Feature Rules in TOPSS™

A Manuscript
Submitted to
the Department of Computer Science
and the Faculty of the
University of Wisconsin-La Crosse
La Crosse, Wisconsin

by
Linda Kay Ingalls
in Partial Fulfillment of the
Requirements for the Degree of
Master of Software Engineering
December 2005
Candidate: Linda Kay Ingalls

We recommend acceptance of this manuscript to the College of Science and Allied Health in partial fulfillment of this candidate’s requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Oral Examination Committee Chairperson
Dr. William Wehrs

Examination Committee Member
Dr. Kasi Periyasamy

Examination Committee Member
Dr. Tom Gendreau

Examination Committee Member
Dr. David Riley

Director, University Graduate Studies
Dr. Vijendra Agarwal
ABSTRACT

INGALLS, LINDA, K., “Feature Rules in TOPSS\textsuperscript{TM},” Master of Software Engineering, December 2005, Advisors: Dr. Kasi Periyasamy and Dr. Tom Gendreau.

This document describes the development of a new data maintenance tool and the enhancement of a product selection tool at Trane, a subsidiary of American Standard Companies, in LaCrosse, Wisconsin. The product selection tool uses several business rule types that assist a salesman to configure a product. The enhancement adds two new rule types to the existing rule base. The data maintenance tool allows input and maintenance of the two new rule types along with supporting data elements which may be used independent of or in conjunction with the two new rule types. These data items are cached and used as input to the product selection tool.
ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my project advisors, Dr. Kasi Periyasamy and Dr. Tom Gendreau, for their instruction and guidance during this project. I would also like to thank the project sponsor, Trane, a subsidiary of American Standard Companies, for initiating the project, and for providing computing resources, tuition reimbursement and supporting materials for this project. At Trane, I would specifically like to express my gratitude to Brad Goetz, Manager of Product Selection Software, and Bill Johnson, Vice President of Global Distribution Systems, for their flexibility and support of this project and for giving me the opportunity to work on this project and to complete it as part of my work responsibilities. I would also like to thank my colleagues on the TOPSS™ team for their excellent work, ideas, and technical expertise in making Project R a success. Finally, I would like to thank my husband, Scott, children, Ethan and Justin, and my extended family for their love, encouragement, patience, understanding, and extra work during the tenure of this project.
# TABLE OF CONTENTS

1. Background Information ........................................................................................................1
   2. Software Life Cycle Overview ...........................................................................................5
3. Feature Rule Types ..................................................................................................................7
   3.1 Field Classifications ........................................................................................................8
   3.2 Feature Dependencies ......................................................................................................9
   3.3 Feature Size Rule ...........................................................................................................11
   3.4 Feature Placement Rule ..................................................................................................14
4. Development of PUMA .........................................................................................................18
   4.1 Requirements Gathering, Analysis, and Specification ......................................................18
   4.2 Design ..........................................................................................................................19
   4.3 Implementation, Testing, and Integration .......................................................................21
5. Development of Feature Rules in TOPSS™ .....................................................................22
   5.1 Requirements Gathering, Analysis, and Specification ......................................................22
   5.2 Design ..........................................................................................................................22
   5.3 Implementation, Testing, and Integration .......................................................................23
6. The PUMA Application .......................................................................................................26
   6.1 General Architecture ......................................................................................................26
   6.2 Detailed Architecture .....................................................................................................27
   6.3 Database Design ............................................................................................................31
   6.4 Graphical User Interface ...............................................................................................34
   6.5 Deployment ...............................................................................................................................
   6.6 Limitations ......................................................................................................................37
   6.7 Continuing Work .............................................................................................................37
7. Feature Rules in TOPSS™ ...................................................................................................39
   7.1 General Architecture of TOPSS™ ...............................................................................39
   7.2 Detailed Architecture of TOPSS™ ...............................................................................41
   7.3 Detailed Design ...............................................................................................................43
   7.4 Database Design .............................................................................................................49
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Border Dependency Examples for Feature Placement Rule</td>
<td>16</td>
</tr>
<tr>
<td>2. Feature Rule Functionality Differences between Phase 1 and Phase 2</td>
<td>48</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1. Feature size and position on module face</td>
<td>7</td>
</tr>
<tr>
<td>3.2. Field Classification Data Element Relationship Diagram</td>
<td>9</td>
</tr>
<tr>
<td>3.3. Feature Dependency Data Element Relationship Diagram</td>
<td>10</td>
</tr>
<tr>
<td>3.4. Feature Rule Data Element Relationship Diagram</td>
<td>14</td>
</tr>
<tr>
<td>6.1. General Architecture of PUMA</td>
<td>26</td>
</tr>
<tr>
<td>6.2. Detailed Architecture of PUMA</td>
<td>28</td>
</tr>
<tr>
<td>6.3. Class Diagram for Feature Size and Placement Rule Class Library</td>
<td>29</td>
</tr>
<tr>
<td>6.4. Class Diagram for Feature Dependency Class Library</td>
<td>30</td>
</tr>
<tr>
<td>6.5. Database Schema for PUMA</td>
<td>33</td>
</tr>
<tr>
<td>6.6. PUMA Login Window</td>
<td>34</td>
</tr>
<tr>
<td>6.7. Window Interaction Map for Feature Rules</td>
<td>36</td>
</tr>
<tr>
<td>7.1. General Architecture and Application Integration of TOPSS</td>
<td>41</td>
</tr>
<tr>
<td>7.2. Detailed Architecture of TOPSS</td>
<td>43</td>
</tr>
<tr>
<td>7.3. TOPSS Classes Diagram for Feature Rules</td>
<td>46</td>
</tr>
<tr>
<td>7.4. TOPSS Class Diagram – Rule Manager and Supporting Classes</td>
<td>47</td>
</tr>
<tr>
<td>7.5. Database Schema for Feature Dimensions &amp; Coordinates</td>
<td>49</td>
</tr>
<tr>
<td>7.6. TOPSS Module Face Editor</td>
<td>50</td>
</tr>
</tbody>
</table>
GLOSSARY

Border
The clearances specified for the top, bottom, left and right edges of a feature.

Clearance
The fixed, imaginary area adjacent to an edge of a feature which defines the minimum distance that the feature’s edge must be positioned from an adjacent feature or module edge.

Citrix
A shared server environment that can be logged into by multiple users who each can run instances of the applications that are installed on the server in their own user session.

Database Replication
The monthly rake-off of product data and rules from the Oracle Enterprise database to the Sybase sales office databases.

Feature
A unique refinement of a product category sometimes referred to as an SI. For example, a feature may be either a left-hand or right-hand door, which are both refinements of the product category called door. A feature has a child relationship to a product category. A product category may include one or more features. A feature inherits the sizeable and moveable properties of a product category.

Module
A section of a unit. Each module is represented as a cube and has 6 module faces (front, rear, top, bottom, right, left).

Module Face
One of the surfaces of a module. Each module face has four edges (top, bottom, left, right).

**Project R**
Large software development project that allows selection of a TCCA unit in TOPSS™. The feature rule functionality referred to in this document is a sub-project to Project R.

**PUMA**
New data entry application developed as part of this project to support the data entry requirements for feature rules.

**TCCA**
Acronym for a new flexible, modular climate changer product family that supports sizeable and positional features.

**TOPSS™**
Acronym that represents the Trane Official Product Selection System, a Windows program that contains information to select and predict performance of Trane products operating under various conditions.

**Trane**
A subsidiary of American Standard Companies and the project sponsor.

**TrapMan**
A data entry maintenance tool that connects to the Oracle Enterprise database and provides the ability to create and maintain product data, rules, security, units of measure and many other types of information that is used by a variety of applications throughout the company.
1. Background Information

As a result of the ever changing business environment, and the desire and requirement of business to maintain a competitive edge, companies continue to develop new and better products and focus on the customer. Trane, a subsidiary of American Standard Companies, is such a company, committed to producing quality heating and air conditioning products and continually improving its service and products for customers.

Trane recently began to manufacture a new custom air handling product, called the Trane Custom Climate Changer, also referred to as TCCA. This product is expected to elevate Trane to a higher position among its competitors and is the product on which this project focuses. Compared to other climate changer products that Trane manufactures, where features like doors and openings have a single size and the position of a feature is static for a given configuration, TCCA will offer more flexibility to the user, not only in how a unit can be configured, but also in selecting the size and setting the position of features.

