2012 IEEE 36th International Conference on Computer Software and Applications Workshops

Rule Based Axiomatic Design Theory Guidance for Software Development

Cengiz Togay*, Emre Selman Caniaz†, and Ali Hikmet Dogru‡
*Computer Engineering Department
Canakkale Onsekiz Mart University, Canakkale, Turkey
Email: ctogay@comu.edu.tr
†Vocational High School of Altinoluk
Balikesir University, Balikesir, Turkey
Email: ecaniaz@balikesir.edu.tr
‡Computer Engineering Department
Middle East Technical University, Ankara, Turkey
Email: dogru@ceng.metu.edu.tr

***Abstract*—This research proposes a rule-based approach to system development with Axiomatic Design Theory. The basic complexity is the resolution of the constraints across the Feature Model and the Design parameters addressed through the Axiomatic Design such as the design matrix and associated components. A rule based approach effectively can support the development process during the configuration of the variability, hence the development of the product. This work can be classified as a product engineering method within Software Product Line Engineering. Axiomatic Design Theory suggests a simultaneous decomposition in different modeling spaces. Incorporating Axiomatic Design principles in Software Product Line Engineering, this research aims to provide a conforming set of development models that co-evolve for an efficient software production.**

***Keywords*-Axiomatic Design Theory; Feature Model; Component Oriented Software Development**

I. Introduction

In component oriented software development, components are composed to produce a new software product. Depending on customer expectations, different variations of the components form distinct software products in a domain. In time, the domains are transformed to mature domains through including various components, design artifacts, dictionaries, etc. and also through the establishment of a common understanding on a dictionary of components, by the developers. Success of the relations between features and products is dependent on domain maturity and domain expert competence. In idealized mature domains, customers determine their expectations through a formal way and systems are developed through composition of the components automatically. At the present time, complexity of the software products is generally beyond the reach of human talents. Therefore, the demand for the guidance tools for each phase of the software development is increased. The first step to aid for this kind of tools is to define end-user expectations in terms of a common language between customers and developers. One of the formal ways is to utilize feature models as this common medium [1] [2]. Features incorporated in these models are defined as end-user visible units of behavior [3]. Features are represented in a feature model as depicted in Figure 1. There are eight features represented in Figure 1. Features F1 and F1.1 are mandatory which means that they have to be part of the requirements. Others are optional. Also an alternative relation is represented in the figure. Users can select only one of the alternatives (F2.1, F2.2, or F2.3). Feature models are generally represented as a tree and they also include rules and constraints [1] [2]. Feature models are prepared by the domain experts to represent commonalities and variability among the domain software products. Customers define their expectations for a specific product, through selection of features from the domain feature models. Selected features should be consistent in terms of the constraints and the tree notation. There are two approaches to provide consistency, one of them is evaluating the consistency after all selections are conducted and the other one is simultaneously testing the selections and guiding the customer after every selection action, based on previous selections. Depending on the size of the domain, relations and dependencies among features, rules, and constraints increase the complexity of providing and managing of their consistency. It should be noted that the number of features in a domain can easily grow up to few thousands [4]. Number of features, constraints, and rules specify the complexity of the selection process in terms of avoiding conflicts and errors [5]. To handle this complexity, there are various studies discussed in the section on feature models. In this study, we are generating rules based on a feature model in order to provide guidance to the stakeholders (customers, developers, etc).

It should be noted that there is a gap between the customer needs and a solution [6] [7]. Identifying features is not enough to find corresponding components and integrate them automatically but can be helpful for a domain expert. Feature models capture fundamental needs, but as discussed in [8] [9], to define behaviors of the components, there is a need for another approach. One approach is proposed in [2] and extended in [10]. In [10], different level features

