Semantic-Aware Feature Modeling and Analysis

by
Taslima Akter
Tasmia Akther
Neeta Sinha

A thesis submitted in partial fulfillment for the
degree of Bachelor of Science in Computer Science and Engineering

in the
Faculty of Science and Engineering
Department of computer Science and Engineering

December 2016

Declaration

We hereby declare that this submission is our own work and that to the best of our knowledge and belief it contains neither material nor facts previously published or written by another person. Further, it does not contain material or facts which to a substantial extent have been accepted for the award of any degree of a university or any other institution of tertiary education except where an acknowledgement.

(**Tasmia Akther**)

(Taslima Akter)

(Neeta Sinha)

Letter of Acceptance

The project entitled "**Semantic-Aware Feature Modeling and Analysis**" submitted by Taslima Akter (ID: 2012-2-60-003) Tasmia Akther (ID: 2012-2-60-009) and Neeta Sinha (ID: 2012-2-60-020), to the department of Computer Science and Engineering, East West University, Dhaka, Bangladesh is accepted by the department in partial fulfillment of requirements for the Award of the Degree of Bachelor of Science in Computer Science and Engineering on December 2016

Abstract

Feature diagrams are the most widely used to model product line variant. Formal Verification of variant requirements has gained much interest in the software product line (SPL) community. Successful development of a software product line (SPL) requires a proper management of product line requirements. Various approaches have been adopted to model both of the requirements of feature diagram. However, most of these approached lack proper formal semantics. This report presents our work in progress semantic web approach to model and verify product line requirements. Logical expressions can be built by modeling variants and their dependencies by using propositional connectives. A case study of two Feature Model (Hall Booking System) variant feature model is presented to illustrate the analysis and verification process.

Contents

List of Figure

Acknowledgements

First of all Thanks to ALLAH for the uncountable blessings on us. Thanks to our Supervisor, **Dr. Shamim Hasnat Ripon** for providing us this opportunity to test our skills in the best possible manner. He enlightened, encouraged and provided us with ingenuity to transform our vision into reality.

Lastly, but deliberately we want to pay tribute to our parents. We call them “Our Heroes, Our mentors”. These two the people that god has used to discover, nurture and deepen our academic career.

To our parents

Chapter 1

Introduction

1.1 Introduction

Software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [1]. Software product lines are emerging as a viable and important development paradigm allowing companies to realize order-of-magnitude improvements in time to market, cost productivity, quality. The main idea of software product line is to explicitly identify all the requirements that are common to all members of the family as well as those that vary among products in the family. This implies a huge model that help the stakeholders to be able to trace any design choices and variability decision. A particular product is then derived by selecting the required variants and configuring them according to the product requirements. Common requirements among all family members are easy to handle and can be integrated into the family architecture and are part of every family member. But problem arises from the variant requirements among family members. Variants are usually modeled using feature diagram, inheritance, templates and other techniques.

Domain and application engineering are the two main phase of SPL development [2]. A detailed domain analysis is performed in domain engineering by identifying the commonalities and variability's of various aspects of the domain. Domain knowledge is captured in a reusable manner. Feature modeling [6] plays an important role for modeling different aspects of family of systems. It models the commonality and variability in a tree structure and describes the interdependencies of product family features. In comparison to analysis of a single system, modeling variants adds an extra level of complexity to the domain analysis. Different variants might have dependencies on each other. Tracing multiple occurrences of any variant and understanding their mutual dependencies are major challenges during domain modeling. While each step in modeling variants may be simple but problem arises when the volume of information grows. As a result, the impact of variant becomes ineffective on domain model.

1.2 Problem and Motivation

There are various reuse mechanism proposed for feature model, such as FODA (Feature Oriented Domain Analysis) [11], FORM9Feature Oriented Reuse Method)[12] . However, dew to the lack of formal semantic for feature models, no automated tools are available to check the consistency and correctness of feature configuration of a particular product. Various approaches have been suggested to model feature diagram. To capture Domain knowledge and common vocabularies in any field ontology's have shown itself an acceptable paradigm [18]. It is also necessary to process and exploit knowledge in a computer system.

Both industry and academia have shown much interest in handling product line in Application domains such as business systems, avionics, command and control systems etc. Today most of the effort in product line development are relating to architecture [13] detail design and code. Common requirements among all family members are easy to handle as they simply can be integrated into the family architecture and are part of every family member. But problem arises from the variant. In a product line, currently variants are modeled using feature diagram, inheritance, templates and other techniques. In comparison to analysis of a single system, modeling variants adds an extra level of complexity to the domain analysis. In any product line model, the same variant has occurrences in different domain model views. Different variants have dependencies on each other. Tracing multiple occurrences in different model views of any variant and understanding the mutual dependencies among variants are major challenges during domain modeling. While each step in modeling variant may be simple but problem arises when the volume of information grows. When the volume of information grows the domain models become difficult to understand. The main problems are the possible explosion of variant combinations, complex dependencies among variants and difficulty in tracing variants from the domain model down to the specification of a particular product. As a result, the impact of variant becomes ineffective on domain model. Therefore, product customization from the product line model becomes unclear and it undermines the very purpose of domain model.

Semantic web technology can provide a meaningful and shared ontological description of the domain. Web Ontology Language (OWL) [19] is one of the most expressive language for specifying publishing and sharing ontology’s. It is therefore evident that semantic web technology, OWL in particular can be used to represent a particular domain and define the relationship of various features within that domain.

1.3 Objectives

This report formally models and verifies the variants of SPL using semantic web mechanism. In developing product line, the variants are to be managed in domain engineering phase, which scopes the product line and develops the means to rapidly produce the members of the family. It serves two distinct but related purposes, firstly it can record decisions about the product as a whole including identifying the variants for each member and secondly ,it can support application engineering by providing proper information and mechanism for the required variants during product generation

- The objective of this work is to provide an approach for modeling variants in the domain model of a product line .This model carries all the variant related information like specifications ,origin of variants and interdependencies etc.
- Semantic web mechanism can integrate meaningful description and semantic information into SPL feature models.
- Our plan is perform these verification by using our Semantic representation.

1.4 Contribution

In order to conduct out experiment we use a hall booking system by analyzing and modeling the variants as well as the variants dependencies.

