Chapter 6

ONTOGRAPHY BASED FEATURE MODELS

This chapter throws light into the use of ontology to achieve dynamic reconfigurations. Feature model can help reconfigurations, it is the model which shows graphically the similarities and variability of SPL. It has its notations. Feature models are being used to represent variability and similarities in SPL and it helps in optimizing configuration management and product derivation. However, in this chapter, the focus is on the usage of ontology to formally represent feature models in order to have a flexible and programmable interface. The programmable interface thus achieved can be used to have automatic reconfiguration of SPL. It also will help in bringing dynamism and speeds up product derivation process. Quality product derivation with optimization is possible with ontology. The rest of the chapter throws light into the usage of ontology for improving configuration management and product derivation.

6.1 ONTOLOGY vs. FEATURE MODEL

The feature model has certain drawbacks with respect to dynamic reconfiguration. [131] Especially the model is not programmable. Therefore it is required to find out a model which is very dynamic and programmable in order to achieve dynamic and automatic reconfigurations in SPL. Since SPL undergoes constant changes, it is inevitable to automate the reconfigurations for the high quality of services. The new model which can represent features and allow easy retrieval of features and possible modifications in real time is an ontology.[132] Ontology is a proven model which can be adapted in order to represent features well and show the relationships among the features besides supporting machine learning or programmability.

Software Product Line (SPL) refers to a set of products with common features and variability to cater to the needs of customers. In a customer-centric approach, the variability in SPL is driven by the genuine needs of loyal and prospective customers. In SPL, at the time of development, both core and custom assets are required. The
SPL development is a complex phenomenon. It needs the support of configuration management and sometimes automatic reconfiguration is a desired feature.

Ontology is best used to formulate knowledge theories. In other words, knowledge representation can be done using ontology. Sharing and reusing knowledge is the fundamental role of ontology. In this thesis, it is used to formally represent features models associated with SPL configuration management. Ontology can graphically represent knowledge in the form of concepts and shows relationships among the concepts. It can be used to represent knowledge in different fields such as healthcare, education, scientific research, and space technology to mention few. It can support concepts pertaining to both space and time. The following are the advantages of ontology.

- Reusability is the greatest advantage which supports the reuse of concepts across different modules.
- Formal representation of knowledge is good to have a graphical view which makes sense and lets people understand just by a glance.
- Interoperability is a very important advantage as it can help different platforms to share knowledge represented in the form of ontology. For instance, Java platform can make use of ontology modeled in .NET platform.
- Ontology can help in the acquisition of model-based knowledge. Model-based knowledge can be obtained and represented in the form of concepts and relationships among the concept.
- Ontology can also help in validating and verification of knowledge-level. This is useful in applications where the ontology is used to follow guidelines or procedures.

Ontology is identified as one of the technologies that can be used to represent the feature models of SPL in the form of concepts and relationships among them for effective variability management. Feature models of SPL are represented in the form of ontology. As the ontology model is programmable, it is exploited to have dynamic and automatic reconfiguration of SPL.[133] The knowledge representation capabilities and visualization mechanisms of ontology made it be intuitive and user-friendly.
Ontology vs. Enterprise Applications

When ontology is considered for knowledge representation a broad spectrum of distributed computing and the technologies come into the picture. Enterprise-wide applications that combine businesses and provide a single point of access irrespective of platforms can be leveraged with ontology. Stating differently ontology usage can improve the fundamental communications in the stakeholders involved in enterprise applications. As such applications are very complex and distributed in nature, it is useful to have ontology to represent the enterprise-wide knowledge, share it and reuse it. It also can help as formal communication to all stakeholders for intuitive understanding of the enterprise. Reusability leads to flexibility and combining different related businesses of enterprises and representing the whole knowledge in the form of ontology is the key for successful modeling and implementation of enterprise-wide applications.