978-0-7695-4758-9/12 $26.00 © 2012 IEEE
DOI 10.1109/COMPSACW.2012.101

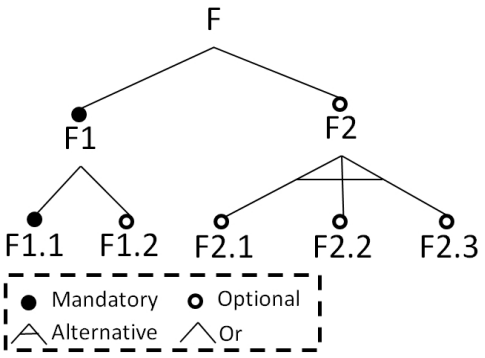

Figure 1. Feature Model

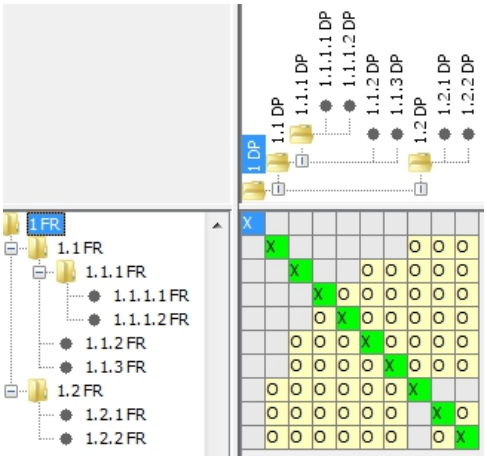

Figure 2. Design Matrix

are mapped to objects. Another approach we used in this study is Axiomatic Design Theory (ADT) [11]. ADT can be utilized to represent functional requirements (FRs) and design parameters (DPs) such as methods, components, etc. and their dependencies in a matrix called FR-DP design matrix as depicted in Figure 2. In the figure, Xs represent dependencies and in this uncoupled design matrix each FR only related with one DP. For instance, FR1.1.1 depends on DP1.1.1. Only a standard interface is not enough to define components' capabilities; especially it's provided behaviors. Therefore, an FR-DP matrix for each component is used to represent which FRs are satisfied and what are the dependencies of these requirements on the design parameters [8] [9]. The mature domain includes a *domain FR-DP matrix* called domain design matrix to represent all FRs, DPs, and their dependencies. Such domain models are developed during the domain engineering phase. This domain model is then utilized in the forming of specialized design matrices for specific products, during the product engineering phase. Selected features are utilized to determine the required FRs and DPs.

In this study, our first contribution is introducing an extension to ADT for design matrices. Depending on the alternative sets of features, FRs and DPs are incorporated in the product matrix. However, there is no representation of *Optional*, *Mandatory*, *Alternative*, *Exclude*, *Require*, and *OR* relations in a design matrix. In [8] [9], FRs or DPs are interpreted to be connected through *OR* and *Optional* relations, if there is no dependency. However, other relations are not included. For instance, *Exclude* relation is no accounted. The standard design matrix cannot represent the excludes relation. Consequently, only some of the FRs can be satisfied in a project based on customer expectations. Since representation of all relations in a design matrix decrease readability, we define new views similar to feature model notation.

The second contribution of this research is the utilization of a rule engine referred to as Drools [12] to provide guidance during design process for all stakeholders involved in the process (customer, developer, domain expert, etc). All relations among FRs, DPs, features, components, and relations (mandatory, optional, exclusive, etc.) are represented in terms of rules so as this tool requires.

A. Feature Models

Feature models [1] [2] are introduced to capture user intents and are used by various software development approaches such as software product line engineering. In [10] that is an extension of FORM [2], an object oriented approach for utilizing feature models is proposed. In the [10], different types of features (capability, operating environment, domain technologies, etc.) are mapped to domain architectures and reusable domain objects. In [8] [9], features are utilized to reach required functional requirements and some features are utilized to select components from a component library. Traceability links are utilized in [7]. Change in the domain can take place especially at the beginning of the domain creation. To decrease the effort for modifying a domain in terms of it's feature model and architectural components, relations between them should be one-to-one (1:1) [7]. The Feature model has to be consistent for a domain. Domain feature models can be formed from various feature models that are prepared independently. However, it should be noted that combination of models can cause consistency problems [4]. Some challenges for feature models are listed below:

- There is no really agreed notation for feature models [4]. However, basically new notations are extensions of FODA [1] such as cardinality based feature models [13].
- Same information can be represented with different feature subtrees. Therefore, combining or integration of more than one feature model causes various consistency problems [4].
- Customers select features from a domain feature model to obtain an application feature model that is subset of

the domain feature model. One method, as preferred by most of the approaches, is the iterative selection of features from a domain feature model. However, a customer can forget his past activity and even the needs can change during the feature selection process. In this situation, previous decisions are effected by a new selection at the higher levels of the feature tree [5].

- Multiple participants can be included in the feature selection process at the same time and depending on their priorities, selected features can be in conflict [14].
- Only feature model utilization for a domain is not sufficient for modeling entity-relationships and a business process. Therefore, feature models should be integrated or should cooperate with other methods [15].

As a definition of the tree structure each feature has to have a parent feature except for the root feature. There are six types of feature relations listed below on a feature tree and some relations are depicted in Figure 1.

1) *Mandatory*: the feature has to be selected.
2) *Optional*: the feature can be selected.
3) *OR*: one or more children can be selected.
4) *Alternative (XOR)*: only one of the children has to be selected.
5) *Require*: a feature requires the selection of another feature. In some notations this relation is directly represented on the diagram and in some notations it is externally represented as a constraint.
6) *Exclude*: a feature, if selected, requires the exclusion of another one. In other words, two features cannot be selected simultaneously for the same product.

