A METHOD FOR DEFINING THE REQUIREMENTS OF A
PRODUCT PLATFORM.

by

Carlos Tapia

Submitted to the
Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

May, 1999

© 1999 Carlos Tapia
All rights reserved

The author hereby grants to MIT permission to reproduce and to distribute publicly paper
and electronic copies of this thesis document in whole or in part.

Signature of Author ____________________________ Department of Mechanical Engineering
May 7, 1999

Certified By ____________________________ Kevin N. Otto
Thesis Supervisor

Accepted By ____________________________ Ernest G. Cravalho
Chairman of the Undergraduate Thesis Committee

JUN 1 7 1999
A METHOD FOR DEFINING THE REQUIREMENTS OF A PRODUCT PLATFORM.

Submitted to the Department of Mechanical Engineering
on May 7, 1999, in partial fulfillment of the requirements for the degree of
Bachelor of Science

by

Carlos Tapia

Abstract

The economic success of a company depends on its ability to produce products
that satisfy the needs of a wide variety of consumers. These products have to be produced
in a cost-effective manner. One approach to this problem is the creation of product
families based on a platform architecture.

A platform architecture is the set of selection and configuration choices shared
among multiple products. This thesis presents a method for designing product platforms
that takes into consideration both the performance requirements as well as the cost of the
product family. The method is used to identify possible subsystems of each product that
could be made common to all products in the family and become part of the platform.

Thesis Supervisor:
Kevin N. Otto
Title: Robert N. Noyce Assistant Professor of Mechanical Engineering
Acknowledgements

I would like to thank several people for their support and counseling throughout the course of this project. First I would like to thank my parents because their trust and enthusiasm motivated me to undertake the MIT adventure. My girlfriend Iliana Salido for understanding and putting up with the monopoly that this school imposes on my time.

I would like to thank my thesis advisor, Kevin Otto for giving me the opportunity to work on this project. He guided me through the whole process while allowing me the freedom of exploring different possibilities. He made me feel this was my project. Thanks to Javier Gonzalez-Zugasti who had the patience to introduce me to the project and helped me along the way.

Special thanks to my roommates who have helped me wake up every morning. This thesis would not have been possible without their invaluable help. And last and possibly least, to my friend Daniel Nelkembaum who is rapidly running out of friends. Without his opportune advice, working on this project would probably not have been as much fun.

MIT has been an interesting experience. It is a place that has people as its most valuable asset. It has destroyed most of my prejudices and opened my mind to almost anything. I think it has made me a better person overall.
# Table of Contents

ABSTRACT ......................................................................................................................... 2

ACKNOWLEDGEMENTS ..................................................................................................... 3

TABLE OF CONTENTS .................................................................................................... 4

TABLE OF FIGURES ....................................................................................................... 5

CHAPTER 1 INTRODUCTION ............................................................................................. 6

1.1 MOTIVATION .............................................................................................................. 6

1.2 THESIS OVERVIEW ................................................................................................... 9

CHAPTER 2 BACKGROUND ............................................................................................. 10

2.1 PRODUCT ARCHITECTURE ................................................................................... 10

2.2 MODULAR ARCHITECTURE .................................................................................. 11

2.3 INTEGRAL ARCHITECTURE .................................................................................. 11

2.4 PRODUCT PORTFOLIOS AND FAMILIES .............................................................. 12

2.5 PRODUCT PLATFORM ............................................................................................ 12

2.6 PRODUCT VARIETY ............................................................................................... 13

2.7 FLEXIBILITY ........................................................................................................... 13

CHAPTER 3 ALGORITHM ................................................................................................. 14

3.1 PROCEDURE ............................................................................................................ 15

3.2 REDUCING THE DESIGN SPACE FOR A PARTICULAR PRODUCT ......................... 15

3.3 IMPROVING THE SOLUTION ................................................................................ 21

3.4 ASSUMPTIONS ....................................................................................................... 23