Moreover, XML can be used by enterprises to have seamless integration on top of ontology. The knowledge represented by ontology can be converted into an intermediate form that is XML. XML is very flexible and useful format which can be understood by any language, any platform, and any device. In fact, XML is the standard for representation of data in interoperable fashion. Therefore it is an ideal candidate to have formal exchange of information between platforms. Thus the usage of XML can help improve the communications among different stakeholders involved in different applications that are part of an enterprise. XML is a W3C standard which contains pure data without any formatting details. Ontology can be converted to XML with automatic programming approach. In other words, it is possible to convert ontology into XML programmatically. The vice versa is also possible as XML is interoperable. From this discussion here it can be concluded that both ontology and XML can leverage the development process and communication among stakeholders in enterprise applications.

Ontology can also be used to represent semantic web. The semantic web is an extension to Web through the W3C given standards. Those standards promote data formats used to exchange data. Specifically, they promote the Resource Description Framework (RDF). The usage of ontology, XML and RDF can be used in enterprise
applications in order to have better data exchange and better communication with all the stakeholders in different SPLs if any.

**Ontology vs. SPL**

Ontology, as stated earlier in this chapter, is to represent knowledge in a structured and hierarchical way. The knowledge of different things can be captured using ontology. Those different things include features, various constraints associated with features and semantic relationships among architectural elements and their features. Ontology can leverage the modeling of knowledge in any domain. With respect to SPL development, ontology can be used. More usage of ontology can be realized using the ontology for configuration management and product derivation. Ontology-based access to artifacts and configurations, ontology-based reasoning and decision making is possible in SPL. Model transformation is one of the usages of ontology where knowledge represented in one model is transformed another model. Transformation rules can be defined and applied and used on product derivation. SPL architecture can be transformed from architecture context to product architecture using ontology. Ontology generation, ontology-based reasoning; ontology based feature selections and ontology-based query interface can be realized. Rule generation engine, model transformation engine, and product derivation can be used in SPL configuration management and product derivation. In this thesis, ontology is used to represent domain ontology in the form of different concepts that reflect underlying artifacts of an SPL for quality product derivation. It is also used to optimize SPL configuration management.

The ontology usage presented in this chapter reveals two steps. The first step is to have domain ontology API which can be reused in different domains. The second step is to actually construct domain ontology by instantiating domain ontology API. The constructed domain ontology can represent knowledge pertaining to an SPL in terms of artifacts and products represented as concepts and relationships among them. Given a set of features in the form of feature models, they are formally represented by an ontology for better processing and optimize the process of configuration management and product derivation.
6.2 STATE-OF-THE-ART

Dynamic Software Product Line (DSPL) is the term used for reflecting an SPL with dynamic reconfigurations and product derivation. As explored in [135,136,137,138] the DSPL becomes more intuitive when the variability and features are visualized. Variability models can exploit machine processable paradigms for better configuration management. In the literature, it is found that ontology is one of the models that can deal with complexities in the systems [139]. Software development environments can be integrated with semantics using ontology. This can help in reusing software component and thus enhance reusability in software development [140]. Feature models and variability models were used in the past for dynamic configuration of SPLs. In software engineering, there was evidence of usage of ontology for better service orientation. The philosophy of SPL is adapted and used for service orientation as explored by Lee and Kotonya [141]. Quinton et al. [142] and Quinton et al. [143] explored the usage of ontology in a model-driven framework for cloud ecosystem and automatic selection of different environments with variability. Variability features are considered for the agile model [144] and the SPLE. Configuration problems are resolved by researchers using different approaches. For instance, Mayer and Albert [145] proposed a model known as Constraint Satisfaction Problem (CSP) that could model issues pertaining to configurations besides helping to solve them.

German et al in [146] exploited Web Ontology Language (OWL) and semantic web by proposing a model known as Ontology-Based Context Model. Their model helped in formal concept analysis and representation of domain knowledge. Ontology is used for other purposes like achieving interoperability among manufacturing entities besides data management without the need for losing semantics [147]. Representation of open source software, feature based configurations, preserving semantics, representation of intelligence in SPL middleware, reconfiguration of service-based processes, evaluation of logistics distribution are other utilities of ontology usage.