In addition to the feature tree structure, some constraints/rules can be defined. In some notations the *Require* and *Exclude* relations are directly represented on the diagram and in some notations they are externally expressed as constraints/rules. To automatically verify selections, there is a need for a rule based system. In this study, Drools rule engine [12] is utilized. Feature models can be prepared by domain experts and they can represent the same set of features with different representations. To be formally consistent, some normalization processes are proposed for feature models in [16]. There is a relationship between the number of optional features and the the number of possible products [16]. There are various studies to handle consistency of a feature model's subtree or itself which includes only the features selected by the customer:

1) Ontology utilization and reasoners: In [8], a feature model and other software assets are defined in ontology and a reasoner is utilized to detect inconsistencies both in the feature model and in assets in terms of selected features. However, this process is achieved after feature selections. There is no guidance during the selection process. In other studies [4] [15] [17] [18], a feature model is also represented in an ontology and ontology reasoning for feature models is used to perform for validating and tailoring guidance.
2) Propositional logic and SAT solvers: Features and their relations are represented as propositional terms and they utilize a SAT solver to automatically classify changes between the original and the updated feature models [16]. In another study, a SAT solver is utilized for debugging feature models [19].
3) Constraint Satisfaction Problem (CSP): Feature models are transformed to CSP and a constraint solver is utilized to find invalid configurations [14]. Also feature models are transformed to CSP and a constraint solver is used to automate feature selection [20].

B. Axiomatic Design Theory

Axiomatic Design Theory (ADT) is introduced by Nam P. Suh and there are various applications in engineering disciplines [11] [8] [9]. There are four concepts in ADT namely domains, hierarchies, zigzagging, and axioms. The domains concept includes the customer domain, the functional domain, the physical domain and the process domain. Customer needs are captured in Customer Needs and they are mapped to Functional Requirements (FRs) in the functional domain. The FRs are then mapped to Design Parameters (DPs) in the physical domain and they are represented in a matrix called FR-DP Design Matrix as depicted in Figure 2 that also provides the dependency information regarding functional requirements. Xs represent dependency between FRs and DPs. Whereas, Os represent independencies. Gray squares are meaningless as far as a design matrix is concerned. In terms of software engineering, DPs can be methods, components or any abstraction of software components such as packages, data abstraction etc. Lastly the DPs can be mapped to Process Variables (PVs) in the process domain but PVs are not used in this study. Problems are considered in terms of different views based on these domains. Each of these domains except the customer domain has been represented as a tree structure. Each item in the domains is decomposed to define more specific items until a solution level entity is represented. This decomposition process is sequentially applied to all four domains. Based on the hierarchy concept, problems and solutions are simultaneously decomposed to simpler parts. Decomposition is conducted simultaneously among neighboring domains with the zigzagging concept. The zigzagging concept introduces a parallel decomposition of all domains except the customer domain. A decomposition in the previous domain (e.g. FR) is immediately followed by a decomposition activity in the next domain. This initial move between the domains is referred to as *zig*. Next, with a *zag* move, previous domain is re-visited for assessing in the compliance of the zig related action and for conducting the next decomposition. Successive zig and zag actions continue until both domains are decomposed to a satisfactory level.