- We define six types of logical notation to represent all the parts in a feature model. Set representation logic has been for this purpose. This notations can be used to define all possible scenarios of a feature model.
- Analyzing the feature model considering the various scenarios the Feature model and we define a set of rules which can be used to verify the feature model.
- We use protégé software for checking the valid or invalid feature model.

1.5 Outline

The report is organized as follows-

In chapter 2 we gave a brief overview of the feature model, feature model notations and logical representation of the feature model and describe some logical operators, domain activities.

In Chapter 3 We have gave an overview hall booking system and hall booking feature tree.

In chapter 4 we discuss about the logical representation and describe their logical relations and analyze the semantic representation.

Chapter 5 Using protégé tool for graphical editing and RACER for consistency checking.

Chapter 6 Concludes the thesis by summarizing our work. Finally we outline our future plan.

Chapter 2

Background

2.1 Software Product Line

Software product lines or software product line development refers to software engineering methods, tools and techniques for managing variability and commonality of core software assets in order to facilitate the development of families of software-intensive products. A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.[1]

A software product line harnesses the principles of industrialization and automation in order to make the development of software more efficient in addition to the resulting artifacts being of higher quality. Also we know that software product line is a software intensive system sharing a common and managed set of feature that satisfy the needs of a particular market segment or mission and that are developed from a set of core assets in a prescribed way. Product line technology is a way of improving the software development lifecycle and refuse by providing facilities to reuse the model of the system family. It is possible to increase the productivity and decrease the possible errors significantly by reusing the products of the system families rather than recreating. The main idea of software product line is to explicitly identify all the activities which are common to all members of the family as well as which are different and arrange them in a model. This implies a huge model which will help the stakeholder to be able to trace any design choices and variability decisions as well. Finally, the derivation of the product is done by selecting the required variants from the model and configuring them according to product requirements.

Manufacturers have long employed analogous engineering techniques to create a product line of similar products using a common factory that assembles and configures parts designed to be reused across the product line. For example, automotive manufacturers can create unique variations of one car model using a single pool of carefully designed parts and a factory specifically designed to configure and assemble those parts. The characteristic that distinguishes software product lines from previous efforts is predictive versus opportunistic software reuse. Rather than put general software components into a library in the hope that opportunities for reuse will arise, software product lines only call for software artifacts to be created when reuse is predicted in one or more products in a well defined product line. Recent advances in the software product line field have demonstrated that narrow and strategic application of these concepts can yield order of magnitude improvements in software engineering capability. The result is often a discontinuous jump in competitive business advantage, similar to that seen when manufacturers adopt mass production and mass customization paradigms

While early software product line methods at the genesis of the field provided the best software engineering improvement metrics seen in four decades, the latest generation of software product line methods and tools are exhibiting even greater improvements. New generation methods are extending benefits beyond product creation into maintenance and evolution, lowering the overall complexity of product line development, increasing the scalability of product line portfolios, and enabling organizations to make the transition to software product line practice with orders of magnitude less time, cost and effort.

2.2 Feature Model

In software development, a feature model is a compact representation of all the products of the Software Product Line in terms of "features".[3] Feature models are visually represented by means of feature diagrams. A feature models are simple, hierarchical models that capture the commonality and variability of a product line. In 1990, Feature models were first introduced in the Feature-Oriented Domain Analysis (FODA) method by Kang .Since then, feature modeling has been widely adopted by the software product line community and a number of extensions have been proposed. Feature models play a central role in the development of a system family architecture, which has to realize the variation points specified in the feature models [4] [5].

In software product line(SPL) implementations are typically feature based as features are logical point of variation for any given group of software products. Therefore , the feature model is an extremely useful method for modeling the commonality and variability within an SPL. Also the key technical innovation of software product lines is the use of features to distinguish product line members. A feature is an increment in program functionality. A particular product line member is defined by a unique combination of features. The set of all legal feature combination defines the set of product-line members. Feature models define features and their usage constraints in product-lines. Current methodologies organize feature into a tree, called a feature diagram (FD), which is used to declaratively specify product-line members .Relationships among FDs and grammars, and FDs and formal models/logic programming have been noted in the past, but the potential of their integration is not yet fully realized.

2.2.1 Feature Modeling Notations

Relationships between a parent feature and its child features (or sub features) are categorized as

- Mandatory: child feature is required.
- Optional: child feature is optional
- Or: at least one of the sub-features must be selected.
- Alternative **(**xor**):** one of the sub _features must be selected.

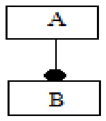
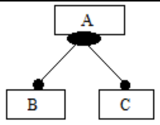
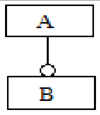
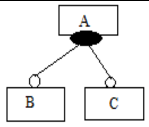
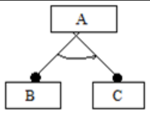
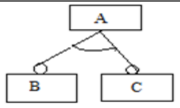
In addition to the parental relationships between features ,cross-tree constraints are allowed. The most common are:

- A requires B – The selection of A in a product implies the selection of B.

A excludes B- A and B cannot be part of the same product.

These relations are shown in Table 1.

Table 2.1 Types of features

Type	Notation	Type	Notation
Mandatory		Or	
Optional		Optional or	
Alternative		Optional Alternative	

2.3 Domain and Application Engineering

Domain is an area of knowledge that uses common concepts for describing phenomena, requirements, problems, capabilities and solution that are interest of some stakeholders. A domain is usually associated with well-defined or partially defined terminology. This terminology refers to the basic concepts in that domain their definition (i.e. their semantic meanings) and their relationships. It sometime also refers to behaviors that are desired, forbidden, or perceived within the domain. Domain engineering is a set of activities that aim at developing, maintaining and managing the creation and evolution of domains. Domain engineering has become of interest to the information systems and software engineering communities for several reasons. These reasons include, in particular, the need to manage increasing requirements for variability of information and software systems(reflecting variability in customer requirements); the need to minimize accidental complexity when modeling the variability of a domain; and the need to obtain, formalize and share expertise in different evolving domains.

Domain engineering provide methods and techniques that may help reduce time-to-market, product cost and project risk on one hand ,and help improve product quality and performance on a consistent basis on the other hand. To improve the quality of developed software products through reuse of software artifacts domain engineering is designed very well. Domain engineering shows that most developed software system which is not a new system but rather variants of other systems within the same field. As a result ,the use of domain engineering business can maximize profits and reduce time -to – market by using the concepts and implementations from prior software systems and applying them to the target system.

