td>13</td>
<td>198</td>
</tr>
<tr>
<td>Zoom Slider</td>
<td>14</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>Top Frame</td>
<td>14</td>
<td>993</td>
</tr>
<tr>
<td>Mic</td>
<td>15</td>
<td>105</td>
<td>0</td>
<td></td>
<td></td>
<td>Flash</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>Speaker</td>
<td>16</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>Capacitor</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Top Frame</td>
<td>17</td>
<td>255</td>
<td>0</td>
<td></td>
<td></td>
<td>Flash PCB</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>Side Frame 2</td>
<td>18</td>
<td>252</td>
<td>0</td>
<td></td>
<td></td>
<td>Main PCB</td>
<td>18</td>
<td>496</td>
</tr>
<tr>
<td>Capacitor Holder</td>
<td>19</td>
<td>12</td>
<td>0</td>
<td></td>
<td></td>
<td>Support Jack</td>
<td>19</td>
<td>67</td>
</tr>
<tr>
<td>Flash</td>
<td>20</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>CCD</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>Capacitor</td>
<td>21</td>
<td>48</td>
<td>0</td>
<td></td>
<td></td>
<td>Filters</td>
<td>21</td>
<td>167</td>
</tr>
<tr>
<td>Flash PCB</td>
<td>22</td>
<td>319</td>
<td>0</td>
<td></td>
<td></td>
<td>Barrier Ring</td>
<td>22</td>
<td>118</td>
</tr>
<tr>
<td>CCD</td>
<td>23</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>Ornamental Ring</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>Filters</td>
<td>24</td>
<td>175</td>
<td>0</td>
<td></td>
<td></td>
<td>Lens Sub-ASM</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>Barrier Ring</td>
<td>25</td>
<td>140</td>
<td>0</td>
<td></td>
<td></td>
<td>Lid</td>
<td>25</td>
<td>200</td>
</tr>
<tr>
<td>Ornamental Ring</td>
<td>26</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>Jack Lid</td>
<td>26</td>
<td>118</td>
</tr>
<tr>
<td>Visor</td>
<td>27</td>
<td>288</td>
<td>0</td>
<td></td>
<td></td>
<td>Main Body</td>
<td>27</td>
<td>3088</td>
</tr>
<tr>
<td>Lens Sub-ASM</td>
<td>28</td>
<td>682</td>
<td>0</td>
<td></td>
<td></td>
<td>Rear PCB</td>
<td>29</td>
<td>283</td>
</tr>
<tr>
<td>Rear PCB</td>
<td>29</td>
<td>283</td>
<td>0</td>
<td></td>
<td></td>
<td>Top Flex PCB</td>
<td>30</td>
<td>222</td>
</tr>
<tr>
<td>Lid</td>
<td>31</td>
<td>123</td>
<td>0</td>
<td></td>
<td></td>
<td>Lid</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>Front PCB</td>
<td>32</td>
<td>343</td>
<td>0</td>
<td></td>
<td></td>
<td>Battery Holder</td>
<td>33</td>
<td>355</td>
</tr>
<tr>
<td>Battery Holder</td>
<td>33</td>
<td>355</td>
<td>0</td>
<td></td>
<td></td>
<td>Main Body</td>
<td>34</td>
<td>741</td>
</tr>
</tbody>
</table>

Figure 2.7: Impact Metric (IM) vectors and decision vectors (Y) for the Digital Cameras.
2.4.3 Threshold Level Selection

The threshold level is a key factor in order to set the desired level of commonality among the products considered in the analysis. The threshold value is calculated as a percentage of the range of the IM values for each product; higher values of IM indicate that the component is not easy to change under the current architecture. The higher the threshold, the greater the number of components that can be considered as suitable candidates for sharing; therefore, it depends on the strategy of the company as to where this level should be set.

Nevertheless, for the purpose of this analysis, the parameter was varied until all the components would become candidates for sharing. The variation on the threshold value went from zero to 5.5 or 4.6, depending on the camera, at which level all the components were candidates for sharing. This variation served well to analyze the effect of the threshold level on the component sharing decision for these three products (see Figure 2.8).

The threshold level was set at 0.5 (i.e. 50% of the range of the IM score per product). At 0.5 some desired characteristics were observed: the components to share have very low IM scores (below 50% of the range); the number of components to share is not too high; and from that point on, big increments are required in order to get other components to be candidates.

On the other hand, the fact that the threshold needs to surpass 4 or 5 times the range of IM to make all components candidates is critical for the decision-making process. Lower thresholds may not capture the full potential for commonality, while higher thresholds may limit the opportunity for sharing components effectively.
the components become candidates for sharing (see highest threshold value for each camera in Figure 2.8) can be understood when the functional matching process is examined. If multiple components are needed in order to match one or multiple functions, it is necessary to add the IM scores for those components; thus it is easy to go above the range.

One important clarification is that, for this case, the threshold was set constant among the three cameras; however, this is not a requirement for the framework application. Individual threshold values can be set for the different products achieving more commonality with a given variant rather than other.

2.5 Results and Discussion

The results obtained by the framework application can be compared to the application of the algorithm proposed by [53] in which there is no consideration regarding component sharing. The summary of the results can be seen in the Appendix A.4.

It can be observed that the clustering strategies are quite different since the decision to share components modifies significantly the clusters that can be formed. For example, in the DSC-W100 the optimum without considering component sharing did not include a cluster with components Capacitor Holder and Capacitor (components 19 and 21 respectively), while in the optimum considering component sharing those components appear in a cluster (see Figure A.9). Similarly, clusters that contained components selected for sharing changed, either reducing the number of components in the cluster (see Figure A.8, cluster with components \{9, 10, 11, 27\} reduces to \{11, 27\}), including additional components, or simply breaking down (see Figure A.7, cluster with components \{5, 6, 7, 9\} breaks down when \{6\} and \{7\} are selected to be shared). These new structures for the products represent the optimal component arrangement in order to facilitate the sharing of the selected components according to the inherent connectivity among them, an aspect that has not been considered in this manner before.

2.6 Conclusions

This paper introduced a framework to select the components to share in an automatic manner, among multiple products based on the architectural information contained in the product DSM and the Function-Component Matrix (FCM). It also proposes the optimization of the component arrangement in each product, finding the clusters that facilitate the component sharing while maintaining the compact modules in the product.

The process of selecting the candidates was based on the IM score of the components and the functional
matching of the components across different products. The first indicated the impact on changing a compo-
ment according to the current structure of the product. The functional matching was the means to identify
components that have the same purpose in different products and may be suitable candidates for sharing.

In order to determine the optimal clustering while considering component sharing, both a product level
representation (MDL) and a component level score (IM) were required and served as the objective function
during the optimization of each product architecture.

Nevertheless, it is well known that product sharing decisions involve many different criteria (material
affinity, serviceability, production costs, etc.) that have been ignored in the current framework. Therefore,
it is necessary to continue to work along this line in order to include some of those factors facilitating the
decision process.
Chapter 3


3.1 Introduction

Product family design is a field that traditionally encompasses the selection of a common set of components or subsystems (product platform) that can be used to easily develop products (variants) [23] in order to fulfill a wider range of market needs. The authors explored these decisions in [35], proposing a framework to obtain the set of components to share across the family, as well as the optimal architecture for each product. The work presented in this article is a logical progression from that framework including a new aspect of securing sensitive components into the decision of establishing the components to share and the optimal architecture for multiple products.

The architecture of a product is understood as the allocation of the functions in the physical components of the product [47], and it is a key aspect for family design since it affects all the variants developed from the platform. However, the connectivity between components is not the only aspect to consider when determining the components to share in a family. Many times, there exists sensitive information in each product that the manufacturer would not want to be accessible by the user or third-party manufacturers; then, the security of those sensitive components or interfaces becomes a concern for the designers.

These security concerns are also evident in the military domain where the term Anti-Tamper is usually used to denote the requirements (or expectations) of a system or subsystem to deter any efforts to reverse engineer, exploit, or develop countermeasures for critical technology and should, in this manner, add longevity to the system [49]. Nevertheless, the guidelines and requirements have been developed mostly for software and electronics in which the programmable nature of these areas facilitate techniques such as encryption, code obfuscation, hardware keys, and dynamic reconfiguration [1, 30]. In the physical/mechanical domain it is still a challenge to translate these protections even when the ideas and concerns are still valid.

Some manufacturing techniques allow encapsulating some components in a very secure manner; custom made integrated circuits, in-mold inserts for injection molding, and Micro Electro-Mechanical Systems (MEMS) are just a few examples. Having these capabilities at hand, it becomes important to identify
which component or information is sensitive and which could be the best way to protect it. This paper aim to help the designers securing those components or relationships that are sensitive by using modularity to hide information, while considering component sharing and optimal architecture for multiple products.

The remaining sections of the paper proceed as follows: a review of the relevant literature in this topic is presented in section 3.2; the sharing decision framework is introduced in section 3.3; which is followed by its application to an illustrative example in section 3.4; and the paper concludes with section 3.5, which summarizes the proposed framework and the implications of the observed results for future work.

3.2 Literature Review

In product architectural analysis, a few terms are commonly used but sometimes the implications may be confusing; therefore, in this paper a **component** is an element (or group of elements treated as a single part) that is a constituting piece of a product; **module** or **cluster** refers to a group of components which ideally are highly interconnected between them but are weakly related with the rest of the product. **Unit** is a generic term to refer to a component which could be seen as component or as module interface depending on the product architecture. An **interface** is a component inside a module connected to a component outside the module. Finally a **product** is a collection of modules and components that interact to perform a common task.

A common tool to model the product is the Design Structure Matrix (DSM) as indicated in [3]. The DSM was introduced by [43] and it captures the interconnections of a product, relating its components in columns and rows of a matrix. Each cell represents a relationship between components whether it means geometrical joints, electrical connections, material flow, etc. Multiple approaches have used this representation of the product to find the optimal module definition by clustering the cells with strong relationship among components (e.g. [53, 52]). This topic will be further explored in section 3.2.2.

When considering a family of products, the architectural analysis is crucial since it is necessary to determine the common structure (platform) for the family. [36] highlights that the main objective when creating a platform is to facilitate the generation of new products by having a common structure from which the new variants can be developed. However, trying to capture the relationships, similarities, or differences among multiple products using multiple DSMs is not an easy task. Each DSM is different in size and content, making it very difficult to relate to other DSMs. In [35], the authors used the Function-Component Matrix (FCM) from [44] to relate the components from different products and found the ones that fulfill identical functions. A FCM is a binary matrix that captures the relationship between the components of a product.
(columns) and the functions they fulfill (rows).

Different criteria have been used in order to determine the components to share: cost considerations [4, 54, 25], Bill of Materials (BOM) [42], product attributes [46], environmental concerns [18, 5, 27], product design variables [16, 17], component interconnections [35], etc. The previous approach to define the components to share is explored in section 3.2.4. Nevertheless, this decision may expose certain information, component, or interface that is sensitive for the manufacturer. [2] generalized the idea of information hiding used in software engineering, proposing that the module definition is a form of abstraction in which the information that is encapsulated or contained in the module becomes hidden for the rest of the system. The only visible information from a module is the one defined by the interfaces, the connections with other components outside the module.

3.2.1 Security and Information Hiding

The security of a product is a concept difficult to quantify. [10] introduced a set of metrics for “Barrier” (the difficulty to extract information out of the product) and “time to reverse engineer” a product based on Ohm’s Law and the model of a RC circuit. Ideally, making a product secure may mean to maximize the barrier to extract the sensitive information; however, the estimation of these parameters depends on skills and resources available to the person performing the information extraction making them very specific and somewhat subjective.

Similarly, the term Anti-Tamper technology has been used in the military context to denote the technologies that prevent unauthorized access to a given information (see [15]). An attempt to design anti-tamper circuits is discussed in [31] in which programmable circuits are dynamically reconfigured to prevent unwanted reconstruction of the functionality of the circuit by reverse engineering techniques. A more general idea is described below.