3.5 DESIGN SPACE FOR THE PLATFORM .................................................................... 25

CHAPTER 4 APPLICATION TO INTERPLANETARY MISSION DESIGN ......................... 27

4.1 PERFORMANCE MODEL ......................................................................................... 28

Table of Figures 4
4.2 MISSION REQUIREMENTS.................................................................................31
4.3 OBTAINING THE PLATFORM.........................................................................32
4.4 ADVANTAGES OF METHOD............................................................................33

REFERENCES.........................................................................................................35

Table of figures

FIGURE 4.1: FORD'S MODEL T .............................................................................6
FIGURE 1.2: BLACK & DECKER'S VERSAPAK PRODUCT FAMILY......................8
FIGURE 3.1: TWO VARIABLE CASE, ORIGINAL BOUNDS ..................................19
FIGURE 3.2: TRAVERSING THE SECOND VARIABLE ALONG THE NEW MIDDLE POINT FOR VARIABLE 1 ........20
FIGURE 3.3: RADIUS REPRESENTATION OF THE CHANGE IN STATE OF THE PRODUCT .........21
FIGURE 4.1: PLATFORM REQUIREMENTS DEFINITION PROCESS OVERVIEW ...............27
FIGURE 4.2: CEM SPACECRAFT PERFORMANCE SYSTEM MODEL ................29
FIGURE 4.3: BOUNDS FOR THE THREE CASE MISSIONS ..................................32
FIGURE 4.4: FINAL PLATFORM DESIGN ................................................................33
Chapter 1  Introduction

1.1  Motivation

In 1908 Henry Ford and his Ford Motor Company introduced the model T automobile, its only product at the time. It was a great product for its time; it was one of the first attempts at mass production. The selling price of the car was a miracle at the time thanks to the exploitation of economies of scale. But it offered no options for the consumers, it was said that it was available in “any color, as long as it was black”. The model T became one of the most successful products of the century, ensuring Ford as a leader in the industry [8]. At that time, consumer preferences were not as important as they are today and the pace of technological advances has no comparison to what firms experience today.

Figure 1.1: Ford’s Model T
The model T stayed in the market until 1927, it remained virtually unchanged for nineteen years. Then things started changing and consumers demanded a wider range of automobiles, which General Motors was able to provide. This forced Ford to introduce a new automobile, the model A. The company was not prepared for this kind of change, and production was halted for six months in order to accommodate the factory for the new product. As we all know, the company survived and now offers a diverse line of automobiles and trucks [8].

The world has changed since the times of Henry Ford, and so have consumers. It is not possible anymore to sustain long term success with a single product. Firms must generate a continuos stream of value-rich products in order to retain their market share and customer loyalty. Furthermore, corporations not only need to rejuvenate their existing products, but need to expand their product diversity in order to penetrate other growing markets [2].

This increase in product variety and responsiveness to the environment is expensive. When products are developed individually, the costs that the firm faces escalate as more products are introduced. The maintenance of these new products also increases the costs because of the need of replacement parts and supporting services. The development of more products also brings competition for resources between these products within the corporation.

When products are developed as part of a family with a product platform architecture, most of these costs can be eliminated. Competition of resources disappears resulting in a more robust family of products. Components that are shared are
manufactured in higher volumes, significantly reducing the cost. In this scheme, products share not only physical components but also manufacturing processes and assembly systems. This allows for a higher quality in the final products due to the better learning of the systems that is acquired in this way.

Black & Decker' consumer power tools are a great example of the advantages of the platform architecture. Currently it produces a family of over 20 products, each of which runs off of one or two VersaPak interchangeable batteries. All of the products share this battery system which forms the platform of the family. Black & Decker has managed to offer quality products at a competitive price by taking advantage of platform architecture. Figure 1.2 shows the VersaPak family of power tools.

Figure 1.2: Black & Decker's VersaPak product family
The goal of this thesis is to provide a design tool that will facilitate the determination of the components of the product platform. This method will provide a list of candidate components or design variables that are able to meet the performance requirements for all the products in the family. The design team of the platform will need to determine the performance criteria for each of the variant products in the family. Also, a system model describing the relationship between inputs and outputs of the product is required to evaluate the performance of the product. The algorithm proposed in this thesis elaborates on this model and based on the requirements for each product narrows the number of design variables or subsystems that can be included as part of the platform.

