Intelligent Assistant Architecture for Complex Configuration Processes

THESIS

Presented in Partial Fulfillment of the Requirements for
the Degree Master of Science in the Graduate
School of The Ohio State University

By

Vasudha Gupta

Graduate Program in Computer Science and Engineering

**********

The Ohio State University

2009

Thesis Committee

Approved By

Dr. Rajiv Ramnath, Advisor

Dr. Neelam Soundarajan

Dr. Jay Ramanathan
Copyright by

Vasudha Gupta

2009
ABSTRACT

Knowledge representation is a difficult task in the field of product configuration. This problem is most prevalent in industries such as manufacturing, sales design and installation to name a few. Existing configuration systems are not functionally complete enough to deal with the complexity of these product configuration processes and provide the desirable configurations to the user. This thesis is an attempt to do an extensive requirement analysis of the configuration process, and motivates the need for incremental re-architecture of existing systems.

There are two approaches to product configuration. In the first, known as the rules based approach, step by step guidance to the user can lead them towards a valid configuration. The other constraint based approach gives the user the flexibility to decide what to select and in what order while limiting the allowed combination of values.

The thesis tries to tackle the different issues involved in a complex configuration process by proposing an architecture termed as the Intelligent Assistant Architecture or IAA. This architecture is based on the idea that effective complex configuration processes need user participation and are best designed using constraint-based approaches with interfaces built using cognitive science principles. The architecture motivates the shift from the product to the process of configuration. It provides a set of desirable qualities to build correct configuration models. It also provides conceptual views based on cognitive
science principles to effectively depict these models to enrich the user experience and help the users in better decision making. Field data from a case study is used to create multiple representations of configuration views which are compared using two different evaluation approaches to demonstrate their usefulness and applicability. The thesis then proposes an integrated method for evaluation by combining two independent approaches.

Finally, the thesis discusses a solution approach based on the idea of interactive constraints using a prototype where the user is closely interacting with the configuration process.
DEDICATION

Dedicated to my family and friends who have been a great support during my thesis.
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Rajiv Ramnath, for giving me the opportunity to work in a very interesting area and for all his help, advice and continued support throughout my graduate studies especially during my thesis. His guidance and assistance during the thesis are much appreciated.

I would like to thank my Co-Advisor Dr. Jay Ramanathan for her input and recommendations in the thesis.

I would also like to thank all the members of the CETI group for their help and support. A special thanks to my friends Aman Kumar and Farha Mukri for their help and support during the thesis.
VITA

August  2007.................B.Tech , Indira Gandhi Institute of Technology, Delhi, India

PUBLICATIONS

1. Rajiv Ramnath, Vasudha Gupta, Jay Ramanathan: RED-Transaction and Goal-Model Based Analysis of Layered Security of Physical Spaces. COMPSAC 2008:


FIELD OF STUDY

Major Field: Computer Science and Engineering
TABLE OF CONTENTS

ABSTRACT ........................................................................................................................................... ii

DEDICATION ........................................................................................................................................ iv

ACKNOWLEDGEMENTS .................................................................................................................. v

VITA .................................................................................................................................................. vi

LIST OF TABLES .......................................................................................................................... xi

LIST OF FIGURES ......................................................................................................................... xii

Chapter 1.  Introduction .................................................................................................................. 1

Chapter 2.  Problem Statement ....................................................................................................... 3

  2.1  What is Configuration? ......................................................................................................... 3

  2.2  Problems encountered during the configuration process .................................................... 5

  2.3  Requirements for a good configuration system ................................................................. 7

  2.4  Configuration Approaches ................................................................................................... 8

  2.4.1  Configuration using Rules .............................................................................................. 8

  2.4.2  Configuration using Constraints .................................................................................... 11

  2.5  Rules versus Constraints from a user perspective .............................................................. 12

  2.6  Thesis Approach .................................................................................................................. 15

Chapter 3.  Related work .............................................................................................................. 16
LIST OF TABLES

Table 1: Constraint Tuples .......................................................... 21

Table 2: Integration of Properties from two comparison approaches ................. 87
LIST OF FIGURES

Figure 1: The Configuration Process ................................................................. 4
Figure 2: Definition of Rules ............................................................................ 9
Figure 3: Rules Structure ................................................................................ 10
Figure 4: Constraint Model ............................................................................ 11
Figure 5: Map of Australia ........................................................................... 18
Figure 6: Solution of the Map Coloring Problem for map of Australia .......... 19
Figure 7: Constraint Network for Map Coloring Problem .............................. 20
Figure 8: Constraint Network for personal computer example ................. 22
Figure 9: Map Coloring by Forward Checking .............................................. 24
Figure 10: Node Consistency ....................................................................... 26
Figure 11: Arc Consistency ........................................................................ 27
Figure 12: Arc Consistency-1 Part 1 .............................................................. 29
Figure 13: Arc Consistency-1 Part 2 .............................................................. 29
Figure 14: Arc Consistency-3 ....................................................................... 30
Figure 15: Network after reduction of domains ........................................... 31
Figure 16: Arc Consistency-3.1 ................................................................. 32
Figure 17: 4+1 architecture model ................................................................. 35
Figure 18: Construction-Integration Model ................................................... 41
Figure 19: Configurations for a cabinet ......................................................... 43
Figure 20: Basic Configuration ..................................................................... 45
Chapter 1. Introduction

The problem of configuration is well known in the field of computer science and spans multiple application areas. It is most prevalent in industries such as manufacturing, sales, design and installation to name a few [1]. Increasing complexity of these industry domains necessitates the need of an interactive configuration process to help the user in effective decision making and producing desirable and valid configurations. Configuration of even a simple product such as a personal computer or camera can become difficult for the user if they are not assisted correctly in the configuration process. The complexity of such a process can cause confusion for the user and may not produce the desired configuration due to lack of guidance from the configuration process. Thus an effective process needs to be designed to solve such complex configuration problems. There are mainly two approaches to solving configuration problems. One approach, called the Rules based approach, guides users towards a correct configuration by giving them step by step guidance. The second approach, namely the Constraint based approach, gives more flexibility to the user with respect to the choices they can make and solves the problem based on user preferences. This thesis concentrates on the latter approach.

Although the work is directed towards product configuration but patterns of this work maybe abstracted to be applied to areas such as service composition [2][3] where the
problem of configuration also exists. My work is motivated largely by the field experience I gathered while working at TDCI, an enterprise solutions provider for product configuration.

The thesis looks at the different problems in the domain of product configuration. The second chapter is a detailed problem statement. The third chapter elaborates on the related and relevant work. The fourth chapter gives a comprehensive overview of the architecture of the product configurator designed by TDCI that is used as a case study to analyze the configuration problem. The case study served as a platform for an extensive requirement analysis of the configuration process. Based on this analysis, a solution approach called Intelligent Assistant Architecture (IAA) is presented in order to tackle the problems existing in the configuration process by incorporating cognitive science principles. This architecture has two main components. Firstly, it elucidates the desirable qualities of a constraint based configuration model and shows how constraint based approaches focus on those aspects of guidance that are relevant to the user. Secondly, it includes the visual aspect to configuration based on architectural and cognitive science principles that can help the user understand the domain and make better selection decisions. This is discussed in the sixth chapter. The thesis provides a novel integrated approach of evaluating visualizations to gain insights into the usefulness and applicability of a visualization from the user perspective that is discussed in the seventh chapter. Chapter seven explains the concept of interactive constraint satisfaction problem that shows how configuration solutions should be designed to take the user participation into account with the help of a prototype. Finally chapter eight concludes the thesis and describes future work.
Chapter 2. Problem Statement

The following section defines in detail the concept of configuration. In particular as mentioned before, the definition is focused towards the area of product configuration [4]. It highlights the problems associated with configuring a product and motivates the need of a configuration process based on the proposed architecture in the thesis called the Intelligent Assistant Architecture or IAA. This architecture motivates the need of a user focused interactive configuration process and tackles its different aspects like designing correct input configuration models and designing different conceptual views of configuration models based on cognitive science aspects to assist the user in decision making. The architecture motivates the need of using interactive constraint based architectures and explains the concept of interactive constraint satisfaction in the last chapter using a prototype.

2.1 What is Configuration?

Configuration can be defined as the arrangement of parts to make up a product by selecting a valid combination of values for each of the part attributes. This arrangement is bound by a set of constraints that must be satisfied while configuring a product. Configuration is a challenge due to the extremely complex domains associated with the products being configured. For example consider the configuration of a door or a window or even the configuration of a car or yacht. Such products are very difficult to configure
not only due to large number A traditional configuration process [6] works as depicted below.

The requirements are captured through the user. A configuration model that represents the existing structure of the components to be configured is constructed based on the available product and knowledge base. Based on user requirements and the available configuration models, the configuration or inference engine solves for valid and consistent configurations and presents them to the user. There are many problems
associated with this traditional configuration process that are discussed in the next section.

of components that make up such assemblies and the constraint that exist across them but also due to the aesthetic complexity of these products to provide the desirable product to the user after configuration. All the values for one component are not compatible with values of others due to existence of intricate relationships. The components are bound by a set of constraints that limit the allowed values of others. When these domains have such large data sets with complex relationships and dependencies, constraints are difficult to model and design and the process of configuration becomes difficult. It is most complex when composite assemblies need to be created from single components and constraints existing across these assemblies that make a configuration model even more complex. This necessitates the need for a configuration process [5] that can be used to model the domain, present the domain and solve the problem in a way that is easy for the user to comprehend and use.

2.2 Problems encountered during the configuration process

1. Bad Configuration Models
   
   a. Incorrect domain model representation

   The domain models are not represented correctly by the existing configuration approaches and thus produce bad outputs when they are used as inputs during the configuration process.

   b. Conflicting conditions

   Due to bad models, a user is not able to achieve a desirable configuration. Increasing complexity of this configuration process makes the problem of
deriving a consistent state during the configuration of a product even more difficult. The configuration system must be designed to handle such conflicting states and derive consistent configurations. This necessitates the need for a good configuration design process that can efficiently solve such problems. This includes avoiding situations such as cycles and deadlocks.