Botzenhardt et al. [148] explored the usage of ontology for representing knowledge related to product management. With respect to testing SPL, feature models are used in [149] for ensuring reduced test cases in test suites. Brian et al.
used the ontology to evaluate SPLs and in [152] it is used to represent knowledge related to ISO standards that leverage high standards in the software industry. In this chapter, the framework proposed in Chapter 3 is enhanced to support knowledge representation of SPL in the form of ontology and improve the dynamic reconfigurations of SPL and product derivation.

To exploit the good features of ontology, in this chapter possible usage of ontology to represent features in the SPL for automatic reconfigurations and evolution of new products in the product line is explored. The framework presented in Figure 3.1 is enhanced in terms of implementation of ontology for further optimization of SPL configuration and product derivation. In the previous chapter, the framework was enhanced with variability–aware design patterns. However, this chapter throws light into ontology usage for enhancement of the framework. The prototype is updated in order to support the ontology feature for representing the features in the form of concepts and the relationships among them. The empirical results revealed that ontology proved to be a useful candidate for automatic reconfiguration of SPL besides deriving a quality product based on customer needs.

6.3 PROPOSED METHODOLOGY

The feature models are formally represented using domain ontology in order to support change management, improving latency, and automatic reconfiguration without human intervention. The procedure using which ontology is used to leverage dynamic reconfigurations and quality product derivation is described here.

Step 1 A feature model from SPL is extracted.

Step 2 Then it is assigned to an instance of domain ontology API.

Step 3 The instance is populated with true dynamics of SPL in terms of features and variability.

Step 4 The runtime instance of domain ontology represents the SPL.

Step 5 Object Graph Navigation approach is followed to navigate through the domain ontology in order to obtain details of assets, there versions and features.

Step 6 The features extracted from the SPL are formally represented in the form of concepts with relationships among them if any.
Thus ontology is constructed which helps in change management.

Once the ontology is constructed it can be used to have dynamic reconfiguration of SPL. Since the ontology is programmable, it is possible to navigate through different parts of it and process the concepts represented. This is why features are formally represented in the form of concepts in ontology model. Thus the variability management is simplified and the variability is found to have real-time decision making in the SPL with respect to configuration management. The variability caused by the production environment and the modifications made if any are reflected in the ontology in real time.

Changes encountered are captured and following necessary steps are used for dynamic reconfiguration in SPL.

Step 1: Construct Ontology.

Step 2: Set the current state property of the features of the SPL to the “selected” value and “eliminated” for the other features.

Step 3: Depending on the requirements query is written for update of the property type or a new feature to be created based on the requirements.

Once the hierarchy of domain API is constructed it can be reconfigured dynamically according to different scenarios. This section describes scenarios directly related to changes in the type of a feature and also changes related to product configuration.

For example, selecting an optional feature in a product: This is one of the most common changing scenarios in SPLs. Usually, a product is firstly generated according to customer’s needs at development time and includes several features and, in such moment, an optional feature would not be selected to be in the product. However, at a later moment, it is possible that a client demands the inclusion of an optional feature that was not previously addressed by the configured product. In the dynamic software product line context, it is important to specify a mechanism which can reconfigure the product to reflect the current requirements of the client. For instance, in our
running example, despite in the TRS cash payment, the feature payment by card was not originally selected to be in the TRS configuration. However, in a changing scenario, the client would like to include it in his product. To achieve such change, it is necessary to change the property that indicates that the feature is present or not in the system in the ontology of the product at runtime. The update of this property can be realized by a query. The ontology of SPL can also help in a faster derivation of quality products based on customer needs.

**6.3.1 Extended SPL Configuration Framework**

The framework was proposed for SPL configuration management and product derivation in [129]. Then the framework is enhanced in [151] for supporting variability-aware design patterns. Afterward, the framework is further improved by using ontology and that is presented in this chapter. Figure 6.1 shows a part of the framework that is to optimize dynamic reconfigurations and product derivation. With this enhancement, the framework now supports four objectives.