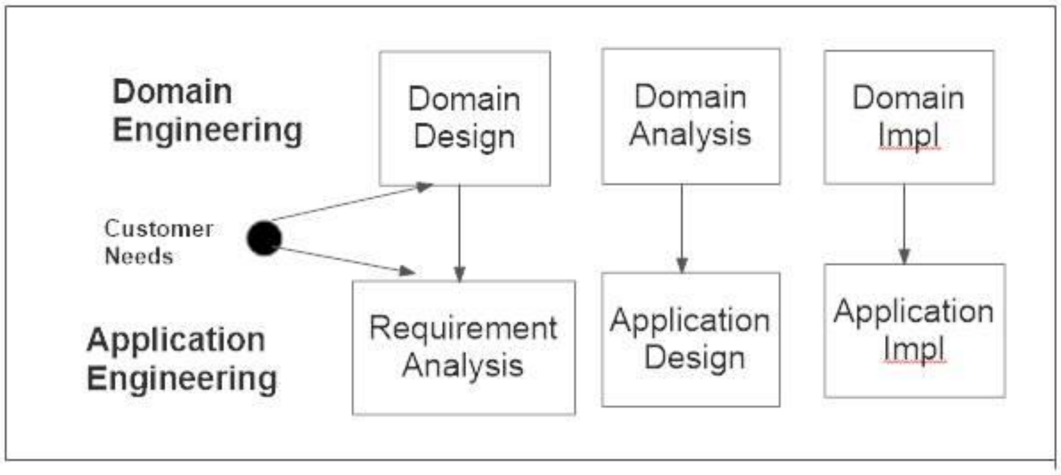


Figure 2.1 domain engineering and application engineering.

The reduction in cost is evident even during the implementation phase. One study showed that the use of domain specific language allowed code size, in both number of methods and number of symbols, to be reduce by over 50% and the total number of lines of code to be reduced by nearly 75%.

Domain engineering focuses on capturing knowledge gathered during the software system. In a domain engineering components can be reused in a new software system at low cost and high quality. Because this applies to all phases of the software development cycle, domain engineering also focuses on the three phases such as analysis, design and implementation, paralleling application engineering. This products not only a set of software implementation components relevant to the domain, but also reusable and configurable requirements and designs .

Also we know that software engineering focuses on single system but domain engineering focuses on a family of system [11]. A good domain model servers as a reference to resolve ambiguities later in the process [12], a repository of knowledge about the domain characteristics and definition, and a specification to developers of products which are the part of the domain .

2.4 Semantic Web

The semantic web is an extension of the Web through standards by the World Wide Web Consortium (W3C). The standards promote common data formats and exchange protocols on the Web, most fundamentally the Resource Description Framework (RDF) [7]. By encouraging the inclusion of semantic content in web pages, the semantic web aims at converting the current web dominated bt unstructured and semi-structured documents into a “web of data”.

The vast majority of the Web is designed to be read and understand by humans. It is, for the most part , not possible for a machine or software agent to freely navigate through the Web and accurately accomplish a task of any significance. Most content on the Web must be viewed by humans and ,with the proper context, understood in order to be of any use. A true Semantic Web would add machine readable structure and encode content in such a manner so that machines, especially intelligent software agents, could navigate and accomplish tasks by reasoning through the meaningful content of the Web pages.

The W3C gives the following definition for the Semantic Web :The Semantic Web is an extension of the current Web in which information is given a well-defined meaning, better enabling computers and people to work in cooperation’s is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. With the SW, the machine can do many complicated tasks which currently can only be performed manually For example ,user can directly send the following request to Web agent Book me a holiday next weekend somewhere warm, not too far away, and where they speak Chinese or English. The Web agent will be able to understand the request and perform it for users. A series of technology has been proposed to realize the vision of the Semantic Web as the next generation Web. It extends the current Web. It extends the current Web by giving the Web content a well defined meaning and representation the information in a machine –understandable form HTML, the current web data standard, is aimed at delivering information to the end user for human-consumption.XML is aimed at delivering data to systems than can understand and interpret information .XML is focused on the syntax(defined but the XML schema or DTD) of a documents and it provides essentially a mechanism to declare and use simple data structure. However there is no way for a program to actually understand the knowledge contained in the XML documents. Resource Description Frame (RDF) is a foundation for processing metadata; it provides interoperability between applications that exchange machine –understandable information on the Web. RDF uses XML to exchange descriptions of Web resources and emphasizes facilities to enable automated processing. The RDF descriptions provide a simply ontology system to support the exchange of knowledge and semantic information on the Web. RDF schema provides the basic vocabulary to describe RDF documents. RDF schema can be used to define properties and types of the Web resources in a similar fashion to XML schema which gives specific constraints on the structure of an XML documents, RDF schema provides information about the interpretation of the RDF statements. The DARPA agent Markup Language (DAML) is an AI-inspired description logic-based language for describing taxonomic information. DAML currently combines Ontology Inference Layer(OIL) and features from other ontology systems .It is now called DAML+OIL and contains richer modeling primitives than RDF schema . the DAML+OIL language builds on top of XML and RDF(S) to provide a language with both a well-defined semantics and a set of language constructs including classes, subclasses and properties with domain and ranges , for describing a Web domain DAML+OIL can further express restrictions on membership in classes and restrictions on certain domain and ranges values. The Semantic Web is highly distributed, and different parties may have different understandings of the same concept .Ideally, the program must have a way t discover the common meaning from the different understandings. It is central to one important concept in Semantic Web system Ontology. The Ontology for a Semantic Web system is a document or a file that formally defines the relations among terms. The most typical kind of ontology for the Web gas a taxonomy and a set of inference rules. Ontology can enhance the functioning of the Web in many ways.