1.2 Thesis Overview

The organization of this thesis is as follows:

*Chapter 2* provides a background on Product Architecture. It presents key definitions and introduces the terminology. It also elaborates in the advantages of product families and platform architecture.

*Chapter 3* presents the proposed algorithm. First it explains the process of reducing the design space for individual products and provides the assumptions made by the algorithm. Then it expands the method to the complete family in order to obtain a candidate platform.

*Chapter 4* illustrates the method by applying it to the design of interplanetary missions. It explains the characteristics of the model used to describe the performance of spacecraft and presents the results provided by the algorithm.
Chapter 2  Background

2.1  Product Architecture

According to Ulrich [5], product architecture is "the scheme by which the function of a product is allocated to physical components." The architecture of the product defines the basic conceptual design of the product and it is created at the beginning stages of the development. It represents the general organization of subsystems without describing much detail. The rest of the design effort builds from it and depends on its design. The definition of a product architecture allows the creation of models that predict the performance of the product early in the development process.

Ulrich also provides a more detailed definition, he identifies three stages, (1) the arrangement of functional elements; (2) the mapping from functional elements to physical components; (3) the specification of the interfaces among interacting physical components. The function of a product is what this product does as opposed to the ways in which the product achieves this function. The second part of product architecture defines how these functions are carried out by physical components. The mapping between functional elements to physical components can be one-to-one, many-to-one or one-to-many. These distinctions produce a modular or integral architecture. The final aspect of product architecture concerns the specification of the interfaces between these components. In this stage, the interactions between components are defined. Designers can choose to adopt standardized protocols to conform to the industry standards or to achieve internal sharing of components.
The architecture of a product represents a very important decision about the design of the product. The ways in which the product can change to evolve or improve are greatly determined by its architecture. The following sections further elaborate on this concept.

2.2 Modular Architecture

In a modular architecture, each functional element of the product is implemented by exactly one physical component. These components interact with each other through a few well-defined interfaces [4]. Modular products can be renovated or improved with relatively little effort. Because each of the functions is performed by separate components changes do not need to involve a redesign of the entire product. The component that implements the feature that needs to be changed or improved can simply be replaced by the new design making sure that the interfaces are preserved. The disadvantage of this type of architecture is that in order to separate each function into a distinct physical component some performance is lost. On the other hand, products with an integral architecture are designed to maximize performance.

2.3 Integral Architecture

A product embodying an integral architecture will often be designed to achieve the maximum performance possible. In this type of architecture, functional elements are performed by more than one physical component. The boundaries between these physical components are not clear or do not exist. More than one function might be implemented into a single component to optimize for size or to achieve a more elegant design. This
implementation is more difficult to modify. In order to change a particular feature or function, the entire product has to be redesigned due to the coupling between functions.

Products are seldom completely integral of modular. In most of the cases, they exhibit a combination of the two. In these cases, it is said that the product has a degree of modularity in comparison to another similar product.

2.4 Product portfolios and families

A product portfolio is the collection of products offered by a company. There is no necessary relationship between these products other than their source. Frequently, products in a portfolio are designed individually and do not share any components. A product family is a type of product portfolio. It is characteristic in that its products share components or resources. The advantage of product families is that they reduce costs through component standardization. The standardization or use of the same component in the products throughout the family allows the company to manufacture the component in higher volumes and take advantage of economies of scale [4]. Also, the development costs are reduced since the total number of components for the family as a whole is also reduced.

2.5 Product Platform

The components or resources that are shared in a product family are grouped into the product platform. Meyer and Lehnerd [2] define a product platform as the “set of common components, modules or parts from which a stream of derivative products can be efficiently created and launched.” Platforms are the tool with which companies are
able to offer a larger product variety to the customer without incurring in prohibitive costs.

Because of the amount of resources devoted to a platform, successful platforms require considerable preliminary planning of the product. The design effort does not rely only on the engineering team, but requires the involvement of other departments such as marketing and manufacturing.

