韓國IT서비스學會誌
第10卷 第3號
2011年 9月

시맨틱 웹 기술을 이용한 특성 모델 및 특성 구성 검증 도구*

최 승 훈**

Verification Tool for Feature Models and Configurations using Semantic Web Technologies*

Seung-Hoon Choi**

■ Abstract ■

Feature models are widely used to model commonalities and variabilities among products during software product line development. Feature configurations are generated by selecting the features to be included in individual products. Automated tools to identify errors or inconsistencies in the feature models and configurations are essential to successful software product line engineering. This paper proposes a verification technique and tool based on semantic web technologies such as OWL, SWRL and Protege API. This approach checks the feature model and configuration based on predefined rules and provides information on existence of errors as well as the kinds of those errors. This approach is extensible due to ease of rule modification and may be easily applied to other environments because semantic web technologies can be easily integrated with other programming environments. This paper demonstrates how various semantic web-related technologies can support automatic verification of one kind of software development artifact, the feature model.

Keyword : Feature Model, Feature Configuration, Verification, Semantic Web, OWL, SWRL

논문투고일 : 2011년 07월 15일 논문수정완료일 : 2011년 08월 16일 논문게재확정일 : 2011년 09월 14일
* 본 연구는 덕성여자대학교 2010년도 교내연구비 지원에 의해 수행되었음.
** 덕성여자대학교 컴퓨터학과 부교수

1. Introduction

Software Product Lines (SPL) refer to a set of software intensive systems developed for a specific aim or market from common software assets [2]. In SPL engineering, it is very important to identify and manage the commonalities and variabilities among potential products.

In general, commonalities and variabilities are analyzed in terms of features or prominent and distinctive characteristics of a system. Feature models are used to represent commonalities and differences in hierarchical form and include composition rules specifying how variable features can be combined to construct valid individual products.

Feature configuration is the selection of a set of features used to produce a particular product from the product line [12]. Feature configuration enables the application engineer to derive an individual product from the product line assets.

The correctness of the feature model and configuration strongly affects the quality of the SPL. In industry, there may be thousands of features (variabilities) involved in large scale software product line development; as a result, a substantial amount of effort is required to correct the associated errors [3].

Many approaches for the automated analysis of feature models and configurations have been proposed [1], most of which have been based mainly on propositional logic, constraint programming and description logic. Of these three kind of approaches, those based on description logic are less common.

The Semantic Web is a group of methods and technologies to allow machines to understand the meaning or “semantics” of information on the World Wide Web. The Semantic Web has emerged as the next generation web technology and includes numerous models and technologies proposed by W3C. Among these technologies, OWL (Web Ontology Language) is most important and provides formal descriptions of concepts, terms and relationships within a given knowledge domain. Numerous other related technologies have been developed including SWRL (Semantic Web Rule Language), SPARQL (SPARQL Protocol and RDF Query Language).

There is a strong relationship between Semantic Web ontology and feature models. Both represent concepts in a particular domain and define their relations to various properties. This paper attempts to verify feature models and configurations using semantic web technologies based on description logic. OWL ontology language is employed to represent and formalize the knowledge contained in feature models and configurations. SWRL is used to implement the verification rules. A pellet reasoner is used to evaluate the rule and analyze the consistency of them. Other semantic web techniques such as SQWRL (Semantic Query-Enhanced Web Rule Language) can be used to extend our approach.

To illustrate the approach, this paper uses an example of the Graph Product Line (GPL) feature model which is proposed in [8] as a standard problem for evaluating software product line technologies. A feature modeling tool to facilitate the development and verification of a feature model and configuration was implemented via Protege API.

