Pipeline Design Modelling Language (PDML): A Formal Analysis Framework Using FODA

Japheth R. Bunakiye and Ogheneovo E. Edward
1. Department of Mathematics/Computer Science, Niger Delta University, Yenagoa, Bayelsa State, PMB 071, Nigeria
2. Department of Computer Science, University of Port Harcourt, Port Harcourt, Rivers State, PMB 5323, Nigeria

Corresponding author: Japheth R. Bunakiye (rb.japheth@ndu.edu.ng)

Abstract: This is a research progress where FODA (feature oriented domain analysis) is used to implement a formal analysis framework that characterized commonality and variability of features for a transmission pipeline design modelling language. The features representing the pipeline physical components were analysed along the line of the system logic being able to model stakeholders design intents. It is demonstrated that FODA methodology for domain analysis fits the needs for the application of a carefully studied semantics to the development of the modelling language. It also best matches with the specifications because representing the components as features in a feature model enabled the formalization of the syntax, the semantics, and the composition constraints. These formal definitions presented useful proofs for managing the variability in the application domain in terms of the components and the requirements. This requirements engineering task will provide a formal analysis model of system information and function behavior, and will bridge the gap between the requirements documentation and the language design.

Key words: FM (feature model), semantics, FODA, pipeline model, domain knowledge.

1. Introduction

This paper presents a formal requirements analysis framework that could enable the development of a pipeline design modelling language.

The structure and behaviour of the system will capture stakeholders design intents, which depicts pipeline design scenarios prevalent in the engineering domain of oil and gas transmission pipelines [1]. Pipeline design includes pipe sections joined with fittings and other supports features. These joints and fittings, which are the pipeline components, are graphics models; they are computer aided design (CAD) objects that depict the typical pipeline fundamentals which are here referred to as the pipeline model [2]. These pipeline models are got from the application domain of pipeline engineering.

To aid modelling devoid of any semantic and syntax difficulties of CAD systems, e.g., AutoCAD, feature diagrams can be used to transform these pipeline models into feature models in the sense of a pipeline product family [2]. This will result into a formal framework that can enable the progression into the actual development of the language metamodel capable of providing the linguistic power to solving the notable complexities with conventional modelling.

The feature models will now be used to represent the characteristically distinctive aspects of the pipeline products in a problem space (i.e. the problem domain), which will capture the commonalities and variabilities between the products that are visible to various stakeholders.

To achieve this, FODA (feature oriented domain analysis) methodology, which uses feature models in the formalizing of domain models, was adopted; primarily to aid in modelling the problem space by abstracting the components as the modelling elements and to bridge the gap between requirements and design [3]. Employing this method, we gave
mathematical proofs and formal definitions of the semantics of features, where each relevant characteristic of the problem space now becomes a feature in the model with composition rules describing which combinations of features are valid or invalid. This will form the breeding ground for proper language design and subsequent development. The remainder of this work is organised as follows: In Section 2, we describe the motivation and scope in the area of application; Section 3 describes the domain analysis; Both the formal analysis and definitions were given in Sections 4 and 5; Followed by related work in Section 6 and the conclusion in Section 7.

2. Motivation and Scope

The need for domain-specific languages has grown significantly over time. Particularly the focus on faster, easier and more consistent use of domain specific knowledge to aid designs.

The direction is how a language tool specific to the domain can enhance the modelling of pipeline designs and related systems. The underlying fact therefore is the ability to be able to process the domain model informal specifications of variability and commonalities into a formal model so that the problem space can be modelled.

This assertion became possible through the use of FODA methodology to formalize the engineering process; the effect is that the domain model is captured during language definition as the core of the concepts and the abstractions [3].

The scope of the system covers three collaborative subsystems. The user could make some input through guided notations from the interface, and the system can then match these inputs with a parsing grammar, with all of this formalism carefully defined in its internal mechanism. Presented in Fig. 1 is the proposed system architecture [4]. There are three collaborative sub-systems: the domain model captures the metrics of the pipeline engineering field; the user interface model enables stakeholder interaction with the system and a solution model that integrates into existing software for code generation and artefact orientation.

3. Domain Analysis

During the domain analysis phase, the BG Technical Nigeria Limited was visited for over a period of three months. BGT (BG Technical Limited) is a pipeline service company located in Nigeria with significant activity in Africa. Quite a lot of inputs regarding pipeline components and design criteria were sequenced through the company’s technical documents [5].