2.6 Product Variety

The diversity of customer backgrounds and preferences creates the necessity to offer product variety. Product variety is the diversity of products that a production system provides to the marketplace [5]. Products based on a platform architecture can be varied more easily by the introduction of variant products without requiring a redesign of the whole product. Variant products make use of the product platform as a starting point and add features or increase performance (variants) by adding or interchanging components to the base product.

2.7 Flexibility

Flexibility, according to Upton’s definition is “...the ability to adapt with little effort, time or penalty”[6]. The flexibility of a firm resides not only in their manufacturing systems, but also in the architecture of its products. As it was mentioned in the previous sections, the easiness with which changes can be made to a particular product is heavily related to its architecture. A modular architecture permits the modification of one component without affecting the rest of the product. On the other hand, on an integral product, a change would be impossible without a complete redesign of the entire product.
Chapter 3  Algorithm

The problem addressed by this algorithm is a design problem. It consists of determining which components or subsystems of each of the members of a product family are viable candidates to be part of the platform. Each product in the family has some specified performance and budget constraints that are specific to that product. The platform has to conform to the performance and budget criteria for all of the products because it will be shared among the members of the family.

The algorithm finds a set of limits on the design variables $\bar{d}_i$ for each individual product $i$. The limits exclude most of the solutions that prove infeasible by the performance parameters $\bar{p}_i$ and budget constraints $B_i$. Finally, it is able to determine which design parameters could be included in the product platform by comparing these reduced ranges in the design variables among each of the family members. If the ranges for one of the design variables intersect across all of the members, then the component or subsystem described by the variable in question is a candidate to be part of the platform. On the other hand, if the ranges for this variable do not describe a common set among the family members, then the component described by this variable cannot be part of the platform and would have to become a variant component different for each of the products in the family.

This approach assumes work done in the early design stages of the project. The designers have to identify the specific requirements of each of the product variants in the family, though these can be negotiated in the course of this algorithm. A general structure for the product has to be determined. This means deciding which are the necessary
components that will be present in the final product, but the details such as the type or size of these components will be left to be decided later on.

The algorithm requires a previously developed model of the product. This model has to be flexible in order to accommodate the different possible architecture choices for the product family that haven’t been fixed yet. Also it has to provide an accurate description of the performance and cost of the product. The model takes the values for each one of the design parameters $\bar{d}_i$ as inputs and reports the performance metrics of interest and the expected cost of the product. The algorithm uses this information to evaluate the design by comparing it to the specified performance parameters $\bar{p}_i$ and budget constraints $B_j$. The expected cost results should be based on the design parameters and should reflect development, manufacturing, operations and other costs incurred in the life cycle of the product.

3.1 Procedure

In order to exclude infeasible architectural choices or design values, the algorithm focuses first on the individual variants of the product family. For each one of the family members, it produces a set of bounds $d_{ij}^{lb}$ and $d_{ij}^{ub}$ for each design variable $j$. These bounds exclude all of the values for which the algorithm has determined that the model would yield unsatisfactory results given the constraints.

3.2 Reducing the design space for a particular product

The algorithm needs some initial information apart from the model and the performance and cost constraints. It starts with an initial range for each one of the design
variables, $d_{ij}^{l,b}$ and $d_{ij}^{u,b}$, determined beforehand by the design team. This range is determined only by common sense. It does not limit the design space in any practical way, it is only meant to provide a reasonable starting point for the algorithm and effectively reduce the complexity of the problem. For example, it is not reasonable to consider negative values for the radius of an antenna plate or masses in the range of dozens of tons for an electronic component. From this state, the algorithm reduces the width of these ranges excluding unfeasible values.