- Software configuration management
- Change management and change propagation
- Enhancement through variability aware design patterns
- Optimization through domain ontology for leveraging automatic reconfiguration of SPL and product derivation.

In this chapter, the focus is on ontology-based improvement to the framework. Especially, it throws light into the formal representation of SPL features in the form of ontology. However, the framework and its technical details can be found in [129] and [151]. Ontology usage leveraged the monitoring, navigation and change management and led to an automatic reconfiguration of SPL and quality product evolution.

As shown in Figure 6.1, the feature models are taken and then ontology is constructed in order to have flexible and programmable navigation to various features. In other words, the ontology construction can help to represent the whole SPL.
This will facilitate the application to have access to any part of the SPL with ease. This is an important observation which leads to the automatic reconfiguration of SPL. Feature models have provision to represent variability in SPL as explored in [147]. When it comes to new product derivation, the variability plays a vital role. The explicit and formal representation of feature models in the form of ontology can improve the utility of the application.

Thus ontology is exploited and used in SPL domain engineering. The Feature-oriented domain analysis (FODA) explored in [148] is the basis for the ontology construction. Features in the feature models can have the types of models associated. The types include alternative, optional, or feature and mandatory. The mandatory features must be part of all products whereas, the optional features are kept in product based on the requirement. Alternative refers to the fact that only a single feature from a set of features is used in the final product. Or-feature feature type refers to the fact that one or more features from the available features can be used in the product from a set of given features. Requires refers to a rule that tells that one feature expects the presence of other features. Exclude refers to the fact that two features are exclusive in nature. If one feature exists, other feature cannot exist.
6.3.2 Ontology API

Application Programming Interface (API) is a set of classes and interfaces. API is generally in the form of libraries that can be reused. In this thesis domain ontology is proposed in order to represent SPL and particularly the feature models. As presented in Figure 6.2, the features of any SPL can be formally represented using the proposed ontology construction API.

![Hierarchy of domain API](image)

Figure 6.2: Hierarchy of domain API
The API is built in Java and can be used to have an instance of ontology construction. Once ontology instance is created, it is possible to have a formal representation of concepts involved and the relationships among the concepts. The API defined for domain ontology construction is shown in Figure 6.2. The root class in the domain ontology API is known as Thing. Its immediate subclasses are Feature Constraint, Features, Software Product Line, and Feature Model.

Table 6.1: API for domain ontology

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoftwareProductLine</td>
<td>Represents an SPL. It has data members like name, FeatureModel, and description.</td>
</tr>
<tr>
<td>FeatureModel</td>
<td>Represents feature model. It has data members like name, Feature, and FeatureConstraint.</td>
</tr>
<tr>
<td>Feature</td>
<td>It represents a resource available in the SPL. It has data members like name and currentState.</td>
</tr>
<tr>
<td>Mandatory</td>
<td>Represents a resource which is mandatory. It has data member name.</td>
</tr>
<tr>
<td>Optional</td>
<td>Represents a resource which is optional. It has data member name.</td>
</tr>
<tr>
<td>Alternative</td>
<td>Represents an alternative resource of the SPL. It has data members like name, alternativeFeature and exclusive.</td>
</tr>
<tr>
<td>OrFeature</td>
<td>Represents an or exclusive resource with data members like name and alternativeFeature.</td>
</tr>
<tr>
<td>RootFeature</td>
<td>Represents a root feature with a data member name.</td>
</tr>
<tr>
<td>FeatureConstraint</td>
<td>Represents a constraint in the feature model. It has a data member name.</td>
</tr>
<tr>
<td>Depend</td>
<td>Represents a constraint which is a dependent type. It has data members like name, TargetFeature and SourceFeature.</td>
</tr>
<tr>
<td>Exclude</td>
<td>Represents a constraint which is excludes type. It has data members like name, TargetFeature and SourceFeature.</td>
</tr>
<tr>
<td>Group</td>
<td>Represents a constraint which is group type. It has data members like name, setFeatures, typeConstraint. The constraint type can be zero or one, at least one, exactly one, any feature and all features</td>
</tr>
</tbody>
</table>
The subclasses of Features class include Exclude, Group, Depend, Root, Or Feature, Optional, Mandatory, and Alternative. The description of these classes is provided in Table 6.1. The classes described in Table 6.1 are appropriately used in the construction of domain ontology. The API is not specific to any domain. In fact, it can be used for any domain. This common API can be instantiated and used for different domains. Thus it is made generic and domain independent. When domain ontology is generic and can support any domain, it is reusable. It promotes reusability. In all kinds of SPL, the domain ontology can be used again in order to represent feature models formally.