The domain of consideration is pipeline design including pipe sections joined with fittings and other supports features such as flanges, fittings, bolting, gaskets, valves, hangers and the pressure containing portions of other piping components. A pipeline design dedicated for transmission of oil and gas from wells to tanks for storage or to refineries for processing [6].

The graphics models which are here referred to as the pipeline model are computer aided design (CAD) objects that depict the typical pipeline fundamentals, materials and joints in situ as shown in Fig. 2 and which invariably forms the instance of the language creation [7].

Fig. 1  Scope and environment of the system.
The domain knowledge acquired is used for the formal analysis framework to form the feature models and as well became the repository for the new systems requirements [8]. Some of the key requirements are as follows:

1. Stakeholders identified interests such as physical attributes of pipeline components should be considered. Physical attributes are those parameters that govern the size, layout, and dimensional limits or proportions of the pipeline components;

2. The DSML has to have sound underlying engineering principles in each area and, illustrate how they are linked to produce a total life cycle approach to pipelines system design and operation;

3. The modelling language should identify functional commonalities of the application domain as an adaptation or refinement of the model;

4. The analysis module should have a feature model to support communication between the requirements analysis and the design phases respectively.

4. Formal Analysis

The first activity in pipeline design after creating the models is by using joints and fittings to fix them together according to specifications [9]. Fig. 3 is the listing of the components features. The design activity is usually performed under some controlled intent (i.e., what component comes in first at what dimension and position and for what pipeline activity to be met) of the domain expert or stakeholder.

Transforming these pipeline models into feature models in the sense of pipeline product family starts with the relationships of the features [9]. The feature model is a pipeline design problem space parent child relationship tree structure; with features forming nodes of the tree (Fig. 4). The root node pipeline(r) represents the concept that refers to the complete product line. The feature relationships are hierarchically arranged with relationships between a parent feature and its child features categorized into mandatory nodes, optional or alternative nodes with the arcs and groupings of features representing feature variability. All mandatory features such as component, fitting, joint and support, whose parent is the pipeline root concept, are common features. Also, the nodes in the feature diagram to which these features are attached are the variation points [10]. In this example, the domain consists of components, component dimensions and types, fittings dimensions and types, piping support features, points of joints and joint types and dimensions.

4.1 The Modelling Primitives

The modelling primitives defines the pipeline design system product line where each application contains five features c, f, j, b, and s, where b is an optional feature. It is an iterative tree grammar such that the optional child features of b, i.e., L, LR, and LN may or may not be included in the configuration of
Fig. 4 Feature model parent-child relationship.

The other parent features $c$, $f$, $j$, and $s$ are compound features that consists of mandatory features $D$, $P$, and $T$, which means that $D$, $P$, and $T$ must be included in the configuration if and only if $c$, $f$, $j$, and $s$ is included in the configuration [11]. The variabilities are now indicating precisely what evidence is needed to specify an instance of a feature that could fit into a particular modelling scenario, and then the commonalities in this respect define the set of common operations and primitives of the language.

4.2 The Composition Rules

In the feature model, there are atomic features and composite features [12]. The atomic features include pipe size in the form of NPS (normal pipe size) or DN (diameter normal), support types, joint types, fittings types and related ones that does not need to be further refined into sub features when there are no variations among different products. The composite features include pipe line components, points of contact, fittings, joints, pipe bed, and related ones that is composed of one or more atomic or composite features.

The composition rules in the form: $<feature1>$ (‘requires’ | ‘mutex-with’) $<feature2>$; which primarily relate these features to one another are clearly the constraints on the use of the features. One feature requires another simply means one feature requires the existence of another feature because they are interdependent, and one feature is mutually exclusive with another simply means they cannot coexist [13]. Examples of composition rules we have used in this analysis include the following:

- $elbowFitting$ requires $jointType$: In this case, the design system cannot have the $elbowFitting$ feature properly oriented unless the system has $jointType$ for complete pipeline design;
- $metreComponent$ mutex-with $pipeSupport$: If the entire pipeline is guarded with supports without flow meters, then there is no adequate position left in the pipeline to fix in metres to check flow properties;
- $pipeComponent$ requires $dimensionUnits$: If all the pipe cross section that made up the entire pipeline is without uniform dimension or is dimensionless, then there is no parameter to draw up the final design attributes from;
- $pipeOrigin$ requires $fittingTypes$: If the pipe origin is to be connected to source, before transmission can occur, then there would be nowhere to do the connection unless, the pipe is fitted to the source. Some branch and bend connections are needed to solve some more problems;
- $jointType$ requires $valveComponent$: If $fittingTypes$
are used, then there must be some arrangement to have control over the flow from origin to termination. To control the flow in the pipeline we need to fit a valve component:

- \texttt{pipeOriginTerminal} requires \texttt{pipeSupport}

Depending on the pipeline designer’s preference and judgment; if the entire pipeline is well planned out, then there is the need to support the pipeline and its components.

4.3 Feature Configurations

The formal definition entails the specification of configurations, since feature models have been semantically related to propositional logic; the analysis is complemented with both the conjunction \( P(x_1) \land P(x_2) \land \ldots \land P(x_n) \), and the disjunction \( P(x_1) \lor P(x_2) \lor \ldots \lor P(x_n) \) \[14\].

The interactive guidance and system response in this context is provided by possible assignments of features given the current state of the system, and propagating information whenever new choices are made.

In the feature tree (top) has a root feature \texttt{Pipeline} (r), mandatory features \texttt{Components} (c), \texttt{Fittings} (f), \texttt{Joint} (j), and \texttt{Support} (s), optional features \texttt{Pipe Bed} (b), alternative features \texttt{NPS} \( (d_1) \) and \texttt{DN} \( (d_2) \) and an exclusive-OR group containing grouped features \texttt{Meter} (m) and \texttt{Gauge} (g) (Fig 4). These are the standard reference attributes and relationships of the features of the family of the modelling system in the domain \[14\].

The feature configurations corresponding to the composition rules and the tree relationships are given below:

\begin{align*}
\text{Cross – tree 1.} & \quad \texttt{elbowFitting} \rightarrow \texttt{jointType} \\
\text{Constraint 1.} & \quad r \rightarrow c \land f \land j \land b \land s \\
\text{Cross – Tree 2.} & \\
\texttt{metreComponent mutex} & \rightarrow \texttt{pipeSupport} \\
\text{Constraint 2.} & \quad c \rightarrow t \land (\exists t \land s) \\
\text{Cross – tree 3.} & \\
\texttt{pipeComponent} & \rightarrow \texttt{dimensionUnits} \\
\text{Constraint 3.} & \quad c \rightarrow d \land (d_1 \times \text{or} \ d_2)
\end{align*}

\begin{align*}
\text{Cross – tree 4.} & \quad \texttt{pipeOrigin} \rightarrow \texttt{fittingTypes} \\
\text{Constraint 4.} & \quad c \land p \rightarrow f \land t \\
\text{Cross – tree 5.} & \\
\texttt{jointType} & \rightarrow \texttt{valveComponent} \\
\text{Constraint 5.} & \quad j \land t \rightarrow c \land t \Rightarrow f \land d \land p \land t \\
\text{Cross – Tree 6.} & \\
\texttt{pipeOriginTerminal} & \rightarrow \texttt{pipeSupport} \\
\text{Constraint 6.} & \quad c \land t_1 \rightarrow c \land t_2 \land (c \land f \land j \land b \rightarrow s)
\end{align*}

5. Formal Definitions

The syntax and semantics of the feature model along with the formal definitions make up the formal definitions. These definitions justify the different design scenarios that could possibly arise as a result of differing design intents of domain experts and stakeholders. The reason is that the modelling language system logic should be able to adapt positively to prompts from the interface with the concretized and familiar notations from the feature variations \[15\].

The syntax and semantic definitions framework represents a synergy between purported design intents, the behaviour of the language to such resource requests and the layers of the design and transformation modules.

The syntax of the feature model is the pipeline domain features organizational structure. The semantics indicates the configuration constraints such as feature attributes, feature interdependencies, and changes in system states due to feature compositions and the pipeline domain specific operational rules. For example, the specification of the positions of either the fittings or any other physical component is the semantics of the composed system \[15\].