The term information hiding was introduced by [28] in the context of software engineering, and has been used as the guideline to define the modules of a program. The idea is to decompose a system into modules in order to hide information that is likely to change, or the information that is not supposed to be accessed by the rest of the program.

By isolating or encapsulating certain information, the developer is restricting access to that information to whatever can be inferred by the interfaces (information exchanged to the rest of the system) as discussed by [14]; therefore, the security of a system can be improved if the module boundaries are carefully defined to isolate the sensitive information. The concept is very general and can be used in other contexts (as suggested in [2]) including product development, as a way to handle the complexity of a design.
Usually, the information/component/interface that is sensitive in a product is easy to identify for the
designer; however, trying to protect it from the user or third party manufacturer is not so easy. Very often,
the core elements of a product are highly interconnected to the rest of the system, making it difficult to
enclose completely in a module. Then it becomes crucial to clearly identify and separate the internal (hidden)
resources of a module, from the ones that can be accessed by the rest of the system or user [26].

A simple case is discussed in [19], in which the tradeoff between modularity and security is explored
for Protected Distribution Systems. These systems are used for protection of networks and are custom
made, where applying a modular design for the conduit leads to the exposure of the network at the module
boundaries compromising the security of the entire system. Then, the challenge for the product designers is
to find an optimal component arrangement that preserves the desired protection for those sensitive elements.

3.2.2 Optimal Clustering for Product Architecture

In [53] and [52] the optimal architecture for a product is achieved through the use of the Minimum Description
Length (MDL) principle. This principle promotes the simplest description for any structure; thence, each
component arrangement of a product can be coded by a model in which the simpler the model is, the shorter
the description is. In the end, the idea is to measure the length of the description and find the minimum,
which corresponds to the simplest (optimal) component arrangement for the product.

Both approaches use this same principle in a different manner. [53] used a description of the overall struc-
ture and accounted for two types of mismatch between a model and the actual product: Type I corresponds
to the relationships between components that are left outside the modules defined by the model; Type II
corresponds to the relationships inside the modules that are weak or do not exist (recall that components
inside the modules should be strongly interconnected). The formulation is as follows:

\[
 f_{MDL} = (1 - \alpha - \beta)(n_{cl} \log_2 n_c + \log_2 n_c \sum_{i=1}^{n_{cl}} cl_i)
 + \alpha(S_1(2 \log_2 n_c + 1)) + \beta(S_2(2 \log_2 n_c + 1))
\]  

(3.1)

In this formulation, the first term corresponds to the length of the overall model, while the second and
third terms corresponds to the mismatch length type I and II respectively. \(n_{cl}\) is the number of clusters, \(n_c\)
is the total number of components of the product, \(cl_i\) is the number of elements in cluster \(i\), and \(\alpha\) and \(\beta\)
are weighting factors for the mismatches. The terms \(S_1\) and \(S_2\) account for the mismatches and are defined
as follows:

\[
 S_1 = \sum_{d'_{ij}} = 1(1 - p_{ij}); \quad S_2 = \sum_{d'_{ij}} = 0(p_{ij});
 p_{ij} = \frac{d_{ij} - d_{min}}{d_{max} - d_{min}}; \quad d_{max} = \max_{ij} d_{ij}; \quad d_{min} = \min_{ij} d_{ij};
\]  

(3.2)
where $d_{ij}$ is the entry of the $i^{th}$ row and $j^{th}$ column of the DSM, and $p_{ij}$ is the same entry normalized. The entry of the ideal binary DSM described by the model is $d'_{ij}$, in which there are no mismatches, i.e. all the entries are one inside the clusters and zero outside the clusters.

On the other hand, [52] used a formulation based on the number of links from each component. This formulation allows a representation of each component (unit) in the product as follows:

$$MDL_j^{(u)} = - \log_2 \left( \frac{N_j^{(u)}}{\sum_{k=1}^{N^{(u)}} N_k} \right) \quad (3.3)$$

in which $N_j^{(u)}$ is the number of units connected to unit $j$, and $N^{(u)}$ is the number of units at the level in which the unit $j$ is. In this context, two units are considered to be at the same level if they are inside the same module or outside every module. Since a unit can not only be seen as a component if it is considered inside a module, but also as an interface if it has connections outside the module, the total representation of unit $j$ in product $k$ would be:

$$MDL_j^k = MDL_j^{(c)} + MDL_j^{(o)} = - \sum_{u=\{c,o\}} \log_2 \left( \frac{N_j^{(u)}}{\sum_{k=1}^{N^{(u)}} N_k} \right) \quad (3.4)$$

where $c$ represents components and $o$ interfaces. This abstraction of the representation becomes very useful when analyzing how tightly connected each element of the product is.

### 3.2.3 Enabling Manufacturing Techniques

The idea of “encapsulating” a group of elements seems clear in the context of software engineering; however, when the concept is translated to other fields, many times there are limitations on the manufacturing processes, the compatibility of the materials, etc.

Nevertheless, advances in the manufacturing processes are making it possible to integrate components to a greater extend than ever before. The Micro-Electric-Mechanical Systems (MEMS) have been in the front end of research on how to integrate electrical and mechanical systems at a microscopic scale. Custom made integrated circuits and programmable circuits have allowed to include elements of a circuit that used to be separate components and now are part of the same “chip.” New techniques in injection molding permit the use of in-mold labeling, inserts for elements of different materials, or simply different colors without creating a new part.

All the cases listed above exemplify how the encapsulation of physical elements is not only feasible in many cases, but even a common practice; therefore, the designers can take advantage of these capabilities
to improve the security of their components and products to a greater extent than ever before.

### 3.2.4 Optimal Sharing Decision Making

Many criteria have been used to define the set of components to share; however, when considering product architecture, in [35], the authors used the Impact Metric (IM) and the Function-Component Matrix (FCM) to establish the suitable candidates for sharing. The IM introduced by [32], gives a proxy for the easiness to change a component under a given architecture. The formulation is as follows:

\[
IM^k_j = MDL^k_j CI^k_j
\]  

in which the \( CI^k_j \) corresponds to the Coupling Index (CI) of component \( j \) in product \( k \). The CI was introduced by [20] and the formulation is based in the DSM that contains the sensitivity analysis for the impact of a change in the specifications of the components. This assumes that the components in the columns supply the specification and the components in the rows receive the specifications. Each row and column of the DSM is added giving the CI-Receiving (CI-R) for the rows, and the CI-Supplying (CI-S) for the columns. The total CI for each component is the sum of the CI-S and the CI-R as follows:

\[
CI^k_j = CIS^k_j + CIR^k_j = n^k_c \sum_{i=1} DSM^k(i,j) + n^k_r \sum_{i=1} DSM^k(j,i)
\]  

The IM for each component of each product is an indication of how easy it is to change a particular component. Nevertheless, in order to identify common components across different products, it is necessary to find a common ground. Therefore, the authors used the FCM to match components from different products by the functions they fulfill. The criterion was to find the set of components that fulfill the same functions in a pair of products, and analyze if the sum of IM scores from the components involved, was below a given threshold set by the designer. In this manner, the pairwise evaluation was conducted for all the products in the family and in the end, the set of components to share from all the products was established and stored in a binary decision vector \( Y^k \) for each product \( k \).

In [35], after the decision vectors are established, an optimization problem is solved for each product of the family with a dual objective: on the one hand, finding an optimal clustering of the components making use of the product MDL representation of equation (3.1); meanwhile, the decision vector is multiplied by the IM vector from equation (3.5) selecting the components to include in the objective function, allowing those components to be easily changed in the product. The optimization problem was solved using genetic
algorithms and the chromosome was a binary string that represents a particular component arrangement in clusters for product \( k \), also represented in a binary matrix \( (X^k) \). In this matrix, the columns are the components and the rows are the clusters; therefore, each cell \((i, j)\) indicates if the component \( j \) belong to the cluster \( i \). The objective function was defined as follows:

\[
f(X^k) = f_{MDL}^k + (Y^k)^T IM^k
\]  

Since the IM is dependent on the current architecture of the product, the process was iterative. At each iteration \((t)\) a new component arrangement was achieved for each product, progressively establishing the final list of components to share. The process was repeated until the change in the decision vectors for all the products being considered was below a given tolerance \( \epsilon \):

\[
\sum_{k=1}^{m} \left\| Y^k(t) - Y^k(t-1) \right\|_2^2 \leq \epsilon
\]  

### 3.3 Generating Security Constraints for Optimal Product Family Design

The framework presented in this section builds upon the work presented in [35] (also described in 3.2.4). It incorporates security considerations when selecting the optimal candidates for sharing across multiple products, simultaneously identifying the optimal component arrangement for each individual product.

#### 3.3.1 Defining Sensitive Components

[35] proposed the use of the functional mapping captured in the FCMs to automatically identify the components that could be matched between products based on the functions they perform. Since the FCM for each product is already available (or at least a requirement for the application of the framework), this information can be used to designate the sets of sensitive components relating the functions that are “critical” to the components that are performing the functions in each product and group them in sets based on the same functional mapping.

The main difference with the approach presented in [33] is that the sets of sensitive components are automatically generated using the FCM and designating the “critical” functions, which are independent of the realization of each product. The term *critical* is used to denote the functions that carry most of the intellectual property of the product and therefore should be protected. These functions are easier to identify.
than individual components in each product since those are the functions in which the companies usually invest more resources and have heavy research and development.

The critical functions are designated by means of a binary vector ($CrFun$) for which a one (1) would indicate that the function is critical. The size of the vector is equal to the number of functions considered for the analysis and the number of rows in the FCMs ($n_f$). $CrFun$ is the input required from the designer and corresponds to the nature of the company and where the intellectual property is.

The automated detection of the sensitive components and its clustering in restricted sets is performed (similarly to the functional matching proposed in [35]) by examining the vector $CrFun$ for critical functions in the following manner:

- A vector keeps track of the critical functions evaluated throughout the selection of the restricted sets, denoted as master function-vector ($f-v$);

- To form each restricted set, another vector marks the critical functions related to the matching components, denoted as local $f-v$;

- A different vector is used to register the components related to the critical functions in the local $f-v$ forming each restricted set, this vector is denoted as component-vector ($c-v$); and

- A matrix is formed row by row with the results from the $c-v$ and it is returned at the end of the identification process, in which the rows indicate the restricted sets ($n_{rcl}$) of components for product $k$. This matrix is denoted as secure matrix ($SecM^k$).

The master $f-v$ and the local $f-v$ are binary vectors with $n_f$ elements, while the $c-v$ is a vector with $n_c$ elements. The Secure Matrix ($SecM^k$), is a binary matrix of dimension $n_{rcl}$ by $n_c$. An example of a SecM for a product with eight components is shown in table 3.1, in which components B, C, and D are required to be inside a module. Similarly components F, G, and H are also required to be in a module. The requirement does not imply that the modules should be different from one another, it could be one module that contains both sets but the sets has to be fully contained in the module.

The selection of restricted sets of components is conducted as follows:
Step 1. Initialize *master f-v* with a zero-vector and start by evaluating the first function \((i = 1)\);

Step 2. check if the function is critical, if yes, continue, if not, go to step 14;

Step 3. check if the function has been evaluated before \((master f-v(i) = 1?)\), if yes, go to step 14, if not, continue;

Step 4. mark the *master f-v* with a 1 for the corresponding function, and initialize the *local f-v* and *c-v* with zero vectors,

Step 5. mark the *local f-v* with a 1 for the corresponding function, and set \(t = i\),

Step 6. check if the function should be considered \((local f-v(t) = 1?)\), if yes, continue, if not go to step 11;

Step 7. find components related to the function following the row of the Function-Component Matrix \((FCM)\) corresponding to the function being evaluated until a 1 is found;

Step 8. mark the corresponding component in the *c-v*;

Step 9. find other critical functions related to the component following the column of the corresponding component and mark them in the *local f-v*;

Step 10. return to step 7 until all the components have been evaluated;

Step 11. continue to the next function \((t = t + 1)\) and return to step 6 until all the functions in the *local f-v* have been evaluated;

Step 12. mark the functions marked in the *local f-v* into the *master f-v*;

Step 13. check if the *c-v* is empty, if not create a new row for \(SecM^k\) and mark the components in the *c-v* into the row of the matrix; and

Step 14. continue the evaluation of the next function \((i = i + 1)\) and return to step 2 until all the functions have been evaluated.