The algorithm starts by focusing on a particular product $i$. The first step is to set all of the variables to the middle point between the upper and lower bounds previously specified, namely to the value $d_{ij}^m$, where

$$d_{ij}^m = \frac{d_{ij}^{l,b} + d_{ij}^{u,b}}{2}$$

Equation 3-1

Note that when the variable represents an architectural choice (i.e. a choice of a standardized component), the values for this variable are bound to be integers, thus the formula becomes,

$$d_{ij}^m = \text{INT}\left(\frac{d_{ij}^{l,b} + d_{ij}^{u,b}}{2}\right)$$

Equation 3-2

Once all of the variables are set to the middle points between their bounds, the algorithm proceeds to traverse the first variable $d_{il}$ from its lower bound $d_{il}^{l,b}$ to its upper limit $d_{il}^{u,b}$ evaluating the performance of the product as reported by the model. The number of steps and thus the size of the steps that it takes to go from the lower to the upper limit define the resolution of the method. At every step, the performance metrics produced by
the model are compared to the performance parameters $\bar{p}_i$ and cost criteria $B_i$ specified for this particular product. The first time that the algorithm encounters a value $d_{ij}^{\text{first}}$ that produces an acceptable outcome according to the specifications, it modifies the lower bound to take this new value so that,

$$d_{ij}^{\text{l.b.}} \rightarrow d_{ij}^{\text{first}}$$

Equation 3-3

After this, the program continues scanning the range of the variable, presumably obtaining satisfactory results from the product model, until it finds a value $d_{ij}^{\text{last}}$ that produces faulty results when it is inputted to the model. At this time, the upper limit is changed to coincide with this value,

$$d_{ij}^{\text{u.b.}} \rightarrow d_{ij}^{\text{last}}$$

Equation 3-4

These new lower and upper bounds for the design variable now produce a new middle point $d_{ij}^{m}$. The value of this new middle point is inputted as variable $d_{ij}$ in the model. The algorithm repeats this process for the next design variable until all the design variables have been traversed and their bounds and middle points updated. The new limits reflect the exclusion of values that produce unsatisfactory results when the rest of the variables are held at the middle points of their bounds.

In this process, each one of the design variables is evaluated at different points throughout their limits while holding the rest of the variables fixed at their middle points. Because of this, the limits that are obtained for each variable depend on the limits for the rest of the design variables. The updated limits obtained for a variable $d_{ij}^{m}$ are only as accurate as the bounds delimiting the rest of the variables at the time these new limits
where obtained. Since the algorithm started reasonable with, but arbitrary values, the bounds obtained after one application of this algorithm do not exemplify a final solution. However, these new bounding sets represent an improvement over the original bounds.

After all of the bounds have been reduced, the new middle points define a different state for the product when inputted to the model. This state can be compared to the original state to obtain a measure of the magnitude of this change.

We will treat the change between the previous state of the product and the new modified state as a ‘distance’ and we will refer to it as the \textit{radius}. We will calculate the change in each one of the design variables and average them to obtain the \textit{radius} thus,

\[
\text{radius} = \sqrt{\frac{\sum_{j=0}^{n} (d_{ij}^{m_0} - d_{ij}^{m_n})^2}{n}}
\]

\text{Equation 3-5}

where \( n \) is the number of design variables, \( d_{ij}^{m_0} \) is the previous middle point for variable \( d_{ij} \) and \( d_{ij}^{m_n} \) is its new middle point. The importance of the \textit{radius} will become more clear in the next section.

Consider an example with only two variables \( d_1 \) and \( d_2 \), performance parameters \( \bar{p} \) and budget constraint \( B \) to illustrate this concept. Figure 3.1 displays the original bounds as well as their middle points for this case.
The algorithm starts with the set of original bounds for these two variables, $d_{1^{lb}} - d_{1^{ub}}$ and $d_{2^{lb}} - d_{2^{ub}}$. It starts by setting $d_2$ to its middle value $d_2^m$ and then scanning the values of $d_1$ starting at the lower limit. When it finds the first value $d_1^{first}$ that satisfies the performance criteria $\bar{p}$ and meets the budget $B$, it moves the lower limit $d_1^{lb}$ to this value. The scanning continues until the last value that produces a satisfactory result according to the specifications is found. The new upper bound $d_1^{ub}$ is assigned this value $d_1^{last}$. Figure 3.2 shows this process.
Figure 3.2: Traversing the second variable along the new middle point for variable 1.