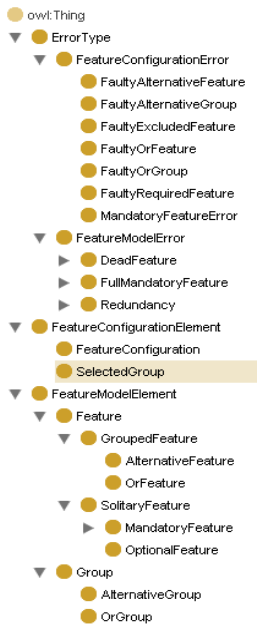
This paper is organized as follows. Section 2 describes the ontology for feature models and configurations. Section 3 investigates the types of errors and inconsistencies that can exist in a feature model and configuration. Then, the verification technique based on SWRL rules is explained. In Section 4, a tool for creating and verifying the feature model and configuration is demonstrated using the GPL product line example. Section 5 discusses

the related works with this paper. Section 6 concludes the paper and describes future work.

2. Ontology for Feature Models and Configurations

2.1 Classes for Feature Models and Configurations

To represent the nodes in the feature models and configurations, classes and class hierarchy should be defined. [Figure 1] displays the classes of a designed hierarchy. The feature model element, represented as FeatureModelElement, is most important in ontology and is classified into two classes, such as GroupedFeature and SolitaryFeature. A GroupedFeature is one for which the feature always exists as a member of a group and is classified as either an alternative feature (represented as AlternativeFeature) or an or feature (represented



[Figure 1] Classes for Feature Model Ontology

〈Table 1〉 Datatype Properties in an Ontology

Datatype Property	Meaning(Domain/Range)
hasName	refers to the name of the current feature (D : *Feature*/R : *string*)
hasSame Group MemberCount	refers to the number of the other features in the same group (D : *GroupedFeature*/R : *int*)
has Selected Member Count	refers to the number of members selected into the current feature configuration among all members of the group (D : *SelectedGroup*/R : *int*)

〈Table 2〉 Object Properties in an Ontology

Object Property	Meaning (Domain/Range)
excludes	refers to the feature excluded by the current feature(D : *Feature*/R : *Feature*)
hasChild	refers to the child feature of the current feature (D : *SolitaryFeature*/R : *SolitaryFeature*); inverse function of hasParent
hasGroup	refers to the group of the current parent feature(D : *SolitaryFeature*/R : *Group*); inverse function of isGroupOf
has Member	refers to a member of the current group (D : *Group*/R : *GroupedFeature*); inverse function of isMemberOf
hasSame Group Member	refers to the member feature included in same group with the current member feature (D : *GroupedFeature*/R : *GroupedFeature*)
has Selected Feature	refers to the feature which was selected into the current feature configuration (D : *FeatureConfiguration*/R : *Feature*)
has Selected Group	refers to the group which was selected into the current feature configuration (D : *FeatureConfiguration*/ R : *SelectedGroup*)
hasUnse lected Feature	refers to the feature which was not selected into the current feature configuration (D : *FeatureConfiguration*/R : *Feature*)
is Identical To	refers to the group in the feature model identical to that selected into the current feature configuration(D : *SelectedGroup*/ R : *Group*)
requires	refers to the feature required by the current feature(D : *Feature*/R : *Feature*)

as OrFeature). A SolitaryFeature is one for which the features exist in isolation and do not pertain to any feature group. A SolitaryFeature is classified as either a mandatory feature (represented as MandatoryFeature) or an optional feature (represented as OptionalFeature). The groups in the feature model are classified into the alternative group (represented as AlternativeGroup) and the or group (represented as OrGroup).

2.2 Properties for Feature Models and Feature Configurations

To represent the relationships between the nodes in feature models and feature configurations, several data type properties and object properties should be defined. <Table 1> displays main data type properties and their domains/ranges. <Table 2> displays main object properties to represent the relationships between the classes in the ontology.