In order to achieve this restriction, a new term is added in the MDL formulation. This term accounts for the mismatch of the current architecture to the constraint designated in the SecM for the product. The formulation in equation (3.1) is redefined as follows:

\[
\hat{f}_{MDL} = (1 - \alpha - \beta - \gamma)(n_{cl} \log_2 n_c + \log_2 n_c \sum_{i=1}^{n_{cl}} c_i) \\
+ \alpha(S_1(2 \log_2 n_c + 1)) + \beta(S_2(2 \log_2 n_c + 1)) \\
+ \gamma(S_3(\log_2 n_c + \log_2 n_{rcl} + \log_2 n_{cl}))
\]

\[3.9\]
in which $\gamma$ is a weighting factor, $n_{rcd}$ is the number of restricted sets of components (rows of SecM), and $S_3$ accounts for the mismatches with the restricted sets. The description of this mismatch would be given by the component number, the restricted set number, and the cluster number; therefore, the description length is given by $(\log_2 n_c + \log_2 n_{rcd} + \log_2 n_{cl})$. $S_3$ counts the number of these mismatches in the following manner:

$$S_3 = \sum_{i=1}^{n_{rcd}} \sum_{j=1}^{n_{cl}} \left[\text{SecM}_i.j(e - X_j)^T \min\{\text{SecM}_i.X_j^T, \text{SecM}_i.(e - X_j)^T\}, 1\} \right] + \sum_{i=1}^{n_{rcd}} \left[\text{SecM}_i.e^T (1 - \min\{\sum_{j=1}^{n_{cl}} \text{SecM}_i.X_j^T, 1\})\right]$$

where $\text{SecM}_i$ corresponds to the $i^{th}$ row vector of $\text{SecM}$, $X_j$ the $j^{th}$ row vector of $X$, and $e$ is a unity row vector of the appropriate dimension. The first term accounts for the mismatches between each cluster ($X_j$) and each restricted set ($\text{SecM}_i$), preventing overcounting mismatches when no component in the restricted set is in the cluster. The second term penalizes the restricted sets that are not included in any of the clusters.

This modification to the original MDL formulation finds an optimal clustering for the product while maintaining the constraints defined in $\text{SecM}$ through the optimization process.

### 3.3.2 Component Sharing

The component sharing decision is also affected by the restriction on the sensitive components. In [35], a candidate for sharing gets isolated in a cluster by itself to facilitate the sharing process. Having a sensitive component selected as a candidate would violate the restriction imposed by $\text{SecM}_i$ unless it is selected along with all the other components of the restricted set and it would have to be shared as a module. Therefore, it is necessary to redefine the decision vector $Y^k$, so this aspect is taken into consideration. The reformulation is as follow:

$$Y^k_c(j) = Y^k(j) \prod_{i=1}^{n_{rcd}} (1 - \text{SecM}_i(i,j))$$

$$Y^k_m(l) = \min\{\sum_{i=1}^{n_{rcd}} (1 - \min\{(\text{SecM}_i(i.,)e - X^k(l,.)\text{SecM}_i(i,.)', 1)\}) \in \{1, ..., n_{rcd}\}, j \in \{1, ..., n_c\}, l \in \{1, ..., n_{cl}\}\}$$

where $Y^k_c$ exclude all the restricted components of product $k$ from $Y^k$. $Y^k_m$ indicates which modules contain restricted sets entirely and are suitable candidates according to $Y^k$. The original decision vector $Y^k$ is still obtained by the pairwise comparisons between products, establishing a functional matching of components from different products and evaluating if the sum of the $IM$ score for the components related is below a given threshold. This analysis determines the components that are easy to change under the current architecture,
making those candidates for sharing across the family of products. (Refer to section 2.3.3 for more details.)

The $IM$ is also computed for the modules defined in $X^k$ by treating each cluster as a component, aggregating the external connections, computing the CI for the cluster, and multiply it by the $MDL$ of the module. The result of this calculation is stored in the vector $IM^k_m$.

The subproblem (clustering optimization for each product $k$) is redefined as follows:

$$
\min f^k = \alpha_1 Y_c^k IM^k + \alpha_2 Y_m^k IM_m^k + \hat{f}_{MDL}^k \tag{3.12}
$$

where $\alpha_1$ and $\alpha_2$ are weighting factors. This objective function determines the optimal clustering strategy for the individual products based on the DSM of the product (fixed), the restrictions for sensitive components defined in the SecM (fixed), the modified decision vector passed from the master problem (Component Sharing) as a parameter, and evaluates the chromosome $(X^k)$ during the process of the optimization until finding the optimal clustering strategy.

### 3.3.3 Framework Overview

The framework defined in [35] requires adjustments to accommodate the new criteria involved in the analysis. A new matrix $SecM$ is required for each product, and the objective function in the optimization process requires two important modifications to include the security considerations: First, to incorporate a mismatch between the restricted sets in $SecM$ and the clustering strategy $X$ for the MDL model; and second, to alter the decision vector to exclude the sensitive components that are selected for sharing without all the elements in the corresponding restricted set. Therefore, the overall framework proceeds as follows (Figure 3.1):

Step 1. Construct the product $DSM$ for each variant of the family;

Step 2. Construct the $FCM$ for each product and include all the functions in the family;

Step 3. Define the set of critical functions in the binary vector $CrFun$;

Step 4. Generate the restricted sets of components ($SecM$) for each product;

Step 5. Initialize the modified decision vectors $Y_c^k$ and $Y_m^k$ with zero vectors;

Step 6. Run the optimization for each product separately. Include both the modified $MDL$ representation of the product ($\hat{f}_{MDL}^k$) and the $IM$ of the modified selection of components and modules for sharing ($\alpha_1 Y_c^k IM^k + \alpha_2 Y_m^k IM_m^k$), following equation (3.12) to obtain a new clustering strategy $X^k$, which considers security of the sensitive components and component sharing;
Step 7. Run the component sharing selection algorithm to obtain the decision vectors \( Y \) for each product in the analysis. This algorithm calculates the IM for all the components and performs a functional matching in order to find candidates for sharing with low IM. The selection of the candidates depends on a threshold level for the IM;

Step 8. Obtain the modified decision vectors \( Y^c_k \) and \( Y^m_k \) for each product;

Step 9. Verify if a stopping criterion is met (i.e. change in all the decision vectors below tolerance \( \epsilon \)), if not return to step 6.

In the end, the proposed framework provides the designers with an optimal component arrangement for each product and a set of candidates for components to share across multiple products in considerations of security (sensitive components), product architecture (internal connectivity), and component sharing (family of products).

### 3.4 Illustrative Example

The framework discussed in the previous section is applied to analyzing the HP inkjet cartridges and their architecture for three distinct printers, HP DeskJet 1220C, HP Photosmart Plus, and HP Business Inkjet 2600 (See figure 3.2).

These three printers exemplify the three strategies that HP currently uses for the inkjet cartridge design; the first uses black (BK) and tricolor (cyan (C), magenta (M), and yellow (Y)) cartridges that include the printhead; the second uses individual ink cartridges for each color but the printhead is fixed to the printer; the
Figure 3.2: Printers and cartridge supplies for each printer: a. Printer HP Deskjet 1220C and cartridges HP45 (BK) and 78 (C, M, Y); b. Printer HP Photosmart Plus and cartridges HP564 (BK, Photo BK, C, M, Y); c. Printer HP Business Inkjet 2600 and cartridges HP10 (BK) and 11 (C, M, Y), Printheads HP11 (BK, C, M, Y).

last printer uses individual ink cartridges as well as printheads that are replaceable separately. The supplies for each printer are shown in figure 3.2, as well as diagrams that shows the ink cartridges, printheads and the carriages. Figure shows the direction the compartment moves and the elements that are carried by it, and how the printing media passes by the printheads. These diagrams do not include the body of the printer or any subsystem other than the elements mentioned above, clarifying the difference in the architecture of the three printers.

Each architecture is significantly different and include distinctive features depending on the intended market for each product. The Deskjet printer is intended for basic color printing for a user at home or the office, the Photosmart is directed to the users that require photo printing as well as documents but not at a professional level; lastly, the business inkjet is intended for higher volume of color printing in an office.

In a recent market report, Gartner, Inc. affirms that Printer OEMs could lose more than $13 Billion
to third-party manufacturers of supplies between mid 2009 and mid 2010 [7]. This potential loss puts in perspective the importance for the OEMs to secure their products and prevent third-party manufacturers or remanufacturers to copy their intellectual properties.

### 3.4.1 Application of the framework

**Steps 1 & 2**: The DSMs and FCMs were developed, only for the subsystems related to the ink cartridges and printheads, for the three printers mentioned above. The matrices were developed based on the service manuals and customer manuals for the printers (see [11], [13], and [12] respectively). The DSMs and FCMs are shown in the appendix (See section B.2 for the DSMs and section B.3 for the FCMs).

**Step 3**: The critical functions were defined in CrFun, indicating the functions that are most sensitive for the printing subsystems, independent of the actual implementation in each printer. The selection of the critical functions is explained further in section 3.4.2 and the CrFun vector can be seen in Table 3.2.

**Step 4**: The Secure Matrices (SecM) were generated for each printer, reflecting the most sensitive components in the cartridges and the architecture of each printer. The SecMs can be seen in the Appendix section, along with a summary of the numerical results for each of the printers.

**Step 5**: The modified decision vectors ($Y^k_c$ and $Y^k_m$) were initialized with zero vectors of the appropriate dimension for each product.

**Step 6**: The subproblems were solved using Genetic Algorithms with the modified decision vector for each printer, the product DSM, and the SecM as parameters for the optimization. The optimal clustering strategy for each product was returned as the optimal chromosome ($X^k$) according to equation (3.12), which captures connectivity, component sharing and security restrictions.

**Step 7**: The optimal chromosomes from all products were passed to the master problem as parameters and the component sharing selection algorithm was executed. The threshold value for the IM was set at 1.0 for this example. The decision vectors ($Y^k$) were returned for each product and can be seen at the Appendix.

**Step 8**: The modified decision vectors ($Y^k_c$ and $Y^k_m$) were obtained according to equation (3.11).

**Step 9**: The tolerance for the stopping criteria, in this case, was selected to be zero ($\epsilon = 0$). Having this tolerance, the steps 6 to 8 were repeated for two iterations after which there was no change in any decision vector.

The details of the algorithms can be seen in the Appendix B.1. The optimization of each product architecture is achieved using Genetic Algorithms and the Simple Genetic Algorithm used is based on the class notes and exercises by Prof. Goldberg at UIUC.
3.4.2 Identifying the Critical Functions

In order to establish the security restrictions in all three printers it was necessary to identify the critical functions of the printing subsystem. First, the functional structure was examined designating the flow of information, energy and material through the printing system. The list of functions came from the FCMs since that is a set of functions considered for all printers and they are independent of the physical implementation of the individual products.

![Functional structure and definition of critical path for printing subsystem.](image)

In Figure 3.3 the functional structure reflects how the functions involved in handling the ink and taking each color from the reservoirs to the paper in a precise manner were the most critical functions in this subsystem. Therefore, these functions are designated as critical functions in the vector \( CrFun \) (see Table 3.2), which is later used to generate the secure matrices for each printer \( \text{SecM}^k \).

In the Appendix, Figure B.5 shows how the critical functions are related to the components of the printer HP Photosmart Plus, generating the Secure Matrix \( \text{SecM}^2 \) which is composed of five restricted sets, the first related to the printheads and the remaining four related with the individual ink cartridges. The generation of this \( \text{SecM} \) was as follows:

**Step 1:** \( i = 0 \) and \( \text{master} \ f-v = 0 \) (\( 0 \) is the zero vector of \( n_f \) elements).
Table 3.2: Critical functions for printing subsystem.

| Function # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| CrFun      | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Step 2:** Is function i critical? $CrFun(1) = 1$.