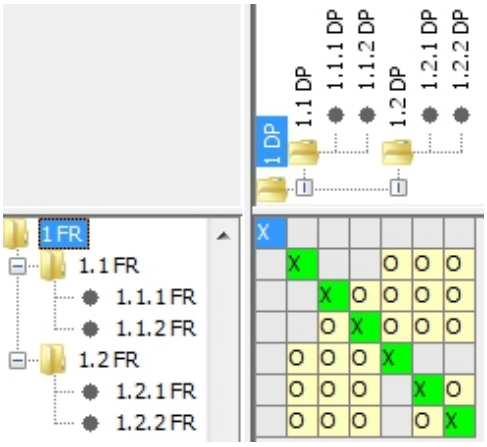

Figure 3. Uncoupled FR-DP Design Matrix

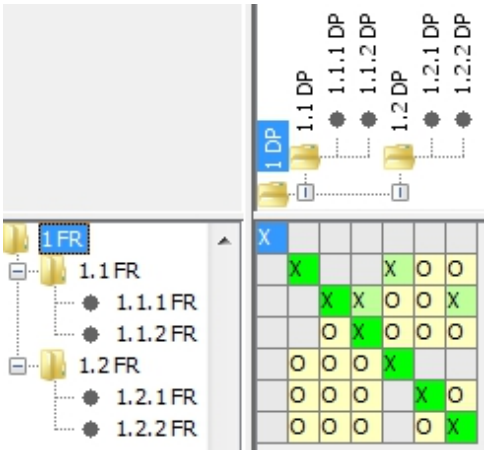

Figure 4. Decoupled FR-DP Design Matrix

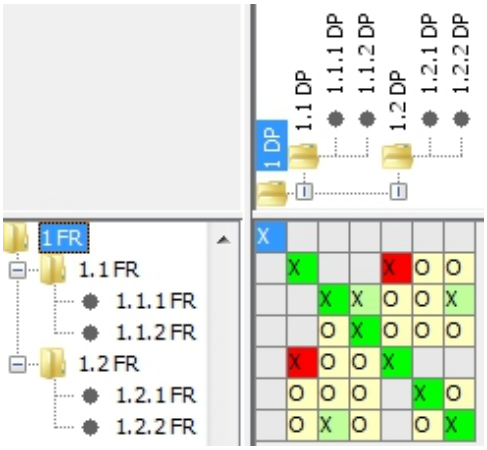

Figure 5. Coupled FR-DP Design Matrix

As the last important concept, ADT has various axioms. There are two basic axioms namely independence axiom and information axiom. In this study, we are interested in the independence axiom which is utilized to determine if the design type is coupled, uncoupled, or decoupled. For the uncoupled designs, each FR is mapped to only one DP in the FR-DP design matrix as depicted in Figure 3. Therefore, if an FR is modified or deleted, other FRs and DPs are unaffected. In a decoupled design, one FR can be mapped to more than one DPs, but there is no cycle between FRs in terms of dependencies as depicted in Figure 4. For instance as represented in Figure 4, *FR1.1.1* is dependent on *FR1.1.2* and *FR1.2.2*. Lastly, a design can be coupled, where FRs are mapped to DPs but there is cycle among them in terms of dependency as depicted in Figure 5. For instance, since *FR1.1.1* is dependent on *FR1.2.2* and *FR1.2.2* is dependent on *FR1.1.1*, there is a cycle in terms of dependencies. If there is a relation between children items, their parents are also related. For instance, *FR1.1* is automatically related to *FR1.2* as depicted in Figure 5. Coupled designs are not preferred by ADT since they indicate extra complexity when modifications, reuse, and maintenance are considered. By means of the information axiom, design is evaluated based on probability of success among FRs and DPs. This axiom can be utilized to measure congruity between a DP and a component during composition [21].

II. Proposed Approach

In this study, we are proposing a rule based approach to guide the stakeholders during their feature selections and design decisions. For instance, if *F2.1* is dependent on *F1.2* as depicted in Figure 1 and *F1.2* is not selected, we can guide the customer to not select *F2.1* during the selection process. This inconsistency can be figured out after all or some of the selections are done, but this situation can cause confusions for customers since they may face many error conditions when they return to review the consequence of their decisions. Therefore, simultaneous guidance is preferred. In a mature domain, all assets (features, FRs, DPs, and PVs) can be connected with each other. These connections increase the complexity of the domain and a stakeholder needs guidance during the design. In this study, we used a rule engine (Drools [12]) to represent all the connections. Based on the connections, stakeholder is guided for selections: the effects of selecting a feature on the components can be displayed. At the end of the design, a prototype can be composed by domain components as depicted in Figure 6. We assume that all the components are compliant with each other, and alternative solutions can be formed by available components. As it can be seen in the figure, stakeholders affect the system through design item (feature, FR, and DP) selections. All the assets such as feature model, domain design matrix including FRs and DPs, component design matrices, and components and their dependencies are added to the rule engine facts and rules respectively. For example, all selectable design items can be enabled automatically through execution of the rule listed in Table I.

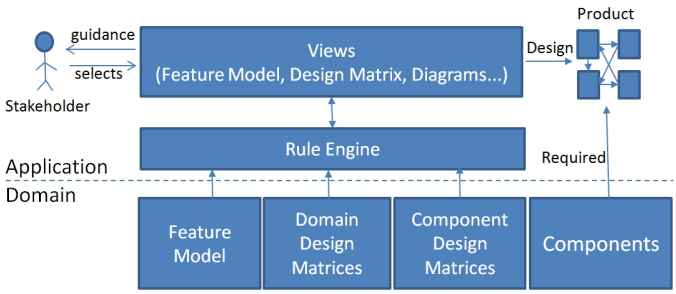

Figure 6. Proposed Approach

Table I
A RULE FOR SPECIFYING SELECTABLE FEATURES

```
rule "Specify selectable items"
salience -90
when
item : DesignItem(status==DesignItem.NOTSELECTABLE, parent!=null);
parent: DesignItem(this==item.parent, status==DesignItem.SELECTED)
then
modify(item)status=DesignItem.SELECTABLE;
end
```

Another contribution of this paper is an extension to the mechanisms utilized by ADT. The ADT proposes to find a solution through specifying FRs, finding DPs, defining dependencies, checking consistency based on the axioms, and reviewing design if required. However there is no support for domain design matrices to include alternative designs. For example, although there is no definition for *Optional* FRs or DPs in terms of mature domains, some FRs are interpreted as optional [8] [9]. In a domain design matrix, if there is a set of FRs that can be included or extracted without affecting the solution, this kind of FRs are referred to as optional FRs. Therefore, in terms of domain, ADT allows to define the *OR* relation. However, there is no support for other kinds of relations (*mandatory*, *optional*, *alternative*, *exclude*, and *require*) among FRs and DPs. We used the feature model notation to create new views represented in Figure 7 and Figure 8 for FRs and DPs corresponding to the design matrix depicted in Figure 2. The same tree structure is used to represent different views for features, FRs, and DPs. Some relations (*mandatory*, *optional*, *or*, and *alternative*) are represented in the tree and some of them (such as *exclusive* and *require*) are defined as rules external to the tree. In our implementation, there are four status for a design item namely, notselectable, selectable, selected, and dependent. The proposed FR and DP views are created and modified with corresponding domain design matrix. Normally, all FRs and DPs are specified as *optional* and *OR* relation. A domain expert can modify these views when required by other relations (such as, *alternative* and *mandatory*). In this study, features, FRs, and DPs are represented as "DesignItem" in the rules. We defined six types of design item relations as listed below:

1) The *Mandatory* relation is used when a design item has to be part of the solution. This type of FRs are used to represent the core product capabilities which provide some basic properties and their relations with DPs and components.
2) The *Optional* relation is used when a design item can be part of the solution. This type of design items are specified through feature selections or developers.
3) The *OR* relation represents that one or more children can be selected.
4) The *Alternative (XOR)* relation represents the case where only one of the children has to be selected. As it can be seen from Figure 7, only one of the *FR1.1.2* or *FR1.1.3* can be part of the solution. Another example is represented in Figure 8, where it can be seen that there are two alternative DPs namely DP1.1.1.2a and DP1.1.1.2b. Depending on the features and developer decisions one of them can be selected. A contribution of the paper, we represent such alternatives in the trees (DP tree or FR tree) as depicted in Figure 7 and Figure 8.
5) The *Require* relation represents that a design item requires the selection of another design item. In some notations, require relation is directly represented on the same view and in some notations it is represented as a constraint rule. We represent this relation as rules to allow a design item to connect with another type of design item. For instance, *F1.2* is dependent on *F1*, *FR1.1.1.2* and *DP1.1.1.2b* as depicted in Figure 1, Figure 7, Figure 8, and Table II. As depicted in Figure 1, when *F1* is selected *F1.2* has to be selected because of the mandatory relation. Also we can conclude that to select the *F1.2* feature, its parent feature (*F1*) has to be selected. *FR1.1.1.2* is dependent on *DP1.1.1.2* as depicted in Figure 2. However, two alternatives of the *DP1.1.1.2* are represented in Figure 8 and it cannot be represented in the design matrix. Therefore, *F1.2* is selected automatically if specified items are selected as listed Table II. It should be noted that any design item (feature, FR, and DP) can require other design items and this relation is represented as rules as listed in Table II.
6) The *Exclude* relation is used if some design items can never be selected at the same time in a specific solution. Some design items (FRs, DPs, and features) cannot be part of the solution with other design items in different branches of the tree. For instance, we assume that *FR1.2.2* and *FR1.1* in Figure 7 cannot simultaneously take part in a solution. In this situation, we can represent this information with lines in the view; however it decreases the readability of the view. For this reason, we defined a rule for this requirement to capture and warn stakeholders as depicted in Table III. It should be noted that if there is a

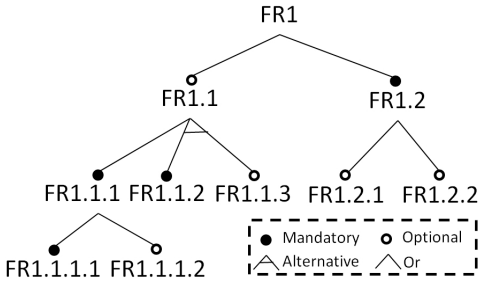

Figure 7. Functional Requirements

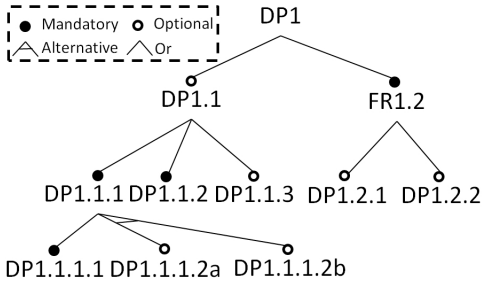

Figure 8. Design Parameters

dependency among design items, to conclude that *FR1.2.2* or *FR.1.1* has to be selected, related one is automatically selected by system. Rules need to be re-evaluated if the system automatically selects both and this situation is not desired.

Table II
DEPENDENCY FR1.2 REQUIRES TO SELECT F1, FR1.1.1.2, AND DP1.1.1.2B

```
rule "FR1.2 requires to select F1, FR1.1.1.2, and DP1.1.1.2b "
salience -90
when
FR12 : DesignItem(name=="FR1.2",status != DesignItem.SELECTED)
F1 : DesignItem(name=="F1",(status==DesignItem.SELECTED))
FR : DesignItem(name=="FR1.1.1.2",(status==DesignItem.SELECTED))
DP : DesignItem(name=="DP1.1.1.2b",(status==DesignItem.SELECTED))
then
modify(FR12)status =DesignItem.SELECTED;
end
```

Table III
EXCLUDE RELATION FOR FR1.2.2 TO FR1.1

```
rule "Exclude relation for FR1.2.2 to FR1.1"
salience -90
when
FR1: DesignItem(name=="FR1.2.2",status==DesignItem.SELECTED)
FR2: DesignItem(name=="FR1.1",(status==DesignItem.SELECTED)
then
System.out.println("There is an exclusion conflict. Only one"+
" of the FR1.2.2 or FR1.1 can be part of the solution" );
end
```

A. Domain Design

We assume that there are various products in a domain that are designed following the ADT and composed from components as proposed in [8]. To create a mature domain, domain experts apply the following steps.