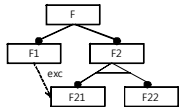
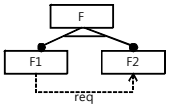
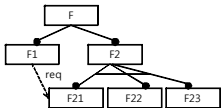
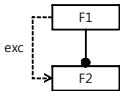
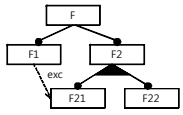
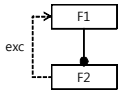
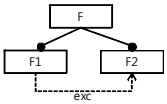
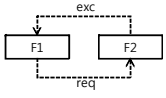
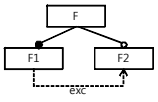
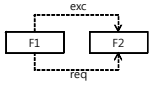
3. Verification of Feature Models and Configurations

3.1 Errors in Feature Models

[13] defined an error in a feature model as an incorrect definition of a relationship that suggests that the set of products described by a feature model may not match the SPL it describes. That study identified three kinds of errors that can be included in a feature model, dead, full-mandatory and void features, as follows.

- **dead features** : A dead feature is one that is not able to be instantiated, i.e., a feature that appears in no feature configuration despite being defined in a feature model.
- **full-mandatory features** : A child feature in a non-mandatory relationship is a full-mandatory feature if it has to be instantiated whenever its parent feature is instantiated.

〈Table 3〉 Errors of a Dead Feature

ID	Feature Model	Dead Feature	ID	Feature Model	Dead Feature
DF1		F21	DF6		F1, F2
DF2		F22, F23	DF7		F2
DF3		F21			F1
DF4		F2	DF8		F1, F2
DF5		F2			F1, F2

- **void feature models** : A feature model is void if it is defined for no product. A feature model is void if its root is a dead feature.

<Table 3> and <Table 4> summarize the cases in which dead features and full mandatory features occur, respectively.

3.2 Constraints for Feature Configurations

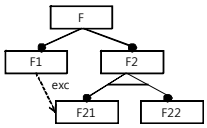
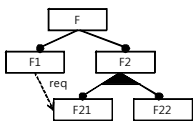
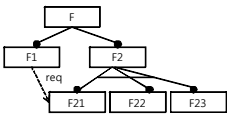
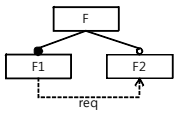
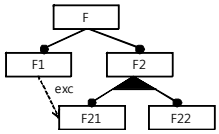
To automate the detection of inconsistencies in feature configurations associated with a feature model, we must confirm that the constraints represented in the feature model are also applied to the feature configurations. The following constraints relating to mandatory, optional, alternative, or feature types and requires/excludes relationships are considered.

- For mandatory features : If Fi is a mandatory feature in feature model FMi, it should be included in the feature configuration FC derived from FMi.
- For optional features : If Fi is an optional feature in feature model FMi, no constraints apply because Fi does not influence the validity of the feature configuration.
- For alternative features : If FGi is an alternative feature group in feature model FMi, only one of the features in FGi should be included in the feature configuration FC derived from FMi.
- For or features : If FGi is an or feature group in feature model FMi, at least one of the features in FGi should be included in the feature configuration FC derived from FMi.
- For requires relationships : If Fi is selected into a feature configuration FC derived from a feature model FMi, all features that have a required relationships with Fi should be included in the FC.
- For excludes relationships : If Fi is selected into a feature configuration FC derived from a feature model FMi, all features that have excludes relationships with Fi should NOT be included in the FC.

3.3 SWRL Rules for Feature Models

For verification of feature models and configurations, most rules should be defined to allow for

〈Table 4〉 Errors of a Full Mandatory Feature

ID	Feature Model	Full Mandatory Feature	ID	Feature Model	Full Mandatory Feature
FMF1		F22	FMF4		F21
FMF2		F21	FMF5		F2
FMF3		F22			

the detection of errors. Whenever an error is found in an ontology, all of the rules which apply to the specific error conditions should be activated.