2.5 Ontology

Ontology is a specification of a conceptualization. Ontology is the philosophical study of the nature of being, becoming, existence or reality as well as the basic categories of being and their relations. Traditionally listed as a part of the major branch of philosophy known as metaphysics, ontology often deals with questions concerning what entities exist or may be said to exist and how such entities may be grouped, related within a hierarchy, and subdivided according. The core meaning within computer science is a model for describing the world that consists of a set of types, properties, and relationship types. There is also generally an expectation that the features of the model in an ontology should closely resemble the real world. In computer science and information science, an ontology is a formal naming and definition of the types, properties, and interrelationships of the entities that really or fundamentally exist for a particular domain of discourse. It is thus a practical application of philosophical ontology, with a taxonomy. An ontology compartmentalizes the variables needed for some set of computations and establishes the relationships between them. to similarities and differences. Actually ontology's are used to capture knowledge about some domain of interest. An ontology describes the concepts in the domain and also the relationships that hold between those concepts. Different ontology languages provide different facilities. The most recent development in standard ontology languages as Owl from the World Wide Web Consortium (W3C). Like protégé, OWL makes it possible to describe concepts but it also provides new facilities. It has a richer set of operators- e.g. intersection, union and negation. It is based on different logical model which makes it possible for concepts to be defined as well as described. Complex concept can therefore be build up in definitions out of simpler concept. Furthermore, the logical model allows the use of the reasoner which can check whether or not all the statement and definition in the ontology are mutually consistent can therefore help to maintain the hierarchy correctly. This is particularly useful when dealing with cases where classes can have more than one parent.

Ontology analysis clarifies the structure of knowledge. Given a domain, its ontology forms the heart of any system of knowledge representation for that domain .Without ontology's, or the conceptualizations that underline knowledge; there can-not be a vocabulary for representation knowledge. Thus, the first step in devising an effective knowledge representation system, and a vocabulary, is to perform an effective ontology analysis of the field , or domain. Weak analyses lead to incoherent knowledge bases. An example of why performing good analysis is necessary comes from the field of database. Consider a domain having several classes of people (for example, students, professors, employee, females ,and males).This study first examined the way this database would be commonly organized: students , employee, professor, males and female would be represented as type of the class humans. However, some of the problems that exist with this ontology are that students can also be employee at times and can stop being students. Further analysis showed that the returns students and employee do not describe categories of humans, but are roles that humans can play, while terms such as females and males more appropriately represent subcategories of humans. Therefore ,clarifying the terminology enables the ontology to work for coherent and cohesive reasoning purposes. Second, ontology's enable knowledge sharing. Suppose we perform an analysis and arrive at a satisfactory set of conceptualizations, and their representative term, for some area of knowledge -for example, the electronic -devices domain. The resulting ontology would likely include domain-specific terms such as transistors and diodes; general terms such as functions, casual processes, and modes; and term that describe behavior such as voltage. The ontology captures the intrinsic conceptual structure of the domain. In order to build a knowledge representation language based on the analysis ,we need to associate terms with the concepts and relations in the ontology and devise a syntax for encoding knowledge in terms of the concepts and relations. We can share this knowledge replication the knowledge analysis process. Shared ontology's can thus the basis for domain-specific knowledge representation languages .In contrast to the previous generation of knowledge-representation languages (such as KL-one), these languages are content –rich; they have a large number of terms that embody a complex content theory of the domain. Shared ontology's let us build specific device manufactures can use a common vocabulary and syntax to build catalogs that describe their products. Then the manufacturers could share the catalogs and use them in automated design systems. This kind of sharing vastly increases the potential for knowledge reuse.

2.5.1 Ontology’s as a Specification Mechanism

A body of formally represented knowledge is based on a conceptualization: the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system or knowledge-level agent is committed to some conceptualization, explicitly or implicitly. A ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where ontology is a systematic account of existence. For AI systems, what “exists” is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects and the describable relationships among them are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe ontology of a program by defining a set of representational terms. In such ontology, definitions associate the names of entities in the universe of discourse (e.g. classes, relations, functions or other objects) with human-readable text describing what the names mean and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, ontology is the statement of a logical theory. We use common ontology’s to describe ontological commitments for a set of agents so that they can communicate about a domain of discourse without necessarily operating on a globally shared theory. We say that an agent commits to ontology if its observable actions are consistent with the definitions in the ontology. The idea of ontological commitments is based on the knowledge-level perspective. The Knowledge Level is a level of description of the knowledge of an agent that is independent of the symbol-level representation used internally by the agent. Knowledge is attributed to agents by observing their actions; an agent “knows” something if it acts as if it had the information and is acting rationally to achieve its goals. The “actions” of agents including knowledge base servers and knowledge-based systems can be seen through a tell-and-ask functional interface, where a client interacts with an agent by making logical assertions (tell) and posing queries (ask). Pragmatically, a common ontology defines the vocabulary with which queries and assertions are exchanged among agents. Ontological commitments are agreements to use the shared vocabulary in a coherent and consistent manner. The agents sharing a vocabulary need not share a knowledge base; each knows things the other does not and an agent that commits to ontology is not required answering all queries that can be formulated in the shared vocabulary. In short, a commitment to a common ontology is a guarantee of consistency, but not completeness, with respect to queries and assertions using the vocabulary defined in the ontology.

Any ontology must give an account of which words refer to entities, which do not, why and what categories result. When one applies this process to nouns such as electrons, energy, contract, happiness, time, truth, causality and God, ontology becomes fundamental to many. In both computer science and information science, ontology is a data model that represents a set of concepts within a domain and the relationships between those concepts. It is used to reason about the object within that domain. Ontology's are used in artificial intelligence, the semantic web, software engineering, biomedical informatics and information architecture as a form of knowledge representation about the world or some part of it. Ontology generally describe:

Individuals: the basic or "ground level" objects.

Classes: sets, collections or types of objects.

Attributes: properties, features, characteristics or parameters that objects can have and share.

Relations: ways that objects can be related to one another.

Events: the changing of attributes or relations.

2.6 OWL

The Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontology's. The OWL languages are characterized by formal semantics They are built upon a W3C XML standard for objects called the Resource Description Framework (RDF).OWL and RDF have attracted significant academic, medical and commercial interest. In October 2007 a new W3C working group was started to extend OWL with several new features as proposed in the OWL 1.1 member submission.W3C announced the new version of OWL on 27 October 2009. This new version, called OWL 2, soon found its way into semantic editors such as Protégé and semantic reasoners such as Pellet, RacerPro, FACT++ and Hermit. The OWL family contains many species, serializations, syntaxes and specifications with similar names. OWL and OWL2 are used to refer to the 2004 and 2009 specifications, respectively. Full species names will be used, including specification version (for example, OWL2 EL). When referring more generally, owl family will be used. The OWL Web Ontology Language is described for use by applications that need to process the content of information instead of just presenting information to humans.OWL facilitates greater machine interpretability of Web content than that supported by XML,RDF ,and RDF schema (RDF-S) by providing additional vocabulary along with a formal semantics.OWL has three increasingly – expressive sublanguages: OWL lite, OWL DL, OWL full.