2. **Inadequate guidance to the user**

   a. *Ineffective conceptual visualizations*

   The configuration process should help to direct the user towards a desirable configuration. It should be intelligently designed to tell the user why a particular conflict has arisen and how can it be resolved. This can be a difficult process and if it is not efficiently designed then the user can get lost and confused. Thus a configuration system must have built in mechanisms for traceability by providing the right guidance and support to the user. This can be achieved by assisting the user with conceptual visualizations that can make it easier for the user to understand the domain and the relationships that exist within it. If the domain is depicted well visually it can help the users in appreciating the relationships that exist in the system and help them in making good choices.

The thesis discusses ways to tackle the above problems by proposing an intelligent assistant architecture (IAA) that can solve complex configuration problems interactively by providing user assistance. It identifies a set of desirable properties for building correct configuration models. It creates different conceptual views to represent a configuration
domain and assist the user in the selection making process. The thesis provides an integrated approach to evaluate visualizations based on both architectural as well as cognitive science principles to demonstrate their usefulness to the users. Finally, the thesis also shows why solving constraints using an interactive approach is better with the help of a prototype.

2.3 Requirements for a good configuration system

Based on experience with configuration related problems the following properties are important for the design of a good configuration system [7]. It should:

a. Meet configuration requirements of different industry applications

Different product configuration companies require the configuration system to be designed for a particular purpose i.e. they are targeted towards a certain application in the industry. Thus the process must be designed to meet these industry requirements. An example that illustrates this is the Dell website (www.dell.com) that is used for configuration of notebooks, desktops, personal computers etc. Another example from the field work is Buy-Design, the TDCI product configurator that serves the door and window industry heavily but are expanding their customer base by configuring items like yachts, air conditioners etc.

b. Handle scalability

The system must be designed so that it can handle complexity if the size of the domain being considered increases and the number of relationships in it increases. In other words it should be able to support larger data sets, multiple users, and more relationships but still maintain optimum performance.

c. Allow customization and user flexibility
The system should be flexible enough to allow the selection of any variable in the system with any value based on the allowed combination of values. Certain combinations may be incompatible with each other and the configuration engine must be designed to deal with such inconsistencies in an efficient way as well as provide good guidance to the user in order to derive a consistent configuration.

d. Provide adequate performance

The engine that is designed as part of the configuration system should be able to solve configuration models input to it, in an optimum time that is acceptable to the user using the system.

The thesis helps to articulate these problems and devises ways to tackle them through a proposed architecture called the Intelligent Assistant Architecture.

2.4 Configuration Approaches

The two most common ways of representational schema for configuration are

a. Rules

b. Constraints

The Constraints based approach is the more useful approach and has more advantages as opposed to the rules based way of resolving configuration issues that are discussed in a later section.

2.4.1 Configuration using Rules

A rule is a condition–action statement. Think of a simple rule as being “If X occurs, then do Y.” E.g. If it is raining then take your umbrella.
2.4.1.1 Rulesets

A ruleset is a collection of rules that perform a series of questions, validations, and calculations to lead a user through the configuration of a valid product.

2.4.1.2 Rules Structure

Rules follow a hierarchical top down pattern. At each step in the hierarchy, conditions are evaluated based on which the rules to be executed are decided. The evaluation of a condition to true or false directs the flow in the hierarchy from top to bottom. Figure 3 shows the example of configuring a personal computer. The circles depict the variables that are evaluated at each step based on which the sub trees that are to be executed are decided.
2.4.1.3 Working of Rules

Figure 3 depicts how rules work. The flow moves through solid black lines in case a condition variable evaluates to true. For instance once a condition on the Processor has been specified, HD Capacity, Memory and their sub trees are evaluated in that order.

In a rules based system, the user is directed towards the final configuration. At each step the user is asked to make a choice based on which the process of configuring the product moves forward. After each choice, domain values of related variables are filtered and
presented to the user. Thereafter, the user selects the next variable value and so on. The process continues until all requirements for the product being configured are met.

2.4.2 Configuration using Constraints

Constraints are conditions that determine what values may be held by different variables for a consistent product configuration.

![Constraint Model](image)

**Figure 4: Constraint Model**

2.4.2.1 Working of Constraints

In a constraint based system, we have a pool of variables that define the system. Each variable has a set of associated values. The user is free to select the value for any variable. Based on the imposed set of constraints, the ranges of the dependent variables are filtered. This filtration is done using constraint satisfaction algorithms that resolve conflicting constraints and make useful suggestions to the user for a correct configuration. Figure 4 again uses the configuration of a personal computer as an
example. The different variables, their domain sets and their dependencies are modeled in the diagram.

2.5 Rules versus Constraints from a user perspective

There are two important users that are using the configuration system namely the actual configuration user and the rule or constraint modeler. The following section compares rules and constraints from the perspective of these users.

1. Order Independence

This is important from a configuration user point of view. A constraint based model does not constrain the user to start the configuration from a particular point. The user picks what matters to him in the order that matters to him. This is advantageous as opposed to the rules based approach where order is important. The user is guided to a solution in a particular direction by choosing value sets for the variables in a certain order. This can constrain the user to a specific ordering and limit user experience.

2. Extensibility

a. Extension of a constraint base is easy.

This is important from the constraint modeler point of view. New constraints can be easily added to the system. Such is not the case with rules. Extending a rules base requires addition and deletion of rules that can get complicated and difficult for the constraint modeler [9].

b. Inheritance
This is important from the constraint modeler’s point of view. It makes the problem of modeling simpler when we talk of inheritance where sub assemblies can be composed separately to make a whole assembly. Each such sub assembly can be modeled as an independent model linked together by constraints. Thus we should have autonomous constraint models in the system to simplify the modeling process.

*Example to illustrate inheritance*

A company is manufacturing a laptop, after selecting the base size the keyboard size has to be selected. Underneath the keyboard may be a slot for the DVD drive. Further there is a rotatable disc inside the DVD drive slot. The following illustrates the example by articulating it as a constraint satisfaction problem.

**Variables:**
- Laptop base: Material, thickness, width, length, height, color
- Keyboard Base: Material, color, width, length, height, keysize
- DVD drive: width, length, height, type, material
- Rotatable disc: Material, diameter, width, height

**Constraints:**
- Width and height of keyboard < Width and height of Laptop base
- Width and height of DVD drive < Width and height of keyboard
- Width and height of rotatable disc < width and height of DVD drive

**Constraint Hierarchy**
- Laptop -> Keyboard size of laptop -> DVD -> Drive

Thus we have different constraint models for the different components of a laptop. There are constraints across the models that need to be accounted for while configuring the
Thus a configuration model based on constraints helps in modeling such a system much better than rules.

3. **Rework**

In a rules based configuration model, changes in user selection need a lot of rework. Changes to values selected previously cause the user to redo their selections in the entire rule hierarchy. The user must make their selections again starting from the beginning where rules originated. In contrast to this, for constraint based models, the domains of variables are filtered based on user selection. The user does not have to redo all his selections. As user preference changes, domains of related variables get filtered to reflect the changing selection.

4. **Flexibility**

Global constraints can be enforced if the configuration model is built using constraint models. Global constraints give more power to the user. For instance, a global constraint such as limiting the cost of the product being configured within a particular value gives the user the flexibility to view a specific subset of scenarios rather than the whole gamut of configurations based on the constraint. Constraints help in defining the relationships much better as compared to rules.

5. **Performance**

Knowledge in constraint models is represented as basically a pool of variables with associated domain values and constraints at any particular instant in time. Thus when a system is to be solved, the entire system needs to be solved to prove consistency. With large data sets, this could hinder performance. This limits the size of the largest domain of a variable. In rules based system the search space reduces to a certain extent at each
step and entire datasets need not be solved. Once a variable value has been set it is not reconsidered in computation unless the user decides to change their selections.

2.6 Thesis Approach

The thesis serves as a way of describing a comprehensive list of requirements in a configuration process. This requirement analysis is motivated by the field work at TDCI. The thesis shows how existing configuration systems must be reengineered incrementally to improve user experience and aid users in the configuration process. It proposes an architecture on which the complex configuration processes should be based. This architecture helps in the design of a good interactive configuration process and focuses on the user requirements. It contains a list of desirable qualities for building configuration models. It also provides different conceptual representations of these models by integrating cognitive science theories to incorporate the user perspective. It supports the constraints based approach to solve the configuration models based on the advantages of constraints over rules that have been identified before. A solution approach called interactive constraint satisfaction to solve these configuration problems interactively in a real time environment is discussed in the last chapter in the form of a prototype. It shows why a dynamic constraint based configuration solution is important to the user.
Chapter 3. Related work

The previous chapter lists the advantages of using constraint based approaches as opposed to the rule based approaches. This chapter starts by discussing how constraint based models are depicted in the form of constraint satisfaction problems and constraint networks. It uses the traditional map coloring problem initially to explain these concepts. Then it illustrates a detailed example i.e. the configuration of a personal computer to explain these ideas from the point of view of a real configuration. Algorithms to actually solve the configuration models are discussed next. A category of algorithms called look-ahead algorithms is the most common approach used to solve constraint based problems. A more recent concept of solving constraint based configuration problems called interactive constraint satisfaction is reviewed after these algorithms and this concept is illustrated using a prototype in the last chapter.

The second part of this chapter provides an overview of the 4+1 architecture model [10] for depicting software intensive systems. This is important because this technique is used in the thesis to illustrate the architecture of Buy-Design, the TDCI product configurator that serves as the case study for requirement analysis of the process of configuration. 4+1 architecture model is an effective way to depict architectures as it covers the different aspects of software systems like the actual process to show system functionality, the deployment process, and the process from a developer perspective and so on. These views are presented as diagrams in the Unified Modeling Language (UML). Thus an
overview of the relevant UML diagrams is also discussed in this chapter after the 4+1 architecture model.