Most of the products that Trane sells can be configured and selected using a sales tool application called TOPSS™, which is the Trane Official Product Selection System. TOPSS is a Windows program that contains sufficient information to select and predict performance of Trane products operating under various conditions. Once the equipment is selected, dimension drawings, pricing, fan curves, schedules and reports can all be generated in TOPSS. Various tools are also available within the TOPSS including mixed air calculators and bid forms. TOPSS can also be fully customized for each individual work environment. TOPSS is primarily used by sales engineers and sales assistants within Trane, but is distributed free to all Trane customers [11].

TCCA is considered to be a “modular” product, meaning that a unit comprises a varying number of modules, such as fans, coils, and air mixing boxes. Each unit is comprised of modules selected by the user while being guided by different types of business rules that ensure a valid unit configuration. In TOPSS™, the user essentially
“builds” the desired unit graphically by adding, deleting, and moving different module types in the unit until the desired configuration is obtained. Each module has the shape of a cube. Thus, each module has six module faces on which features, such as doors or dampers, can be located.

In 2004, Trane initiated a project called “Project R” to enable TCCA to be configured and selected in TOPSS, and to be supported by other sales tools applications in the ordering process [1]. Due to the complexity of the TCCA product and how it would need to be selected and configured by the user, TOPSS would require a number of extensive changes, some of which included graphical changes, new views, and the ability to add, modify, delete, size, and position features on a unit based on business rules. As a subproject to Project R, the developer was given the responsibility to define two new business rule types – one rule type to define the available sizes for a particular option on a given module face, and the other rule type to define how close features could be placed to other features or to the edges of a module face. Since the new business rule types apply to features, the new rules were dubbed “feature rules”, more specifically, feature size rules and feature placement rules. These feature rules would prevent the user from incorrectly sizing and positioning features on the unit. Unique to the TCCA selection program, the user would have the ability to size a feature and change the position of it on a module face, subject to the feature rules [3]. The feature size rules define which sizes of the feature are valid, and the feature placement rules define how close the feature can be to the edge of the module face and to other features which also reside on the module face [2]. Once a TCCA unit is configured and selected, it was the developer’s responsibility to save and update the coordinates and dimensions of all sizeable or positional features for the unit on the database, and then to load those same coordinates and dimensions when the unit is reloaded from the database.

The developer was also given the responsibility to create a new data entry tool to allow the feature rules to be created and defined by the factory. The tool is expected to be supported within the TOPSS development team which will make maintenance and enhancements easier and quicker to implement than attempting to schedule changes with the team who supports the current data entry tool. Product selection data that is
maintained in the current data maintenance tool will be migrated to the new data entry
tool as time, priorities, and resources allow in the future.

The project described in this document focuses on the following areas: the definition
of the feature rules, the new data entry tool to support the creation and maintenance of the
feature rules, and the application of the business rules within TOPSS™ as a user
configures and selects a TCCA unit. As part of the selection of TCCA, a graphical user
interface would need to be developed to allow the user to size and move features, and it
would be supported by the feature rule application logic as defined in this document. The
development of the graphical user interface is not reported in this document because it is
not the responsibility of the developer.

The implementation of Project R was planned in two Phases. Phase 1 would provide
the ability to select a TCCA unit in TOPSS. For purposes of the developer’s
responsibilities, this required the completion of the data entry tool, called PUMA, which
would support the creation and maintenance of feature rules and supporting data, such as
field classifications and feature dependencies, and for TOPSS it would require the
application of the feature rules to support the graphical user interface that allows a user to
size and move features on a unit. Phase 2 would include enhancements to the feature
rules, the modification to PUMA to support the enhancements, modifications to the
feature rule application logic to support the enhancements, and modifications to the
functionality that saves feature dimensions and coordinates in the unit. The
aforementioned changes would be made available to a limited, select, trained group of
users following Phase 2, with a full release of the TCCA selection program following
future phases of the project.
2. Software Life Cycle Overview

The development of PUMA and the feature rule functionality in TOPSS™ were completed using two different software development life cycle models. This section provides a brief overview of each of the models.

The waterfall model is a simple process model in which the phases of development are organized and completed in a linear order [5]. One phase is completed before the next phase is started. The model includes the requirements phase, design phase, coding phase, testing and integration phase, and installation phase, being completed in order as listed. Each phase produces an output which is considered complete and static for the remainder of the project. For example, the output from the requirements phase is the Software Requirements Specification (SRS), the output from the design phase is the Design document, the output from the coding phase is the program, the output from the testing phase is a test plan, and the output from the installation is a completed, deployed application. A variation of the waterfall model allows the developer to return to a previous phase to make modifications based on the feedback, and changes made in the current phase.

The second approach is to use the prototype model. Once an initial set of requirements is defined, the design, coding and development of a prototype of the application are begun. Although this model contains the same phases of development as the waterfall model, the phases are completed more quickly and less thoroughly. By using this prototype, the client can get an actual feel of the system, because the interactions with the prototype can enable the client to better understand the requirements of the desired system [5]. Following the initial phase of development of the prototype, the iterative process of evaluating the prototype, defining and changing requirements, and incorporating the changes into the prototype continues. The development process may stop for a number of reasons: if the prototype contains all of the desired functionality it may be accepted as a completed application; the prototype may be discarded because it
doesn’t meet the customer’s needs or it has proven that the approach is not a viable solution; the prototype gives a clue that the chosen approach is appropriate for the product under development.
3. Feature Rule Types

This section provides an overview of the feature size and placement rule types and the supporting data types that are incorporated as part of the rules, namely field classifications and feature dependencies.

To understand how a feature is sized or positioned on a module face, consider Figure 3.1. This figure illustrates a module, as pictured by a cube, and a feature which is placed on the right face of the module (according to air flow). A feature is shown with the dimensions to size and position the feature. The dotted rectangle shows an imaginary border that TOPSS™ will use to ensure that the feature is positioned with sufficient space from adjacent features and module edges [2].

![Figure 3.1. Feature size and position on module face](image)
3.1 Field Classifications

This section describes the classification of fields across all products.

Different products can be configured using the same types of fields, such as a door. For example, product A and product B, although very different, may both be configured with doors that provide access into the unit. Similarly, the fields themselves may be very different. For example, three different types of doors may be configured onto the same unit, yet each is still generically considered to be a door. This concept of generically classifying fields is the basis for a field classification. A field classification is a generic identifier that is used to group similar fields, both within a product family and also across product families.

The data elements contained in a field classification along with the relationship of each data element are shown below and in Figure 3.2:

- **Classification name:** The name of the feature classification.
- **Classification description:** The description of the feature classification.
- **Unique identifier:** Unique identifier for the feature classification.
- **Status:** The status of the feature classification.
- **Display Sequence:** The order in which a feature classification is listed with respect to the other feature classifications.
- **Metafile name:** A reference to the file that contains the icon that represents the feature classification.
- **Last modified user id:** The user id of the last user that modified the size rule.
- **Last modified date:** The date and timestamp of when the feature classification was last modified.
The long term goal is to have each factory assign a field classification for every field in a product family. Due to time and resource constraints, the field maintenance functionality in TrapMan couldn’t be modified for this project to allow the assignment of a field classification to a field. As a short-term workaround, a field classification was added as a data element to both the feature size and placement rule types. Since each rule applies to only one field, and each rule contains one field classification, the field to field classification relationship could be established.

### 3.2 Feature Dependency

A feature dependency is a set of features that are associated with a particular module and/or any non-positional module that is defined as part of the product data. Once created, a feature dependency may be referenced in one or more feature size or placement rules for the product with the purpose of defining which borders or sizes are used based on whether the unit being evaluated contains the features that are included in the feature dependencies. The use of feature dependencies in a feature size or placement rule provides the ability to dynamically determine the size and position of features based on the features configured in the unit at the time the rules are evaluated.

By creating feature dependencies independent of the feature size and placement rules, the feature dependencies can not only be used and reused in different feature rules, but they can also be used in the future when adding or replacing other functionalities in TOPSS™ where the same feature dependency is needed.