**Step 3:** Is $master\ f-v(i) = 1$? $master\ f-v(1) = 0$, then continue.

**Step 4:** $master\ f-v(1) = 1$, $local\ f-v = \hat{0}$, and $c-v = \hat{0}$.

**Step 5:** $local\ f-v(1) = 1$, $t = i = 1$.

**Step 6:** Is $local\ f-v(t) = 1$? $local\ f-v(1) = 1$, then continue.

**Step 7:** Follow row 1 of $FCM$ until a 1 is found, $FCM(1,6) = 1$.

**Step 8:** $c-v(6) = 1$.

**Step 9:** Other related functions of component 6 which are also critical: functions 13 and 24, then $local\ f-v(13) = 1$, $local\ f-v(24) = 1$.

**Step 10:** Return to step 7 to continue to follow row 1 looking for a 1; since no other 1 is found, continue to step 11.

**Step 11:** Increment $t$ and return to step 6 looking for $local\ f-v(t) = 1$.

**Step 6:** At $t = 13$, $local\ f-v(13) = 1$, then continue.

**Step 7:** Follow row 13 of $FCM$ until a 1 is found, $FCM(13,7) = 1$.

**Step 8:** $c-v(7) = 1$.

**Step 9:** Other related functions to component 7: no other.

**Step 10:** Return to step 7 to look for other components related to function 13, since no other function is found continue to step 11.

**Step 11:** Increment $t$ and return to step 6 looking for $local\ f-v(t) = 1$.

**Step 6:** At $t = 24$, $local\ f-v(24) = 1$, then continue.

**Step 7:** Follow row 24 of $FCM$ until a 1 is found, $FCM(24,8) = 1$.

**Step 8:** $c-v(8) = 1$.

**Step 9:** Other related functions to component 8: no other.
**Step 10:** Return to step 7 to look for other components related to function 24, since no other function is found continue to step 11.

**Step 11:** Increment \( t \) and return to step 6 looking for local \( f-v(t) = 1 \), since no other \( t \) makes local \( f-v(t) = 1 \) then continue to step 12.

**Step 12:** \( \text{master } f-v(13) = 1 \), \( \text{master } f-v(24) = 1 \).

**Step 13:** Since the \( c-v \) is not empty then the \( c-v \) is copied into the first row of \( SecM \), \( SecM(1, 6) = 1 \), \( SecM(1, 7) = 1 \), and \( SecM(1, 8) = 1 \).

**Step 14:** Increment \( i \) and return to step 2 looking for critical functions to generate other secured sets.

Following the same steps four other secured sets were identified for this printer: components 9, 10, and 11; components 12, 13, and 14; components 15, 16, and 17; and components 2, 3, and 4. Just for visualization purposes, the secured sets were rearranged, placing the secured set of components 2, 3, and 4 in the first row of the \( SecM^2 \), instead of the fifth row, and displacing the other rows down. This is why, in Figures B.4 and B.5, the set of components 6, 7, and 8 appear as Set 2, but the order is irrelevant in the application of the framework. In this manner the secured sets are defined for all printers.

In this work it is assumed that, usually, the critical functions in a product are related to the functions that required more research, development, and resources from the company; thus, identifying those functions tends to be a straightforward decision. The analysis performed to establish the set of critical functions for this illustrative example, is an approximation to the company’s perspective on the functional structure of the printing subsystems considered in this paper.

### 3.4.3 Results

The results for the model of the HP Deskjet 1220C are shown in Figure B.4, for the HP Photosmart Plus are in Figure B.4, and for the HP Business Inkjet 2600 are in Figure B.4. The tables include (in addition to the \( SecMs \)) the decision vectors at each iteration \( (Y^k(t)) \), the final modified decision vectors \( (\hat{Y}^k(2)) \), and the final chromosomes \( (X^k(2)) \) with the optimal clustering strategy.

The secure matrices were generated for the critical functions selected in the \( CrFun \) vector, using the individual mapping from functions to components for each printer captured in the \( FCMs \). Most of the secure sets are related to the ink cartridges and the printheads as expected; however, for the Business Inkjet 2600 the ninth set include components 18 and 20 (Ink Delivery System and Ink Pump respectively) which could have been easily overlooked if a manual selection was done instead of the automated generation (see Figure B.4).

The overall process was completed in only two iterations of the master problem (Component Sharing).
The decision vectors reflect the suitable candidates for sharing throughout the process without the restriction on security. These decision vectors are shown to illustrate how sensitive components could become suitable candidates for sharing if no consideration for security is taken. The final modified decision vector indicates the recommended set of components to share in each printer. When the security restrictions are involved in the optimization process, the decision vectors are modified and the constraint imposed with the SecM is followed strictly, which leads to a different component arrangement.

The restricted sets from the SecMs appear preserved in the optimal clusters of each printer (see Figures B.4, B.4, and B.4). One of the components to share is common for the three printers, the Sponge; some other components are shared only between two of the printers. The set of components to share between printers 2 (Photosmart Plus) and 3 (Business Inkjet 2600) include the Carriage Belt, the Belt attach, and the Carriage Motor. These components are not sensitive and are not restricted by the sets defined. However, the ink cartridges are also candidates for sharing between these two printers, which are sensitive components. Sharing these elements is done in modules rather than the individual components; therefore, all the components in the restricted set should be candidates for sharing, which is the case for the ink cartridges.

Cluster 3 of printer 1 does not include any sensitive element or candidate for sharing, but corresponds to an optimal arrangement according to the connectivity of the system; thence, it is preserved by the algorithm.

When the security restrictions were not considered (following strictly the framework in [35]), the results showed a more complicated module definition in the sense that some modules overlap to minimize mismatches type I and II; however, when security is taken into account, the overlap is forbidden by the constraints in SecM or some modules break down leading to a simpler structure but greater mismatches type I and II. The main difference was that the carriage of each printer was integrated with the components that constitute the cartridges or the printheads. (The results of this analysis following [35] are not shown in this paper due to space limitations.) These results confirm how the security considerations affect the architecture of the product, and how the framework presented in this paper, allows the designers to define sets of components that are known to be sensitive and therefore need to be enclosed in a module.

3.5 Conclusion and Future Work

The methodology proposed in this paper takes component/module security into considerations for component sharing and optimal architecture decision making in a family of products. This new security criterion affects significantly the outcome of the optimal component sharing decision making process in two ways. First, it preserves the restricted sets of components together while achieving optimal architecture. This may
involve adding related components to the restricted sets to form a cluster, or even join restricted sets into a bigger cluster, as long as it contributes to achieving optimal architecture. Second, it prevents the sensitive components from becoming available candidates for sharing due to security concerns defined by the manufacturer. Instead, the sensitive components are allowed to be shared only as modules confined within the restricted set(s).

The illustration design example of inkjet printer family shows how the security concern can be preserved while allowing component sharing across multiple products. This example validates the proposed framework by obtaining a different product architecture that considers both the security of the sensitive components and the component sharing among the three printers. The results were compared against the application of a similar framework without the security considerations and not only they were different but the results from the proposed framework reflect much closer the actual implementation by HP and suggests which components could be shared.

It is not clear if the product architecture decisions that HP has made, at least for the three printers considered in this example, were indeed influenced by the security considerations but they certainly make sense. The cartridge of the entry level printer (which is also the most probable to be copied due to high volume sales and high willingness from the user to buy the cheapest option) is the most complex type that they offer, enclosing the printhead as well as the ink reservoir under the same case, making it more difficult to copy. On the other hand, the high-end printer offer simpler cartridges which only contain the ink and a separate cartridge for the printhead. This high-end type of printers is also intended for professional users or at least users with very high concerns for quality and performance, who are less likely to buy third party cartridges that may not meet the performance offered by the OEM.

Future work involves developing new representation with flexible degrees of security, which would capture security sensitive design decisions in a more comprehensive manner. These design decisions involve evaluating the tradeoffs between compromising the security of the system and gaining the advantages from sharing components in a family of products, for which there is not a methodology established in the literature. Similarly, further research is necessary to expand this framework to design products or subsystems that comply with the Anti-Tamper requirements for products or components that are not strictly software or electronics for military applications.
Chapter 4

General Conclusions and Future Work

The frameworks presented in this thesis offer insights into the opportunities by sharing components across different products. In both applications, products as established as printers and digital cameras from companies like Sony and HP, evidence lots of room for improvement in the sense that there could be much more sharing across products in the different families and across the entire portfolio of products. This suggests that companies could adapt the proposed design frameworks to improve their design practices.

This work is a first step into considering the security of the product (or its elements) into the design decision process for a family of products. It is still a challenge to implement the encapsulation of physical elements that is suggested by the second framework but, it is clear that there are some alternatives manufacturing technologies that can make it possible in many areas to a greater extent than ever before. The insights gained with this first approach and the importance of the subject will keep motivating the work on this track.

Having these considerations helps to model, in a more realistic way, the designer’s challenge at the time to define the architecture of multiple products. These types of decisions involve multiple factors and this work only considers three - connectivity, security, and component sharing. The more comprehensive analysis that the designer may be able to make, the more informed decisions he/she may be able to make. Therefore, it will be always a challenge for the research in this field, to provide a model as realistic as possible.

It would be interesting to relate how securing products may have a negative effect on market share if the costumers think that by not having other options (cheaper generic alternative consumable) they are being forced to buy everything (original product and replacement parts) from the OEM and therefore, opt to buy from a different brand. In the case of inkjet printers, it was known that the third-party (re)manufacturers play a significant role (specially at the entry-level printers) and the companies selling the printers below cost suffer a great deal to recover the investment; however, for some customers having the third-party option may be something appealing and would ultimately make them buy printers from that brand. This type of market insights would be of great value for a company trying to decide up to which extent to protect their product or even change their business model.
A possible extension to the frameworks would be to analyze component sharing for dissimilar products, or in other words, different types of products that would still share some functionalities (like camera cell phones and digital cameras). This analysis would uncover new possibilities for a company to save costs and would offer a more complete analysis throughout the company’s product portfolios in the market.

Other areas that require further work is the representation of the relationships among components of multiple products. It would be useful to develop a different way that may facilitate the analysis among multiple products rather than having multiple DSMs which sometimes are difficult to relate.

One important factor that has not been included in this analysis is cost. The frameworks proposed in this thesis have the premise that sharing is something desirable and would reduce manufacturing costs; however, it is necessary to explore the situations in which this applies, and when it is better to have separate designs. The inclusion of cost would also serve to determine the extent of securing sensitive components since it is very likely that encapsulating or integrating sensitive components would increase production costs.
Appendix A

Numerical Results for the Illustrative Example of Chapter 2 (Digital Cameras)

A.1 Algorithms used in the framework application

The algorithms used for the application of this framework is composed of a main program that handles the iterations of the master problem and calls other functions for specific calculations. The subproblem is solved with a simple GA based on the class notes and exercises by Prof. Goldberg at UIUC. The security restrictions are generated from the FCMs and the definition of the critical functions. The components to share are selected based on the IM score and the functions the components fulfill in the product.