As discussed in the earlier chapter, the proposed configuration solution integrates conceptual views to depict configuration models. These views are based on architectural as well as cognitive science principles. So the last section of this chapter discusses some of the frameworks used in cognitive science to create effective visualizations that can help the user in the decision making process.

3.1 Ways of Representing Constraint Models

3.1.1 Constraint Satisfaction Problems (CSP)

A constraint based model is represented as a constraint satisfaction problem. A constraint satisfaction problem is defined by a set of variables, a set of values associated with those variables and a set of constraints that limit the combination of values of these variables. A solution to this problem is a set of values for these variables that satisfies all constraints. In other words, a solution is a way for assigning a value to each variable in a way that all constraints are satisfied.

3.1.2 Constraint Networks

A constraint network is an instance of a constraint satisfaction problem. It consists of a set of variables associated with a domain, containing a set of values that models the interrelationship between these variables and defines valid values for these variables. Mathematically constraint networks can be represented as a triple (V, D, C) where

a) V: Set of Variables

b) D: Domain or a Set of valid values.
c) C: Set of Constraints that determine the set of values that different variables take that can coexist with each other. For instance if (V1, V2, V3, V4, V5…Vn) are a set of variables and ( D1,D2,D3,D4,D5…Dn) are the domain values, then constraints C determine what combinations of D1 x D2 x D3 x D4 ……x Dn are valid. The domains are filtered based on constraints in the constraint network to derive a consistent state.

3.1.3 Example of a constraint satisfaction problem: Map Coloring Problem

Figure 5 depicts the map of Australia containing its different states. These states need to be colored using minimum number of colors such that the adjacent states are not represented in the same color. This problem is represented as a constraint satisfaction problem below.

![Figure 5: Map of Australia](image)

**Representation as a CSP**
1. **Variables (V):** The variables are the different states of Australia that needs to be colored. Thus, the variables here are *Western Australia (WT), Northern Territory (NT), Queensland (Q), New South Wales (NSW), Victoria (V), Southern Australia (SA), and Tasmania (T).*

2. **Domain (D):** Each of the variables can take the following values: *Red, Green, and Blue*

3. **Constraints (C):** Adjacent regions must have different colors.

   Mathematically:

   $V = \{WA, NT, Q, NSW, V, SA, T\}$

   $D(V) = \{\text{red, green, blue}\}$

   $C = \forall (X, Y) \text{ Adjacent } (X, Y) \rightarrow [\text{Color}(X) \neq \text{Color}(Y)]$

Figure 6 depicts a possible solution to the above problem.

![Figure 6: Solution of the Map Coloring Problem for map of Australia](image)

Figure 7 represents the problem as a constraint network.
3.1.4 Detailed Example – Configuration of a PC

Consider the modeling of a Personal Computer/Laptop with the help of a constraint network. This modeling facilitates easy configuration of the system. It helps in filtering the domain values of each of the variables based on user selection. The example below shows the construction of a constraint network and how it can be represented given the variables, domain values and constraints for the example of a PC.

Variables and Domain Values:

- Processor Speed: 1.66, 2, 2.6, 3 Ghz
- Hard Drive Capacity: 40, 60, 120, 160 GB
- Memory Capacity: 1, 2, 4, 8 GB
- Processor Type: AMD, Intel Dual 2 Core, Intel Dual 4 Core
- Warranty: Yes, No

Constraints

- C1: For Processor Speed € (1.66, 2) : Hard Drive capacity € (40, 60)
- C2: For Processor Speed €(2.6, 3) : Hard Drive capacity € (120, 160)
• C3: For Processor Speed $\in (1.66, 2)$: Memory Capacity $\in (1, 2)$

• C4: For Processor Speed $\geq 2.6$: Hard Drive Capacity $\geq 120$

• C5: Hard Drive Capacity $\in (120, 160)$: Memory Capacity $\geq 4$

• C6: For Processor Type $\in (\text{Intel Dual 2 Core, Intel Dual 4 Core})$: Processor Speed $\in (2.6, 3)$

<table>
<thead>
<tr>
<th>Processor Speed</th>
<th>Hard Drive Capacity</th>
<th>Memory Capacity</th>
<th>Processor Type</th>
<th>Warranty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.66</td>
<td>40</td>
<td>1</td>
<td>AMD</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>1</td>
<td>AMD</td>
<td>Yes</td>
</tr>
<tr>
<td>1.66</td>
<td>60</td>
<td>1</td>
<td>AMD</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>1</td>
<td>AMD</td>
<td>Yes</td>
</tr>
<tr>
<td>1.66</td>
<td>40</td>
<td>2</td>
<td>AMD</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>2</td>
<td>AMD</td>
<td>Yes</td>
</tr>
<tr>
<td>1.66</td>
<td>60</td>
<td>2</td>
<td>AMD</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>2</td>
<td>AMD</td>
<td>Yes</td>
</tr>
<tr>
<td>1.66</td>
<td>40</td>
<td>1</td>
<td>AMD</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>1</td>
<td>AMD</td>
<td>No</td>
</tr>
<tr>
<td>1.66</td>
<td>60</td>
<td>1</td>
<td>AMD</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>1</td>
<td>AMD</td>
<td>No</td>
</tr>
<tr>
<td>1.66</td>
<td>40</td>
<td>2</td>
<td>AMD</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>2</td>
<td>AMD</td>
<td>No</td>
</tr>
<tr>
<td>1.66</td>
<td>60</td>
<td>2</td>
<td>AMD</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>2</td>
<td>AMD</td>
<td>No</td>
</tr>
<tr>
<td>2.6</td>
<td>120</td>
<td>4</td>
<td>Intel Dual 2 Core</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>4</td>
<td>Intel Dual 2 Core</td>
<td>Yes</td>
</tr>
<tr>
<td>2.6</td>
<td>160</td>
<td>4</td>
<td>Intel Dual 2 Core</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>160</td>
<td>4</td>
<td>Intel Dual 2 Core</td>
<td>Yes</td>
</tr>
<tr>
<td>2.6</td>
<td>120</td>
<td>8</td>
<td>Intel Dual 2 Core</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>8</td>
<td>Intel Dual 2 Core</td>
<td>Yes</td>
</tr>
<tr>
<td>2.6</td>
<td>160</td>
<td>8</td>
<td>Intel Dual 2 Core</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>160</td>
<td>8</td>
<td>Intel Dual 2 Core</td>
<td>Yes</td>
</tr>
<tr>
<td>2.6</td>
<td>120</td>
<td>4</td>
<td>Intel Dual 2 Core</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>4</td>
<td>Intel Dual 2 Core</td>
<td>No</td>
</tr>
<tr>
<td>2.6</td>
<td>160</td>
<td>4</td>
<td>Intel Dual 2 Core</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>160</td>
<td>4</td>
<td>Intel Dual 2 Core</td>
<td>No</td>
</tr>
<tr>
<td>2.6</td>
<td>120</td>
<td>8</td>
<td>Intel Dual 2 Core</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>8</td>
<td>Intel Dual 2 Core</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 1: Constraint Tuples
<table>
<thead>
<tr>
<th>Processor Speed</th>
<th>Hard Drive Capacity</th>
<th>Processor Type</th>
<th>Warranty</th>
<th>Memory Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6, 2, 2.6, 3 (GHz)</td>
<td>40, 60, 120, 160</td>
<td>AMD, Intel Dual 2 Core, Intel Dual 4 Core</td>
<td>Yes, No</td>
<td>1, 2, 4, 8 (GB)</td>
</tr>
</tbody>
</table>

Table 1: Continued

<table>
<thead>
<tr>
<th>Processor Speed</th>
<th>Hard Drive Capacity</th>
<th>Processor Type</th>
<th>Warranty</th>
<th>Memory Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6</td>
<td>160</td>
<td>Intel Dual 2 Core</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>160</td>
<td>Intel Dual 2 Core</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>120</td>
<td>Intel Dual 4 Core</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>Intel Dual 4 Core</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>160</td>
<td>Intel Dual 4 Core</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>160</td>
<td>Intel Dual 4 Core</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>120</td>
<td>Intel Dual 4 Core</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>Intel Dual 4 Core</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>160</td>
<td>Intel Dual 4 Core</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>160</td>
<td>Intel Dual 4 Core</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>120</td>
<td>Intel Dual 4 Core</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>Intel Dual 4 Core</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>160</td>
<td>Intel Dual 4 Core</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>Intel Dual 4 Core</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>160</td>
<td>Intel Dual 4 Core</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>160</td>
<td>Intel Dual 4 Core</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Constraint Network of the Example

Figure 8: Constraint Network for personal computer example
3.2 Ways of Solving Constraint Models

3.2.1 Look Ahead Algorithms

Look-ahead algorithms attempt to foresee the effect of selecting a value for a variable on a set of dependent variables. These look at the effects of an assignment by evaluating its effects on the related variables before making the assignment.

3.2.1.1 Forward Checking

The simplest technique in the look-ahead category is Forward Checking. Forward checking looks at the assignment of variables at each step. Whenever a variable $V_i$ is assigned a value, the forward checking algorithm checks against the remaining $i-1$ variables to make sure that the assignment of the $i$th variable can still lead to a viable solution. Thus at each step, the partial solution is checked to ensure that valid assignments are still possible for the remaining variables.

3.2.1.1.1 Solving Map coloring problem using Forward Checking

Figure 9 illustrates how the map coloring problem is solved by using the forward checking approach.
Here WA is colored red. Next the effect of such an assignment on its neighbors is evaluated and consequently the domains of NT and SA are reduced. Since each variable still has a set of valid values associated with it, the assignment of red to WA is valid. Next when we select green for NSW, the domains of NT and SA are reduced to Blue. Again, since each variable has an associated set of valid values, this assignment is also considered as valid. Note however that this selection of values will not lead to a viable solution since NT and SA are adjacent and their only color choices are blue. This problem is undetected by forward checking. This is because forward checking only checks the immediate neighbors.