6.4 DOMAIN ONTOLOGY CONSTRUCTION

Domain ontology refers to the ontology that reflects the concepts and their relationships found in a particular domain. The construction of domain ontology is made easy with the API defined. There are many steps in the construction of domain ontology.

In step 1, core assets available are obtained.
In step 2, custom assets available are obtained.
In the third step, all the products available in the given SPL are retrieved.
In step 4, domain concepts are identified and relationships among them are identified.
In step 5, ontology is built with concepts and the relationships among them.

The five steps mentioned above can be used to build an ontology with concepts and possible relationships among them. This actually needs a more formal means in order to realize the construction of ontology for given domain and SPL. To achieve this, an algorithm named Ontology Construction Algorithm is proposed and implemented. The algorithm is common and generic for any domain and any SPL. Therefore its usability is maximized and it can be reused in different domains instead of reinventing the wheel. Once domain ontology is constructed based on the feature models of given SPL of a particular domain, it is possible to use it for further processing with respect to configuration management and also product derivation. The algorithm is designed in such a way that it avoids usage of statements that are
SPL dependent or domain dependent. The aim of the algorithm is to support a formal construction of ontology i.e, knowledge representation that represents formally the underlying feature models of SPL in order to have better navigation and dynamic operations with ease. The construction of ontology is subjected to the identification of concepts and relationships among them. Once the domain ontology is constructed, it supports a programmable interface which can help navigate and identify different concepts and they can be used for optimization of configuration management and product derivation.

The ontology construction algorithm is implemented using domain ontology API build in Java. Thus the domain ontology API can be reused in many applications. Moreover, the API built in Java can be used on any platform by exploiting the WORA features of Java. Since Java is platform independent it is possible to reuse in different platforms. Moreover, the technologies like CORBA and Web Services can be used in order to have perfect interoperability in distributed computing environment. As distributed technologies are prevailing and widely used in the real world now, it is important to have interoperability and the ability to be reused in different applications. Since ontology can be reused the reusability is promoted and taken to the next level. The purpose of domain ontology is realized by formally representing feature models and having machine processed approach for the modifications to be made in terms of reconfiguration of SPL artifacts. Moreover, the algorithm proposed in this chapter paves way for optimization of the framework proposed and described in Chapter 3. Different approaches are there in the real world for knowledge representation while the ontology is the most used and preferred approach for enabling platforms to share and reuse concepts and the underlying details in the form of relationships. Therefore it is imperative that ontology is best used to promote better navigation, programmability, and machine readable approach to realize the optimization of SPL configuration management and quality product derivation. The algorithm proposed below is in the form of pseudo code that is used to build such functionality in the prototype application developed in this thesis for proof of concept.
**Algorithm:** Ontology Construction Algorithm