5.1 Syntax Definition

Definition 1

The FM (feature model) representing the features of the pipeline design modelling language is a labelled graph with nodes \( n \in N \):

\[ n \in N \Rightarrow \{n/n \text{ is all pipeline features}\} \]

\[ \forall n \in N(n) \equiv \text{True for all features considered} \]
From the feature model in Fig. 4 there are six possible considerations for variability:

(a) Exclusive Or Node: denoting features that are realized by selecting exactly one child sub feature. They are drawn with an arc joining their outgoing links as shown in Dimensions in Fig. 4.

(b) Or Node: denoting features that are realizable by selecting one or more child sub feature, as seen in Meter and Gauge sub features under Component Type (T) in Fig. 4.

(c) And Node: denoting features that are realized by selecting all child sub features such as Joint, in Fig. 4.

(d) Nodes have appropriate arity:

(e) A node which is conventionally an and-node type is the unique concept of the feature diagram or root \( r \in X \) drawn at the top of the tree. An example is Pipeline \( r \in P \) in Fig. 4.

(f) Leaf \( f \in X \) are features drawn at the bottom of the diagram.

Definition 2

The FM (feature model) representing the features of the pipeline design modelling language is a labeled graph with edges \( \rightarrow N \times N \):

\[ x \in N \Rightarrow x \text{ is all directed edges of feature model} \]

\[ = N(x) \forall x \text{ in the domain} \]

\[ \Rightarrow x N(x) = \text{TRUE for all considered edges} \]

\[ = N(x_1) \lor N(x_2) \lor \ldots \lor N(x_e) \]

Referring to Fig. 4, the edges have directions shown by the arrows. Starting from random node \( N \) and walk through all possible edges \( x \) (following the edge directions), and never return to \( N \). It implies then that the edges \( \rightarrow N \times N \) labelled mandatory \( m \) from root concept \( r \); is a 5-tuple such that \( FM = (N, P, r, \lambda, DE) \);

where

\[ N = \{ n_1, n_2, \ldots, n_e \} \text{ is the set of nodes}; \]

\[ P \in N \text{ is the set of primitive nodes}; \]

\[ r \in N \text{ is the root of the FM}; \]

\[ \lambda: N \rightarrow NT \text{ labeled node type}; \]

\[ DE \in N \times N \text{ is the set of decomposition edges}. \]

From the tuple, five evidences to specify an instance of a feature that could fit into a particular modelling scenario are as follows:

(a) Only \( r \) has no parent:

\[ \forall n \in N, (\exists n' \in N, n' \rightarrow n) \Leftrightarrow n = r \]

(b) \( DE \) is a tree:

\[ \exists n_1, n_2, n_3 \in N, n_1 \rightarrow n_2 \land n_3 \rightarrow n_2 \land n_1 \neq n_3 \]

(c) \( DE \) hierarchy is acyclic:

\[ \exists n_1, \ldots, n_e \in N, n_1 \rightarrow \ldots \rightarrow n_e \rightarrow n_1 \]

\[ \Leftrightarrow \neg(n_1 \rightarrow \ldots ightarrow n_e \rightarrow n_1) \]

(d) Nodes have appropriate arity:

\[ \forall n \in N, \lambda(n) = p_e \rightarrow n' \]

(e) \( n \in N \rightarrow \) constraints

\[ n^1 \text{ mutex } n^2, \text{ or } n_1 \text{ requires } n_2 \]

5.2 Semantics Definition

The formal semantics definition is a pipeline design product line model (PM). Where a product (P) of the feature model holds information regarding a set of leaves, the model (M) of the FM holds values of products as a subset of primitive nodes, and some constraints (C) as the rules guiding the system specific operations.

A PM is a triple \( (P, M, C) \)

where

\[ P = \{ p^1, p^2, p^3, \ldots, p_e \in P \} \text{ is the set of leaves } P = p(F); \]

\[ M = m \in M \subseteq \text{ nodes of the FM: } M = pN; \]

\[ C = \phi \in \Phi \text{ are the satisifiable constraints.} \]

Definition 1

The product line \( P \) with a set of leafs (F) of the FM is a valid model \( m \in M \) if