In this part of this report, describes the OWL Web Ontology Language. OWL is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called ontology. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web. OWL is a revision of the DAML+OIL web ontology language incorporating lessons learned from the design and application of DAML+OIL.

2.6.1 Why OWL

The semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. The semantic Web will build on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data. The first level above RDF required for the semantic web is an ontology language what can formally describe the meaning of terminology used in Web documents. If machines are expected to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDF schema. The OWL use cases and requirements Documents provides more details on ontology's, motivates the need for a Web Ontology Language in terms of six use cases, and formulates design goals, requirements and objectives for OWL.OWL has been designed to meet this need for a Web Ontology Language .OWL is part of the growing stack of W3C recommendations related to the Semantic Web.

- XML provides a surface syntax for structured documents, but impose no semantic constraints on the meaning of these documents.XML schema is a language for restricting the structure of XML documents and also extends with data types.
- RDF is a data model for objects ("resources") and relations between them, provides a simple semantics for this data model , and these data models can be represented in an XML syntax.RDF schema is a vocabulary for describing properties and classes of RDF resources , with a semantics for semantics for generalization – hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others relations between classes (e .g disjoints), cardinality (e.g. "exactly one"), equality riche
- typing of properties ,characteristics of properties (e.g symmetry), and enumerated classes.

Chapter 3

Hall Booking System Overview

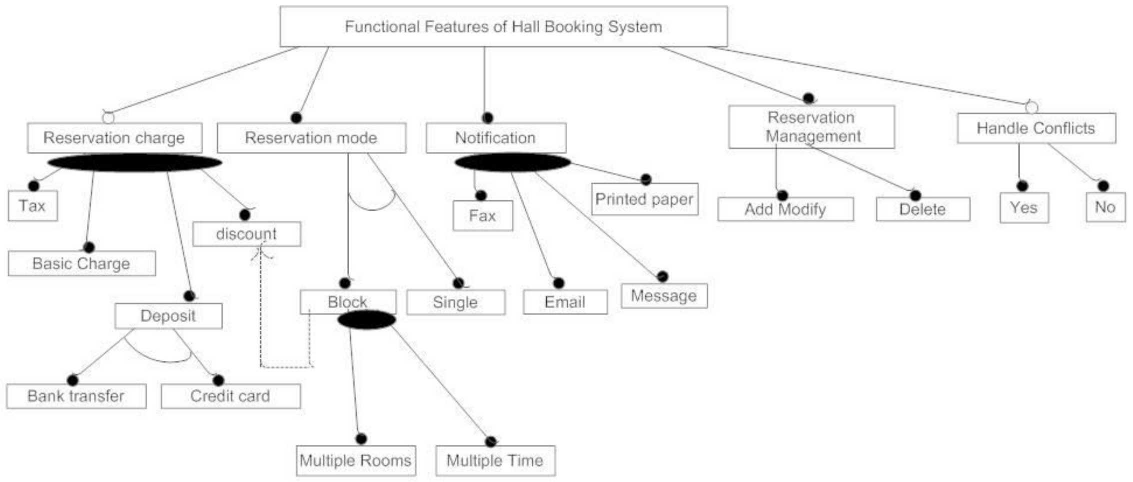
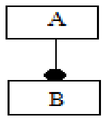
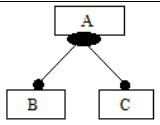
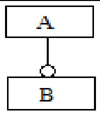
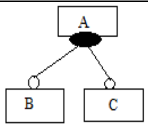
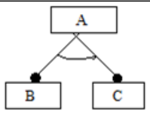
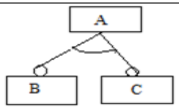


Figure 3.1: Hall Booking System Feature Tree

Hall Booking software is online or manual booking software for room and conference facility reservations. This software makes hall booking system more efficient for clients, staff, and conference facilities. People hiring the hall get all the information they need at the right time. This software simplifies the process; maximize capacity of booking the hall. The main purpose of this software to Improvements in efficiency of hall booking, Prevention of double bookings, Quick and Simple Booking Process, Block bookings for daily, weekly, monthly or ad hoc multiple day bookings, Automatic email or sms confirmations to customers. The system can be used for either academic or non academic purposes. The system can be used in academic institutions to reserve tutorial rooms and lecture halls, at companies to reserve meeting rooms, and at hotels to reserve rooms and conference facilities etc. Users can be able to manage their own reservation with the system easily. The main purpose and the core functionality are similar across the Hall Booking System however; there are many variants on the basic theme. One of the basic variants is the charging of the booking system. Our system is design as like when the system is used for academic purposes no charge is needed for booking halls but there may be a need to charge for booking halls in other areas like booking hall for non academic purpose. In some systems, there are facilities available for seasonal booking as well as multiple bookings.

Table 3.1 :Types of features

Type	Notation	Type	Notation
Mandatory		Or	
Optional		Optional or	
Alternative		Optional Alternative	

We use Hall Booking System to illustrate our variability modeling mechanism. A part of the features of Hall Booking System in shown in Fig 3.1.extensions of feature diagram described in [4] have been used here. The Root of this system is Functional Feature of Hall Booking System. It has Five Direct Feature. There are three mandatory features and two optional features in our feature tree Three Mandatory Features are Reservation Mode, Reservation Management and Notification. Two Optional features are Reservation Charge and Handle Conflict. Mandatory features appear in all the members of the system on the other hand variant features appear in some member of the system .Variant feature are classified as optional. Alternative, and Or feature An example of .In Reservation Charge feature there has four child feature and they are Deposit, Tax, Basic Charge and Discount and all of them are in Or relationship with their parent feature Reservation Charge. Under Deposit Parent feature Bank transfer and Credit cards are in Or relationship. In Reservation Mode there has two child feature and they are Block and Single. The Block and Single feature are in Alternative Relationship with their Parent Feature Reservation Mode and Under Block Parent feature Multiple Rooms and Multiple Time are in Or relationship. On the Other hand Block under Reservation Mode and Discount feature under Reservation Charge are in Require Relationship. Notification contain four Child feature in Or relation and they are Fax, Printed Paper, Email And Massage. Reservation Management contains Add Modify and Delete and they are two mandatory Features under the Reservation Management Parent Feature. Optional feature Handle Conflicts contains two children yes or no and they are two mandatory Feature under the Handle Conflicts Parent Feature.