In this paper, SWRL is employed as a rule language for verification of feature models and configurations. SWRL is based on a combination of the OWL DL and OWL Lite sublanguages of the OWL with Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. These rules take the form of an implication between an antecedent (body) and a consequent (head); if the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

In our approach, an OWL ontology contains a sequence of axioms and facts about feature models and feature configurations. A semantic reasoner infers logical consequences from a set of asserted facts or axioms. An SWRL reasoner analyzes the facts and axioms of the feature model and feature configurations. Whenever the antecedent of the rule matches the facts and axioms, it infers a new instance of an error class. The fact that a new instance is inferred by the current ontology indicates that the feature model or feature configuration has errors represented by a new instance.

This section describes the SWRL rule which defines the OWL DL rules used to detect the various errors contained in the ontology.

• Rules for dead features

The definitions of the SWRL rules used to detect the various kinds of dead feature errors in section 3.1 are described here. The symbol → separates the IF- clause (left-hand side) and the THEN-clause (right-hand side). The IF-clause represents the situation in which a dead feature error occurs. When the IF clause is true, the THEN clause is also true.

Let's consider the case of DF1 in <Table 3>. SRWL rule for DF1 is defined as follows :

MandatoryFeature(?f2) ∧ hasGroup(?f2, ?group)
∧ hasMember(?group, ?f21) ∧
AlternativeFeature(?f21) ∧
MandatoryFeature(?f1) ∧ excludes(?f1, ?f21)
→ DeadFeature1(?f21)

In above rule, IF part represents the situation in which a feature represented by the ?f2 variable is a mandatory feature and the feature represented by the ?f21 variable is one of the alternative features contained in the group including ?f1. When the reasoner determines that the IF part is true, then the assertion that ?f21 is an instance of DeadFeature1 is added into the current ontology. The other DFRules are defined in the same way.

• Rules for full mandatory features

There are five SRWL rules for the full mandatory features. Let's consider the case of FMF4 in <table 4>. SRWL rule for FMF4 is defined as follows :

MandatoryFeature(?f2) ∧ hasGroup(?f2, ?group)
∧ hasMember(?group, ?f21) ∧ OrFeature(?f21)
∧ MandatoryFeature(?f1) ∧ requires(?f1, ?f21)
→ FullMandatoryFeature4(?f21)

In above rule, the IF part of the rule tests whether the feature represented by the ?f21 variable is included in the list of features required by the ?f1 feature. When the IF part of FMFRule 4 is satisfied, the feature represented by the ?f21 variable is included in the FullMandatoryFeature4 class and is added into the current ontology. The other rules are defined in the same way.

3.4 SWRL Rules for Feature Configurations

SWRL rules to detect the errors which violate the constraints of the feature configurations are defined in the same way as are the rules for dead and full mandatory features. Let's consider the SWRL rule for *requires-relationship* in feature configuration. This rule is defined as follows :

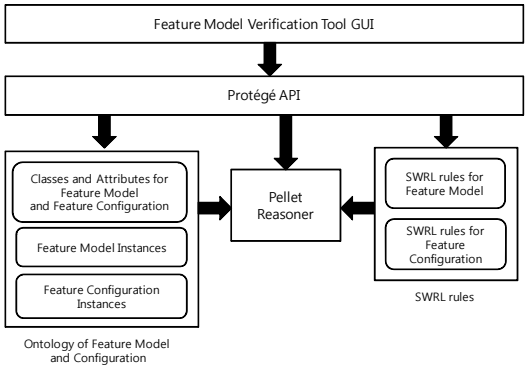
$$Feature(?f1) \wedge Feature(?f2) \wedge FeatureConfiguration(?fc1) \wedge hasSelectedFeature(?fc1, ?f1) \wedge requires(?f1, ?f2) \wedge hasUnselectedFeature(?fc1, ?f2) \rightarrow FaultyRequiredFeature(?f2)$$

According to above rule, if a feature required by some feature already selected in the feature configuration was not selected, that feature is an instance of FaultyRequiresFeature. The other SWRL rules for feature configurations are defined in the same way.