The data elements contained in a feature dependency along with the relationship of each data element are shown below:
- Name: The name of the feature dependency.
- Unique identifier: Unique identifier for the feature dependency.
- Target Module Identifier: Unique identifier of the module to which this feature dependency applies.
- Status: The status of the feature dependency.
- Last modified user id: The user id of the last user that modified the feature dependency.
- Last modified date: The date and timestamp of when the feature dependency was last modified.
- Set of Features: The set of features contained in the feature dependency.

Figure 3.3. Feature Dependency Data Element Relationship Diagram

**3.3 Feature Size Rule**

A feature size rule defines the available sizes of features for specific module faces. Depending on which module face a feature is located, the set of available sizes may vary.

A feature size rule consists of three levels. The general rule information defines the field to which the rule applies, such as a door, along with the rule name, description, and field classification. The second level of the rule, called the size specification, defines the
unique features, module faces, and feature dependency combinations for which sizes will be defined. For example, if a door is available with both right hand and left hand options on only the top, left, and right module faces, one size specification may be created to define a set of sizes that would apply to all of these combinations, or multiple size specifications may be created for each combination that requires a different set of sizes.

The third level is the set of sizes that pertain to a size specification. Dimensions may be defined as individual values, a range or a range with an increment. For example, a left hand door on a right module face may have a height of 5-10”, a width of 30-40” with increments of 2, which indicates 30, 32, 34, 36, 38 and 40 are valid widths, and a depth of 1”.

The addition of a feature dependency to a size specification means that the sizes will only be applicable if the unit being evaluated contains the features defined in the feature dependency. For example, if a size rule contains a specification with a feature dependency of a unit size of 10, and the unit being evaluated has a unit size of 10, the specification will be selected and the set of sizes for that specification will be applied to the feature. Thus, if multiple specifications are set up for the same feature and module face combination, one specification may be selected depending on both the feature dependency that is assigned to the size specification and on the features that exist on the unit at the time it is being evaluated.

The following data elements define the three levels of a feature size rule and Figure 3.4 shows the relationship for the same:

Level 1: General Rule Information

- Rule name: The name of the rule.
- Rule description: The description of the rule.
- Field: Field to which the rule pertains.
- Field classification: Unique field classification identifier.
- UOM: Unit of measure.
- Debug flag: Indicator to output debug information when this rule runs.
- Last modified user id: The user id of the last user that modified the size rule.
- Last modified date: The date and timestamp of when the rule was last modified.
• Rule status: The status of the rule.
• Rule sequence: The order in which the rule is evaluated with respect to the other size rules with the same product category but different status.
• Rule identifier: Unique identifier for the rule.
• Set of size specifications: Each size specification consists of a set of one or more features and a set of one or more module faces to which the set of sizes applies.

Level 2: Size Specification Detail:

• Set of features: The set of features to which the set of sizes applies.
• Set of module faces: The set of module faces to which the set of sizes applies.
• Feature Dependency: Unique feature dependency identifier.
• Set of Sizes: The available sizes for the features when located on any of the module faces in the set of module faces.

• Rule message: A message relayed to the user if the rule fails.
• Specification status: The status of the size specification.
• Specification sequence: The order in which this size specification is evaluated with respect to the other size specifications in the rule.
• Specification identifier: Unique identifier for the specification.

Level 3: Size Detail:

• Set of Dimensions The height, width and depth dimension of a size may be specified in any of the following combinations:
  1) As a set of actual dimension values.
     An example of this would be a feature whose actual height could be 46, 50, 53 or 58 inches.
  2) As a size range, where a minimum value, maximum value and increment value are specified.
     An example of a size range for the height would be a minimum height of 12 inches, maximum height of 24 inches with an increment of 2 inches, meaning that the valid heights would be 12, 14, 16, 18, 20, 22 and 24.
• Size sequence: The order in which a size is evaluated with respect to the other sizes in the size specification.
• Size status: The status of the size.
3.4 Feature Placement Rule

A feature placement rule is similar to a feature size rule except that, instead of sizes, it defines the border limitations that must be enforced around a feature with respect to the top, bottom, right and left edges of a module face and to the adjacent features on the same module face. A feature placement rule also consists of three levels – the general rule...
information, the placement specification(s), and the set of borders that pertain to each placement specification.

In Phase 1, only one border could be defined for a placement specification. The dimensions of the border define the minimum distance between both the feature and the module edges, and the distance between the feature and all adjacent features on the same module face. Using the same distance for both module edges and for all adjacent features did not provide the amount of accuracy needed; so this functionality was expanded for Phase 2. For example, one adjacent feature may be valid at 3” from the feature while another adjacent feature must be 5” from the feature to be valid.

In Phase 2, borders were enhanced with border dependencies which allowed multiple borders to be defined for a placement specification. This enabled the factory to define border distances for module face edges and adjacent features by assigning dependencies to each border to identify in what case(s) the border should be applied. Border distances can further be specified for all adjacent features or for groups of adjacent features, based on the field classification of the feature. For example, a border could be set up to apply to only those adjacent features that have a field classification of “Door”. Another border may be set up to apply to only those adjacent features that have a field classification of “Opening”. A third border could be defined to specify all field classifications. Depending on the rank of these three borders, the border that specifies all field classifications would be used for all adjacent features whose field classification was not “Door” or “Opening”. Table 1 below illustrates the flexibility this provides.
Example #1:
Placement Specification

<table>
<thead>
<tr>
<th>Top</th>
<th>Bottom</th>
<th>Left</th>
<th>Right</th>
<th>Rank</th>
<th>Status</th>
<th>Border dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>2&quot;</td>
<td>2&quot;</td>
<td>2&quot;</td>
<td>2&quot;</td>
<td>1</td>
<td>Current</td>
<td>module edges and all field classifications</td>
</tr>
</tbody>
</table>

Example #2:
Placement Specification

<table>
<thead>
<tr>
<th>Top</th>
<th>Bottom</th>
<th>Left</th>
<th>Right</th>
<th>Rank</th>
<th>Status</th>
<th>Applies to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2&quot;</td>
<td>2&quot;</td>
<td>2&quot;</td>
<td>2&quot;</td>
<td>1</td>
<td>Current</td>
<td>module edges</td>
</tr>
<tr>
<td>3&quot;</td>
<td>3&quot;</td>
<td>4&quot;</td>
<td>6&quot;</td>
<td>2</td>
<td>Current</td>
<td>All Field Classifications</td>
</tr>
</tbody>
</table>

Example #3:
Placement Specification

<table>
<thead>
<tr>
<th>Top</th>
<th>Bottom</th>
<th>Left</th>
<th>Right</th>
<th>Rank</th>
<th>Status</th>
<th>Applies to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2&quot;</td>
<td>3&quot;</td>
<td>2&quot;</td>
<td>3&quot;</td>
<td>1</td>
<td>Current</td>
<td>module edges</td>
</tr>
<tr>
<td>3&quot;</td>
<td>3&quot;</td>
<td>3&quot;</td>
<td>3&quot;</td>
<td>2</td>
<td>Current</td>
<td>Field Classifications (201, 202)</td>
</tr>
<tr>
<td>5&quot;</td>
<td>5&quot;</td>
<td>5&quot;</td>
<td>5&quot;</td>
<td>3</td>
<td>Current</td>
<td>Field Classifications (203, 204)</td>
</tr>
<tr>
<td>4&quot;</td>
<td>4&quot;</td>
<td>5&quot;</td>
<td>5&quot;</td>
<td>4</td>
<td>Current</td>
<td>All Field Classifications</td>
</tr>
</tbody>
</table>

Table 1. Border Dependency Examples for Feature Placement Rule

The following data elements define a feature placement rule and Figure 3.3 shows the data element relationship diagram for the same:

Level 1: General Rule Information

- Rule name: The name of the rule.
- Rule description: The description of the rule.
- Field: Field to which the rule pertains.
- Field classification: Unique field classification identifier.
- UOM: Unit of measure.
- Debug flag: Indicator to output debug information when this rule runs.
- Last modified user id: The user id of the last user that modified the placement rule.
- Last modified date: The date and timestamp of when the rule was last modified.
- Rule status: The status of the rule.
- Rule sequence: The order in which the rule is evaluated with respect to the other placement rules with the same product category but different status.
- Rule identifier: Unique identifier for the rule.
- Set of Placement Specifications: Each placement specification consists of a set of one or more features and a set of one or more module faces to which the set of borders applies.
Level 2: Placement Specification Detail:

- Set of features: The set of features to which the set of borders applies.
- Set of module faces: The set of module faces to which the set of borders applies.
- Feature Dependency: Unique feature dependency identifier.
- Set of Borders: The available borders for the features when located on any of the module faces in the set of module faces.
- Rule message: A message relayed to the user if the rule fails.
- Specification status: The status of the placement specification.
- Specification sequence: The order in which this placement specification is evaluated with respect to the other placement specifications in the rule.
- Specification identifier: Unique identifier for the specification.