Chapter 4

Semantic Representation

We use protégé software to represent our System. Using OWL language constructs we modeled various feature relations. By using OWL-Dl language we model six different types of relations namely mandatory, optional, alternative, or, optional or, alternative or. There are one dependency in our system which is called requires are also modeled.

Representation of Various Types of Features

4.1 Mandatory

A mandatory feature is included if its parent feature is included. Mandatory feature is represented by a small circle on the child node. A filled bullet denotes a mandatory (In Table : 3.1) feature and features that are required. There are three mandatory features in our system which is Reservation Mood, Reservation Management, and Notification.

Mandatory features are defined as follow:

```
Hall Booking that
hasBookingSystem some ReservationMood
hasBookingSystem some ReservationManagement
hasBookingSystem some Notification
```

4.2 Optional

An optional Feature may or may not be include if its parent is included. Optional Feature is represented by a small circle on the child node. A empty bullet denotes(In Table : 3.1) a optional feature and features that are optional .There are two optional feature in our hall booking system they are Reservation Charge and Handle Conflicts and they may or may not be included in a configuration of Hall Booking system.

Optional features are defined as follow:

```
Hall Booking that
hasBookingSystem some ReservationCharge
hasBookingSystem some HandleConflict
```

4.3 Alternative

One and only one feature from a set of alternative features are included when parent feature is included that means exactly one sub-feature must be selected. Feature is represented by a unfilled (In Table : 3.1) triangle denotes the alternative.

Block and Single are alternative features of Reservation Mode. We model this relation as follow:

```
Hall Booking that
hasBlock some MultipleRoom
hasBlock some MultipleTime
ReservationMood

hasReservation only(Block or Ssingle)
hasReservation some single
has Reservation some block
```

Bank transfer and credit Transfer are alternative feature of deposite under reservation charge.we model this relation as follow

```
Hall Booking that
hasReservation only(BankTransfer or CreditTransfer)
ReservationCharge
```

4.4 Or

At least one from a set of or feature is included when parent is included and one or more features can be selected when the parent feature appears. Feature is represented by a filled triangle (In Fig : 3.1) denotes the or Feature.

There are four child which are in or relation with each others under Reservation Charge and they are Basic charge, Deposit, Discount and Tax. We defined this relation as follow:

```
Hall Booking that
hasReservationCharge some BasicCharge
hasReservationCharge some Deposit
hasReservationCharge some discount
hasReservationCharge some Tax
```

There are four child of Notification which are Fax, Email, Massege and Printed Paper also in Or relation.we defined this relation as follow:

```
Hall Booking that
hasNotification some Email
hasNotification some fax
hasNotification some PrintedPaper
hasNotification some massege
```

Multiple Room and Multiple time under Block parent are also in Or relation.we defined this as follow:

```
Hall Booking that
hasBlock some MultipleRoom
hasBlock some MultipleTime
ReservationMood
```

4.5 Optional Alternative

One feature from a set of alternative features may or may not be included if parent included. Feature is represented by a unfilled triangle and empty bullets (In table: 3.1) denotes the optional alternative.

Optional alternative can be defined as:

```
hasBooking some ReservationMood
and(not Block (or Single) or not single(or block))
```

4.6 Optional Or

One or more optional feature may be included if the parent is included. Optional Or Feature is represented by a filled triangle (In table : 3.1) and filled bullets denotes the optional or.

Optional Or can be defined as follow:

```
hasBooking some Notification
(not(fax or Email or PrintedPaper or Massage))
```

4.7 Exclude

Some feature cannot be together in a feature tree.
exclude relation can be defined as:

```
ReservationMood and
(Block or (not Single)
And (not Block)or Single))
```

4.8 Requires

The feature may depend on some other feature; hence its present in a feature configuration requires the appearance of the others. In our hall booking system there are dependency on block under parent Reservation Mood and Discount under parent Reservation Charge, Block and Discount are in requires relation.

we defined this relation as follow:

```
Hall Booking and(hasReservationMood some block)
And(hasReservationCharge some Discount)
```

Chapter 5

Consistency analysis of feature model

We input our ontology into protégé and use RACER [8] to check its consistency. For the initially encoded ontology, RACER checks for consistency and show that encoded definition are consistent (Fig 5.1).