1) Create a feature model to capture commonalities and variability among products.
2) Create a domain design matrix.
3) FR and DP trees (views) corresponding to the design matrix can be automatically created. These trees include more information than a standard design matrix because mandatory, optional, require, exclude, etc. relations can be represented in trees.
4) Set relations (mandatory, optional, require, exclude, etc.) among FRs
5) Set relations (mandatory, optional, require, exclude, etc.) among DPs
6) Create relations among design items (features, FRs, and DPs).
7) Create rules based on relations among the design items.
8) Create rules for feature, FRs, DPs, and relations of components. Therefore, when a DP in domain design matrix should be part of the solution, related component can be identified. Also features can affect the selection of correct components.

B. Application Design

As depicted in Figure 6, stakeholders utilize the modeling constituents such as feature model, design matrix, and new defined FR and DP views as proposed in the following list.

1) Stakeholders select features with guidance from the proposed approach. Depending on the preselected features and relations among features, succeeding selections will be constrained
2) Depending on the selected features, system offers a set of FRs and DPs depending on the rules created during domain creation.
3) Stakeholders (especially developers) select FRs and DPs among the alternatives.
4) System identifies components based on the selected DPs.
5) Components are composed and integration problems are solved.
6) If available components are not congruent in terms of non-functional requirements, new components can be developed as proposed in [8] and added to the domain.

III. CASE STUDY

In this section, we present a robot development process as a case study. A feature model for the example domain is represented in Figure 9. A customer can select required features from the feature model. Some relations among features are represented in the following list:

- Customers have to select the *Environment* feature and then they have to choose between the *Outdoor* or *Indoor* features. In this case study, we assume that robots can be designed for either *Outdoor* or *Indoor*.
- Another mandatory feature is the *Orientation* feature. Customers have to select *Sensor Based* or *Image Processing*.
- We defined a *require* relation between *Sensor Based* and *Sensors*. When the *Sensor Based* feature is selected, *Sensors* feature also has to be selected.
- Another *require* relation is defined between *Teleoperation* and *Communication*. When *Teleoperation* is selected, *Communication* also has to be selected.
- In our domain, robots can be oriented in either one of the two alternatives modeled by *Self Oriented* or *Teleoperation*.It should be noted that there is no exactly one correct domain. It depends on available products and domain experts approach. For instance, in another domain, a robot can be produced having both *Self Oriented* and *Teleoperation* capabilities.
- There is an *exclude* relation between the *Indoor* feature and the *GPS* feature. When the *Indoor* feature is selected, the *GPS* feature can not be selected.

Partial domain design matrix and corresponding views are depicted in Figure 10, Figure 11, and Figure 12 respectively. Corresponding views can be automatically produced from the design matrix. We have represented only id numbers of the FRs and DPs in views. For instance, the design matrix includes FR1, and eight sub FRs. *FR1.5* includes three sub FRs. The same information is represented in the FR view as depicted in Figure 11. Similarly, DP view can be produced from design matrix. As introduced in the Domain Design section, a domain expert defines the relations among FRs (mandatory, optional, etc.), DPs (mandatory, optional, etc.), and rules for all types of the design items (features, FRs, and DPs). Relations among the design items are listed in the following list:

- Except *FR1.8*, all FRs are defined as *optional* as depicted in Figure 11. Selected features affect FRs. For instance, "FR1.1: Obtain meaningful information from an image" is an optional FR as depicted in Figure 11. It is activated when *Image Processing* feature is selected.
- *FR1.1* is dependent on "FR1.2: Capture an image" as depicted in Figure 10.
- There are two ways to represent the *alternative* relation for DPs. One of them is to represent it on the view and another, as a rule. Alternative DPs *Wireless* and *Ethernet* are represented in Figure 12. They are effected by *Wireless* and *Wire* communication features. When *Wire* is selected, automatically the *Ethernet* DP is activated as a part of the solution. We defined another rule for the *Exclude* relation between "FR1.4: Get commands from the base" and "FR1.6: Orient itself". Only one of them can be part of the solution and they are affected by the "*Control*" feature. For example, when *Self Oriented* is selected, FR1.6 is activated.