A.1.1 Main algorithm for multiple product analysis

mainmp.m

% This is the main algorithm to solve the product architecture for a family of products.

tol=0; % initialize the tolerance and decide to keep iterating
y1=zeros(1,size(prdsm(1),1)); %initial decision vector y1 for product 1
y2=zeros(1,size(prdsm(2),1)); %initial decision vector y2 for product 2
y3=zeros(1,size(prdsm(3),1))); %initial decision vector y3 for product 3

while tol==0
    for p=1:3
        product.id=p;
        product.dsm=prdsm(p); %get the DSM for product p
        product.n=size(product.dsm,1); %number of components in product p

        if p==1 %establish decision vector y for the product in the iteration

        end
    end
end
product.y=y1;
elseif p==2
    product.y=y2;
elseif p==3
    product.y=y3;
end

accept=0;
while accept==0
    product.maxcr=sga(product.y,p); %run the GA for product p
    accept=input('Accept result?(y=1/n=0): '); %Accept or reject the result of GA
end
if p==1 %Assign the result of the GA to the appropriate X matrix
    x1=product.maxcr;
elseif p==2
    x2=product.maxcr;
elseif p==3
    x3=product.maxcr;
end

[y1, y2, y3] = com_sel(x1,x2,x3) %establish the components to share

A.1.2 Simple Genetic Algorithm

( sga.m)

% A Simple Gentic Algorithm (SGA) in Matlab to get the optimum clusters for
% the DSM of product p defined by prdsm(p)
function glmaxchrom = sga(y,p)

maxpop = 100;
maxstring = 30;
% -------------------------------
%         Main Program          
% -------------------------------

gen = 0;
maxeq=0;
glmax=-100000;
glmaxchrom=0;
max1=0;

%initialize the parameters of the GA
% SGA Parameters

global popsize;
popsize = 50; %Enter population size
global maxgen;
maxgen = 900; %Enter max. generations
global pcross;
pcross = 0.8; %Enter crossover probability
global pmutation;
pmutation = 0.01; %Enter mutation probability
global maxncl;
maxncl = input('Enter maximum number of clusters > ');

global n;
n = size(prdsm(p),1);
global lchrom;

lchrom = n*maxncl; %chromosome length
% # of mutations
global nmutation;
nmutation = 0;

% # of crossovers
global ncross;
ncross = 0;

% For tournament selection without replacement
global nextpos;
nextpos = 1;

% population statistics
global maximum;
global minimum;
global avg;
global sumfitness;

randomize();

for j = 1:popsizen
    individual.chrom = 0;
    individual.x = chromgen(maxncl,p);
    individual.chrom = decode1(individual.x,maxncl);
    individual.fitness = objfunc(individual.x,y,p);
    individual.parent1 = 0;
    individual.parent2 = 0;
    individual.xsite = 0;
    oldpop(j) = individual;
end

statistics(oldpop);
while (gen < maxgen) && (maxeq<70)
gen = gen + 1;
generation;
maxchrom = statistics(newpop);
report;
    if maximum == max1
        maxeq=maxeq+1;
    else
        if maximum > glmax
            glmax=maximum;
            glmaxchrom = maxchrom;
        end
        maxeq=0;
        max1=maximum;
    end
oldpop = newpop;
end

A.1.3 Initial chromosome generation

chromgen.m

function x = chromgen(maxncl,p)

n = size(prdsm(p),1);
for i=1:n
    chd=fix((maxncl)*(rand())+1);
    for j=1:maxncl
        if j==chd
            ch(j,i)=1;
        else
            ch(j,i)=0;
    end
end
A.1.4 Population statistics


A.1.5 Tournament Selection with Replacement method


50
ind1 = rnd(1,popsize);
ind2 = rnd(1,popsize);
while ind2 == ind1
    ind2 = rnd(1,popsize);
end
winner = maxid(population(ind1).fitness, population(ind2).fitness, ind1, ind2);
else
    winner = rnd(1,popsize);
    for i = 2:S
        pick = rnd(1,popsize);
        while pick == winner
            pick = rnd(1,popsize);
        end
        winner = maxid(population(pick).fitness, population(winner).fitness, pick, winner);
    end
end

maxid.m

function index = maxid(f1, f2, id1, id2)
    if f1 > f2
        index = id1;
    else
        index = id2;
    end

A.1.6 Uniform crossover method

uniformcrossover.m

function [child1 child2 jcross] = uniform_crossover(parent1, parent2)
    global pcross;
    global ncross;
    global lchrom;
if flip(pcross) == 1
ncross = ncross + 1;
for i = 1:lchrom
if flip(0.5) == 0
p1 = parent1;
p2 = parent2;
else
p1 = parent2;
p2 = parent1;
end

child1(i) = mutation(p1(i));
child2(i) = mutation(p2(i));
end
else
for i = 1:lchrom
child1(i) = mutation(parent1(i));
child2(i) = mutation(parent2(i));
end
end

jcross = -1; % Don't keep track of which bits come from which parent

flip.m

% Returns a 1 with probability p, and a 0 with probability (1-p)
function bit = flip(p)
if p == 1
bit = 1;
elseif p == 0
bit = 0;
else
bit = (random() <= p);
A.1.7 Mutation method

mutation.m

function new_allele_value = mutation(old_allele_value)
global pmutation
global nmutation

if flip(pmutation);
    nmutation = nmutation + 1;
    new_allele_value = ~ old_allele_value;
else
    new_allele_value = old_allele_value;
end

A.1.8 Coding the chromosome

decode1.m

function chrom = decode1(x, nclust)

% Converts from a matrix to a binary string
n = size(x,2);

k = 1;
for i = 1:nclust
    for j = 1:n
        chrom(k) = x(i,j);
        k = k + 1;
    end
end

A.1.9 Decoding the chromosome

decode.m
function x = decode(chrom, nclust)
% converts from a binary string to a matrix
n=size(chrom,2)/nclust;

k=1;
for i=1:nclust
    for j=1:n
        x(i,j)= chrom(k);
        k=k+1;
    end
end

A.1.10 Objective function calculation

objfcn.m

function fitness = objfunc(x,y,p)
%Calculates the objective function for a chromosome x, matrix y denotes the
%IM of components to include in the obj. function, and p is the product

alpha=1/6;
beta=1/2;

DSM=prdsm(p);
%nclust: max number of clusters | n: number of components
[nclust,n]=size(x);

%Vector with number of components per cluster
cl=x*ones(n,1);

%nc %number of non-empty clusters
nc=0;
for i=1:nclust
    if cl(i)~=0

nc=nc+1;
end
end

%sum of components per clusters
acumcl=ones(1,nclust)*cl;

% DSM1 % Ideal DSM according to current clustering
for i=1:n
    for j=1:n
        if i~=j
            for k=1:nclust
                if (x(k,i)==1)&&(x(k,j)==1)
                    DSM1(i,j)=1;
                end
            end
        else
            DSM1(i,j)=0;
        end
    end
end

s1=0; s2=0;
% calculate mismatches for non-binary DSM
dmin=min(min(DSM));
dmax=max(max(DSM));
for i=1:n
    for j=1:n
        p(i,j)=(DSM(i,j)-dmin)/(dmax-dmin);
    end
end

for i=1:n
for j=1:n
    if (DSM1(i,j)==1)
        s1=s1+(1-p(i,j));
    end
    if (DSM1(i,j)==0)
        s2=s2+p(i,j);
    end
end
end

%calculate CI
CI=(DSM*ones(n,1))+(ones(1,n)*DSM)';

%calculate MDL
MDL=zeros(n,1); %Vector for the MDL representation of each component
ninl=zeros(n,1); %Vector with number of internal links per component
nexl=zeros(n,1); %Vector with number of external links per component
for i=1:n %Account for external links
    for j=1:n
        if (DSM1(i,j)==0)&&(DSM(i,j)~=0)%external
            nexl(i)=nexl(i)+1;
        end
        if (DSM1(j,i)==0)&&(DSM(j,i)~=0)%external
            nexl(i)=nexl(i)+1;
        end
    end
end

%creating a binary DSM
for i=1:n
    for j=1:n
        if DSM(i,j)~=0
            if DSM1(i,j)==1
                DSM1(i,j)=0;
            end
            if DSM1(j,i)==1
                DSM1(j,i)=0;
            end
        end
    end
end
BDSM(i,j)=1;
else
  BDSM(i,j)=0;
end
end
end

nlpc=zeros(nclust,1); %number of links per cluster
nipc=zeros(nclust,1); %number of interfaces per cluster
for i=1:nclust
  for j=1:n
    if x(i,j)==1
      ninl(j)=(x(i,:)*BDSM(:,j))+(BDSM(j,:)*(x(i,:)'));
    end
  end
  nlpc(i)=x(i,:)*ninl; %links per cluster
  nipc(i)=x(i,:)*nexl; %interfaces per cluster
  for j=1:n %MDL for the elements in at least one cluster
    if x(i,j)==1 && nlpc(i)~=0 && ninl(j)~=0
      MDL(j)=MDL(j)+(log2(nlpc(i)/ninl(j))); %MDL
    end
    if x(i,j)==1 && nipc(i)~=0 && nexl(j)~=0
      MDL(j)=MDL(j)+(log2(nipc(i)/nexl(j))); %MDL
    end
  end
end
end

cwcl=ones(1,nclust)*x; %component without clusters
lml=ones(1,nclust)*nipc; %links at the main level
for i=1:n
  if cwcl(i)==0
    lml=lml+nexl(i);
for i=1:n %MDL for the elements that don't belong to any cluster
    if cwcl(i)==0
        MDL(i)=log2(lml/(nexl(i)));
    end
end

for i=1:n
    IM(i)=CI(i)*MDL(i);
end

%calculate fitness for IM
fitness = -(y*IM')-((1-alpha-beta)*(nc*log2(n)+(log2(n)*acumcl))
              +(alpha*s1*(2*log2(n)+1))+(beta*s2*(2*log2(n)+1)));
y2=zeros(1,n2);
y3=zeros(1,n3);

IM1=cal_IM(x1,1);
IM2=cal_IM(x2,2);
IM3=cal_IM(x3,3);

max_im1=max(IM1); % getting the IM max and min for each product
min_im1=min(IM1);
max_im2=max(IM2);
min_im2=min(IM2);
max_im3=max(IM3);
min_im3=min(IM3);

threshold=0.5; % defining the threshold

fcn=zeros(1,f);

for i=1:f % comparison between product 1 and 2
    if fcn(i)==0
        fcn(i)=1;
        temp_fcn=zeros(1,f);
        com1=zeros(1,n1);
        com2=zeros(1,n2);
        temp_fcn(i)=1;
        for t=i:f
            if temp_fcn(t)==1;
                for j=1:n1 % check the row of RM1
                    if RM1(t,j)~0
                        com1(j)=1;
                    
                for k=1:f % check the column of the RM1
                    if RM1(k,j)~0

59
temp_fcn(k)=1;
end
end
end
end

for j=1:n2 %check the row of RM2
    if RM2(t,j)^=0
        com2(j)=1;
        for k=1:f %check the column of the RM2
            if RM2(k,j)^=0
                temp_fcn(k)=1;
            end
        end
    end
end
end
end

end

if ((IM1*com1')<(min_im1+threshold*(max_im1-min_im1))&&
((IM2*com2')<(min_im2+threshold*(max_im2-min_im2)))
    for t=1:n1
        if com1(t)==1
            y1(t)=1;
        end
    end
    for t=1:n2
        if com2(t)==1
            y2(t)=1;
        end
    end
end

for t=1:f %pass temp fcn vector to fcn vector to keep track of fcns checked
    if temp_fcn(t)==1