4. Feature Model Verification Tool

4.1 Architecture

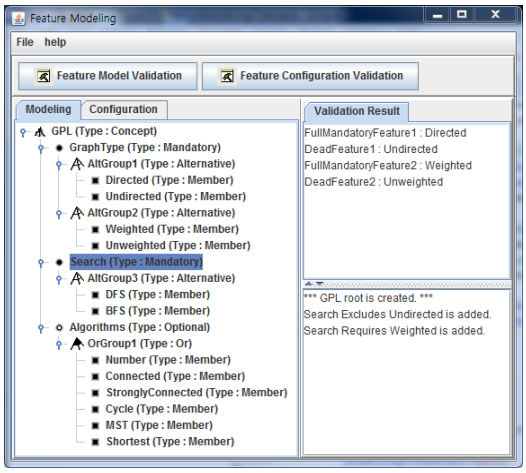
[Figure 2] displays the overall architecture of the feature model and configuration verification using semantic web technologies. The ontology consists of the meta-model for feature models and feature configuration, feature model instances and feature configuration instances. The ontology and SWRL rules for feature models and feature configuration are used to identify errors in the current feature model and feature configurations.



[Figure 2] Overall Architecture of the Semantic Web-based Feature Modeling Tool

4.2 Case Study : GPL Feature Model

[Figure 3] shows the GUI of the verification tool. The GPL feature model, a standard problem for evaluation of software product line technologies proposed by [8], is constructed by this tool. To test this tool, some feature model errors were inserted into the original GPL feature model, as shown in <Table 5>. Our approach should identify all of the errors listed in <Table 5>, and the validation result tab of [Figure 3] shows that our tool detected all feature model errors (DF1, DF2, FMF1, FMF2).

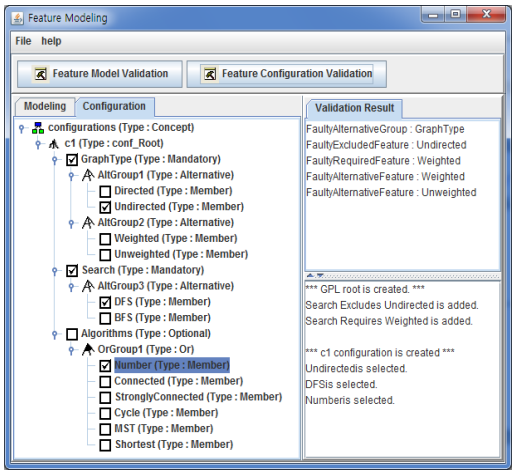


[Figure 3] GPL Feature Model

〈Table 5〉 Test Case for Feature Configuration

Inserted Feature Model Error	Expected Verification Results
1) *Search* feature is changed from optional to mandatory. 2) *Search* feature has a excludes relationship with an *Undirected* feature. 3) *Search* feature has an *requires* relationship with an *Weighted* feature.	1) *Undirected* feature is a dead feature according to DeadFeature1 rule. 2) *Directed* feature is a full mandatory feature according to FMF1 rule. 3) *Weighted* feature is a full mandatory feature according to FMF2 rule. 4) *Unweighted* feature is a dead feature according to DeadFeature2 rule.

The feature configuration from the GPL feature model was then determined. To test the verification function of the tool, some errors were inserted into a feature configuration example, as in <Table 6>. Figure 4 displays a feature configuration corresponding to the test case in <Table 6>. The validation result tab of [Figure 4] shows that our tool identified the feature configuration errors (errors relating to constraints and alternative relationships).