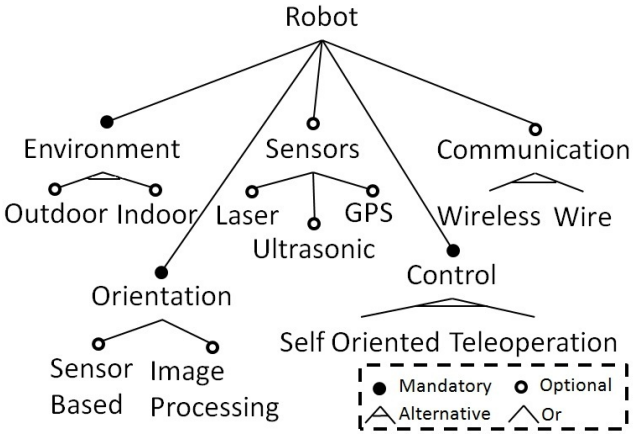

Figure 9. Feature Model of robot domain

Depending on the defined relations, stakeholders select the solution design items. For instance, when *Environment*, *Indoor*, *Orientation*, *Sensor Based* features are selected, *Sensor*, *Control*, and *Communication* features are enabled. When *Sensor* feature is selected, only *Laser* and *Ultrasonic* features are enabled. *GPS* is disabled because of *Exclude* relation between *Indoor* and *GPS*. When *Control* and *Teleoperation* selected, *Communication* feature can be also selected automatically. Lastly, customer selects *Wire* feature. Depending on these features, developers evaluate FRs and DPs to find a solution. When rules are executed, *FR1.4*, *FR1.5*, *FR1.5.1*, *FR1.5.2*, *FR1.7*, and mandatory *FR1.8* and corresponding DPs are selected automatically as a part of the solution. For *FR1.7*, there are two alternative solutions and *ethernet* is activated because of *wire* feature. Based on the DPs, sets of components are specified and composed.

IV. Conclusion

In this study, we introduced an extension to Axiomatic Design mechanisms and supported them with a rule engine to increase guidance for development in a mature domain. In addition to standard representations of ADT, we proposed two new views corresponding to FR-DP design matrix for FRs and DPs to represent *Optional*, *Mandatory*, *Alternative*, and *OR* relations. Therefore, commonality and variability can be represented for mature domains. Since these views also increase the complexity of the mature domain, relations are represented as rules. Also, *Exclude* and *Require* relations among all design items (features, FRs, and DPs) are defined as rules. As a result of this study, all stakeholders of the development (customers, developers, etc.) can benefit from the guidance that is produced by the rule base system. Customers will be informed about allowed feature selections