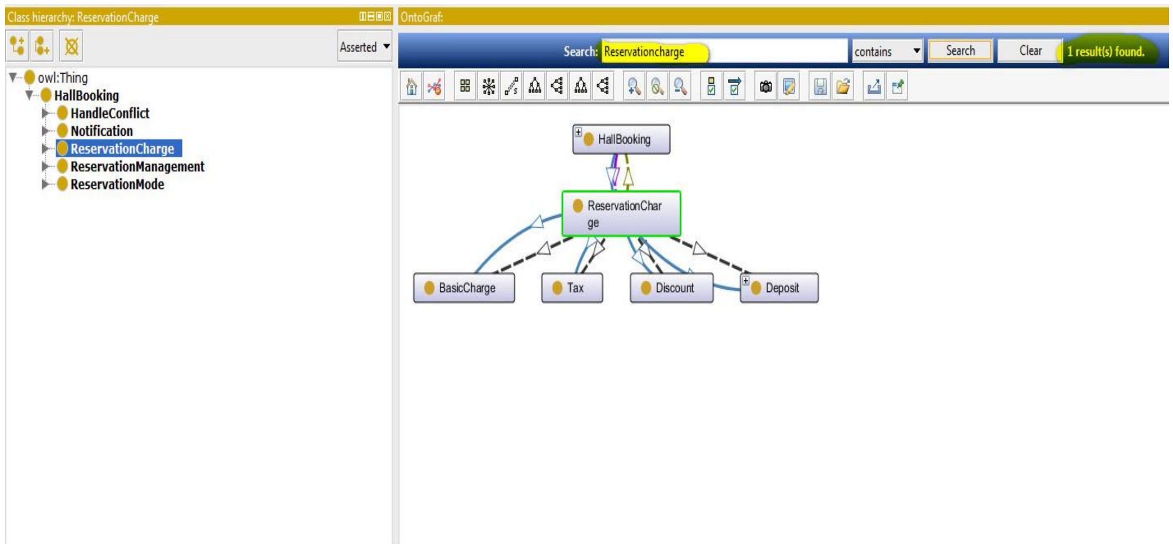


Figure 5.1: Consistency checking in RACER

In feature modeling an instance of a concept is a configuration derived from the feature model. In order to detect inconsistency in a configuration OWL classes are used, and features and concepts instances are then simulated. When an instance is checked, the reasoner not only check inconsistency but also shows which class/classes are inconsistent.

In feature modeling, a feature configuration derived from a feature model represents a concrete instance of a concept (i.e., a specific system in a domain). Intuitively given a feature ontology, features and concepts in a configuration should be ground instances (OWL individuals) of the OWL classes defined in the ontology. Hence modeling feature configurations using individuals is a straight forward approach.

Here we search for ReservationCharge and RACER give us tree saying that 1 result found. The feature configuration can be modeled as follow,

```
Hall Booking that
hasReservationCharge some BasicCharge
hasReservationCharge some Deposit
hasReservationCharge some discount
hasReservationCharge some Tax
```

For inconsistency we search for notification type from our hall booking system feature tree. The RACER now shows that the configuration in inconsistent. The inconsistent classes are marked as red. There is no class name notification type so it's give 0 result found

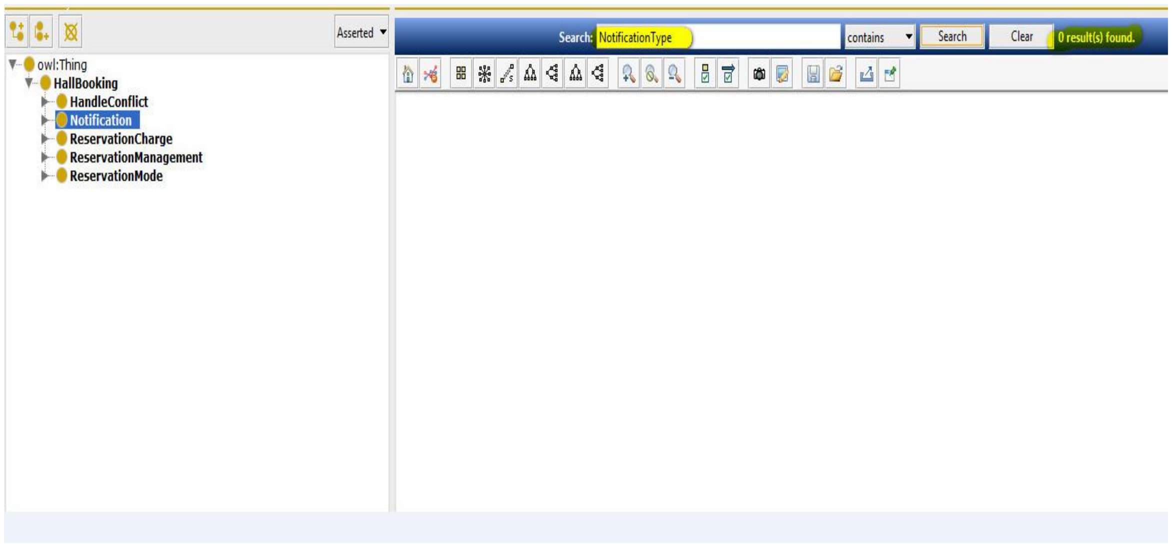


Figure 5.2: Inconsistency checking in RACER

Chapter 6

Conclusion

We presented an approach to formalizing and verifying SPL feature models to be able to create a decision table to generate customized product by using formal reasoning techniques. We provided formal semantics of the feature models by using, set representation first-order logic and specified the definitions of six types of variant relationships. We have six notations mandatory, optional, or, alternative, optional alternative, optional or we also defined cross-tree variant dependencies. Examples are provided describing various analysis operations, such as validity. We have addresses most of the analysis questions mentioned in. Finally, we encoded our logical notations into Semantic Representation to be able to automatically verify any analysis related queries. A knowledge-based approach to specify and verify feature models is presented in Comparing to that presentation.

OWL ontology's provide a suitable platform for the development of semantically aware software product line allowing the knowledge within the feature model to be shared among the reusable features of the SPL. We represented our preliminary result of consistency checking using RACER. A through consistency checking is currently undergoing.

Appendix

A.1 RDF/XML SOURCE CODE

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<rdf:RDF
xmlns="http://www.semanticweb.org/emon/ontologies/2016/11/untitled-
ontology-19#"

xml:base="http://www.semanticweb.org/emon/ontologies/2016/11/untitled-
ontology-19"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology
rdf:about="http://www.semanticweb.org/emon/ontologies/2016/11/untitled-
ontology-19"/>



    <!--

////////////////////////////////////////////////////////////////////////
///////////////
    //
    // Object Properties
    //

////////////////////////////////////////////////////////////////////////
///////////////
     -->
    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasBlock
-->

    <rdf:Description
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasBloc
k"/>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasDeposite -->

    <rdf:Description
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasDepo
site"/>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasHandleConflicts
-->

    <rdf:Description
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasHand
leConflicts"/>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasNotification --
>

    <rdf:Description
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasNoti
fication"/>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasReservation -->

    <rdf:Description
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasRese
rvation"/>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasReservationChar
ge -->

    <rdf:Description
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasRese
rvationCharge"/>
```

```
    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasReservationMana
gement -->

    <rdf:Description
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#hasRese
rvationManagement"/>



    <!-- http://www.w3.org/2002/07/owl#topObjectProperty -->

    <rdf:Description rdf:about="&owl;topObjectProperty">
        <rdf:type rdf:resource="&owl;ReflexiveProperty"/>
        <rdf:type rdf:resource="&owl;SymmetricProperty"/>
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    </rdf:Description>