### 3.2.1.2 Arc Consistency Techniques

The other category of look-ahead algorithms is the arc consistency algorithms. These are more common and probably the better approach for solving configuration models. These algorithms can foresee the effect of the assignment by the user and the effect is propagated to all the variables. Thus it looks forward several steps as opposed to forward
checking that can see only the effect on the immediate neighbors. Thus the inconsistencies that could arise later in the selection process are resolved earlier. Here we discuss three variations of arc consistency techniques namely node consistency, arc consistency and path consistency. Node consistency is for those constraints that effect only one node. Such constraints are called *unary constraints*. Arc consistency is used for consistency checking between two variables. It ensures that all values for one variable are consistent with values of another variable if a constraint exists between them. Path consistency is an extension of arc consistency but it contains pairs of variables instead of involving only one variable. A pair of variables is path-consistent with a third variable if each consistent evaluation of the pair can be extended to the other variable in such a way that all *binary constraints* i.e. those involving two variables are satisfied.

3.2.1.2.1 Node Consistency

The easiest form of arc consistency is node consistency [11]. A node representing a variable V is said to be node consistent if for every value in the domain V, each unary constraint on V is satisfied. So if the values associated with a variable do not satisfy the unary constraint defined for that variable, then that value must be removed from the domain of the variable.

The following pseudo code explains the idea of node consistency where G is the set of variables in the system, V is a variable in G, D is the domain associated with the variable V and X is a value in the domain D.
Example to illustrate node consistency

For example, given a variable $V$ with a domain of \{1, 2, 3, 4\} and a constraint \{\(V \leq 3\)\} node consistency would restrict the domain to \{1, 2, 3\}.

3.2.1.2.2 Arc Consistency

Let $x,y$: \{ Set of variables\} and $C$: \{Set of Constraints\}

The constraint network consists of a set of nodes representing variables each associated with a set of values and the constraints represented as arcs between the variables.

A variable $x$ is arc consistent\[12\] with respect to “$y$” if for every value in the domain of $x$ there is a corresponding value in the domain of “$y$” such that $(x,y)$ is allowed by $C$ i.e. the set of constraints. The arc $(x,y)$ in a constraint network is consistent if $x$ is consistent with respect to $y$ and $y$ is consistent with respect to $x$. A network is consistent if all its arcs are consistent. In other words, arc consistency checks the mutual consistency between all the other unassigned variables.
3.2.1.2.3 Arc consistency in the Map Coloring example

The figure below shows a sequence of steps for enforcing arc consistency on the previously discussed map coloring example where:

\[ V = \{\text{Western Australia (WA), Northern Territory (NT), Queensland (Q), New South Wales (NSW), Victoria(V), Southern Australia (SA), Tasmania(T)}\} \]

\[ D (V) = \{\text{Red, Green, Blue}\} \]

\[ C = \forall (X, Y) \text{Adjacent (X, Y)} \rightarrow [\text{Color(X) ≠ Color(Y)}] \]

Let the arcs be considered in the following order:

a) SA→ NSW

b) NSW→ SA

c) V→ NSW

d) SA→ NT

![Figure 11: Arc Consistency](image-url)
Steps 1 to 4 illustrates how the domains are filtered based on the constraint such that once a color is picked for a state the adjacent states should have different colors. Due to propagation of constraints, we see that the domain of Southern Australia (SA) becomes empty and thus the assignments are shown to be invalid.

**3.2.1.3 Progression of Arc Consistency Algorithms**

This section describes the important arc consistency algorithms. Assume the following convention when reading about the algorithms;

- \( V_i, V_j \ldots V_n \): {Set of variables}

- \( G \): {Set of all arcs stored as \( \{V_i, V_j\} \) pairs}

- \( D_i, D_j, \ldots D_n \): {Domains associated with the variable \( V_i, V_j \ldots V_n \)}

**3.2.1.3.1 Arc consistency- 1 (AC-1)**

Arc \( (V_i, V_j) \) is arc consistent when for each value of \( V_i \) in its domain there is a corresponding value for \( V_j \) in its domain. The pseudo code in Figure 12 describes the first arc consistency algorithm [13].
The procedure REVISE shown above is used to check the consistency of an arc between two variables. If for an x in $D_i$, there is no y in $D_j$ then x must be deleted from the domain $D_i$. When the domain of a variable is reduced due to filtering based on constraints, arcs must be revisited. Because reduction in the domain of variable $V_j$ may lead to
inconsistency with \( V_i \), all arcs \((V_j, V_i)\) need to be revisited for reevaluation. Thus the procedure Revise must be executed each time the domain of a variable changes.

The disadvantage here is that all arcs are evaluated again even if they are not relevant although only a subset of the arcs should be revisited. To avoid this problem Arc Consistency 3 was developed.

### 3.2.1.3.2 Arc consistency – 3 (AC-3)

AC-3 [13][14] is the most frequently applied algorithm of all the arc consistency algorithms due to its simplicity of implementation. The pseudo code for the algorithm is specified as below.

**Procedure AC-3**

```plaintext
Q = \{(V_i, V_j) \in \text{arcs}(G) \mid \#j\};
While not Q empty
  Select and delete any \((V_k, V_m)\) from Q;
  If REVISE\((V_k, V_m)\) then
    Q = Q union \{(V_i, V_k) \text{ such that } (V_i, V_k) \in \text{arcs}(G), \#k, \#m\}
  Endif
Endwhile
End AC-3
```

*Figure 14: Arc Consistency-3*

### 3.2.1.3.3 Example to demonstrate Arc Consistency-3

The following arcs exist in the queue. Consider the following arcs in this order:

1. Processor → HD Capacity
2. HD Capacity → Memory Capacity
3. Memory Capacity → Network Capacity

4. Processor → Memory Capacity

Note: Each arc must be consistent in both directions

Consider the following scenario: If the value “2” is selected for Memory Capacity, then the network is affected in the following way:

- All arcs are visited. The domains are filtered based on the constraints that have been specified before.
- Each time a value from a variable is removed, all arcs to that variable are added in the queue so that they can be revisited later to check for consistency.
- The figure reduces to Figure 15 after filtering domains.

![Figure 15: Network after reduction of domains](image)
3.2.1.3.4 Arc Consistency 3.1

It extends AC-3 and achieves better performance, at the same time maintaining the simplicity of implementation. In order to avoid revisiting the same arc multiple times, Arc consistency 3.1 [15] introduces the concept of a support that it stored with the variable $V_i$ in the arc $(V_i, V_j)$.

The support value for a variable $V_i$, is a value in the domain of $V_j$ that is consistent with the value that has been selected for $V_i$ in the arc $(V_i, V_j)$. Arc consistency stores this support value called “Last” so that it does not have to revisit all values while evaluating an arc. This reduces the time complexity of this algorithm as compared to Arc consistency-3 and makes it a much more efficient algorithm in worst case. The pseudo code for this algorithm is shown below.

```
Procedure REVISE2001/3.1(xi; xj)
Begin
DELETE false
for each a belongs to Di do
    b <- Last((xi, a); xj)
    if b does not belong to Dj then
        b <- succ(b; Dj)
    while (b <> NIL) and (not cij(a, b)) do
        b <- succ(b; Dj)
        if b <> NIL then
            Last((xi, a); xj) <- b
    else
        delete a from Di
        DELETE true
return DELETE
end
```

Figure 16: Arc Consistency-3.1
3.2.1.3.5 Path Consistency

Path consistency [11] is a property similar to arc consistency, but considers pairs of variables instead of only one. A pair of variables is path-consistent with a third variable if each consistent evaluation of the pair can be extended to the other variable in such a way that all binary constraints are satisfied. Formally, $V_i$ and $V_j$ are path consistent with $V_k$ if, for every pair of values $(a,b)$ that satisfies the binary constraint between $V_i$ and $V_j$, there exists a value $c$ in the domain of $V_k$ such that $(a,c)$ and $(b,c)$ satisfy the constraint between $V_i$ and $V_k$ and between $V_j$ and $V_k$, respectively.

3.3 Interactive Constraint Satisfaction

Many real world applications, however, require interactive decision [16] support rather than automatic problem solving. This is the case where the initial constraint network admits many possible solutions and human guidance is needed to select a solution based on some additional criteria. These criteria cannot be modeled as constraints in the original network since they are not yet known; the user can only identify these criteria when consequences of the initial constraints are revealed. Constraint satisfaction problems work well in a static environment. Many filtering techniques for CSP’s have been developed, the most common ones being arc consistency. But real world problems need to be addressed for a highly dynamic environment. Almost all reasoning techniques deal with static CSP’s, so a new concept of dynamic CSP’s was developed and algorithms achieving arc consistency in a dynamic CSP were developed. DCSP’s are a common way of solving real configuration problems. The CSP definitions and algorithms presented in the previous chapter are for batch processing where the machine is intended to solve the problem autonomously.
Mathematic definition of dynamic constraint satisfaction problem:

A dynamic constraint satisfaction problem (DCSP) \[17\] is a sequence of static CSPs each resulting from a change in the preceding one imposed by "the outside world". This change can be a restriction (a new constraint is imposed on a pair of variables) or a relaxation (a constraint that was present in the CSP is removed because it is no longer interesting or because the current CSP has no solution). This is an extension of the concept of a static constraint satisfaction problem.

3.4 Architecture Description Techniques

3.4.1 The 4+1 architecture model

4+1 is a architecture view model designed by Philippe Kruchten\[18\] for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views. Thus this model is reviewed here to help in illustrating the Buy-Design architecture. Four views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers and project managers. The four views of the model are logical, development, process and physical view. In addition selected use cases or scenarios are utilized to illustrate the architecture, this use case view is the 'plus one' view. Hence the model contains 4+1 views:
Figure 17: 4+1 architecture model

- **Logical view**: The logical view is concerned with the functionality that the system provides to end-users.

- **Development view**: The development view illustrates a system from a programmer’s perspective and is concerned with software management. This view is also known as the implementation view.

- **Process view**: The process model deals with the dynamic aspect of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc.