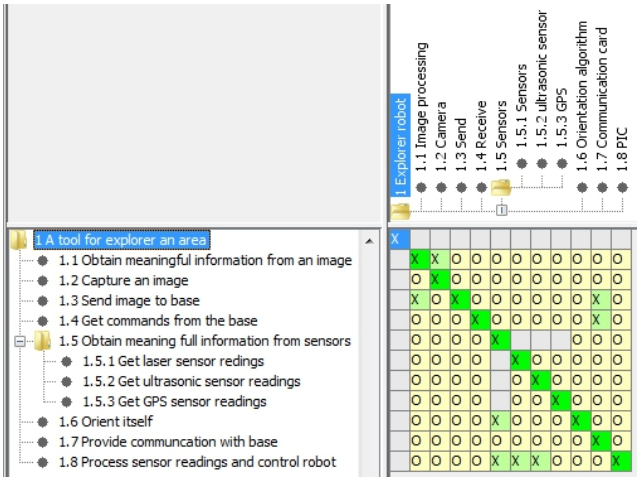

Figure 10. Design matrix of robot domain

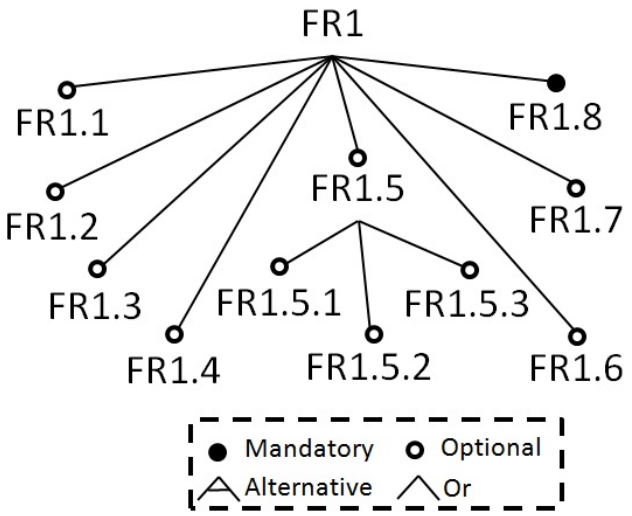

Figure 11. FR view corresponding to domain design matrix

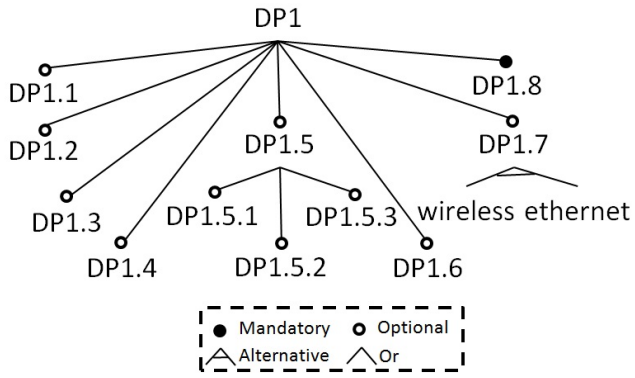

Figure 12. DP view corresponding to domain design matrix

at a stage of development and developers are informed about which components can be used. Our experiments have indicated that proposed guidance can be helpful in increasing a design's correctness for mature domains that include various features, components, etc. However, we need more experimentation especially to verify our approach on huge models to assess its scalability. As a future work, we are also planning to specify a domain specific language to define rules.

REFERENCES

[1] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.

[2] K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim, and E. Shin, "Form: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, pp. 143–168, 1998.

[3] S. Apel, H. Speidel, P. Wendler, A. von Rhein, and D. Beyer, "Detection of feature interactions using feature-aware verification," in *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, November 2011, pp. 372 –375.

[4] L. A. Zaid, F. Kleinermann, and O. De Troyer, "Applying semantic web technology to feature modeling," in *Proceedings of the 2009 ACM symposium on Applied Computing*, ser. SAC '09. New York, NY, USA: ACM, 2009, pp. 1252–1256. [Online]. Available: http://doi.acm.org/10.1145/1529282.1529563

[5] J. White, D. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortes, "Automated diagnosis of product-line configuration errors in feature models," in *Software Product Line Conference, 2008. SPLC '08. 12th International*, September 2008, pp. 225–234.

[6] L. Shen, X. Peng, and W. Zhao, "A comprehensive feature-oriented traceability model for software product line development," in *Proceedings of the 2009 Australian Software Engineering Conference*, ser. ASWEC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 210–219. [Online]. Available: http://dx.doi.org/10.1109/ASWEC.2009.27

[7] M. Riebisch and R. Brcina, "Optimizing design for variability using traceability links," in *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*, April 2008, pp. 235 –244.

[8] C. Togay, "Systematic component oriented development with axiomatic design," Dissertation, Middle East Technical University, July 2008.

[9] C. Togay, A. H. Dogru, and J. U. Tanik, "Systematic component-oriented development with axiomatic design," *J. Syst. Softw.*, vol. 81, pp. 1803–1815, November 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1412750.1412833

[10] K. Lee, K. C. Kang, W. Chae, and B. W. Choi, "Featured-based approach to object-oriented engineering of applications for reuse," *Softw. Pract. Exper.*, vol. 30, pp. 1025–1046, July 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=350554.350569

[11] N. P. Suh, *Axiomatic Design: Advances and Applications (The Oxford Series on Advanced Manufacturing)*. Oxford University Press, May 2001.

[12] M. Bali, *Drools JBoss Rules 5.0 Developer's Guide*. Packt Publishing, 2009.

[13] K. Czarnecki and C. H. Kim, "Cardinality-Based Feature Modeling and Constraints: A Progress Report," 2005.

[14] J. White, D. Benavides, D. C. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes, "Automated diagnosis of feature model configurations," *J. Syst. Softw.*, vol. 83, pp. 1094–1107, July 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2010.02.017

[15] G. Ying, Z. Xiao-yan, W. Jun, and Y. Meihong, "Domain service acquisition and domain modeling based on feature model," in *Computational Science and Engineering (CSE), 2011 IEEE 14th International Conference on*, August 2011, pp. 26 –33.

[16] T. Thum, D. Batory, and C. Kastner, "Reasoning about edits to feature models," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 254–264. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070526

[17] X. Peng, W. Zhao, Y. Xue, and Y. Wu, "Ontology-Based Feature Modeling and Application-Oriented Tailoring," 2006, pp. 87–100. [Online]. Available: http://dx.doi.org/10.1007/11763864_7

[18] C. Xin, M. Huadong, and W. G. Wang, "Feature-ontology based semantic specification model for simulation component," in *System Simulation and Scientific Computing, 2008. ICSC 2008. Asia Simulation Conference - 7th International Conference on*, October 2008, pp. 1163 –1167.

[19] D. Batory, "Feature models, grammars, and propositional formulas." Springer, 2005, pp. 7–20.

[20] D. Benavides, P. Trinidad, and A. Ruiz-Corts, "Automated reasoning on feature models," in *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAISE 2005*. Springer, 2005, p. 2005.

[21] C. Togay, O. Aktunc, M. Tanik, and A. H. Dogru, "Measurement of component congruity for composition based on axiomatic design," in *The Ninth World Conference on Integrated Design and Process Technology, 2006.*, June 2006.