Now that the bounds for $d_1$ have been updated, there is a new middle point $d_1^m$ for this design variable. The second variable $d_2$ is now traversed in the same way $d_1$ was scanned; only this time $d_1$ is held at its new middle point (see Figure 3.2). Once the bounds for $d_2$ have been reduced, the middle points for the two variables represent a different state of the product. The *radius* for this new state can be calculated using equation 3-5, for this case

$$
\text{radius} = \sqrt{\frac{(d_{11}^m - d_{10}^m)^2 + (d_{12}^m - d_{12}^m)^2}{2}}
$$

Equation 3-6

The *radius* that represents this change is displayed in Figure 3.3.
3.3 Improving the solution

The reduced bounds obtained with the process described in the previous section represent an improvement over the original bounds, but are far from being the final solution. We can take advantage of this and apply the same process again to obtain better results. These iterations can be repeated until the process fails to produce a change over the previous iteration. In other words, once the radius between two continuous iterations becomes zero, a final solution has been found. Figure 3.4 illustrates this process of gradual improvement until the final solution is reached.

There is one modification to the original algorithm followed in the first iteration. The bounds obtained in each iteration are based on the middle points that were fixed at
that time. However, one or more of those middle points could be excluded on further iterations and thus not be part of the feasible intervals produced as the final solution. Therefore the reduction in the bounds produced in each iteration should not be considered final. The whole extent of the original bounds is searched with each subsequent iteration to avoid the premature exclusion of possible values for the design variables. The real value of the iterations previous to the final solution is the change in the middle points.

Figure 3.4: Iterations to reach final result

Once there is no change in the state of the product to within some iteration limit $E$, the bounds limiting the range of the design variables represent the final solution. These values within those bounds are the feasible intervals for the design variables. These
are values that will generate a product that will satisfy the performance and budget constraints. Note that not all of the combinations of design parameters within these bounds are guaranteed to be feasible. There might be some combinations that fail to produce a satisfactory result by not meeting one or more of the constraints. However, we have determined that all of the values outside of these bounds produce products that do not satisfy the performance requirements. In this way, the algorithm has narrowed the design space for each of the products, thus simplifying the design problem.

3.4 Assumptions

It is assumed that for every variable range, $d_{ij}^{lb} - d_{ij}^{ub}$, there is one and only one continuous subset $d_{ij}^{lb}_{final} - d_{ij}^{ub}_{final}$ such that $d_{ij}^{lb} \leq d_{ij}^{lb}_{final}$ and $d_{ij}^{ub}_{final} \leq d_{ij}^{ub}$ that meets the performance criteria while the rest of the variables are set to their middle value. Figure 3.5 illustrates this assumption. This assumption applies to the final iteration. However, it is possible that before reaching the final state, in one of the intermediate iterations, one of the variables fails to produce a subset of values that meets the performance criteria in combination with the rest of the variables at their middle points. In this case the interval would be the null set, i.e., there is no platform design which can meet all the variant requirements. The algorithm handles this case by setting the upper bound at a value 1% higher than the lower bound. This is done in order to arbitrarily change the middle point of that variable to ensure the continuous evolution in the bounds. Note that this is only done in intermediate iterations and never when the final solution is presented.
Some of the design variables could represent a choice for a standardized component from a catalog. In this case, it is assumed that the choices are placed on a list arranged in a monotonic fashion according to the component characteristic that has a greater effect on the performance of the product. A numerical value (integer) is assigned to each component according to their order in the list (see Table 3.1). The algorithm traverses the range for these variables in the same way it does with the others except that it makes sure that it does so in a discrete fashion. It only assigns integer values to these variables and makes sure to consider all of the choices regardless of the step size or number specified to traverse the variables.

<table>
<thead>
<tr>
<th>No.</th>
<th>Component</th>
<th>Primary characteristic</th>
<th>Secondary characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Component A</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Component B</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Component C</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Component D</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>Component E</td>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>

*Table 3-1: Example list of components arranged monotonically on the primary characteristic*
3.5 Design space for the platform

Once a set of bounds on the design variables is established for each of the products in the family, the information can be used to obtain bounds of the same type for the platform.