- **Physical view**: The physical view depicts the system from a system engineer's point-of-view. It is concerned with the topology of software components on the physical layer, as well as communication between these components. This view is
also known as the deployment view. UML Diagrams to represent physical view include the Deployment diagram.

- **Scenarios**: The description of architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype.

The next explains the diagrams in UML that are used to describe the above views.

### 3.5 The Unified Modeling Language (UML)

UML stands for unified modeling language, is a standardized general-purpose modeling language in the field of software engineering. It is an open method used to specify, visualize, construct and document the artifacts of an object-oriented software-intensive system under development.

UML diagrams represent two different views of a system model:

- **Static (or structural) view**: Emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.

- **Dynamic (or behavioral) view**: Emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.
The diagrams that are used in the thesis to describe the architecture of Buy-Design configurator include the following:

1. Class diagrams: Describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.


3. Deployment diagrams: Serves to model the hardware used in system implementations, and the execution environments and artifacts deployed on the hardware.

4. Use Case diagrams: Shows the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.

5. Package diagrams: Depicts how a system is split up into logical groupings by showing the dependencies among these groupings.

### 3.6 Visualization

Understanding a configuration domain through good conceptual views is an important aspect of the configuration problem. This section explains the work in cognitive science relevant to visualization of graphs.  

The visualizations that have been compared in the sixth chapter are motivated partly by the work done in the field of cognitive science.

---

Cognitive science may be defined as the study of the nature of intelligence. It draws on multiple empirical disciplines, including psychology, philosophy, neuroscience, linguistics, anthropology, computer science, sociology and biology. Cognitive studies helps to capture the information model of human thinking and how humans interpret information.
3.6.1 Graphical Perception

Cleveland and Gill [19] have done useful research in the field of graphical perception and graphical methods for analyzing scientific data. Graphical perception can be defined as the visual decoding of information encoded on graphs and it includes both theory and experimentation to test the theory. The first step is the identification of elementary perceptual tasks that are carried out when people want to extract quantitative information from graphs. The second step is an ordering on the tasks based on how accurately people perform them. The main elementary perceptual tasks in the order from most to least accurate include:

- a. Positions along a common scale
- b. Position along nonaligned scales
- c. Length, direction and angle
- d. Area
- e. Volume, curvature
- f. Shading, color saturation

These elementary tasks have been selected based on how a person viewing the graph performs these tasks to extract the values of real variables from graphs. Interpretation based on these elementary tasks helps in better organization and increases the chances of correct interpretation of patterns and behavior. Simon and Hastie [20] have done experimental work to examine the expectations of what types of information people want in graphs. These suggest how graph types and judgment types interact to determine the speed and accuracy of information extraction.
3.6.2 Models for perception and understanding of graphs

a. UCIE[21]

There has been extensive work in the design of information systems so that people can perceive and interpret information. Lohse developed a computer program by the name of UCIE, known as Universal Cognitive Information Engineering to help in graphical perception. This was used to model the cognitive science processes so that people could decode information from a graph easily and much faster. To determine the usefulness of the graph, predictions such as the eye time fixations, short term memory capacity and duration limitations, and the relative level of difficulty to acquire information are made. The model can help in advising the construction of graphs based on the results collected from the above predictions.

b. Kosslyn Framework [22] [23]

Kosslyn has also done some work in the field of cognitive science to understand and interpret graphs and derive useful information. He developed a framework to analyze information so that design flaws in a display could be identified. The scheme examines four kinds of display types and specifies their structure at a syntactic, semantic and a pragmatic level. This structure is then checked for any violations based on the gathered information. He uses the ideas of human information processing, short term and long term processing of information to help in the analysis.

c. Construction Integration model [24]
Real world interpretation requires more than just doing data analysis. There are many factors that need to be considered if graphs are to be interpreted and understood. Knowledge based graph comprehension have a deep impact on the interpretation of graphs. Domain knowledge, graphical skills and explanatory skills all contribute towards graph comprehension. Prior knowledge guides the processing of visual features. Freedman and Shah propose a construction-integration model that helps in graph comprehension. There are two main phases in this model.

a) Construction phase – This phase takes place in two stages, a construction and a comprehension phase. In the construction part, the reader activates the available textual information and the prior knowledge associated with that. A coherent representation for this information is constructed. The prior knowledge provides the guidance to represent this information better.

b) Integration phase – Here the knowledge is combined into a coherent representation. This helps in drawing inferences, which is actually the useful part of the integration phase.
These frameworks have assisted us in deciding the properties for evaluating visualizations based on cognitive science principles for configuration related problems. They have helped us in deciding what is important to the user and what he perceives most when he looks at the visualization.
Chapter 4. Current Architecture of Buy-Design System

This chapter elaborates on the architecture of the Buy-Design product configurator designed by TDCI. Buy-Design is a guided selling and product configurator that can simplify many selling processes. The configuration engine of Buy-Design is the heart of the configurator. Since this engine is a guided selling and configuration engine, it prevents the user from making incorrect selections so that only valid combinations of features are ordered.

Figure 19 provides an example of the kind of configuration issues that this configuration engine has to deal with. Different configurations of a cabinet, with varied sizes and shapes are shown in the figure below. Some configurations may be able to fit inside another depending on the constraints that have been defined on the products. These configurations can get complex due to composite assemblies created from independent units as shown in the Figure 19. The artistic complexity of these shapes makes the configuration process tough and user can get confused while configuring a product.
The important characteristics of this configuration engine that can be enumerated as follows:

a) Product selection – helps users identify and select the product(s) that best fits their needs by providing attribute/requirement questions, results filtering, and match ranking capabilities.

b) Product configuration – This ensures that the customer makes a full valid configuration and configures a product with the right size, dimensions and features. Features include: multi-level configuration, dynamic presentation of options based on previous selections, global options for rapid configuration and mass change of related items, and more.

c) Product Pricing – Provides the capability to calculate an accurate price for a configuration. To ensure a flexible system, customers are provided with various options such as list price, discounted price, dealer cost etc. that enable flexible calculations for the price of a configuration.
d) Product visualization – Enables a visual representation of the configured product.

e) Dynamic user interface makes the entire user experience highly intuitive, productive, and personalized. The ready-to-use UI is dynamically-rendered based on the product model, user type, and task.

4.1 BuyDesign 4+1 architecture model

The 4+1 architecture model explained earlier is now used to explain the architecture of Buy-Design. The different views of this software system are illustrated in this section [25].

4.1.1 Process View

This view is illustrated using the three process figures below. The process views show the interactions taking in the system and how components interact with each other to solve the problem. Figure 20 describes the basic flow in a typical configuration process. The user enters the input parameters through the user interface. The rules engine in the backend uses configuration rules to configure items. The input parameters, existing rules and the decision to configure a new product or reconfigure existing products determines decide whether a previous configuration is to be reloaded or a new configuration is to be created.
4.1.1.1 The Basic Configuration Process

![Diagram of Basic Configuration Process]

**Figure 20: Basic Configuration**

4.1.1.2 The Interactive Configuration Process

This view is a subset of the previous view. It illustrates the process of interactive configuration when the user interacts with the user interface to configure a new product.
4.1.1.3 Post Configuration Process

This is also a subset of the first view. This illustrates the process of post configuration when existing products need to be reconfigured. The existing rulesets are loaded and the user makes the appropriate changes by interacting with the user interface.
4.1.2 Logical View

Figure 23 below depicts that part of the class diagram which is relevant to constraints.
The important concepts to keep in mind while looking at this view are the following:

1. **Constraint Trigger Rule**
   
   Consists of a variable and a specific property on that variable such that when the variable property changes it will cause the *Constraint Loop Rule Condition* to be evaluated.

2. **Constraint Loop Rule**
   
   a. Consists of a condition that evaluates to true or false
   
   b. Represents a condition or set of conditions evaluating the present state of one or more variables in the *Constraint Model*. 
c. Variables used in the condition may or may not be defined as a *Constraint Trigger Rule* for the condition.

d. Sets the limit to the maximum number of times the actions can be executed and the loop condition reevaluated.

3. **Constraint Action Rule**

   a. Consists of a set of variables and a specific property on that variable that will be updated to have a new value specified as part of the action.

   b. It is valid to define an action that affects a variable that is also part of the *Constraint Trigger Rule*. However during evaluation of the constraint any actions that would update the variable that triggered the will be ignored to prevent circular references and invalid state conditions.

4.1.3 **Development View**

This view illustrates the development view. It illustrates the folder structure used in the deployment of a Buy Design application through a package diagram. TDCI is the root folder under which the Framework and BuyDesign folders are designed. Under the Framework folder and BuyDesign folder, exists a BIN folder to describe the business objects and another folder describing the web objects.
4.1.4 Physical View

This figure illustrates the physical view of the product configurator by depicting the different components of the system present in different layers and how these components interact with each other. The flow starts at the client layer. The client (could be rich client or browser client) makes a call to the Web server that contains the host application, in Buy-Design this is the simulator. The user enters the input parameters. Based on the input parameters passed that includes the Application ID, a call is made to the configuration engine. A set of parameters in the form of rapid options are input to the configuration engine. These options are evaluated using the constraint engine. Based on the input parameters, the correct rulesets are loaded and the default interface is displayed. These rulesets are created in the Design Studio and are physically stored on the model server in
the database layer. There also exists the enterprise manager that stores instances of the input model server and the associated output server to store configuration outputs.

![Deployment diagram- Buy-Design](image-url)

Figure 25: Deployment diagram- Buy-Design
4.1.5 Scenarios

Figure 26: Use Case diagram- Buy-Design

Figure 26 illustrates the use cases in a configuration process. A rules developer/modeler is used to add, edit and remove rules. The rules engine that is a part of the configuration process is used to solve input rulesets, detect conflicts and give possible solutions to the user. The configuration user is responsible for configuring or re-configuring an item.