    <!--

///////////////////////////////////////////////////////////////////////
///////////////
    //
    // Classes
    //

///////////////////////////////////////////////////////////////////////
///////////////
     -->




    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#AddModify
-->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#AddModi
fy">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Rese
rvationManagement"/>
    </owl:Class>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#BankTransfer -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#BankTra
nsfer">
```

```
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Depo
sit"/>
    </owl:Class>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#BasicCharge -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#BasicCh
arge">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Rese
rvationCharge"/>
    </owl:Class>



    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#Block -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Block">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Rese
rvationMode"/>
    </owl:Class>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#CreditTransfer -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#CreditT
ransfer">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Depo
sit"/>
    </owl:Class>



    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#Delete --
>

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Delete"
>
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Rese
rvationManagement"/>
    </owl:Class>
    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#Deposit -
->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Deposit
">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Rese
rvationCharge"/>
    </owl:Class>



    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#Discount
-->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Discoun
t">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Rese
rvationCharge"/>
    </owl:Class>



    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#Email -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Email">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Noti
fication"/>
    </owl:Class>



    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#Fax -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Fax">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Noti
fication"/>
    </owl:Class>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#HandleConflict -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#HandleC
onflict"/>
```

```
    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#MultipleRoom -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Multipl
eRoom">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Bloc
k"/>
    </owl:Class>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#MultipleTime -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Multipl
eTime">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Bloc
k"/>
    </owl:Class>



    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#No -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#No">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Hand
leConflict"/>
    </owl:Class>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#Notification -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Notific
ation"/>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#PrintedPaper -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Printed
Paper">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Noti
fication"/>
    </owl:Class>
    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#ReservationCharge
-->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Reserva
tionCharge"/>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#ReservationManagem
ent -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Reserva
tionManagement"/>



    <!--
http://www.Hallbooking.com/ontologies/Hallbooking.owl#ReservationMode --
>

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Reserva
tionMode"/>



    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#Single --
>

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Single"
>
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Rese
rvationMode"/>
    </owl:Class>



    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#Sms -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Sms">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Noti
fication"/>
    </owl:Class>



    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#Tax -->
    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Tax">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Rese
rvationCharge"/>
    </owl:Class>



    <!-- http://www.Hallbooking.com/ontologies/Hallbooking.owl#Yes -->

    <owl:Class
rdf:about="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Yes">
        <rdfs:subClassOf
rdf:resource="http://www.Hallbooking.com/ontologies/Hallbooking.owl#Hand
leConflict"/>
    </owl:Class>
</rdf:RDF>



<!-- Generated by the OWL API (version 3.4.2)
http://owlapi.sourceforge.net -->
```

Bibliography

[1] Software product lines: practices and patterns. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[2] Pohl, Klaus, Böckle, Günter, van der Linden, Frank J.. Software Product Line Engineering Foundations, Principles, and Techniques. Springer-Verlag New York, Inc. Secaucus, NJ, USA ©2005

[3] Greenfield, J., Short, K.: Software Factories: Assembling Applications with Patterns,Models, Frameworks, and Tools. Wiley (2004) To be published.

[4] Czarnecki, K., Eisenecker,U.W. Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)

[5] Bosch, J.: Design and Use of Software Architecture: Adopting and evolving a product-line approach. Addison-Wesley (2000)

[6] Feature model: https://en.wikipedia.org/wiki/Feature_model (Last Visited in 12.11.2016)

[7] Semantic web: https://en.wikipedia.org/wiki/Semantic_Web (Last Visited in 12.11.2016)

[8] Volker Haarslev and Ralf Moller RACER User's Guide and Reference Manual Version 1.7.19. April 26, 2004

[9] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Fergerson, and Mark A. Musen. Creating Semantic Web Contents with Protégé-2000

[10] Monica Shekhar and Saravanaguru RA. K, Semantic Web Search based on Ontology Modeling using Protégé Reasoner

[11] Kyo C. Kang, Sholom G. Cohen, James A. Hess ,William E. Novak A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, Software Engineering Institute Carnegie Mellon University, November 1990

[12] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, Moonhang Huh, FORM: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering 5 (1), 143-168, January 1998.

[13] Jan Bosch. Design and use of software architectures: adopting and evolving A product-line approach. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[14] Ontology: https://en.wikipedia.org/wiki/Ontology(Last Visited in 25.10.2016)

[15] Jan Bosch. Design and use of software architectures: adopting and evolving A product-line approach. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.
[16] http://www.sei.cmu.edu/productlines/ (Last Visited in 12.11.2016)

[17] http://www.tdgseville.info/topics/spl (Last Visited in 12.11.2016)

[18] Shusheng Zhang, Weiming Shen, and Hamada Ghenniwa.A review of Internet-based product information sharing and visualization. May 2004, Pages 1–15

[19] Ian Horrocks, Peter F. Patel-Schneider,and Frank van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. Journal of Web Semantics, 2003.

[20] M. Bernardo, P. Ciancarini, and L. Donatiello.Architecting families of softwaresystems withprocessalgebras.ACM Transactions on Software Engineering and Methodology, 11(4):386–426, 2002.

[21 Lee, K., Kang, K.C., Lee, J.: Concepts and guidelines of feature modeling for product line software engineering. In Gacek, C., ed.: Software Reuse: Methods, Techniques, and Tools: Proceedings of the Seventh Reuse Conference (ICSR7), Austin, USA, Apr.15-19, 2002. LNCS 2319, Springer-Verlag (2002) 62–77

[22] Protégé: http://protegewiki.stanford.edu/wiki/RacerProTG (Last Visited in 20.11.2016)

[23] Andreas Hein, John MacGregor, and Steffen Thiel. Configuring software product line Features. In ElkePulvermller, Andreas Speck, James Coplien, Maja D Hondt, and Wolfgang De Meuter, editors, Proceedings of the ECOOP 2001 Workshop on Feature

Interaction in Composed Systems (FICS 2001), Budapest, Hungary, June 18-22, 2001, volume 2001-14 of Technical Report, pages 67–69

[24] Griss, M., Favaro, J., d‘ Alessandro, M.: Integrating feature modeling with the RSEB. In: Proceedings of the Fifth International Conference on Software Reuse(ICSR), IEEE Computer Society Press (1998) 76–85