If a variable is going to common to all of the products in the family, then it must satisfy the requirements for each of these products to be a feasible platform. This is equivalent to finding the intersection between the design variable ranges, found with the process described in the previous section, on each product (see Figure 3.6).

![Figure 3.6: Finding intersections to design the platform](image)

Equation 3-7 and 3-8 describes the process that finds the intersections for each design variable \( j \) across all of the products in the family,

\[
    d_{yj}^{ub, platform} = \max(d_{1j}^{ub}, d_{2j}^{ub}, ..., d_{yj}^{ub})
\]

Equation 3-7

\[
    d_{yj}^{lb, platform} = \min(d_{1j}^{lb}, d_{2j}^{lb}, ..., d_{yj}^{lb})
\]

Equation 3-8
for all products $i$. If the intersection defines a non-empty interval, then the variable is a viable candidate to become part of the platform. There is a range of values in this variable that satisfies the performance and budget constraints for all of the products in the family. On the other hand, if the intersection turns out to be the null set, then there is no value in that variable that is able to meet the specifications for all of the members of the product family.
Chapter 4  Application to Interplanetary Mission Design

The process described in the previous section was applied to the design of interplanetary missions. Figure 4.1 shows a simplified diagram of the process. First, a system performance model was obtained. Then, the requirements for three fictitious missions were determined. Using the algorithm from the preceding section, a set of bounds for the design variables of each of the missions was obtained. These bounds were later used to find their intersection and determine which variables, and the subsystems they describe, were viable candidates to become part of the platform.

![Diagram of Mission Models, Individual Bounds, Platform Bounds](image)

**Figure 4.1:** Platform requirements definition process overview
4.1 Performance Model

The products that compose a product family are meant for different purposes. Because of this, each has its own requirements, a set of performance parameters that it needs to satisfy. In order to evaluate the performance and determine if the product in question meets its requirements, a system model of the product needs to be created.

Depending on the nature of the product, the model can become quite complex. Products such as spacecraft require a very elaborate model that incorporates the coupled nature in the subsystems of the product. In such an intricate product, a change in one subsystem or component usually affects the performance of other subsystems of the product. For example, increasing the transmission data rate of the spacecraft may mean an increase in the size of the antenna, which would increase the system mass, and this in turn would require a different propulsion system, attitude control and other subsystem parameters. Therefore, in order to accurately measure the performance of products such as spacecraft, an inter-linked system performance model is needed.
### Figure 4.2: CEM spacecraft performance system model

<table>
<thead>
<tr>
<th>System</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mission Model</strong></td>
<td></td>
<td>(Based on Interplanetary Concurrent Engineering Methodology - CEM)</td>
</tr>
<tr>
<td><strong>State and Orbit</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Altitude Correction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orbit Correction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parking Maneuver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orbit Insertion Delta-V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Earth-Moon</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gravitational Center</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Lunar</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gravitational Center</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Luna Navigation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gravitational Center</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Moon Parameter</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gravitational Center</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Lunar Navigation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gravitational Center</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Launch Vehicle</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mass to Orbit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mass to Moon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mass to Earth</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Propulsion</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Propellant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pressurant Tank</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Telemetry</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Antenna</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High Gain Antenna</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Amplifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mission Encoder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Telemetry Encoder</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Energy Storage</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solar Array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Battery</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Command &amp; Data Handling</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Launch Vehicle</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Launch Vehicle Mass</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Power</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUCLEAR OPTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Power [W]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peak Power [W]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Conditioning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOLAR OPTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solar Array [W]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Power [W]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peak Power [W]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Conditioning</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Structure</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structure Mass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cabling</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Launch Vehicle</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mass</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** All values and parameters are subject to change based on mission requirements and spacecraft design.
Currently, the Jet Propulsion Laboratory (JPL) uses a model originally created by the Aerospace Corporation to develop early-stage designs for interplanetary missions. This model is named "Concurrent Engineering Methodology" or CEM.

The platforming performance model used for this application was built based on the CEM model. The CEM model consists of several linked components:

- A system model, where spacecraft requirements are entered and architecture choices are made, such as the selection of hardware components such as engines, sensors, and type of propulsion system, power source, etc.;

- A database with performance and cost data for hardware as well as planetary information;

- A summary display of the key system-level characteristics of the spacecraft.