Level 3: Border Detail

- Set of edges: Four edges of the feature (top, bottom, left, right).
- Set of clearances: A clearance is the minimum distance that the edge of a feature must be from another feature or module edge. A clearance is defined for each edge.
- Border rank: The order in which a border is evaluated with respect to the other borders in the placement specification.
- Border status: The status of the border.
- Set of border dependencies: The settings that define which field classifications and/or module edges this border applies.
4. Development of PUMA

The data entry tool, PUMA, was developed using a combination of both the waterfall model and the prototype model for Phase 1, and the waterfall model for Phase 2. This section describes the various phases of development of PUMA.

4.1 Requirements Gathering, Analysis, and Specification

When the project began, the expectations for the new business rule types were summarized in the Project R Specification document [1]. From that point onward, the developer met with the project team -- product engineers for TCCA and the project manager for Project R -- who provided the requirements for the business rules, or feature rules. The project team held requirements meetings so the developer could learn about the TCCA product, understand how and when the features rules would be applied, and identify architecture and data elements for each feature rule type.

As part of documenting the requirements, the developer created a Data Element Relationship Diagram, as shown in Figure 3.3, of all data elements that would comprise a feature size rule and a feature placement rule. This visual depiction of the rule types proved to be very helpful for discussions and for verifying that the rule types were correct and complete, and that they contained the necessary data elements to be applied as defined in the requirements. The Data Element Relationship Diagram was also reviewed in a discussion that previewed requirements for Phase 2 of the project to ensure at a high level that the rule types would support the anticipated enhancements for Phase 2.

A prototype of the data entry screens that would be used to create the feature rules was created. This prototype contained the proposed screen designs and linkage between the screens for the feature rule creation. This prototype was presented to the Project R project team and those individuals in the factory who would be creating the rules. The screen designs in the prototype were approved. Due to time constraints imposed by Project R, the prototype was created by another member of the Project R team who
worked very closely with the developer to design the screens and to determine the screen flow.

Based on the prototype, the Data Element Relationship Diagram and the documented requirements from the project meetings, the developer created a requirements document which followed the Institute of Electrical and Electronics Engineers (IEEE) Standards on Requirements Specification [8]. The requirements document for Phase 1 was reviewed and approved by the project team.

The requirements, analysis and specification for Phase 2 of PUMA followed the same process as described above for Phase 1. First, the requirements were defined as to how the feature rules were to be enhanced, and then the requirements were defined for PUMA to support the rule changes. The prototype and Data Element Relationship Diagram were updated and another requirements document in IEEE format was created by the developer to document the PUMA requirements for Phase 2.

4.2 Design, Coding, and Testing

Although the original intent was to use the waterfall model for Phase 1, the design of PUMA followed the prototype model. Due to the time constraints imposed by the Project R schedule and due to the request of the executive management team to see tangible results and progress for Project R, it was decided that the design, coding and testing of PUMA would be done in three phases. PUMA would be developed in three different prototype phases, where the culmination of the development would produce a version of PUMA with the complete set of functionalities. The final PUMA prototype would be the completed application, at least for Phase 1 of Project R.

The prototypes were defined as follows: Prototype 1 would consist of the design, coding and unit testing of Field Classifications, the generic categorization of features which is also a data element in both feature size and feature placement rule types. Completing the Field Classification functionality in PUMA was the logical first starting point and would be stepping stone for the data entry creation and maintenance of the feature rule general information and specification functionality, which would be Prototype 2. Prototype 3 would complete the size and border portion of the feature rule functionality.
A number of benefits were realized from following the prototyping approach. The three different prototypes provided logical milestones for the development of the data maintenance tool. For each prototype, the developer created an object-oriented design document and then completed the coding and unit testing. Following the completion of each prototype, the Quality Assurance team began their testing of the prototype while the developer worked on the next prototype. The completion of each prototype produced a more complete version of PUMA that represented tangible progress for this project and for Project R.

As part of the design for Phase 1, the developer created class diagrams and database schema diagrams. The classes were implemented by the developer, and the database diagrams and schema request forms were submitted to the Database Administration team to implement the schema changes that would store the field classifications, feature size rules, feature placement rules, and the size and position of features on a TCCA unit.

The majority of the coding was completed by the developer, but due to the time constraints and size of the project, some coding was completed by two team members who worked closely with the developer and followed the design that had previously been approved.

For Phase 2 of the PUMA development, the waterfall model was used because the enhancements to the feature rules were not as extensive as in Phase 1 and the time constraints were not as tight. The developer modified the class diagrams to reflect the changes for Phase 2, updated the database schema diagram, and created another object-oriented design document. Following the completion of coding and unit testing, the new version of PUMA for Phase 2 was turned over to the Quality Assurance team for further testing.

### 4.3 Implementation

For Phase 1 and Phase 2, the implementation of PUMA would not be expected to be released in a production environment. Two versions of PUMA are available in a Citrix test environment, which is accessible by everyone on the Project R team. “PUMA Test”

---

1 A shared server environment that can be logged into by multiple users who each can run instances of the applications that are installed on the server in their own user session.
is the latest version of the PUMA application and contains recent functionality changes. Whenever a new version of PUMA is built, it is made available in “PUMA Test”. Once the “PUMA Test” version is tested and approved by the Quality Assurance team, it is copied to the “PUMA Production” version, which represents the most stable, production-ready version of the application. As changes and bug fixes are made to PUMA, the iterative cycle of using the “PUMA Test” and then “PUMA Production” versions continues. The factory uses the “PUMA Production” version to create and maintain field classifications and feature rules for the Project.

In an upcoming phase, yet to be determined, one of the requirements will be to release “PUMA Production” into a production environment. Since PUMA requires Oracle 9 to be installed, and individual personal computers throughout Trane do not yet have the Oracle 9 installation, the project to include Oracle 9 in the distributed architecture would be a large undertaking by those teams in Trane who support the infrastructure. This could not be achieved during the timeframe allowed for Project R and will be planned for in a future phase of Project R.
5. Development of Feature Rules in TOPSSTM

The feature rule functionality in TOPSS™ was developed using the waterfall model for both phases of development. Although many areas of TOPSS needed to be modified for Project R, this section describes only the phases of development for the application of feature rules in TOPSS along with the load/save logic of the feature dimensions and coordinates for Phase 1 and Phase 2 of the project.

5.1 Requirements Gathering, Analysis, and Specification

The requirements gathering process to determine how feature rules would be applied in TOPSS was done in conjunction with and in the same manner as the requirements gathering that was done in Phase 1 to define the feature rules and the requirements for PUMA as documented in section 4.1. The proposed modifications for the TOPSS GUI were discussed in detail and these discussions identified specific functionality that the feature rule functionality would need to provide to the GUI [4]. The feature size and placement rule types were defined from the combined results of the requirements process as defined in section 4.1 and as a result of the requirements gathering for TOPSS.

For Phase 1, the developer created a requirements document in IEEE format that documented the changes that would be made to TOPSS to apply the feature rules in the selection process. Another requirements document was created to define the feature rule enhancement rules for TOPSS. Both requirements documents were reviewed in detail several times by the project team and were approved.

5.2 Design

Since TOPSS is an existing application, the design of the feature rules in TOPSS was approached much differently than the design of new application PUMA. The design was approached by the developer from the perspective of “How can the application of feature rules best be incorporated in the existing TOPSS functionality, yet be as independent and self-contained as possible?” The design decision was made to create a Feature Rule Manager as a sub-manager within the general Rule Manager so the feature rules would be
accessible like other rule types via the Rule Manager, yet would be relatively independent and self-contained within the Feature Rule Manager.