(a) The concept is in the model: \( r \in m \)
(b) i. with a xor - node in the model, exactly one of its sub features is $q \in m$
\[ \Rightarrow \exists \ n. \ q \rightarrow n \land n' \in m \]
ii. with and - node in the model, all its sub features are $k \in m$
\[ \Rightarrow \forall n. \ (k \rightarrow n \Rightarrow n' \in m) \]
(c) The model satisfies
i. the constraint set: $\forall \phi \in \Phi, m \vdash \phi$
where
$m \vdash n_1 \text{mutex } n_2 \rightarrow n_1 \text{ and } n_2 \notin m$
$m \vdash n_1 \text{ requires } n_2 \rightarrow n_1$
\[ \text{ and } \forall \phi \in \Phi, m \vdash \phi, \text{ where } \]
where $m \vdash \phi_1 \rightarrow p_1, m \Rightarrow p_2, m \text{ is true}
iii. the graphical constraints:
$\forall (n_1 \ast n_2) \in C, (n_1 \in m, n_2 \in m) \text{ is true}
(d) Two possible semantics:
\begin{itemize}
  \item i. node - based:
  \[ \forall \text{ mandatory edge } (e) \in m \]
  \[ \Rightarrow n_1, n_2, n_3, \ldots, n_e \in m \]
  \[ e_1, e_2, e_3, \ldots, e_m \in m \rightarrow (e \in m, \ldots, e \in m) \]
  \item ii. edge - based:
  \[ \forall \text{ mandatory edge } (e) \in m \exists! \ e \in m \]
\end{itemize}
Definition 2
The product line $P$ with a set of leafs $(F)$ of the FM is the set of leaves $P = p(F)$ if
(a) $P$ is a set of products: $p(P) = p(p(F))$
(b) $P$ is the products of the FMs valid models, such that
$[p] = (m \cap F \setminus m \vdash p)$
(c) A product $k$ is a set of primitive nodes
\[ k \in p(P) \]
6. Related Work
A PDML (pipeline design modelling language) is simply a domain specific modelling language whose pre-constructed notations and abstractions only offer expressive power to the pipeline engineering domain. As usual it has its own syntax and semantics definition, which we have presented in this paper using feature oriented domain analysis. We highlighted the usefulness of feature models or diagrams to formalizing the syntax and semantics definitions of a domain specific modelling language in the context of a product line; however a few more formally defined schemes have been identified.
Zhao et al. [16] defined the domain directly as a language. They called it an open operator definition, which gives much flexibility to the language for the evolution of operators of a domain, and any compositions in the domain are the relationships presented by the grammar rules and are not physically represented by any symbols. The identified differing point is that, they were simply interpreting the feature compositions grammatically, while in ours, the composition rules of the features were clearly set out for a separate grammar construction action applying relevant language development methods.
Batory [17] translated a feature diagram to both grammars and propositional formulae. The semantics of the grammars is a set of iterative tree with string tokens, and thus repetition was possible. His definitions are close to ours but differs in the respect that there is no clear separation of decomposable features. Shan et al. [18] presented a framework for the formal definition of the abstract syntax and type systems of modelling languages that facilitates formal specification and automatic checking of consistency and completeness constraints on graphic models. The approach illustrated a robust specification of the consistency and completeness constraints of a modelling language, but we are much more interested in providing a modelling tool for domain experts.
Czarnecki et al. [19] define a new FD language to account for staged configuration. They introduce feature cardinality (the number of times a feature can be repeated in a product) in addition to the more usual (group) cardinality. Foremost, a new semantics is proposed where the full shape of the unordered tree is important, including repetition and decomposable features.
From our point of view, we can argue that related approaches have some varying grounds with our approach on generality, and abstraction. In most of the
approaches the semantics is tool-driven, while we are concentrating on building a tool on a carefully chosen and well-studied semantics. For others a FD language is defined with feature cardinalities to be explicitly applied, currently, we are limited in defining cardinalities rather we focused more on interdependencies in the formalization scope.

7. Conclusions and Future Work

Throughout this paper, we have focused on the use of FODA (feature oriented domain analysis) formal analysis method in defining the syntax and semantics of domain specific modelling language. We have thus tried to present a formal framework of the syntax and semantics definitions of a pipeline design modelling language.

The primitive types and semantics will formalize the structure, behaviour and requirements within the domain of pipeline engineering as the system development progresses to offering a meta-language whose semantics are familiar only to pipeline design mechanisms.

These formal definitions presented useful proofs for variability management in the application domain in terms of the components and the requirements, and will bridge the gap between the requirements documentation and the language design. Future work is concentrated on the language design based on the carefully studied and defined semantics.

References