60
fcn(t)=1;
end
end
end
end

fcn=zeros(1,f);

for i=1:f % comparison between product 1 and 3
    if fcn(i)==0
        fcn(i)=1;
        temp_fcn=zeros(1,f);
        com1=zeros(1,n1);
        com3=zeros(1,n3);
        temp_fcn(i)=1;
        for t=i:f
            if temp_fcn(t)==1;
                for j=1:n1 % check the row of RM1
                    if RM1(t,j)~=0
                        com1(j)=1;
                        for k=1:f % check the column of the RM1
                            if RM1(k,j)~=0
                                temp_fcn(k)=1;
                            end
                        end
                    end
                end
            end
        end
        for j=1:n3 % check the row of RM3
            if RM3(t,j)~=0
                com3(j)=1;
                for k=1:f % check the column of the RM3
                    if RM3(k,j)~=0
                        temp_fcn(k)=1;
                    end
                end
            end
        end
    end
end
temp_fcn(k)=1;
end
end
end
end
end
end
end

if ((IM1*com1')<(min_im1+threshold*(max_im1-min_im1)))&&
((IM3*com3')<(min_im3+threshold*(max_im3-min_im3)))
for t=1:n1
if com1(t)==1
y1(t)=1;
end
end
for t=1:n3
if com3(t)==1
y3(t)=1;
end
end
end
for t=1:f %pass temp fcn vector to fcn vector to keep track of fcns checked
if temp_fcn(t)==1
fcn(t)=1;
end
end
end
end

fcn=zeros(1,f);

for i=1:f % comparison between product 2 and 3
if fcn(i)==0
fcn(i)=1;
temp_fcn=zeros(1,f);
com3=zeros(1,n3);
com2=zeros(1,n2);
temp_fcn(i)=1;
for t=i:f
    if temp_fcn(t)==1;
        for j=1:n3 %check the row of RM3
            if RM3(t,j)~0
                com3(j)=1;
                for k=1:f %check the column of the RM3
                    if RM3(k,j)~0
                        temp_fcn(k)=1;
                    end
                end
            end
        end
        for j=1:n2 %check the row of RM2
            if RM2(t,j)~0
                com2(j)=1;
                for k=1:f %check the column of the RM2
                    if RM2(k,j)~0
                        temp_fcn(k)=1;
                    end
                end
            end
        end
    end
    end
end
if ((IM3*com3')<(min_im1+threshold*(max_im3-min_im3)))&&
((IM2*com2')<(min_im2+threshold*(max_im2-min_im2))
    for t=1:n3
end
if com3(t)==1
    y3(t)=1;
end
end
for t=1:n2
    if com2(t)==1
        y2(t)=1;
    end
end
end
for t=1:f %pass temp fcn vector to fcn vector to keep track of fcns checked
    if temp_fcn(t)==1
        fcn(t)=1;
    end
end
end
end

A.1.12 Calculating IM for components

calIM.m

function IM = cal_IM(x,p)

%Calculates the IM for a chromosome x

DSM=prdsm(p);
%nclust: max number of clusters | n: number of components
[nclust,n]=size(x);

%Vector with number of components per cluster
cl=x*ones(n,1);

%nc  %number of non-empty clusters
nc=0;
for i=1:nclust
    if cl(i)~=0
        nc=nc+1;
    end
end

%sum of components per clusters
acumcl=ones(1,nclust)*cl;

%DSM1 %Ideal DSM according to current clustering
for i=1:n
    for j=1:n
        if i~=j
            for k=1:nclust
                if (x(k,i)==1)&&(x(k,j)==1)
                    DSM1(i,j)=1;
                end
            end
        else
            DSM1(i,j)=0;
        end
    end
end

%calculate CI
CI=(DSM*ones(n,1))+(ones(1,n)*DSM)';

%calculate MDL
MDL=zeros(n,1); %Vector for the MDL representation of each component
ninl=zeros(n,1); %Vector with number of internal links per component
nexl=zeros(n,1); %Vector with number of external links per component
for i=1:n %Account for external links
    for j=1:n

if (DSM1(i,j)==0)&&(DSM(i,j)~=0)\%external
    nexl(i)=nexl(i)+1;
end
if (DSM1(j,i)==0)&&(DSM(j,i)~=0)\%external
    nexl(i)=nexl(i)+1;
end
end
end

\%creating a binary DSM
for i=1:n
    for j=1:n
        if DSM(i,j)~=0
            BDSM(i,j)=1;
        else
            BDSM(i,j)=0;
        end
    end
end

nlpc=zeros(nclust,1); \%number of links per cluster
nipc=zeros(nclust,1); \%number of interfaces per cluster
for i=1:nclust
    for j=1:n
        if x(i,j)==1
            ninl(j)=(x(i,:)*BDSM(:,j))+(BDSM(j,:)*(x(i,:)'));
        end
    end
    nlpc(i)=x(i,:)*ninl; \%links per cluster
    nipc(i)=x(i,:)*nexl; \%interfaces per cluster
    for j=1:n \%MDL for the elements in at least one cluster
        if x(i,j)==1 && nlpc(i)~=0 && ninl(j)~=0
            66
        end
    end
end
MDL(j)=MDL(j)+(log2(nlpc(i)/ninl(j))); \%MDL
end
if x(i,j)==1 && nipc(i)~=0 && nexl(j)~=0
    MDL(j)=MDL(j)+(log2(nipc(i)/nexl(j))); \%MDL
end
end
end
cwcl=ones(1,nclust)*x; \%component without clusters
lml=ones(1,nclust)*nicpc; \%links at the main level
for i=1:n
    if cwcl(i)==0
        lml=lml+nexl(i);
    end
end
for i=1:n \%MDL for the elements that don't belong to any cluster
    if cwcl(i)==0
        MDL(i)=log2(lml/(nexl(i)));
    end
end
for i=1:n
    IM(i)=CI(i)*MDL(i);
end
### A.2 Product DSM for Digital Cameras

![DSM for Digital Camera Sony DSC-T30](image)

*Figure A.1: DSM for Digital Camera Sony DSC-T30.*
Figure A.2: DSM for Digital Camera Sony DSC-S730.
Figure A.3: DSM for Digital Camera Sony DSC-W100.
### A.3 Function-Component Matrices for Digital Cameras

![Figure A.4: FCM for Digital Camera Sony DSC-T30.](image)

---

71
Figure A.5: FCM for Digital Camera Sony DSC-S730.
| Feature      | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Flash        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Speaker      | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mic          | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SD card slot | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Shutter      | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| LCD screen   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Lens         | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Tripod mounting point | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Figure A.6: FCM for Digital Camera Sony DSC-W100. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

73
A.4 Optimum clustering and component sharing candidates for Digital Cameras

<table>
<thead>
<tr>
<th>Component #</th>
<th>Buttons</th>
<th>Control Switch</th>
<th>CCD</th>
<th>Lens Sub-ASM</th>
<th>Flash</th>
<th>Capacitor</th>
<th>Flash PCB</th>
<th>LCD Plate</th>
<th>LCD</th>
<th>LCD PCB</th>
<th>Rear Cabinet</th>
<th>Cabinet Front</th>
<th>Main Body</th>
<th>Plate</th>
<th>Flex PCB</th>
<th>Main PCB</th>
<th>CCD Holder</th>
<th>Filters</th>
<th>Lens Cover</th>
<th>LCD Window</th>
<th>Insulating sheet</th>
<th>Battery Lid</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSC-T30</td>
<td>6 7 12 14 17 19 20 21 3 4 5 9 2 23 15 16 18 11 13 1 8 10 22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buttons</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Control Switch</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CCD</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lens Sub-ASM</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Speaker</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Flash</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Capacitor</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Flash PCB</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LCD Plate</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LCD</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LCD PCB</td>
<td>5</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rear Cabinet</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cabinet Front</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Main Body</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Plate</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Flex PCB</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Main PCB</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CCD Holder</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Filters</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lens Cover</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LCD Window</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Insulating sheet</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Battery Lid</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Optimum Clusters with Component Sharing: (3, 4, 5), (2, 9, 23), (15, 16, 18, 23)
Components to share: (6), (7), (12), (14), (17), (19), (20), (21)
Optimum Clusters (following Yu et al. 2007): (3, 4, 5), (5, 6, 7, 9), (2, 9, 14, 23), (12, 14, 18), (15, 16, 18, 23)

Figure A.7: Optimum DSM for Digital Camera Sony DSC-T30.
![Optimum DSM for Digital Camera Sony DSC-S730](image)

**Figure A.8:** Optimum DSM for Digital Camera Sony DSC-S730.
Figure A.9: Optimum DSM for Digital Camera Sony DSC-W100.

Last 8 columns are excluded since the DSM is symmetric and all the elements in the diagonal block are zero.
Appendix B

Numerical Results for the Illustrative Example of Chapter 3 (Inkjet Printers and Cartridges)

B.1 Algorithms used in the framework application

The algorithms used for the application of this framework is composed of a main program that handles the iterations of the master problem and calls other functions for specific calculations. The subproblem is solved with a simple GA based on the class notes and exercises by Prof. Goldberg at UIUC. The security restrictions are generated from the FCMs and the definition of the critical functions. The components to share are selected based on the IM score and the functions the components fulfill in the product.

B.1.1 Main algorithm for multiple products analysis

mainmp.m

% This is the main algorithm to find a product architecture that facilitate
% the security of the components in "secm" for multiple products

tol=0; % initialize the tolerance and decide to keep iterating
y1=zeros(1,size(prdsm(1),1)); %initial decision vector y1 for product 1
y2=zeros(1,size(prdsm(2),1)); %initial decision vector y2 for product 2
y3=zeros(1,size(prdsm(3),1)); %initial decision vector y3 for product 3

while tol==0
    for p=1:3
        product.id=p;
        product.dsm=prdsm(p); %get the DSM for product p
        product.n=size(product.dsm,1); %number of components in product p
        product.secm=genSecM(p); %get the SecM for product p
    end
end

77
if p==1 %establish decision vector y for the product in the iteration
    product.y=y1;
elseif p==2
    product.y=y2;
elseif p==3
    product.y=y3;
end

accept=0;
while accept==0
    product.maxcr=sga(product.y,p); %Run the GA for product p
    accept=input('Accept result?(y=1/n=0): '); %Accept or reject the result of GA
end
if p==1 %Assign the result of the GA to the appropriate X matrix
    x1=product.maxcr;
elseif p==2
    x2=product.maxcr;
elseif p==3
    x3=product.maxcr;
end
end
[y1, y2, y3] = com-sel(x1,x2,x3) %Establish the components to share
tol=input('Stop iterations?(y=1/n=0): '); %Decide to continue with the iterations
end

B.1.2 Simple Genetic Algorithm

sga.m

% A Simple Genetic Algorithm (SGA) in Matlab to get the optimum clusters for
% the DSM of product p defined by prdsm(p)
function glmaxchrom = sga(y,p)
%clear;
%clc;

%set parameters for the GA
maxpop = 100;
maxstring = 30;
gen = 0;
maxeq=0;
glmax=-100000;
glmaxchrom=0;
max1=0;

%initialize the parameters of the GA
% SGA Parameters

global popsize;
popsize = 50; %Enter population size
global maxgen;
maxgen = 900; %Enter max. generations
global pcross;
pcross = 0.8; %Enter crossover probability
global pmutation;
pmutation = 0.01; %Enter mutation probability
global maxncl;
maxncl = input('Enter maximum number of clusters > ');

global n;
n = size(prdsm(p),1);
global lchrom;
lchrom = n*maxncl; %chromosome length
% # of mutations
global nmutation;
nmutation = 0;

% # of crossovers
global ncross;
cross = 0;

% For tournament selection without replacement
global nextpos;
nextpos = 1;

% population statistics
global maximum;
global minimum;
global avg;
global sumfitness;

randomize();

%initialize population
for j = 1:popsiz
    individual.chrom = 0;
    individual.x = chromgen(maxncl,p);
    individual.chrom = decode1(individual.x,maxncl);
    individual.fitness = objfunc(individual.x,y,p);
    individual.parent1 = 0;
    individual.parent2 = 0;
    individual.xsite = 0;
    oldpop(j) = individual;
end
statistics(oldpop);

while (gen < maxgen) && (maxeq<70)
gen = gen + 1;

SELECTION_CHOICE = 2;  % (1=>Tournament Selection w/o Replacement,
                        % 2=>Tournament Selection with Replacement, 3 => Proportionate Selection)
CROSSOVER_CHOICE = 3;  % (1=>Single Pt XO,  2=>Two pt XO, 3 => Uniform XO)

j = 1;
while j <= popsize

% ------------ Selection -----------------------
if SELECTION_CHOICE == 1
mate1 = tswor_selection(oldpop, 2);
mate2 = tswor_selection(oldpop, 2);
elseif SELECTION_CHOICE == 2
mate1 = tswr_selection(oldpop, 2);
mate2 = tswr_selection(oldpop, 2);
elseif SELECTION_CHOICE == 3
mate1 = proportionate_selection(popsize, sumfitness, oldpop);
mate2 = proportionate_selection(popsize, sumfitness, oldpop);
end