Two class diagrams were created for the design. One class diagram documented the existing TOPSS™ classes that were affected by the project and the modifications that would be needed to those classes, including a new member variable of the Feature Rule Manager in the Rule Manager class. Another class diagram specifically documented the Feature Rule Manager and its supporting feature rule classes.

The database schema change to add the table attributes that would store a feature’s dimensions and coordinates are defined in Figure 7.5.

For Phase 2, since the feature rule manager was already defined in TOPSS, the design was easier to complete than the design in Phase 1 because the changes were contained within the Feature Rule Manager.

In each of the two phases, an object-oriented design document was created by the developer to document the proposed changes to the TOPSS application. The design was reviewed by several of the developers on the TOPSS team and was approved.

5.3 Coding, Testing, and Implementation

The coding of the feature rule manager was done in sections, to support the simultaneous development of the GUI, which needed to call various functions within the Feature Rule Manager. Once coded, the functionality was tested both by the developer and by the GUI developer.

As in all projects for TOPSS, the developer’s unit testing is followed by testing by the Quality Assurance team to verify that the functionality works properly and as stated in the requirements. The Quality Assurance team was pleased and appreciative of the thorough requirements document that was created by the developer as it made their work of creating testing plans easier.

Since the Project R changes for TOPSS™ were not intended to be released to the sales force until Phase 3, yet code changes needed to be made for both Phase 1 and Phase 2, the decision was made by the project team to implement the Project R changes in the TOPSS source code by surrounding the code changes with a unique preprocessor definition. By using the preprocessor definition, builds of TOPSS™ could continue to be made for on-going release changes and for Project R, and the Project R changes could be
excluded from a build by removing the preprocessor definition from the build parameters. Thus, using this technique, on-going development and bug fixes not related to Project R could continue to be made and released to the sales force while still supporting the development of Project R. The Feature Rule Manager changes were made in TOPSS using the preprocessor definition.

For Phase 1 and Phase 2, separate Project R builds of TOPSS to use for testing were created using the preprocessor definition. As with PUMA, two versions of TOPSS were available in the same Citrix test environment as PUMA, so that the test versions of TOPSS could also be accessed by everyone on the Project R team. “TOPSS Project R” is the latest Project R test version of TOPSS and contains recent functionality changes and bug fixes. Once the “TOPSS Project R” version is tested and approved by the Quality Assurance team and by the (TCCA) Product Manager at the factory, and is deemed a stable version. It is then copied to the “TOPSS Project R Prod” version, which represents the most stable, production-ready version of TOPSS. As changes and bug fixes are made to TOPSS for Project R, the iterative cycle of using the “TOPSS Project R” and then “TOPSS Product R Prod” versions continues. The (TCCA) Product Manager uses the “TOPSS Project R Prod” version to test and select TCCA units to assist with the pre-release orders of TCCA.

The developer worked with the Citrix team to setup the various versions of TOPSS in Citrix. Since TOPSS can be run using different settings as defined in the TOPSS.ini file, and because the team needed a convenient way to test TOPSS using various application settings, the developer created a number of TOPSS desktop icons that would trigger command files and launch TOPSSTM using master topss.ini files containing the appropriate settings. This testing approach in Citrix ensures that all Project R team members are testing the application using the same environment and the same settings, and ultimately reduced the amount of time that would normally be spent setting up and maintaining the test environments for each project team member.

The testing setup of “TOPSS Project R” and “TOPSS Project R Prod” in Citrix will likely be used for future phases of Project R. However, since the Project R functionality and TCCA selection program is scheduled to be released in a production environment to a limited portion of the sales force in August, 2005, the “TOPSS Project R Prod” version
will be the build that is tested and released as part of the normal release schedule for August, 2005.
6. The PUMA Application

This section presents the general and detailed architecture of the data entry tool, PUMA, in addition to the database design, graphical user interface, deployment, limitations and continuing work on the application.

6.1 General Architecture

PUMA was developed using the Microsoft .NET framework and it is written in C#. An installation of the PUMA executable resides on a Citrix server. As each user signs onto the Citrix server, a separate user session is created for each user. Within that user session, the user can run an instance of the PUMA and other applications that are available on the Citrix server. The PUMA application is a simple, data entry application that retrieves and stores information to an Oracle database based on the Add, Update and Delete requests or commands requested by the user. The general architecture of PUMA is presented in Figure 6.1.

![Figure 6.1. General Architecture of PUMA](image)

6.2 Detailed Architecture
The PUMA application is structured as a collection of libraries, where each library contains a set of classes. Figure 6.2 illustrates the detailed architecture of PUMA. At the highest level, the PUMA application contains the user login functionality and the main menu which connects to the various functional areas of the application. Each functional area is divided into two libraries – a user interface library, which contains all of the classes used in the user interface, and a class library, which contains the application classes that provide the bridge between the user interface and the database for that functional area.

Each class library contains a database manager which is derived from a base class database manager in the database utilities library. The database manager contains the functions that select, update, and delete information to and from the database specific to that functional area. When data is retrieved from the database, the database manager formats the data into an XML format, which is then returned to the class library. This approach provides a seamless interface from the application class to the database manager and removes database specific commands from the application classes.

Figure 6.2. Detailed Architecture of PUMA
Each user interface library contains the window designs and user interface classes specific to a functional area. To promote re-use of controls and to provide a common look and feel to the application, a global user controls library is used to store controls that may be accessed by any of the user interface libraries.

Besides providing a well-organized application that is easy to maintain, this architecture provided the benefit of being able to access a function in one library from another library. For example, the Field Classification library contains a function to retrieve a Field Classification from the database using its unique database identifier. The same function was used in the Feature Size and Placement Rules class library to retrieve and evaluate the Field Classification associated with a feature rule to determine whether the Field Classification’s status would support a feature rule status change from Pending to Current.

Figure 6.3 and 6.4 present the detailed class diagrams for the Feature Dependency functionality and the Feature Size and Placement Rule functional areas in PUMA.
Figure 6.3 Class Diagram for Feature Size and Placement Rule Class Library
Figure 6.4. Class Diagram for Feature Dependency Class Library
6.3 Database Design

The database design for field classifications, feature dependencies, and feature size and placement rules is based on the data element relationship diagrams as shown in Figures 3.2, 3.3, and 3.4 respectively. Although the data maintenance of these items is done in separate areas of PUMA, the database diagram in Figure 6.5 shows how closely integrated they are. In the design, each database table is represented as a rectangular box that lists the table name, primary key (PK) and each attribute on the table. The lines connect the tables to show foreign key (FK) relationships between the tables. Attributes and combinations of attributes that are commonly used for retrieving data are noted as indexes (I).

Nine new tables were created and implemented in the Oracle production database for Phase 1. Three additional tables were created and three of the original tables were modified in Phase 2 to store the feature dependency and border dependency enhancements. Data changes to these tables continue to be replicated monthly to approximately 132 sales office Sybase databases.

All functional areas of PUMA use the pessimistic approach to locking records for update. This ensures that only one user can update a particular database record at a time. When data is in edit mode, the relevant database record(s) is locked for update. When a record is locked, another user can read the record but cannot update the row. If edit mode is canceled without making changes, the lock on the database record(s) is released. If the changes are made while in edit mode, the data changes are applied to the record(s) and committed which inherently releases the lock so the record(s) becomes available again. Any errors that occur during an Insert, Update or Delete are returned and displayed in the user interface. When a record is inserted, no locking is done as Trane’s Data Services team policy prohibits an entire table lock on an insert.

To ensure that PUMA does not access or modify data on any tables unexpectedly, a database role is used to define the tables to which PUMA has visibility. All views and tables that are accessed by PUMA are defined in the database role along with the Select, Update and Delete permissions for each, as needed. If a table or view is not listed in the role, PUMA does not have access to it. The PUMA application has its own application account and password hard-coded into the application. The application account has been
granted the database role. At application startup, PUMA uses the application account to connect to the database and the role gives PUMA the necessary access it needs to make the user requested updates.