A section of the system model is shown in Figure 4.2. Each CEM model contains information on ten major spacecraft subsystems (payload, propulsion, attitude control, telecommunications, command and data handling, thermal control, power, structures, cabling, and launch vehicle), as well as system variables (contingency margins, mass, launch vehicle cost, etc.). Figure 4.2 shows some of the input variables for the propulsion subsystem, such as number, mass, power consumption, and type of components such as thrusters, tanks, etc. Note that engineers using this model can make architectural choices by selecting actual components from a database (see the pick lists in the “Transfer Thruster” and “RCS Thrusters” rows), and also enter specifications for the input variables in the colored cells. These inputs are used to calculate system performance values through formulas built into the model that link the behavior of the spacecraft subsystems to one another. For example, increasing the number of thrusters also increases the dry
mass of the spacecraft, requiring it to carry more fuel, structural mass, etc. These changes are automatically calculated and propagated through the whole system model of the spacecraft.

The original CEM model contains over 50 design variables (inputs) that are used to calculate the performance of the spacecraft as well as its cost. For the purposes of this project, this number was reduced to 10. The rest of the input variables were either fixed to a reasonable value, or made dependant on other parameters through formulas that represent realistic values. The algorithm discussed in the previous section can be extended to any number of input variables, but we simplified the example to save time and computational resources.

The remaining 10 design variables were chosen according to their impact on the performance indices (i.e. mass, cost, etc.) of the mission. Also, four of these input variables represent a choice of a hardware component (e.g. choice of launch vehicle, star tracker, etc). Information about the characteristics of these components is stored in the database section of the model.

4.2 Mission Requirements

Ideally, the algorithm would be applied to a number of space missions with real objectives and specifications to match these goals. In this case, it was not possible to obtain information about real interplanetary missions. The algorithm was used to find the possible components or subsystems that could become part of the platform of three dummy missions. Nevertheless, we tried to conceive three missions that would differ in their main performance parameters although these do not delineate real missions. These
fictitious missions were given cost, mass, attitude control and data rate requirements, mimicking real interplanetary missions.

4.3 Obtaining the Platform

In order to obtain the candidate subsystems to form the platform each of the three missions was independently evaluated. A set of ranges for the design variables that would yield feasible solutions was found for every mission. Figure 4.3 shows the CEM model reports that were obtained for these three simulated missions.

![Table of mission parameters]

**Figure 4.3: Bounds for the three case missions**

The ranges obtained for each individual mission were compared to determine which design variables were able to meet the constraints on all three missions. Using equations 3-7 and 3-8 the intersections in these ranges across the three cases was obtained. Those variables that produced non-empty intervals are possible members of the
platform. Figure 4.4 shows these variables marked with the word “true”, and the rest of them for which no intersection was found are marked “false”.

These results were tested using the CEM model. For the variables that will be part of the platform, the middle points of their final bounds (intersection of the bounds for the individual products) were inputted to the model. The test produced satisfactory results; each mission exhibited performance indices that met the corresponding specified requirements. Therefore, the components described by these design variables could be architected into a platform without affecting the desired performance of the individual products in the family.

![Table Image]

**Figure 4.4: Final platform design**

### 4.4 Advantages of method

The number times that the algorithm had to calculate the performance and cost of the product using the CEM model in order to obtain the results for these three missions is small. In fact, it is orders of magnitude smaller than it would be required if the approach
were to apply "brute force", that is to evaluate every possible combination of values in these 10 variables.

In this case, the algorithm searched each variable range using 20 values, equally spaced, to traverse the range from its lower to its upper bound. It took 6 iterations to find the final answer on average. That requires a total of 200 calculations per iteration for a total of 1200 calculations. On the other hand, the "brute force" approach requires $1024 \times 10^{10}$ calculations. A difference of ten orders of magnitude! As we mentioned before, the system was simplified from more than 50 input variables to only 10. In that case, the algorithm would provide an even larger advantage, as the "brute force" approach would become prohibitive.
References