**Inputs:** SPL

**Output:** Domain Ontology \( \text{DO} \)

01 Initialize vectors \( \text{CA}, \text{CA}_1, \text{P}, \text{DO} \)

02 Obtain core assets to \( \text{CA} \)

03 Obtain custom assets to \( \text{CA}_1 \)

04 Obtain products into \( \text{P} \)

05 For each \( p \) in \( \text{P} \)

06 For each \( ca \) in \( \text{CA} \)

07 IF \( ca \) belongs to \( p \) THEN

08 Associate \( ca \) with \( p \)

09 END IF

10 End For

11 End For

12 For each \( p \) in \( \text{P} \)

13 For each \( \text{ca}_1 \) in \( \text{CA}_1 \)

14 IF \( \text{ca}_1 \) belongs to \( p \) THEN

15 Associate \( \text{ca}_1 \) with \( p \)

16 END IF

17 End For

18 End For

19 For each \( p \) in \( \text{P} \)

20 Add \( p \) to \( \text{DO} \)

21 End For

Return \( \text{DO} \)

Listing 6.1: Algorithm for domain ontology construction

As shown in Listing 6.1, the SPL is taken as input and the domain ontology is constructed as output. The algorithm extracts core and custom assets from the loaded domain ontology. Then it obtains the available products associated with SPL. It follows an iterative process which identifies the usage of assets with different products. Once concepts are constructed in the form of different products, then the concepts are added to the domain ontology. There are relationships between the
concepts. The data source maintains in the backend is the source to know the artifacts and their relationship with different products. Therefore, the data source is the basis for feature models and ultimately for ontology construction. In the algorithm different vectors are used to hold data about products, core assets, custom assets and the relationship among them. After construction of product vector denoted by P, the domain ontology is built. Each product is represented by a concept in the domain. Everything else is associated properly with the product. Once the ontology is constructed, it is visualized and the visualization can help intuitive understanding. On the other hand, the application can read data from ontology instance so as to process data as per the requirement. The ontology constructed for LMS SPL is as shown in Figure 6.3.

As shown in Figure 6.3, it is evident that LMS SPL is visualized with concepts and relations. A concept represent a set of things in the domain. For example, in the Figure 6.3 CoreAss 1.0, CoreAss 1.1, CustomAss1.1.1 etc., are the concepts in this particular case. The runtime instance of domain ontology API facilitates the essence of an SPL. After construction of an instance of the domain ontology, the instance is
populated with \textit{DO} object returned by the algorithm appropriately. The domain ontology constructed can be used for having a programmable interface for features available in the SPL. The reason behind this is that ontology is capable of providing an interface that can be exploited for machine processing without human intervention. The core assets, custom assets, and products involved in an SPL comes under the purview of knowledge. This knowledge is made visible and graphical as shown in Figure 6.3. When knowledge is properly represented and relationships are established, it is easier to have programmable navigation among the concepts and obtain intelligence as required for automatic reconfiguration and quality product derivation.

Another advantage of domain ontology built here is to provide intuitive understanding to stakeholders of the SPL. All stakeholders can have a formal means of understanding knowledge, sharing it and reusing it to the extent possible. When developers in the SPL development team can have a view of the system as expressed, it is possible to reinstate and speed up ascertaining of facts to make well-informed decisions. The mapping of concepts or knowledge components to specific products can reveal the underlying knowledge that can help in processing pertaining to configuration management of SPL. This knowledge can also help to have formal means of accessing artifacts of different versions while constructing new products. New product derivation is thus made easier besides being able to identify the best possible product to be derived.

\section*{6.5 RESULTS AND DISCUSSION}

The aim of experiments presented in this chapter is to construct domain ontology for different SPLs and make important observations pertaining to domain ontology construction and the usage of the same for automatic reconfiguration of SPL and the quality product derivation. The proposed framework presented in Chapter 3 is enhanced in order to accommodate domain ontology construction and demonstrate the utility of such representation in order to simplify reconfiguration of SPL. As the domain ontology is machine readable, the application has improved flexibility and performance. Human experts are involved in evaluating the proposed framework keeping the focus on domain ontology construction and its utility in mind. The experimental observations also include the execution time taken for domain ontology
construction and modification besides reconfiguration. The time taken in milliseconds is considered as a measure to evaluate the performance of the proposed system. The performance reveals the difference between the system with and without ontology usage for optimizing configuration management and quality product derivation. Optimization of the proposed framework was done already with the usage of variability-aware design patterns covered in chapter 5. In this chapter further optimization of the framework and prototype are made using ontology. The prototype can now demonstrate the usefulness of ontology for improving performance in configuration management of SPL and quality product derivation.