Figure 6.5. Database Schema for PUMA
Database Sequences are defined for each new database table. When a new record is to be inserted into a table, the Sequence is read, the value is retrieved and the Sequence is set to the next incremental value. The retrieved Sequence value is used as the primary key value for the new record. The Sequence ensures that each primary key value is unique and that key values are incremented by one for each new record.

6.4 Graphical User Interface

The graphical user interface for PUMA is series of dialog windows. At application startup, the user must log in by providing a user id, password, and database name. Verification is done to ensure that the user id and password are valid and that the user has permission to log into PUMA while connected to the specified database. In order for a user to be able to log into PUMA, the user account must be granted the PUMA database role. This role is not a default role for the user, but is granted to the user account. During login, PUMA verifies that the user account has the database role and then allows the log in to complete successfully. Otherwise, a message is displayed to let the user know they do not have access to use PUMA.

![Figure 6.6. PUMA Login Window](image)

Figure 6.6. PUMA Login Window
On the login window, the user may choose the option for PUMA to remember the user id and password. This provides a convenience for the user and works well when PUMA is installed locally on a PC, but is not secure when PUMA is used in a Citrix environment where the single installation of the application is launched by many different users. This option will be removed in Phase 3 as it is likely that PUMA will always be released in a Citrix environment.

Following a successful login, a main menu is displayed which contains links to the various functional areas of PUMA, such as Field Classifications, Feature Dependencies, and Feature Size and Placement Rules. After pressing a link, the main window of the requested functionality is displayed. The Field Classification data maintenance consists of one window, and the Feature Dependency data maintenance consists of two windows. The Feature Size and Placement data maintenance is much more complex and contains a number of interrelated windows, as shown in Figure 6.7. Multiple windows in any functional area can be open and active at any given time.

Addition window screenshots can be found in the Appendixes. Appendix A contains sample Field Classification windows, Appendix B contains Feature Dependency windows and Appendix C contain Feature Rule windows.
6.5 Deployment

The PUMA application has not yet been released in a production environment. It is currently available in a Citrix testing environment that is primarily accessed by the Project R development team and business unit product data testers, both domestic and international. Although PUMA resides in a test environment, the data updates are made to the production Oracle database, whose data gets replicated monthly to the sales office databases. Phase 1 and Phase 2 focused on the development of PUMA and ensuring that it provided the necessary data maintenance functionality, so little or no priority has been given to the actual deployment of PUMA.

Since PUMA requires Oracle 9 to be installed and individual personal computers do not have Oracle 9, it is expected that PUMA will eventually be released in a Production Citrix environment that has Oracle 9 installed, where PUMA would be available to any user who has access to the Citrix environment and whose user account has been granted
the proper database role. Deployment of PUMA to a production environment will likely occur as part of Phase 3 or Phase 4.

6.6 Limitations

As the functionality in PUMA expands, security will be a key issue and will need to be revisited and modified. Currently a single database role allows a user access to all functionality within PUMA. This works well as all functionality to date pertains to Project R. As functionality expands, either more database roles will be needed to secure functional areas or a new security approach will be needed to meet future security needs.

Speed is an issue that will continue to be addressed. As more and more data is entered into PUMA, window refreshes take longer, so shortcuts are desired to connect from one screen from another. As a result of the screens being used over a several months, several screens are scheduled to be redesigned based on user feedback to implement suggested shortcuts and enhancements that will provide quicker feature rule creation.

6.7 Continuing Work

As stated in the limitations in Section 6.6, several feature rule screens will be redesigned and combined to provide faster and more efficient rule creation.

Five reports that were originally identified in the requirements for Phase 2 are needed to monitor and report on feature rules, feature dependencies and field classifications. The creation and implementation of these reports is scheduled for Phase 3.

The deployment of PUMA into a production Citrix environment will become a higher priority as the number of users of PUMA and diversity of functionality expands to areas outside of Project R. This will also require version updates of PUMA to be applied during the monthly update schedule.

Feature Dependencies will be enhanced so that they include performance fields. For Phase 2, only product category fields were required to be included in a feature dependency, but a need has already been found to include performance fields and to allow them to be evaluated with comparison operators on decimal values or values of other fields, such that a feature dependency would be met if the condition evaluated to TRUE. An example of a feature dependency with performance fields would be “if the length of the damper is greater than the width of the damper” or “if the length of the damper is greater than or equal to 50 inches”.
Continuing work for PUMA, well into the future as time and resources permit, will be the migration and redesign of existing product data maintenance from TrapMan to PUMA.
7. Feature Rules in TOPSS™

This chapter presents the general and detailed architecture of the TOPSS application and how the feature rule functionality fits into that architecture. The database design, graphical user interface, design decisions, deployment, limitations and continuing work of the feature rule functionality are also presented.

7.1 General Architecture of TOPSS™

TOPSS is supported by an eleven member software development team and a five member quality assurance team along with personnel in each factory who create and maintain the supporting data, rules and pricing for each product. The application was originally developed by a consulting company but the development and maintenance was brought in-house about five years ago. The TOPSS application is written in C++ using Microsoft Foundation Classes with development using Microsoft Visual Studio version 6.0.

The general architecture and integration of TOPSS with other sales tool applications is presented in Figure 7.1, which is derived from [9]. The following paragraphs discuss the architecture from both data and user perspectives.

The product data, including business rules, that supports the selection and pricing of a product is updated as needed by the factory on the production Oracle database using either the TrapMan or PUMA data entry tools. Once a month, the product data changes that were made during the past month are captured, or replicated, and applied to each of the sales office databases along with any new versions of the sale tool applications so that the product data and applications stay up to date and in sync in all sales office locations. New product cache files are generated from the replicated data so they contain the latest product data, and are be distributed as part of the monthly update for TOPSS. The product data from the cache files pertaining to that product is loaded into memory when a user selects a new product and is used throughout the selection process of units for that product type.
A TOPSS™ user selects and configures units using Windows dialogs and has the option to save the selections as a job to either local storage to a PSD file, or the user may save the job to the sales office (Sybase) database. All sales tools applications connect to the sales office database. Once the job is saved to the database, the job and selections are available to all sales tools applications or can be re-loaded to TOPSS at a later time. A PSD file is typically used in one of three ways: as a work area for a job, as a backup of a job, or as a means to send a job to another user who can load the PSD file into TOPSS in their sales office location.

The COM interface for TOPSS is used by a number of other applications who launch an instance of TOPSS in the background and then use COM to access functionality in TOPSS. For example, if a user modifies the unit configuration in a sales tool other than TOPSS, the product selection is no longer considered valid. By using COM, that application can call the TOPSS functionality to generate performance, validate the new configuration and generate product drawings.
In addition to being able to be run in the background via COM, the TOPSS application can be run by sales engineers and sales assistants in different sales office locations by logging into a common production Citrix environment. Each user gets their own session and different instances of the sales tools applications, including TOPSS, can be launched with a connection to the user’s sales office database.

TOPSS is also distributed external to Trane to customers who install TOPSS in their location and are able to run TOPSS without a database connection due to the product data cache files which contain the product data.

**7.2 Detailed Architecture of TOPSS**

The first version of Trane’s product selection programs were independent programs for each product and were run in a DOS environment. Depending on who developed the program, the interface design of each program was different which led to confusion and training issues as it required a user to learn the unique characteristics of each selection program. The TOPSS™ executable was designed with the goal of providing a generic, data-driven, interface from which all Trane products could be selected and configured. Based on this approach, the application was named the “TOPSS Official Product Selection Program” or TOPSS. As a result, whenever a new selection program is developed for a product, the selection program is developed as a DLL and can essentially be “plugged into” the existing architecture and user interface.

Figure 7.2 shows the detailed architecture of TOPSS which can be viewed as three levels [10]. At the top level, the TOPSS application and user interface is supported by global, user interface, unit of measure, and database recordset DLLs. The middle level consists of the product selection DLLs, where each product has its own DLL which contains the functionality to generate the performance for the unit at specified conditions. These C++ DLLs are developed by members of the TOPSS development team working closely with a product engineer/product manager in the factory. The lowest level contains the engineering models which are called by the product DLLs to run the mathematical calculations specific to a fan, coil, etc. These engineering models are developed and maintained by the engineering teams. Each product DLL calls one or more engineering
models, as needed, providing input conditions and obtaining performance values from the models which may be used in the performance calculations in the product DLL and ultimately gets returned to the GUI and displayed to the user.