Based on the case study of the current Buy-Design architecture we have come up with the following solution approach to help in re-engineering the design of existing configuration tools incrementally so that they fill the gaps of a traditional configuration process. Traditionally the configuration process was based only on physical models that may be incorrectly built and are difficult to visualize conceptually and the users ended up making assumptions. The new approach called the Intelligent Assistant Architecture attempts to address this gap.

5.1 What is IAA?

IAA stands for Intelligent Assistant Architecture and is based on the principle that effective complex configurations require user participation in configuration processes that in turn requires a constraints based architecture focused on cognitive principles. Using constraint based approaches allows us to focus on those aspects of guidance that enable user decisions. Integrating cognitive science principles help to effectively showcase the configuration problems and their solutions by assisting the user in careful decision making.

Thus is it different from the existing configuration engine designs that do not consider this aspect of the problem. The new configuration process can be modeled as in Figure 27.
The architecture contains the following components:

a. A list of desirable properties for building correct configuration models that help in producing good outputs.

b. Effective views of configuration models to guide users and help them in decision making. The thesis provides a method for evaluating the effectiveness of the visualization based on a set of architectural properties and cognitive science principles. Such evaluations demonstrate the usefulness of a visualization and how it can be used to assist the user in effective decision making.

The first component is discussed in this chapter. The next chapter focuses on the conceptual views. Based on the case study, we also develop a prototype focused around
the idea of interactive constraint satisfaction. This approach attempts to solve configuration problems interactively using constraint based approaches due to reasons discussed in the second chapter. This can be considered as the first step in moving towards an interactive constraint based solution.

5.2 Desirable Qualities of Constraint Models

This sub section discusses three properties namely path invariance, cyclic permutations and reachability that are important for developing correct constraint based models.

5.2.1 Path invariance

A flexible configuration system allows a user to select the options in the order that they matter to him. Different users have different degree of importance attached to the variables in the system and may start at different points to configure the same product with the same combination of values. Thus a particular configuration must be derivable independent of the order in which the variables are selected. This property is termed as path invariance as the configuration does not depend on the path that the user takes to configure the same product.

Path invariance must also guarantee satisfiability. This means that no variable must be associated with an empty set of values in the problem set. If the user is directed through the system such that the order of selections produces an empty set, then such a solution is invalid or unsatisfiable. Constraint propagation can help in identifying unsatisfiable constraints.

Consider the personal computer example discussed in Chapter 3. Let us take the following constraints:
1. If (Processor Speed=2.6 Ghz) then Hard Drive Capacity =120GB, Processor Type =Intel Dual 4 Core, Memory Capacity=4 GB, Warranty =Yes.

2. If (Hard Drive Capacity =120GB) then Memory Capacity=3 GB, Warranty =Yes, Processor Speed=3 Ghz, Processor Type =Intel Dual 2 Core.

If the user selects the Processor Speed with a value of 2.6 Ghz then the configuration that would be derived would be

Processor Speed: 2.6 Ghz
Hard Drive Capacity: 120GB
Processor Type: Intel Dual 4 Core
Memory Capacity = 4 GB
Warranty: Yes

If the user selects the HD Capacity =120GB first then the second and third constraint are fired and we derive a different configuration that is

Processor Speed: 3 Mghz
HD Capacity: 120GB
Processor Type: Intel Dual 2 Core
Memory Capacity = 3 GB
Warranty: Yes

So if the user wanted to reach to a same configuration but could not due to different starting points then this means path invariance has not been achieved. A configuration should be derivable starting from any variable and should be independent of the order of
selection. This means that whether the value for Processor or the value for Hard Drive Capacity is selected first, the user must be able to reach the desired configuration. The configuration must also be satisfiable i.e. while reducing the domain values; a variable should not be associated with an empty set. Constraint propagation algorithms propagate constraints and thus can identify early in the selection process whether a particular selection will lead to an empty set or not.

5.2.2 Cycles

A cycle can arise based on user selection of values for different variables. User should not be allowed to go down a route leading to a cyclic condition. The configuration process must be intelligently designed to warn the user of such situations and providing them with alternatives.

A cycle exists if and only if the domains of two or more variables affect each other in a circular manner. In a configuration, cyclic conditions have to be avoided. If a set of variables affect each other in a cyclic manner, then the system must make arrangements to avoid such a state and user must be directed somehow to take an alternative route.

Consider the following scenario that could lead to a deadlock condition:
If the user selects values such that a cycle exists, the configuration engine should be able to detect such conditions and inform the user. It must prevent such cyclic conditions to take place by giving reasons and suggestions to the users of the configuration process and guiding them in the configuration process.

5.2.3 Reachability

In graph theory, reachability [26] is defined as the notion of being able to reach from one vertex in a directed graph to some other vertex. If the rules are defined such that a particular variable can never be accessed, then there may be bad rules in the model. A variation of this is that a particular element in the domain set of a variable can never be accessed, and then such a value is redundant and can be eliminated from the variable set. Constraint based configuration models should allow the user to reach all the variables. This property is termed as reachability.
Consider the following example to illustrate the problem of reachability.

If (Processor Speed=2.6 Ghz) then Hard Drive Capacity =120GB, Processor Type=Intel Dual 2 Core, Warranty = Yes else Memory Capacity=4 GB, Processor Type =AMD, Warranty = No.

A rules modeler may forget to specify rules to set Memory Capacity or Hard Drive Capacity independently. If the condition evaluates to true, then Hard Drive Capacity, Processor Type and Warranty are set otherwise Memory Capacity, Processor Type and Warranty are set. The modeler may forget to specify rules involving Hard Drive Capacity and Memory Capacity independently thus leaving unassigned variables in the system. A good constraint based configuration model guarantees reachability.
Chapter 6. Visualization

The second component of IAA is the creation of effective views of configuration models for the user. This chapter focuses on this aspect of configuration by providing different conceptual views of the configuration domain and then evaluating these views by integrating cognitive science and architectural principles together.

Let us refer back to the use cases that were described for a configuration system in the fourth chapter. In case of a constraints based architecture, the constraint developer would be responsible for adding, removing and editing constraints. Visualization of a configuration domain with all variables and their associated constraints help the developers in analyzing the existing domain better. If the domain can be represented visually, then it becomes easy for developers to add, edit and remove constraints as they will be able to see the effect of their incorporated changes. Drag and drop features for adding, removing constraints help to ease the task of a constraint developer.

The other set of use cases involves the configuration engine that solves configuration models, detects possible conflicts and provides alternate solutions to the user. Again, by looking at a configuration domain visually, conflicts can be detected by tracing the path of variable selection based on user constraints.

Thus the importance of visual configuration models is evident from the above examples. This chapter elaborates on the requirements for depicting these models. Different views...
of the same problem domain are created. These are then compared using two different approaches. The first approach, an architectural based approachdifferentiates them using a list of passive and active properties that have been identified for a good representation of a configuration domain. The other approach is the cognitive science focused approach, where the accuracy with which a set of questions about the configuration can be answered, determines the importance of the view. Integrating the cognitive science aspects with the architectural concepts helps to enrich the quality of the views thus improving user experience and makes the process of decision making easier for them.

Next we present a set of conceptual views that have been created using Microsoft Visio and a visualization tool called Startlight. The Visio representations are views from real industry domains. The Starlight visualizations have been created using the personal computer configuration example as the basis. These visualizations are then compared using the two approaches mentioned before. The thesis then proposes an integrated approach to evaluating the visualizations in the latter part of the chapter.
6.1 Conceptual Constraint Model Views

6.1.1 Visualization - 1

Figure 29: First Constraint Model for example domain-1
6.1.2 Visualization - 2

Figure 30: Second Constraint Model for example domain-1
6.1.3 Visualization - 3

Figure 31: Third Constraint Model for example domain-1
6.1.4 Visualization - 4

Figure 32: First Constraint Model for example domain-2
6.1.5 Visualization - 5

Figure 33: Second Constraint Model for example domain-2
6.1.6 Visualization - 6

Figure 34: Third Constraint Model for example domain-2
6.1.7 Visualization - 7

Figure 35: Link Array View
Figure 36: Network View
6.1.9 Visualization - 9

![Visualization Diagram]

Figure 37: Category View

6.2 Architectural Approach for Evaluation

Below is a list of passive and active properties that we have identified to compare the constraint networks. While the passive requirements are listed in [10] the active requirements have been identified based on our experience with configuration problems.
6.2.1 Passive properties

a. Fidelity - How closely is the system represented by the underlying model?

b. Consistency - How well can a good interface represent similar concepts?

c. Comprehensibility - How easy is it for the stakeholders to understand the constraint model?

d. Dynamism - How dynamic is the user interface in order to incorporate changes?

e. View Coordination - How well can this visualization be coordinated with other visualizations?

f. Extensibility - How easy is the user interface to modify to take on new capabilities?

g. Aesthetics - How pleasing is it to the users?

6.2.2 Active properties

a. Order Independence - The interface should allow the user to select variables independent of order during a configuration.

b. Provide guidance to the user during configuration i.e. support for flexibility -

The interface should help the user in making selections by giving useful suggestions. The configuration engine should be able to trace user path and allow the user to retract their previous selections if needed. It should allow locking and unlocking of variables to see the effects of changes in user selection. Locking of variables allows the user to fix certain domain values and analyze the effects due to change in other variables. This helps in dealing with global constraints such as minimization of cost.
2. Idiot proofing- The interface should have the ability to gray out values from lists rather than eliminate them completely so that all options are visible for selection at a later time if required.

6.2.3 Evaluation of visualizations

6.2.3.1 Evaluation – Visualization 1, 3

- Scope/Purpose
- Graphical view of the configuration domain
- User event/selection traces
- Basic Type
  - Graphical
- Depiction
  - Directed acyclic graph type network depicting a configuration model.
- Interaction
  - Point and click, drag and drop direct interactions with symbols augmented by menus and dialogs.
- Fidelity
  - Diagrams are canonical. It presents a simplistic view of the domain under consideration. It does not show the actual constraints. It just shows that which variable affect other variables but does not depict how one variable affects the other.
- Consistency
A small symbol vocabulary used to represent the model ensures consistency. Generally good, there could be exceptions.