% ------------ Crossover ------------------------
if CROSSOVER_CHOICE == 1
[newpop(j).chrom newpop(j+1).chrom jcross] =
    single_pt_crossover(oldpop(mate1).chrom, oldpop(mate2).chrom);
elseif CROSSOVER_CHOICE == 2
newpop(j).chrom newpop(j+1).chrom jcross] =
    two_pt_crossover(oldpop(mate1).chrom, oldpop(mate2).chrom);

elseif CROSSOVER_CHOICE == 3
newpop(j).chrom newpop(j+1).chrom jcross] =
    uniform_crossover(oldpop(mate1).chrom, oldpop(mate2).chrom);
end

% Decode string, evaluate fitness, & record parents on both children
newpop(j).x = decode(newpop(j).chrom, maxncl);
newpop(j).fitness = objfunc(newpop(j).x, y, p);
newpop(j).parent1 = mate1;
newpop(j).parent2 = mate2;
newpop(j).xsite = jcross;

newpop(j+1).x = decode(newpop(j+1).chrom, maxncl);
newpop(j+1).fitness = objfunc(newpop(j+1).x, y, p);
newpop(j+1).parent1 = mate1;
newpop(j+1).parent2 = mate2;
newpop(j+1).xsite = jcross;

% elitism 1
    if newpop(j).fitness < oldpop(j).fitness
        newpop(j)=oldpop(j);
    end
    if newpop(j+1).fitness < oldpop(j+1).fitness
        newpop(j+1)=oldpop(j+1);
    end

% Increment population index
j = j + 2;
end
maxchrom = statistics(newpop);
    if maximum == max1
        maxeq=maxeq+1;
    else
        if maximum > glmax
            glmax=maximum;
            glmaxchrom = maxchrom;
        end
        maxeq=0;
        max1=maximum;
    end
oldpop = newpop;
end

B.1.3 Initial chromosome generation

chromgen.m

function x = chromgen(maxncl,p)

n = size(prdsm(p),1);
for i=1:n
    chd=fix((maxncl)*(rand())+1);
    for j=1:maxncl
        if j==chd
            ch(j,i)=1;
        else
            ch(j,i)=0;
        end
    end
end
x=ch;
B.1.4 Population statistics

\texttt{statistics.m}

\begin{verbatim}
function maxchrom = statistics(pop)
    \% Calculates population statistics

    global sumfitness;
    global maximum;
    \%global maxchrom;
    global minimum;
    global avg;

    sumfitness = sum([pop.fitness]);
    [maximum,i] = max([pop.fitness]);
    maxchrom = pop(i).x;
    minimum = min([pop.fitness]);
    avg = mean([pop.fitness]);
\end{verbatim}

B.1.5 Tournament Selection with Replacement method

\texttt{tswrselection.m}

\begin{verbatim}
function winner = tswr_selection(population, S)
    global popsize;

    if S == 2
        ind1 = rnd(1,popsize);
        ind2 = rnd(1,popsize);
        while ind2 == ind1
            ind2 = rnd(1,popsize);
        end
        winner = maxid(population(ind1).fitness, population(ind2).fitness, ind1, ind2);
    \end{verbatim}
else
winner = rnd(1,popsize);
for i = 2:S
pick = rnd(1,popsize);
while pick == winner
pick = rnd(1,popsize);
end
winner = maxid(population(pick).fitness, population(winner).fitness, pick, winner);
end
end

maxid.m

function index = maxid(f1, f2, id1, id2)
if f1 > f2
index = id1;
else
index = id2;
end

B.1.6 Uniform crossover method

uniformcrossover.m

function [child1 child2 jcross] = uniform_crossover(parent1, parent2)
global pcross;
global ncross;
global lchrom;

if flip(pcross) == 1
ncross = ncross + 1;
for i = 1:lchrom
if flip(0.5) == 0
p1 = parent1;
p2 = parent2;
else
  p1 = parent2;
  p2 = parent1;
end

child1(i) = mutation(p1(i));
child2(i) = mutation(p2(i));
end
else
  for i = 1:lchrom
    child1(i) = mutation(parent1(i));
    child2(i) = mutation(parent2(i));
  end
end

jcross = -1; % Don't keep track of which bits come from which parent

flip.m

% Returns a 1 with probability p, and a 0 with probability (1-p)
function bit = flip(p)
if p == 1
  bit = 1;
elseif p == 0
  bit = 0;
else
  bit = (random() <= p);
end

B.1.7 Mutation method

mutation.m

function new_allele_value = mutation(old_allele_value)
  global pmutation
global nmutation

if flip(pmutation);
    nmutation = nmutation + 1;
    new_allele_value = ~ old_allele_value;
else
    new_allele_value = old_allele_value;
end

B.1.8 Coding the chromosome
decod1.m

function chrom = decode1(x, nclust)
% Converts from a matrix to a binary string
n=size(x,2);

k=1;
for i=1:nclust
    for j=1:n
        chrom(k)=x(i,j);
        k=k+1;
    end
end

B.1.9 Decoding the chromosome
decode.m

function x = decode(chrom, nclust)
% Converts from a binary string to a matrix
n=size(chrom,2)/nclust;

k=1;
for i=1:nclust
for j=1:n
    x(i,j)= chrom(k);
    k=k+1;
end
end

B.1.10 Objective function calculation

objfcn.m

function fitness = objfunc(x,y,p)
%Calculates the objective function for a chromosome x, matrix y denotes the
%components candidates to share, and p is the product

alpha=1/4;
beta=1/8;
gamma=3/8;

DSM=prdsm(p);
secm=genSecM(p);

%nclust: max number of clusters | n: number of components
[nclust,n]=size(x);

%Vector with number of components per cluster
cl=x*ones(n,1);

%nc: number of non-empty clusters
nc=0;
for i=1:nclust
    if cl(i)>0
        nc=nc+1;
    end
end
end

%sum of components per clusters
acumcl=ones(1,nclust)*cl;

%DSM1 %Ideal DSM according to current clustering
for i=1:n
    for j=1:n
        if i~=j
            for k=1:nclust
                if (x(k,i)==1)&&(x(k,j)==1)
                    DSM1(i,j)=1;
                end
            end
        else
            DSM1(i,j)=0;
        end
    end
end

s1=0; s2=0; s3=0;
%calculate mismatches 1 and 2 for non-binary DSM
dmin=min(min(DSM));
dmax=max(max(DSM));
for i=1:n
    for j=1:n
        pr(i,j)=(DSM(i,j)-dmin)/(dmax-dmin);
    end
end

for i=1:n
    for j=1:n
        if (DSM1(i,j)==1)
\[ s_1 = s_1 + (1 - p_{r(i,j)}) \]
end

if (DSM1(i,j) == 0)
\[ s_2 = s_2 + p_{r(i,j)} \]
end
end
end

% calculate mismatch 3 of restricted modules
nrc = size(secm,1); % number of restricted clusters

for i = 1:nrc
    cstr = secm(i,:) * ones(n,1);
    k = 0;
    for j = 1:nclust
        auxv = secm(i,:) * x(j,:);
        if auxv > 0 && auxv < cstr
            s3 = s3 + (cstr - auxv);
        elseif auxv == 0
            k = k + 1;
            if k == nclust
                s3 = s3 + cstr;
            end
        end
    end
end

% Calculate IM
[IMc, IMm] = cal_IM2(x, p);

% modified fitness to include required modules
modyc = y; % modyc identify components to share not in the secm
for i=1:nrc
    %cstr=secm(i,:)*ones(n,1);
    auxv=secm(i,:)*modyc';
    if auxv>0
        for k=1:n
            if secm(i,k)==1
                modyc(k)=0; % if y include components from the req. clusters then ignore them
            end
        end
    end
end

modym=zeros(1,nclust); %modym identify modules to share also restricted
for j=1:nclust
    for i=1:nrc
        auxv=x(j,:)*secm(i,:);*
        if auxv==(secm(i,:)*ones(n,1)) && (y*secm(i,:))=(secm(i,:)*ones(n,1))
            modym(j)=1;
        else
            % modym(j)=0;
        end
    end
end

% Calculate objective function: first components to share, then restricted modules to share, % then MDL of overall structure + mismatches type 1, 2, and 3
fitness = -(modyc*IMc') -0.001*(modym*IMm') -((1-alpha-beta-gamma)*(nc*log2(n)+(log2(n)*acumcl)) +(alpha*s1*(2*log2(n)+1)) +(beta*s2*(2*log2(n)+1)) +(gamma*s3*(log2(nrc)+log2(n)+log2(nc))));

B.1.11 Generating the security restriction matrix (SecM)
genSecM.m
function SecM = genSecM(p)

% Denote the functions in the critical path

SecF=[1 1 1 0 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1];

% Functions
% 1 Store K ink
% 2 Store C ink
% 3 Store M ink
% 4 Store Y ink
% 5 Move carriage along paper
% 6 Deliver K Ink precisely into paper
% 7 Deliver C Ink precisely into paper
% 8 Deliver M Ink precisely into paper
% 9 Deliver Y Ink precisely into paper
% 10 Send/Receive signals
% 11 Provide support for movement
% 12 Provide support for cartridges
% 13 Protect K ink from environment
% 14 Protect C ink from environment
% 15 Protect M ink from environment
% 16 Protect Y ink from environment
% 17 Protect K printhead from environment
% 18 Protect C printhead from environment
% 19 Protect M printhead from environment
% 20 Protect Y printhead from environment
% 21 Clean printheads
% 22 Prevent ink spills
% 23 Absorb extra ink
% 24 Sense K ink level and cartridge info
% 25 Sense C ink level and cartridge info
% 26 Sense M ink level and cartridge info
% 27 Sense Y ink level and cartridge info
% 28 Transport ink from reservoir to printhead

FCM=prrelm(p); % get the FCM for product p
[nf,nc]=size(FCM); % get the number of functions and components

q=1;
SecM=zeros(1,nc); % initialize output as a zero vector
fcn=zeros(1,nf); % initialize the vector to keep track of functions (master f-v)

for i=1:nf
    if SecF(i)==1 % evaluate if the function is critical
        if fcn(i)==0 % has it been evaluated before?
            auxf=zeros(1,nf); % auxiliary vector to group functions (local f-v)
            auxc=zeros(1,nc); % auxiliary vector to group components (c-v)
            fcn(i)=1;
            auxf(i)=1;
            for t=i:nf % evaluate if other functions should be considered
                if auxf(t)==1
                    for j=1:nc % find the components related to those functions
                        if FCM(t,j)==1
                            auxc(j)=1;
                            for k=1:nf % find other critical functions related to that component
                                if (FCM(k,j)==1)&&(SecF(k)==1)
                                    auxf(k)=1;
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
for t=1:nf %pass temp fcn vector to fcn vector to keep track of fcns checked
    if auxf(t)==1
        fcn(t)=1;
    end
end
for t=1:nc
    if auxc(t)==1
        SecM(q,t)=1;
    end
end
q=q+1;
end
end

B.1.12 Selecting components to share

comsel.m

function [y1, y2, y3] = com_sel(x1,x2,x3)

n1=size(x1,2); %number of components of product 1
n2=size(x2,2); %number of components of product 2
n3=size(x3,2); %number of components of product 3

FCM1=prrelm(1); %Function Component Matrices
FCM2=prrelm(2);
FCM3=prrelm(3);

f=size(FCM1,1); % number of functions
y1=zeros(1,n1);
y2=zeros(1,n2);
y3=zeros(1,n3);

IM1=cal.IM2(x1,1);
IM2=cal.IM2(x2,2);
IM3=cal.IM2(x3,3);

max_im1=max(IM1); % getting the IM max and min for each product
min_im1=min(IM1);
max_im2=max(IM2);
min_im2=min(IM2);
max_im3=max(IM3);
min_im3=min(IM3);