7.3 Detailed Design

As the user inputs the conditions to configure a unit, and as the selection is run and performance is generated, various types of business rules are run to guide the selection process. The feature rules are a new business rule type that is now included as part of the selection process for modular products that support the ability to size and position features. This section describes the design of feature rule related functionality in TOPSS.

With the exception of one rule type, the rule functionality in TOPSS is contained within a Rule Manager class. While wanting to continue the philosophy of keeping the rules in one location via the Rule Manager, the developer also wanted the feature rule
functionality to be an independent and self-contained, so the design decision was made to create a Feature Rule Manager to act as a sub-manager within the Rule Manager. When the Rule Manager receives a request to run feature rules or to retrieve feature size or position information for a feature, the request is passed on to the Feature Rule Manager. The results are returned from the Feature Rule Manager to the Rule Manager and then returns to the calling function.

The class diagram for the Feature Rule Manager is shown in Figure 7.3. The primary class in the feature rule functionality is the Feature Rule Manager, which contains the logic to: fire the appropriate feature rule at the appropriate time, determine the available sizes for a feature on a given module face based on the unit configuration, and determine whether the position of a feature on a module face is valid with respect to how close the feature is to the edges of the module face or to other adjacent features on the same module face. Other classes accessed by the Feature Rule Manager are product related classes, container classes, and classes specific to the structure of a feature size or placement rule.

Some classes that hold information which is directly used by the feature rules, such as the Field Classification and Feature Dependency classes, are not located within the Rule Manager or Feature Rule Manager. Field Classifications are generic categories and, since they can be applied to any product, they are stored globally so that both the TOPSS™ application and the product DLLs have access to them. Feature Dependencies, which are specific to a product but yet can be used in any functionality to the product -- not just in the feature rules -- are stored as part of the static product data. The class diagram for the Rule Manager, Feature Dependencies, Field Classifications, is shown in Figure 7.4.

Another set of existing classes contained in Figure 7.4 are the DrawingInfo classes. The DrawingInfo classes hold feature and module dimensions and coordinates for a unit. The data in these classes gets updated from the GUI after a user has sized or positioned a feature or module. Although the size and position of some features for TCCA will now originate from the user rather than being static, the usage of these classes did not need to change. As the user changes the size and position of a feature, the size and position is updated in these classes.
When saving a job, either to a PSD file or to the database, the sizes and coordinates are obtained from the DrawingInfo classes. When loading a job, the sizes and coordinates are either deserialized from the .psd file and loaded into the DrawingInfo classes, or they are read from the database and loaded into the DrawingInfo classes. Once loaded, the DrawingInfo classes are used by the GUI to display a visual picture of the unit with its modules and features in proportion to the user.

A common set of settings are added to the registry for each product when TOPSS™ is installed. Since each product is different, the value of the registry settings for each product is different. As part of this project, a new registry setting was created to identify whether a product supports sizeable and positional features. TCCA is the only product that currently supports sizeable and positional features, so all other products have this registry setting turned off. Throughout TOPSS, to support new functionality that is now specific to TCCA or but is available to other products in the future that may support sizeable and positional features, this registry setting is checked prior to executing the code. The load and save functionality make use of this functionality. For example, feature sizes and coordinates are only loaded and saved for products that have this new registry setting.
Figure 7.3. TOPSS™ Class Diagram for Feature Rule Manager
Figure 7.4. TOPSS™ Class Diagram – Rule Manager and Other Support Classes to Feature Rule Functionality

The feature rule, field classification and feature dependency information for TCCA is read from the database and stored in cache files. When the TCCA product is loaded, this information along with other product information from the cache files is loaded into
memory to be used as needed. Table 2 shows the functional differences for the rule, specification, size and border levels of the feature rules and the differences in how they were loaded into memory, based on their status for Phase 1 and Phase 2.

<table>
<thead>
<tr>
<th></th>
<th>Phase 1</th>
<th>Phase 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rule</strong></td>
<td>ShowPending=0 (Current only)</td>
<td>ShowPending=1 (Current &amp; Pending)</td>
</tr>
<tr>
<td></td>
<td>Load one rule per field - the rule with Current status.</td>
<td>Load one rule per field -- the pending or current rule that has the lowest rank.</td>
</tr>
<tr>
<td></td>
<td>ShowPending=0 (Current only)</td>
<td>ShowPending=1 (Current &amp; Pending)</td>
</tr>
<tr>
<td></td>
<td>Load one rule per field - the pending or current rule that has the lowest rank.</td>
<td>Load one rule per field -- the rule with Current status.</td>
</tr>
<tr>
<td><strong>Spec</strong></td>
<td>Load all Current specs for a rule. (Each spec contains a unique SI/Module Face combo).</td>
<td>Map each unique SI/Module Face combo (that is defined in a Pending or Current spec) to the Pending or Current spec that contains the combo and has the lowest rank.</td>
</tr>
<tr>
<td></td>
<td>Load all Current specs for a rule. Evaluate in rank order. (Each spec contains a unique SI/Module Face/Feature Dependency combo. The spec will not be loaded if the Feature Dependency or any SI in the Feature Dependency is not Current.)</td>
<td>Load all Current and Pending specs for a rule. (Evaluate in rank order.)</td>
</tr>
<tr>
<td><strong>Sizes</strong></td>
<td>Load all Current sizes for the spec. (Sizes are loaded into an array and evaluated in rank order.)</td>
<td>Load all Pending and Current sizes for the spec. (Sizes are loaded into an array and evaluated in rank order.)</td>
</tr>
<tr>
<td></td>
<td>Load all Pending and Current sizes for the spec. (Sizes are loaded into an array and evaluated in rank order.)</td>
<td>Load all Current sizes for the spec. (Sizes are loaded into an array and evaluated in rank order.)</td>
</tr>
<tr>
<td><strong>Borders</strong></td>
<td>Load the border that has the Current status.</td>
<td>Load one -- the Pending or Current border that has the lowest rank.</td>
</tr>
<tr>
<td></td>
<td>Load all borders with a Current status. (Evaluate in rank order.)</td>
<td>Load all Pending and Current borders. (Evaluate in rank order.)</td>
</tr>
</tbody>
</table>

Table 2. Functional Differences Loading Feature Rules from Cache File Between Phase 1 and Phase 2.
7.4 Database Design

With the exception of one table, the database design for feature rules in TOPSS™ is the same schema used by PUMA as defined in Section 6.3. The feature rule tables, field classification tables, and feature dependency tables are populated by PUMA. TOPSS reads these tables and stores the data in product cache files and loads the data to memory.

Since, for TCCA, the feature size and position is determined by the user, the size and position of each sizable and moveable feature now was required to be stored as part of the job information. The Selected_Item table is an existing table that stores those feature’s that are selected by the user to be included in the unit. The schema for this table was modified to add attributes to hold the height, width and depth of the feature, the x, y and z position of the feature, and the distance the feature extends externally from the module face. The Load/Save functionality in TOPSS was modified to read/write these attributes. The schema for the Selected_Item table is presented in Figure 7.5.

```
    selected_item
   PK selected_item_id
    vpc_id
    si_id
    plane
    x-dim
    y-dim
    z-dim
    x-offset
    y-offset
    z-offset
```

Figure 7.5. Database Schema for Feature Dimensions & Coordinates

7.5 Graphical User Interface

The feature rules are tightly coupled with the (Module) Face Editor, which is the only portion of the TOPSS user interface that directly interacts with the feature rules. This editor allows a user to add, modify or delete a feature on a module face, and also select the size and the location of the feature on the module face. A sample screenshot of the Face Editor is shown in Figure 7.6.
Each field classification from the feature size and placement rules for the TCCA product has an icon representation in the feature toolbar at the top of the window. Beneath each field classification icon, a drop down list of all sizeable and positional features for that field classification that can be added to the current module face are listed. The feature toolbar is dynamically updated, based on the module face selected in the Face Editor and the features that are available to be added to the module face as defined in the feature rules.