- Comprehensibility
  - Easy to see causal relationships but difficult to understand why they’re there because it does not model the actual constraints in the system.
  - Also depends on the skills of the modeler and the use of symbols/patterns used to depict the actual visualization.

- Dynamism
  - Animation, use of point and click interfaces, drag and drop mechanisms all help in incorporating dynamism. It can get difficult if the domain complexity increases.

- View coordination
  - Depends on the graphical editor.

- Aesthetics
  - Simple unadorned directed acyclic graph type network consisting of nodes representing variables and edges representing constraints.

- Extensibility
  - Tool set is effectively a ‘black box’. New variables, relationships can be added but how they are handled depends on the available graphical tools.

### 6.2.3.2 Evaluation – Visualization 2, 4

- Scope/Purpose
  - Graphical view of the configuration domain
  - User event/selection traces
• Basic Type
  o Graphical

• Depiction
  o Directed acyclic graph type figure depicting constraint networks.

• Interaction
  o Point and click, drag and drop direct interactions with symbols augmented by menus and dialogs.

• Fidelity
  o Diagrams are canonical. It presents a simplistic view of the domain under consideration. This model also does not show the actual constraints. It just shows that which variable affects what other variables but does not depict how one variable affects the other.

• Consistency
  o A small symbol vocabulary used to represent the model ensures consistency.

• Comprehensibility
  o Easy to see causal relationships but difficult to understand why they’re there because it does not model the actual constraints in the system.
  o Also depends on the skills of the modeler and the use of symbols/patterns used to depict the actual visualization.

• Dynamism
Animation, use of point and click interfaces, drag and drop mechanisms all help in incorporating dynamism. It can get difficult if the domain complexity increases.

- View coordination
  - Depends on graphical editor.

- Aesthetics
  - Simple directed acyclic graph type network consisting of nodes representing variables and edges representing constraints. Uses color to represent different constraints that add to the appeal of the figure. It is less rigid and has less cross-over lines representing relationships as compared to the first visualization. The first model has lines that can only be at right angles to one another thus reducing the aesthetic appeal of the figure.

- Extensibility
  - Tool set is effectively a ‘black box”. New variables, relationships can be added but how they are handled depends on the available graphical tools.

6.2.3.3 Evaluation – Visualization 5, 6

- Scope/Purpose
  - Graphical view of the configuration domain
  - User event/selection traces

- Basic Type
  - Graphical

- Depiction
  - Directed acyclic graph type picture depicting constraint networks
• Interaction
  o Point and click, drag and drop direct interactions with symbols augmented by menus and dialogs.

• Fidelity
  o It represents the underlying domain more closely as it represents the actual constraints in the system. The interface can be designed to show the actual relationship (“And” versus “Or”) that exists between variables.

• Consistency
  o A small symbol vocabulary used to represent the model ensures consistency.

• Comprehensibility
  o It is easy to see the actual relationships in the system. It can show which variables are the driving variables and which variables are the driven variables in the system. Again comprehensibility also depends on the skills of the modeler and the use of symbols/patterns used to depict the actual visualization.

• Dynamism
  o Animation, use of point and click interfaces, drag and drop mechanisms all help in incorporating dynamism. It can get difficult if the domain complexity increases. Handling dynamism is more difficult in the 3rd model due to more information being represented and more re-arrangements being required.

• View coordination
• Depends on graphical editor.

• Aesthetics
  o Simple directed acyclic graph type network consisting of nodes representing variables and edges representing constraints. Uses color to represent different constraints that add to the appeal of the figure. It is less rigid and allows for re-arrangement to reduce the number of cross over lines that exist to represent relationships. It can get complicated due to more information being depicted on the picture.

• Extensibility
  o Tool set is effectively a ‘black box’. New variables, relationships can be added but how they are handled depends on the available graphical tools.

6.2.3.4 Evaluation – Visualization 7

• Scope/Purpose
  o 3D view of the configuration domain
  o Possible sets of solutions based on variable selection

• Basic Type
  o Graphical Network

• Depiction
  o Called a link array view showing the different variables, their values and solution sets that exist between them based on user selection.

• Interaction
  o Point and click interface, symbols augmented by menus and dialogs, color coding schemes to show user selections.
• Fidelity
  o It shows the variables in the configuration domain and plots the associated values. If the user selects a particular value, then it can show the different solutions that exist. Thus this figure models the domain in terms of available solution sets based on user selection.

• Consistency
  o A small symbol vocabulary used to represent the model ensures consistency.

• Comprehensibility
  o Easy to see the possible solution sets after user makes certain selections.
  o Depends on the use of symbols/patterns and good color coding schemes used to depict the actual visualization.
  o Can help the user in decision making in case of inconsistent selections.
  o Understanding solution sets may become complex in case of large sets but can model subsets efficiently.

• Dynamism
  o Animation (Availability of a 3D view), use of point and click interfaces, help in incorporating dynamism. It can get difficult if the domain complexity increases.

• View coordination
  o Depends on the graphical editor.

• Aesthetics
  o Good use of color coding mechanisms to look at the solutions.
3D view available to look at the visualization from different angles.

- **Extensibility**
  - New variables, values can be added and imported easily.

### 6.2.3.5 Evaluation – Visualization 8

- **Scope/Purpose**
  - 3D view of the configuration domain
  - Network showing valid combinations of values for variables based on user preference.

- **Basic Type**
  - 3D Graphical Network

- **Depiction**
  - Called a network view showing the different variable values in forms of nodes and valid combinations between them in the form of edges.

- **Interaction**
  - Point and click interface, symbols augmented by menus and dialogs, color coding schemes to show different edge types.

- **Fidelity**
  - It shows the variables in the configuration domain and plots the associated values.
  - It can show the different solutions by connecting edges between variable values if the combinations are valid.
  - Thus it models the domain in terms of the available solution sets in the form of connected edges in a network.
• Consistency
  o A small symbol vocabulary used to represent the model ensures consistency.

• Comprehensibility
  o Easy to see the possible solution sets between the different users selected variables in the form of a network.
  o Depends on the use of symbols/patterns and good color coding schemes used to depict the actual visualization.
  o Can help the user in decision making during configurations.
  o Understanding solution sets may become complex in case of large sets but can model subsets efficiently.

• Dynamism
  o Animation (Availability of a 3D view), use of point and click interfaces, selection and de-selection of edges help in incorporating dynamism. It can get difficult if the domain complexity increases.

• View coordination
  o Depends on the graphical editor.

• Aesthetics
  o Good use of color coding mechanisms to look at the solutions.
  o 3D view available to look at the visualization from different angles.

• Extensibility
  o New variables, values can be added and imported easily.
6.2.3.6 Evaluation – Visualization 9

- Scope/Purpose
  - 3D view of a subset of the configuration domain
  - Possible sets of solutions between value sets of two variables.

- Basic Type
  - 2D view of a subset of a configuration domain

- Depiction
  - Called a category view showing the variables and the associated values for the other variable in the form of clusters.

- Interaction
  - Point and click interface, symbols augmented by menus and dialogs, color coding schemes to depict different variables.

- Fidelity
  - Uses cluster kind of figure to show the combination of values of different variables.
  - Helps to model a subset of the domain.

- Consistency
  - A small symbol vocabulary used to represent the model ensures consistency.

- Comprehensibility
  - Easy to see the possible solution sets between two variables.
  - Depends on the use of symbols/patterns and good color coding schemes used to depict the actual visualization.
Can help the user in decision making.

Understanding solution sets may become complex in case of large sets but can model subsets efficiently.

- **Dynamism**
  - Animation use of point and click interfaces, help in incorporating dynamism.

- **View coordination**
  - Depends on the graphical editor.

- **Aesthetics**
  - Good use of color coding mechanisms to look at the configuration.

- **Extensibility**
  - New variables, values can be added and imported easily.

### 6.3 Cognitive Science Approach

The second approach used to compare visualizations is cognitive science focused. Here we try to use cognitive science principles to interpret and compare the visualizations. Based on the experience with constraint models, below are listed some properties that can be used to compare the conceptual views that have been created.

1. **Order Of Variables On The Constraint Network (Top Down Versus Bottom Up)**

It is easy to visualize a constraint network top down rather than bottom up. This means it is easier to analyze how one variable affects another by placing the driving variable above the driven variable rather than in the opposite direction. By placing the driving
variables above the driven variables, the relationships on the networks flow top down as opposed to bottom up, making it easier for the user to understand the domain.

2. **Color Coding Schemes**

It is easier to interpret visualizations if they have some form of color coding schemes associated with them. A simplistic view without any color scheme makes it difficult to interpret visualizations.

3. **Positioning of variables/Rearrangement of variables**

The positioning of variables on the visualization affects its readability. The variables that are closely related to one another should be naturally placed closer together to give more meaning to the network being considered.

4. **Lines representing relationships, their angles to one another and their directions**

The more the information, the more difficult it is to model the domain. As complexity increases, the number of variables and the number of relationships increase manifold and hence the pictorial representation becomes too cluttered. This leads to lines representing relationships to cross each other and reduces the comprehensibility of the visualization being considered.

5. **Prior Domain knowledge**

Having prior knowledge about the domain helps to decipher and interpret a constraint network more easily as compared to a complete novice who has no understanding of the domain and is looking at the picture for the first time with no prior domain knowledge. Also apart from this, the graph reading skills of humans determine how well a person is able to comprehend a diagram and derive useful results from it.
6.3.1 Evaluation of Visualizations

We can compare the views through a set of questions that we would like to answer by looking at the view. Below are listed some of these questions. This list is not comprehensive but contains some of the most important questions that need to be addressed. By determining the accuracy with which the questions are answered, the representations can be compared. The lesser time it takes to answer a question, the more effective the visualization.

6.3.1.1 List of questions

1. Which variables are affected due to change of a variable? Or in other words what are the driven variables for a particular driving variable?