[Figure 4] Feature Configuration Example

〈Table 6〉 Test Case for Feature Configuration Verification

Inserted Feature Configuration Errors	Expected Verification Results
1) *Undirected* feature that is excludes by the *Search* feature is selected for feature configuration. 2) *Weighted* feature that is required by the *Search* feature are not selected for feature configuration. 3) Both the *Weighted* and *Unweighted* features are not selected for feature configuration.	1) The feature configuration generates following *requires* error : FaultyExcludedFeature : (*Undirected*). 2) The feature configuration generates following *excludes* constraint : FaultyRequiredFeature (*Weighted*). 3) The feature configuration generates following alternative-related errors : FaultyAlternativeGroup (*GraphType*), FaultyAlternativeFeature (*Weighted*), FaultyAlternativeFeature (*Unweighted*)

4.3 Evaluation

Our approach has several advantages compared with other methods. These advantages are as follows.

• Extensibility

In our approach, the meta-model of the feature model, the feature model and feature configuration instances, rules for feature model verification and feature configuration verification are constructed separately. This separation allows each part to be easily modified and extended. For example, one undesirable property of the feature model is the redundancies that occur when semantic information is modeled in multiple ways [14]. This kind of error can be detected by defining the following SWRL rules to identify redundancy, as follows :

MandatoryFeature(?f1) ∧
MandatoryFeature(?f2) ∧ hasParent(?f1, ?p) ∧
hasParent(?f2, ?p) ∧ requires(?f1, ?f2) →
Redundancy1(?f1)

- Applicability of various recent semantic web technologies

Recently, semantic web technologies have become a research topic of interest, and various new applicable technologies have been developed. These technologies are applied in our proposed approach. For example, SQWRL facilitates the query for the feature model. If we want to search all of the sub?features of the GPL feature in the GPL feature model, the following SQWRL query can be utilized :

hasParent(?f, GPL) ∧ hasName(?f, ?name) ∧
swrlb : stringConcat(?featureName, "Feature :", ?name)
→ sqwrl : select(?featureName)

- Integration with other tools

The Protege framework that is utilized in our approach was developed in Java. Through this framework, we can programmatically build and manage the ontology using Java. This characteristic allows us to integrate our approach into other software development environments.

5. Related Works

[4] proposed an approach to determine the consistency of the feature model via formalization with description logic. They translated the feature model into the ALCQI knowledge base and performed consistency reasoning via the description logic reasoner RACER. Their approach can determine whether or not the feature model is consistent but cannot provide any information or explanation about errors that can be detected. In addition, their approach cannot verify the feature configuration.

[11] proposed a feature meta-model for verification of feature models based on ontology. Their paper analyzed the problem domain in terms of business, regarding features as business operations and defining classes and attributes to represent these operations. This method is different from our approach in that it saved a feature model in ABox by representing an instance as a sub-class in the meta-model. This approach defined the rules between the features mainly based on binding and verified the feature model using these rules. This approach only describes the validity of the feature model and does not provide an explanation of the errors. In addition, this approach does not provide rules or techniques for verification of the feature configuration.

[15, 16] represented the feature model and configuration in OWL [10] using the Protege ontology development tool [7] and then verified the feature configurations using reasoning engines like RACER [5] or FaCT [6]. This approach is similar to that in the present paper in its use of a reasoning engine; however, it does not define the meta-model of the feature model and does not verify the feature model. In addition, explicit explanation of the causes of inconsistency is not provided, and an additional OWL debugging tool hints at inconsistencies.

[9] implemented the techniques of [15, 16] using JENA frameworks and a Pallet reasoning engine. This method supports only the verification of the feature configuration and is not different from that of [16] except that it solves the problem programmatically.

[17] developed the ontology framework for the

feature model and proposed a method to identify potential logical errors in one feature model or in the integrated distributed feature models. This method is similar to our approach with regard to the definition of the feature model ontology in OWL and the verification rule for the feature model in SWRL, but did not provide a method for verification of the feature configuration.

The comparison of our approach and other approaches of feature model and configuration verification based on ontology and the semantic web is summarized in <Table 7>.

6. Conclusions

In a software product line paradigm, the commonalities and variabilities among the products should be systematically analyzed and managed. Feature models are widely used to manage variabilities and commonalities.