As part of each of the user actions, the feature rules are called in the background to return information to the user interface. When a module face is selected, the Feature Rule Manager is called to determine all of the features that may be placed onto the module face. Those features are grouped by field classification icons in the feature toolbar and are available as drop-down options from the toolbar for the user to click or drag to add the feature to the module face. As part of the logic to add a feature, the Feature Rule Manager determines the initial, default size for the feature by taking into
account the feature rules for the field, the feature dependencies and configuration of the unit. When a feature edge is dragged to change the size of the feature, the Feature Rule Manager is called to identify the next available width, height or depth. Anytime a feature is added, resized or repositioned on the module face, the Feature Rule Manager is called to validate the position of the feature with respect to the edges of the module face and to other sizeable or positional features on the module face. An additional check to validate the position of all sizeable and positional features on the unit was added to the existing logic that determines whether the unit has a valid configuration.

7.6 Design Decisions

This section lists the design decisions that were made to support the design of feature rules in TOPSS™.

- A new cache file will be created to hold field classifications. Since field classifications are not product specific, they cannot be added to an existing cache file.
  - The cache file will have an XML format, which provides the benefit of making the contents readable.
  - The cache file will be serialized and deserialized using the same .ini file settings that are currently used when serializing and deserializing other TOPSS cache files.
  - All field classifications, with a Pending and Current status, will be read from the database and directly serialized to the cache file. To load the field classifications to memory, the cache file will be deserialized using the .ini settings to dynamically determine which field classifications will be loaded to memory. Although this approach increases the size of the cache file, this approach is desired because it enables the user to control which field classifications get loaded to memory by simply changing .ini settings rather than having to regenerate the cache file.
  - The new cache file will be called “FieldClass.dbi”, where “dbi” represents database information and is the common file extension for TOPSS™ cache files.
- A new cache file will be created for each product to hold the feature size and placement rule specific to that product.
o The cache file will have an XML format, which will provide the benefit of making the contents readable.

o The file will initially contain only feature rules but will include other rules for the product in the future as other existing rule types are redesigned in the future and serialized to an XML format.

o The new cache file will be serialized and deserialized using the same .ini file settings that are currently used when serializing and deserializing other cache files for TOPSS™.

o All feature rules, with a Pending and Current status, for the product will be read from the database and serialized directly to the cache file. To load the feature rules to memory, the cache file will be deserialized using the .ini settings to dynamically determine which feature rules will be loaded to memory. Although this approach increases the size of the cache file, this approach is desired because it enables the user to control which feature rules get loaded to memory by simply changing .ini settings rather than having to regenerate the cache file.

o The name of the new cache file for each product will have the format “<Product>Rules.dbi”, where <Product> is the acronym that represents the product in TOPSS.

- Field classifications will be available globally within the application. Although only the user interface and the rule manager need access to them for this project, it is anticipated that product DLLs will need access to them in the future as product DLLs gain the ability to run rules.

- Only one occurrence of a feature may exist on a module at any given time. For example, a left-hand door, may be place on any face of the module, but on no more than one module face at any given time. This decision was made so that the job information that gets saved to the database does not cause issues in other business processes and applications that use the job information.

- The feature rules will be accessed via a Feature Rule Manager contained in the rule manager. The advantages of using a Feature Rule Manager will be that the feature rules will be self contained but yet have the same accessibility as other rules types.
within the rule manager. The feature rule manager will handle all requests to retrieve feature size and position information and to validate feature size and position at the feature, module, module face and unit levels.

- The naming scheme of the module faces used internally in TOPSS™ differs from the naming scheme used by the Business units to write feature rules and other rules. As a result, the same module face may be referred to as the Left module face internally in TOPSS and as the Front module face in the feature rules and other rule types. To accommodate this, the module face will be converted prior to validating the feature size and position using the feature rules.

### 7.7 Deployment

Since the feature rule functionality is a sub-project of Project R, and Project R changes are contained within the TOPSS application, the deployment of a new version of TOPSS will remain basically unchanged. The first time TOPSS is deployed with Project R functionality, the following additional actions will be needed:

- Distribute the feature rule cache files for each product. These files have the name “<Product>Rules.dbi”.
- Distribute the Field Classification cache file. This file has the name “FieldClass.dbi”.
- Run the TOPSS registry file that contains a setting for TCCA to indicate that the product supports sizeable and positional features.

The initial release of TOPSS™ Project R functionality will be released in August 2005 with a limited number of sales engineers have security access to load TCCA and who have had training on how to select a TCCA unit. The team is also looking for these users to provide feedback about the TCCA selection process. A full release of TCCA to all Trane users is scheduled for Phase 3 of Project R in December 2005.

### 7.8 Limitations

The majority of the feature functionality is complete, yet some areas are not yet working to the full expectations of the original Project R requirements. Some of the limitations that currently exist in the feature rule functionality in TOPSS are discussed in this section.
A UOM, or unit of measure, is part of each feature rule, yet the Feature Rule Manager does not yet handle UOM conversions. The borders and sizes that are passed into the Feature Rule Manager for validation are assumed to have the same UOM as the sizes and dimensions in the feature rules for the product. This poses some problems and will be an issue if the user changes the UOM scheme for the product. This issue will be a high priority to be addressed before TCCA is released to all users in Phase 3.

Another limitation is the feature dimensions and coordinates are not saved as part of the selection data that is passed to the Spectrum application, which is the application used by the international user community to price selections. A hand-off file is used to transfer information back and forth between TOPSS™ and Spectrum created by TOPSS and this hand-off file has not been updated to handle feature dimensions and coordinates. Due to time constraints, this portion of the project was assigned a lower priority. The impact of this missing functionality affects the international users of TOPSS who will not be able to select and save a TCCA unit from TOPSS and pass it to Spectrum for pricing. This item will be a high priority to be addressed before TCCA is released to all users in Phase 3.

An original requirement was to be able indicate as part of the feature rule whether or not to log debug messages anytime the rule was fired. During the development period, the logging of debug messages was handled by a simple on or off setting in the .ini file. As a result, logging is either done for all messages or for none. As part of the next Phase of Project R, the logging logic should be revisited to allow the logging of an individual rule to be available as the original requirement intended.

The validation of adjacent features on a module face is working properly except in a couple of cases where the diagonal distance between the corners of the adjacent feature is validated instead of the direct linear distance. These cases occur when the adjacent features share the same X or Y position. Although the validation isn’t exactly correct, it ensures that more than enough distance is present between the two features. This issue will also need to be addressed in the next Phase of Project R.

**7.9 Continuing Work**

In addition to addressing the limitation of the Feature Rule Manager as stated in Section 7.8, one of the upcoming phases of Project R will be to merge the Feature Rule
Manager with the unit optimization logic. Unit optimization is another piece of functionality in Project R that attempts to shrink a unit to the smallest possible size while still maintaining the spacing requirements defined by the feature rules and other spacing limitations. Eliminating unnecessary space and materials impacts cost, which is key to the user. Since feature rules are part of optimization and both areas look for areas where features reside too closely, there are opportunities to merge these two areas or at least integrate them more closely.
8. Bibliography

9. Appendices

Appendix A. Sample PUMA Field Classification Screen Shots

PUMA Field Classification Summary Window
PUMA Field Classification Detail Window

Field Classification Details

Classification Name: 

Classification Description: 

Display Sequence: 

Status: Pending 

Metadata Name: 

Last Modified Info
User Id:
Date:

Save  Cancel
Appendix B. Sample PUMA Feature Dependency Screen Shots

PUMA Feature Dependency Summary Window
PUMA Feature Dependency Detail Window
Appendix C. Sample PUMA Feature Rule Screen Shots

PUMA Feature Rule Filter Window
PUMA Feature Rule Detail Window
PUMA Specification Summary Window
Damper banks and openings require 6 in. of clearance from the end of the unit.

Rank: [Blank]
Status: Pending

Features:
- No damper or opening (P)
- Fouth damper - opening only (P)
- Fouth damper - opposed blades (P)
- Fouth damper - parallel blades (P)
- Fouth damper - TRAQ (P)
- Fouth damper - louver (P)

Module Faces:
- Back
- Bottom
- Front
- Left
- Right
- Top

Feature Dependency:
- Name: Air section: Entering EDU
PUMA Size and Dimension Detail Window
PUMA Border Summary Window
PUMA Border and Clearance Detail Window
PUMA Copy Placement Specification Window