   **Importance:**
   
   • This is a common question that needs to be asked from the user perspective. It helps a user to analyze which variables would be affected from their selection.
   
   • Relating back to the use cases, knowing what the affected variables are can assist a constraint developer in adding, removing and editing constraints.
   
   • This question can be answered by looking at the first six visualizations that are shown previously.

2. How can we determine the most critical variables in the domain? Or in other words which variables are most affected by the set of constraints in the domain?

   **Importance:**
   
   • Again, this question is important from the constraint developer perspective. It helps them to identify the critical variables in the system. The user path can be traced in case of abnormal behavior, thus it helps in debugging.
• This question can be answered by looking at the first six visualizations that are shown previously.

3. What are the conflicting paths in the view and do variables need to be added/removed?

**Importance:**

• This is important from the constraint developer point of view. It helps to identify the design issues in the domain. There could be conflicting paths in the configuration model that can be identified by looking at the conceptual views.

• This question can be answered by looking at the second and the fourth visualization.

4. What kind of relationship exists between variables i.e. for instance is it an “And” or an “Or” relationship that relates the variables? For instance, if we have a constraint like (If “X=2” AND “Y=3”) then “Z=6”, most representations would represent a direct relation between (“X and Z”) and (“Y and Z”) without showing the “AND relationship. So, if constraints are modeled as a part of the diagram as depicted in the third visualization, then the actual relationship that exists between the variables can be identified.

**Importance:**

• This is important from the user/constraint developer perspective as it can trace user selection and thus help in debugging in case of abnormalities.

• This question can be answered by looking at the third and sixth visualization.
5. What are the specific values for the set of variables and what are their valid combinations?

**Importance:**

- This is important when the user wants to know the possible routes that he can take based on his selection. This is to find the specific values for a set of variables that are compatible with his selection.
- This question can be answered by looking at the starlight visualizations.

The degree of accuracy with which these questions are answered determines the usefulness of the visualizations keeping the different actors namely the configuration user and constraint modeler in mind.

### 6.4 Integration of the two approaches

The figure below depicts whether cognitive science properties identified for better visualization of constraint networks, affects the visualization properties such as fidelity, consistency, comprehensibility etc positively (+), negatively(-) or in a neutral way(N).
<table>
<thead>
<tr>
<th>PROPERTY(1st Approach)</th>
<th>PROPERTY(2nd Approach)</th>
<th>Ordering of variables – Top down</th>
<th>Color Coding Schemes</th>
<th>Positioning of variables</th>
<th>Lines, their angles and directions</th>
<th>Prior domain knowledge and graph reading skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fidelity</td>
<td></td>
<td>+</td>
<td>N</td>
<td>+</td>
<td>N</td>
<td>+</td>
</tr>
<tr>
<td>Consistency</td>
<td></td>
<td>+</td>
<td>N</td>
<td>N</td>
<td>_</td>
<td>N</td>
</tr>
<tr>
<td>Comprehensibility</td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Dynamism</td>
<td></td>
<td>N</td>
<td>N</td>
<td>_</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>View Coordination</td>
<td></td>
<td>N</td>
<td>+</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Extensibility</td>
<td></td>
<td>_</td>
<td>_</td>
<td>+</td>
<td>N</td>
<td>_</td>
</tr>
<tr>
<td>Aesthetics</td>
<td></td>
<td>+</td>
<td>+</td>
<td>N</td>
<td>+</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 2: Integration of Properties from two comparison approaches

Ordering of variables top down rather than bottom up has a positive effect on the fidelity, consistency, comprehensibility and aesthetics of a constraint view representation. The ordering of variables by the top-down approach helps in making the view easier to read and understand. It adds to the appeal of the figure as it is more intuitive for the user. It affects the fidelity positively as it is easier to determine what variables affect what other variables in such an organization. But this top down approach could be disadvantageous when the number of variables and relationships increase due to more information being represented on the view.
Color coding schemes have a positive effect on the aesthetic appeal of a view and increase its comprehensibility. But it could affect the extensibility negatively due to lot more information representation thus necessitating the use of additional color coding schemes.

Positioning of variables based on logical reasoning increases the fidelity of the system and helps in understanding the domain better. It also makes the system extensible as the variables are grouped together based on some reasoning making it easier to add or remove variables later on. Positioning of variables in a certain way may make the view rigid in terms of incorporating changes to the user interface, thus affecting its dynamism negatively.

The angles and directions of lines also help to reduce the amount of clutter in a view and make it more comprehensible, when a user is trying to answer the questions we have discussed previously. The lines, their angles and directions need not be consistent across all views. Thus it may potentially affect the consistency in a negative way.

Prior domain knowledge and better graph reading skills help in deciphering the network in a more time effective way. But again it could limit the extensibility if it is given high importance.

6.4.1 Useful Insights from the integration

1. The more the positive effect of a cognitive science property on a visualization property, the higher the importance of that cognitive science property. Thus it helps in ordering cognitive science properties based on greater number of + signs in a column.
2. It helps in identifying which properties have been handled well by cognitive science and which are yet to be tackled. Thus it helps in identifying potential areas of research in the cognitive science area.
Chapter 7. Interactive Constraint Satisfaction

There are many techniques for solving product configuration problems. Some of the techniques are discussed in [27]. This chapter focuses on an approach called interactive constraint satisfaction problems to solve complex product configuration problems. This approach is motivated due to the interactive nature of configuration problems that depend largely on user participation. This approach is proposed based on the analysis of the current architecture of Buy-Design product configurator and the experience gathered by working on configuration problems. This chapter provides an approach of solving configuration problems using the method of interactive constraints by means of a prototype.

Constraints for a configuration problem can be solved using interactive constraint satisfaction problems [28]. This chapter illustrates a prototype that demonstrates how interactive means can be used to resolve constraints by adding dynamic user imposed constraints to a set of initial constraints.

When the user is interacting with the user interface, user selection for a variable can be described as a dynamic constraint that is added to a set of initial constraints. These constraints have a higher priority than the existing constraints as they are a part of the user selection process during the configuration of a product.

Sample XML files representing variable input structure for the configuration of a bike.
7.1 XML input set for variables and associated values

This XML file shows the list of variables and associated set of values that serve as input while constructing a configuration model.

```
<?xml version="1.0" encoding="UTF-8"?>
<Variables>
  <Variable>
    <Name>Gear</Name>
    <ValueSet>
      <Value>1</Value>
      <Value>2</Value>
      <Value>3</Value>
    </ValueSet>
  </Variable>
  <Variable>
    <Name>Speed</Name>
    <ValueSet>
      <Value>40</Value>
      <Value>60</Value>
      <Value>80</Value>
      <Value>100</Value>
    </ValueSet>
  </Variable>
  <Variable>
    <Name>Engine</Name>
    <ValueSet>
      <Value>2</Value>
      <Value>4</Value>
    </ValueSet>
  </Variable>
</Variables>
```

Figure 38: XML file for variables

7.2 XML input set for Initial Constraints

This XML shows the initial list of constraints that serves as input while constructing a configuration model. The XML file here shows only a single kind of constraint called “FilterValueConstraint” that is used to filter domain values based on variable selection. Similarly other kinds of constraints like LessThanConstraint, GreaterThanConstraint etc can be defined in the XML file to handle different scenarios of filtering domains.
7.3 Solving the configuration model

The process of solving a configuration

1. The configuration models are constructed based on the initial XML files consisting of a set of variables and values and the set of constraints defined on those variables. This forms the knowledge database to construct configuration models.
2. The user interacts with the user interface to configure products. When the user makes a selection during the configuration process, the selection is added as a dynamic constraint to the existing set. The dynamic constraints have a higher priority than the initial constraints and are placed higher up in the order while solving for constraints.

3. Each time the user makes a selection, the constraints are solved by the configuration engine and the solution is presented to the user. This means that the variables are populated with the compatible values based on a set of constraints that exist in the system.

4. If the selection is inconsistent then the user is warned of the wrong selection. If the selection of the user changes for a particular variable, then existing constraints are removed to the system and new constraints can be added. Thus this method allows the user to retract from their previous choices.

Thus, this chapter presents an approach for solving the configuration problems in a dynamic way and gives higher priority to the user selection. If this method is incorporated for solving constraints along with proposed architecture, it can greatly improve the quality of configuration systems for the user.
Chapter 8. Conclusion and Future Work

The thesis provides a means of extensive requirement analysis of the configuration process by using the field work at TDCI as a case study. The study of the TDCI product configurator helped in gaining insights to the complexity of the configuration process and the problems associated with it. Standard software engineering techniques were used for the architecture description that helped in analyzing the TDCI system closely and thus identify the improvement areas.

From the study, it is proved that the complexity of the configuration domains necessitates the need for a user based interactive configuration process that can solve the identified problems effectively. The thesis covers the requirements for such a process by describing an architecture termed as Intelligent Assistant Architecture that motivates the shift from the product to actual process configuration. Firstly, it provides a list of desirable properties to help in the building of correct configuration models. Secondly, it looks at how to display these constraint based configuration models from a user perspective by incorporating cognitive science principles so that these views are effective and convey maximum information.

Based on this architecture, recommendations were made for incremental improvements to the existing configuration engine. Firstly, the architecture proposes the use of a dynamic constraint based engine in addition to the existing rules engine to give more flexibility to
the user and make the system more powerful. Secondly, it also motivates the need for conceptual views to assist the user in the selection making process.

The thesis also provides an integrated approach to evaluate visualizations using both architectural as well as cognitive science principles. The integration of the two approaches clearly highlights those areas of visualization that cognitive science has not yet addressed and need to be researched to come up with better and more effective visualizations. So in future, these areas can be explored in order to make the visualizations more complete.
References


[10] Chapter 5 AI Book


[21] Jerry Lohse, A cognitive model for the perception and understanding of graphs, Cognitive Science and Machine Intelligence Laboratory.


[25] Buy design development guide, TDCI

[26] Luis Quesada, Solving Constraint Graph problems using reachability constraints based on transitive closure and dominators, 2006.