NIM1=(IM1-min_im1).*(1/(max_im1-min_im1));
NIM2=(IM2-min_im2).*(1/(max_im2-min_im2));
NIM3=(IM3-min_im3).*(1/(max_im3-min_im3));

threshold=1.1;

fcn=zeros(1,f);

for i=1:f % comparison between product 1 and 2
    if fcn(i)==0
        fcn(i)=1;
        temp_fcn=zeros(1,f);
        com1=zeros(1,n1);
        com2=zeros(1,n2);
        temp_fcn(i)=1;
        for t=i:f
            temp_fcn(t)=1;
            fcn(t)=1;
            com1(t)=1;
            com2(t)=1;
        end
    end
end
if temp_fcn(t)==1;
    for j=1:n1 %check the row of FCM1
        if FCM1(t,j)<>0
            com1(j)=1;
            for k=1:f %check the column of the FCM1
                if FCM1(k,j)<>0
                    temp_fcn(k)=1;
                end
            end
        end
    end
end
for j=1:n2 %check the row of FCM2
    if FCM2(t,j)<>0
        com2(j)=1;
        for k=1:f %check the column of the FCM2
            if FCM2(k,j)<>0
                temp_fcn(k)=1;
            end
        end
    end
end
end
end

if ((NIM1*com1')<threshold)&&((NIM2*com2')<threshold)
    for t=1:n1
        if com1(t)==1
            y1(t)=1;
        end
    end
    for t=1:n2
        if com2(t)==1
            y2(t)=1;
        end
    end
end
end
y2(t)=1;
end
end
end
for t=1:f %pass temp fcn vector to fcn vector to keep track of fcns checked
    if temp_fcn(t)==1
        fcn(t)=1;
    end
end
end

fcn=zeros(1,f);

for i=1:f % comparison between product 1 and 3
    if fcn(i)==0
        fcn(i)=1;
        temp_fcn=zeros(1,f);
        com1=zeros(1,n1);
        com3=zeros(1,n3);
        temp_fcn(i)=1;
        for t=i:f
            if temp_fcn(t)==1;
                for j=1:n1 %check the row of FCM1
                    if FCM1(t,j)~0
                        com1(j)=1;
                        for k=1:f %check the column of the FCM1
                            if FCM1(k,j)~0
                                temp_fcn(k)=1;
                            end
                        end
                    end
                end
            end
        end
    end
end
%check the row of FCM3
for j=1:n3
    if FCM3(t,j)~=0
        com3(j)=1;
        for k=1:f
            if FCM3(k,j)~=0
                temp_fcn(k)=1;
            end
        end
    end
end
end

if ((NIM1*com1')<threshold)&((NIM3*com3')<threshold)
    for t=1:n1
        if com1(t)==1
            y1(t)=1;
        end
    end
    for t=1:n3
        if com3(t)==1
            y3(t)=1;
        end
    end
end
for t=1:f
    if temp_fcn(t)==1
        fcn(t)=1;
    end
end
end
end
fcn=zeros(1,f);

for i=1:f % comparison between product 2 and 3
    if fcn(i)==0
        fcn(i)=1;
        temp_fcn=zeros(1,f);
        com3=zeros(1,n3);
        com2=zeros(1,n2);
        temp_fcn(i)=1;
        for t=i:f
            if temp_fcn(t)==1;
                for j=1:n3 %check the row of FCM3
                    if FCM3(t,j)~=0
                        com3(j)=1;
                        for k=1:f %check the column of the FCM3
                            if FCM3(k,j)~=0
                                temp_fcn(k)=1;
                            end
                        end
                    end
                end
            end
        end
    end
end
for j=1:n2 %check the row of FCM2
    if FCM2(t,j)~=0
        com2(j)=1;
        for k=1:f %check the column of the FCM2
            if FCM2(k,j)~=0
                temp_fcn(k)=1;
            end
        end
    end
end
end
if \((\text{NIM3} \times \text{com3'}) < \text{threshold})\&\&(\text{NIM2} \times \text{com2'}) < \text{threshold}\)
for \(t=1:n3\)
    if \(\text{com3}(t) == 1\)
        \(y3(t) = 1\);
    end
end
for \(t=1:n2\)
    if \(\text{com2}(t) == 1\)
        \(y2(t) = 1\);
    end
end

for \(t=1:f\) %pass temp fcn vector to fcn vector to keep track of fcns checked
    if \(\text{temp}_\text{fcn}(t) == 1\)
        \(\text{fcn}(t) = 1\);
    end
end

B.1.13 Calculating IM for components and modules

calIM2.m

function \([\text{IMc},\text{IMm}] = \text{cal}_\text{IM2}(x,p)\)
%Calculates the IM for a chromosome x

\(\text{DSM}=\text{prdsm}(p)\);
%\text{nclust}: max number of clusters \(\mid\) \(n\): number of components
[\text{nclust},n]=\text{size}(x);
%calculate CIC
CIC=(DSM*ones(n,1))+(ones(1,n)*DSM)';

%Vector with number of components per cluster
cl=x*ones(n,1);

%sum of components per clusters
acumcl=ones(1,nclust)*cl;

%DSM1 %Ideal DSM according to current clustering
for i=1:n
    for j=1:n
        if i~=j
            for k=1:nclust
                if (x(k,i)==1)&(x(k,j)==1)
                    DSM1(i,j)=1;
                end
            end
        else
            DSM1(i,j)=0;
        end
    end
end

%calculate MDL
MDLC=zeros(n,1); %Vector for the MDL representation of each component
MDLM=zeros(nclust,1); %Vector for the MDL representation of each cluster

%creating a binary DSM
for i=1:n
    for j=1:n

if DSM(i,j) ≠ 0
    BDSM(i,j) = 1;
else
    BDSM(i,j) = 0;
end
end
end

nlpc = zeros(nclust,1); % number of links per cluster
nipc = zeros(nclust,1); % number of interfaces per cluster
CIm = zeros(nclust,1); % CI per cluster

for i = 1:nclust
    ninl = zeros(n,1); % Vector with number of internal links per component per cluster
    nexl1 = zeros(n,1); % Vector with number of external links per component per cluster
    for j = 1:n
        if x(i,j) == 1
            ninl(j) = ninl(j) + (x(i,:) * BDSM(:,j)) + (BDSM(j,:) * (x(i,:)'));
        end
    end
    nlpc(i) = x(i,:) * ninl; % links per cluster
    for j = 1:n % interfaces per cluster
        if x(i,j) == 1
            nexl1(j) = nexl1(j) + ((ones(1,n) - x(i,:)) * BDSM(:,j)) + (BDSM(j,:) * (ones(1,n) - x(i,:))');
        end
    end
    nipc(i) = x(i,:) * nexl1; % interfaces per cluster
    for j = 1:n % MDL for the elements in at least one cluster
        if x(i,j) == 1 && nlpc(i) ≠ 0 && ninl(j) ≠ 0
            MDLc(j) = MDLc(j) + (log2(nlpc(i)/ninl(j))); % MDL
if x(i,j)==1 && nipc(i)~=0 && nexl1(j)~=0
    MDLc(j)=MDLc(j)+(log2(nipc(i)/nexl1(j))); %MDL
end
end

for j=1:n %calculate CIm (CI per clusters)
    CIm(i)=CIm(i)+x(i,j)*(((ones(1,n)-x(i,:))*DSM(:,j))+(DSM(j,:)*(ones(1,n)-x(i,:))'));
end
end

nexl=zeros(n,1); %Vector with number of external links per component
for i=1:n %Account for external links
    for j=1:n
        if (DSM1(i,j)==0)&&(DSM(i,j)~=0) %external
            nexl(i)=nexl(i)+1;
        end
        if (DSM1(j,i)==0)&&(DSM(j,i)~=0) %external
            nexl(i)=nexl(i)+1;
        end
    end
end

cwcl=ones(1,n)-(ones(1,nclust)*x); %component with clusters
for i=1:n
    if cwcl(i)<1
        cwcl(i)=0;
    end
end

lml=cwcl*nexl+ones(1,nclust)*nipc; %links at the main level
for i=1:n %MDL for the elements that don't belong to any cluster
    if cwcl(i)==1
        MDLc(i)=log2(lml/(nexl(i)));      
    end
end

MDLm=zeros(nclust,1);
for i=1:nclust %MDL of each cluster
    if nipc(i)==0
        MDLm(i)=0;
    else
        MDLm(i)=log2(lml/nipc(i));
    end
end

%Calculate IM for components
for i=1:n
    IMc(i)=CIc(i)*MDLc(i);
end

%Calculate IM for clusters
for i=1:nclust
    IMm(i)=CIm(i)*MDLm(i);
end
### B.2 DSM for Printers and Cartridges

![Diagram of DSM for Printing Subsystem in Printer HP Deskjet 1220C](image)

*Figure B.1: DSM for printing subsystem in printer HP Deskjet 1220C.*
Figure B.2: DSM for printing subsystem in printer HP Photosmart Plus.
Figure B.3: DSM for printing subsystem in printer HP Business Inkjet 2600.
### B.3 Function-Component Matrices for Printers and Cartridges

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Printer</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Cartridge</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Figure B.4: FCM for printing subsystem in printer HP Deskjet 1220C.
Figure B.5: FCM for printing subsystem in printer HP Photosmart Plus.

| Function # | Component # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|------------|-------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1          | Store K ink |   |   |   | X |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 2          | Store C ink |   |   | X |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 3          | Store M ink |   | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 4          | Store Y ink |   | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 5          | Move carriage along paper |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 6          | Deliver K ink precisely into paper | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 7          | Deliver C ink precisely into paper | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 8          | Deliver M ink precisely into paper | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 9          | Deliver Y ink precisely into paper | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 10         | Send/Receive signals | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 11         | Provide support for movement | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 12         | Provide support for cartridges | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 13         | Protect K ink from environment | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 14         | Protect C ink from environment | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 15         | Protect M ink from environment | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 16         | Protect Y ink from environment | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 17         | Protect K printhead from environment | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 18         | Protect C printhead from environment | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 19         | Protect M printhead from environment | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 20         | Protect Y printhead from environment | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 21         | Clean prinheads | X | X | X |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 22         | Prevent ink spills | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 23         | Absorb extra ink | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 24         | Sense K ink level and cartridge info | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 25         | Sense C ink level and cartridge info | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 26         | Sense M ink level and cartridge info | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 27         | Sense Y ink level and cartridge info | X | X |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 28         | Transport ink from reservoir to printhead | X | X | X |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
Figure B.6: FCM for printing subsystem in printer HP Business Inkjet 2600.
### B.4 Optimum clustering and component sharing candidates for Printers and Cartridges Considering Security Restrictions

<table>
<thead>
<tr>
<th>Comp. #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

| Y1      | Y1(0) | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
|         | Y1(1) | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   |
|         | Y1(2) | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

| Yc1(2)  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

| X1(2)   | Cluster 1 | 0  | 1  | 1  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
|         | Cluster 2 | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
|         | Cluster 3 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 1   | 1   | 1   | 0   | 0   | 0   | 0   | 0   |
|         | Cluster 4 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure B.7: Results summary for printer HP Deskjet 1220C.
Figure B.8: Results summary for printer HP Photosmart Plus.
Figure B.9: Results summary for printer HP Business Inkjet 2600.
B.5 Security constraints (SecM) generation example

Figure B.10: FCM, CrFun and SecM generation for printer HP Photosmart Plus.
References


P. W. O’Hearn, H. Yang, and J. C. Reynolds, On the criteria to be used in decomposing systems into modules, ACM Transactions on Programming Languages and Systems, 31 (2009), pp. 1 – 50.


D. L. Parnas, On the criteria to be used in decomposing systems into modules, Communications of the ACM, 15 (1972), pp. 1053 – 1058.


116


A. J. Rojas-Arciniegas and H. M. Kim, *Optimal component sharing in product family by simultaneous consideration of minimum description length and impact metric*, Engineering Optimization, iFirst (October 11, 2010), pp. 1 – 18.