As shown in Figure 6.4, the horizontal axis represents different SPLs while the vertical axis represents the time taken for construction and modification of domain ontology. As observed in the graph there is a difference in execution time and modification time for different SPLs based on the differences among them in terms of size. In fact, the SPLs do have similarities and variability and differences in degrees of complexity. Dr. School SPL exhibited more time taken when compared with that of others. Least time is taken by LMS when compared with that of others in the

![Figure 6.4: Performance evaluation in terms of ontology construction and modification for different SPLs](image-url)
experimental results. Considering the bulk of SPLs in the real world, the measure used here to differentiate the performance of the proposed algorithm for different SPLs has got the significance. An interesting observation is that the modification of already constructed ontology took less time when compared with the time taken to construct domain ontology from the scratch. From the results, it is understood that usage of domain ontology not only brings about intuitiveness in the optimization of the framework but also improves performance in terms of time taken. As shown in Figure 6.5, there are three SPLs plotted in horizontal axis while the vertical axis represents reconfiguration time. Reconfiguration is the process of updating the configurations as per the change dynamics in SPL. Since the changes are made constantly in a product line, it is essential to have an automated and effective approach for reconfiguration. The changes made to the artifacts in terms of features are to be updated in the configuration automatically thus making configuration management consistent and in tandem with the ongoing development efforts.
As shown in the results there is a significant time difference between Dr. School SPL and others when compared in terms of reconfiguration time. The differences in the time taken are attributed to the level of complexity existed in the SPLs. As said earlier, the time taken is the measure which got importance as the real world SPLs has very complex SPLs with diversified functionalities in terms of artifacts. The reconfiguration time is based on the number of concepts in the underlying domain ontology constructed. As there are SPLs which are highly complex, the time expended for reconfiguration differs.

The number of features present in an SPL can have its influence on the time expended for constructing ontology. Since the feature models are the basis for ontology construction it is quite natural that features have a significance impact on the time taken for constructing domain ontology. Therefore it is observed in the experimental results that the time taken is directly proportional to the number of features in the SPLs. This was found true with all the case studies considered. When the number of features is increased, the concepts and the underlying relationships may increase. Thus the domain ontology brings about complexity in the SPL. In other words, the complexity of the SPL is revealed in the form of the domain ontology. However, it can be said that very complex systems are made intuitive to understand with domain ontology as it provides visibility to stakeholders of SPL. Thus the
communication among the stakeholders can improve significantly. When ontology can be used to have formal communication, sharing technical information and reusing them, it is evident the utility of the configuration management and product derivation is improved.

![Bar chart showing optimization of product derivation with ontology](image)

**Figure 6.7: Optimization of product derivation with ontology**

As shown in Figure 6.7, the SPLs are presented in horizontal axis while the vertical axis represents the time taken for new product derivation. Since time is a very important measure for real world activities, it is considered here to know how much time is required in order to derive a new product in given SPL. Since ontology is programmable, it is considered to have a certain analysis made by the algorithms proposed in Chapter 4. Therefore product derivation time is an important measure to know how quickly the product can be derived as per the needs of customers. Keeping this in mind, experiments are made using the application with and without domain ontology usage.

Results revealed that the time taken for product derivation with ontology is significantly less for all SPLs. Therefore, it can be concluded that domain ontology usage has optimized the product derivation process. Without ontology, the product derivation took more time as the process is not optimized. Ontology usage has improved the system in analyzing and producing different permutations and combinations prior to final best product derivation. The product derivation with domain ontology has shown significant performance enhancement in SPLs.
6.6 THREATS TO VALIDITY

The focus of the thesis is to propose a comprehensive framework for configuration management and quality product derivation in product line software systems. In the chapters three, four, five and the present chapter, the above task was accomplished by implementing quality product derivation and composition algorithms. A prototype was built using Java platform to evaluate the above functionality. The prototype was further enhanced with introducing variability-Aware-Design Patterns and ontology based feature model for improving configuration management and product derivation. There are four types of threats to validity. They are construct validity, conclusion validity, internal validity and external validity.

Construct validity concerns whether a test measures intended construct. Constructs used in the questionnaires for ground truth evaluation of the prototype have been well defined. The scale used to compare evaluation parameters like configuration identification, change control, usability etc was explained to all participants. Thus, the results from each respondent are not dependent on a single answer. There is always a risk that respondents guess the hypotheses at hand. By making sure questions are answered in different order, we balance both the threat of hypotheses guessing as well as the threat of interaction between different questions and evaluation parameters.

Conclusion validity is the degree to which conclusions we reach about relationships in the parameters used and the outcome. Also conclusion validity ensures use of adequate sampling procedures. The parameters used in the study were appropriate to the problem to be solved. The data collection was based on the participants who were well experienced software engineers. But the sample size was ten participants thus, there might be insufficient statistical power on the effects of the results.

Internal Validity refers to how well the parameters chosen for the study are appropriate for the study. In the present research the main focus was on configuration management and quality product derivation. The parameters used for the study are configuration identification, change control and maintainability for assessing configuration management and the parameters like reliability, usability efficiency etc.,
are taken for measuring quality. Firstly, the parameters chosen for the study are very much appropriate for the study under consideration. Secondly, Testing effects have been minimized by making sure that all the questions are answered by all the participants.

External validity concerns generalization of the findings to other contexts and environments than the one studied. All participants are well experienced software engineers with expertise. They are actively working at various software companies in diversified job roles in software development. In this thesis, external validity is considered based on two important aspects. First, there is no statistically significant difference among the participants of evaluation. Second, which is more important for the external validity, is on generalization of findings. Although industrial software case studies are used to evaluate the prototype, they may not be representative of SPLs in different domains. Therefore, the generalization of findings is the main threat to external validity.

SUMMARY

This chapter has provided details of the proposed framework with enhanced functionality in terms of domain ontology usage for improving product derivation and configuration management. Ontology is used to represent feature models in order to have more flexible and programmable approach for automating product derivation and configuration management. Since ontology provides a formal way to represent feature models besides supporting easy navigation among concepts associated with the constructed domain ontology, it has got significance in its utility. In other words, ontology represents variability and similarities in SPL in a better way. This is evident in experiments as the representation of domain knowledge had its impact on the performance. The programmable interface provided by domain ontology API defined using Java language helped in performance improvement. The API, in turn, uses predefined ontology functionality available in Java libraries. Usage of this methodology is the main reason behind the enhanced functionality of the proposed framework and the application. The process became more dynamic and consistent with domain ontology usage.
The formal approach to instantiate domain ontology and represent features of SPLs made it possible to optimize the two important tasks. With the improved prototype, experiments are made and evaluated the extended framework. The empirical results showed that there was a significant impact on the product derivation and automatic reconfiguration of SPLs when domain ontology was used. When domain ontology is not used the performance is less as there was no formal approach used for representing features. This is the main reason for performance improvement. It is evident in the results of evaluation made by human experts. The results presented in this chapter show the utility of ontology in the proposed framework. The generic framework proposed in this thesis is improved gradually and the final functionality of it is realized. Though in this chapter ontology is focused on enhancing the framework, the previous two chapters threw light into product derivation, usage of variability-aware design patterns for optimization of the framework. The work presented in this chapter reflects the complete optimization of the framework and the prototype application. This thesis brought forth a systematic and generic approach to configuration management of SPL and quality product derivation. The approach has been realized incrementally. Moreover, the proposed framework is extendable so as to accommodate new extensions in future.

The research done has realized all parts of the proposed framework, which led to well-informed conclusions. Besides it also helped to have directions for future work. The ensuing chapter focuses on conclusions and future work.

******