As the number of features in a product line increases, the probability of invalid feature models and feature configurations also increases. As a result, the necessity for an automated tool to verify the feature models and feature configuration incre-

〈Table 7〉 Comparison with other Approaches

Item		Our Approach	[9, 16]	[17]
Purpose		Verification of feature model and feature configuration	Verification of feature configuration	Verification of feature model
Semantic web technologies applied		OWL, SWRL, SQWRL	OWL	OWL, SWRL
Meta-model for feature model		TBox(OWL classes and attributes)	no	TBox(OWL classes and attributes)
Feature model instances		ABox(OWL individuals)	TBox(OWL classes and attributes)	ABox(OWL individuals)
Feature configuration instances		ABox(OWL individuals)	TBox(OWL classes and attributes)	no
Verification rules		SWRL rules (Because the verification rules are separately defined in SWRL, more complex verification rules are easily added to the existing rules.)	OWL classes and attributes (Because the verification rules are defined by OWL classes and attributes, insertion of the new verification rules causes changes in the TBox.)	SWRL rules (only for verification of the feature models)
Reasoning engine		Pellet (Instances representing the causes of errors are added into the current ontology)	FaCT++ (Only hints for cause of error are provided by an independent feature model debugger.)	Pellet (Explanation about the reason for the error is not provided.)
Query for feature model		SQWRL	no	no
Operations	Dead features	O	X	O
	Full mandatory features	O	X	X
	Valid product	O	O	X
	Explanation	O	O	O

ases. This paper attempted to verify the feature models and feature configuration using semantic web technologies.

Our approach constructed an ontology for the feature model and feature configurations using OWL, and the rules used to identify the various errors in the feature model and feature configuration were defined using SWRL. When a developer builds a feature model and feature configurations, these are changed into the ontology. Then, SWRL verification rules are applied to this ontology, and if errors are detected, instances representing these errors are added to the ontology.

The feature model verification tool is implemented based on this approach using the Protege framework. The Graph Product Line, a standard for evaluating software product line technologies, was used to prove the capabilities of this tool. The advantages of our approach include applicability of several recent semantic web-related technologies such as SQWRL, extensibility of the verification rule set, and integrability with other tools.

This paper demonstrates that various semantic web-related technologies can be applicable for feature modeling and verification. Future works will include application of this approach to the larger feature set, extension of this approach for the extended feature model, and utilization of this approach for code generation.

References

[1] Benavides, D., S. Segura, and A. Ruiz-Cortes, "Automated Analysis of Feature Models : A Detailed Literature Review", *In Technical Report ISA-09-TR-04,* Applied Software Engineering Research Group, University of Seville, Spain, 2009.

[2] Clements, P. and L. Northrop, *Software Product Lines : Practices and Patterns*, Addison Wesley, 2002.

[3] Deelstra, S., M. Sinnema, and J. Bosch, "Experiences in software product families : problems and issues during product derivation", SPLC, *LNCS*, Springer, Vol.3154(2004), pp. 165-182.

[4] Fan, S. and N. Zhang, "Feature Model Based On Description Logics", *In Knowledge-Based Intelligent Information and Engineering Systems*, 2006.

[5] Haarslev, V. and R. M¨oller, *RACER User's Guide and Reference Manual* : Version 1.7.6, 2002.

[6] Horrocks, I., Fact++ web site. http://owl.man.ac.uk/factplusplus/.

[7] http://protege.stanford.edu/.

[8] Lopez-Herrejon, R. E. and D. S. Batory, "A standard problem for evaluating productline methodologies", *Proceedings of the Third International Conference on Generative and Component-Based Software Engineering*, Springer-Verlag.pages, (2001), pp.10-24.

[9] Matcha, V. B. et al., "Software Reuse : Ontological Approach to Feature Modeling", *IJCSNS(International Journal of Computer Science and Network Security)*, Vol.9, No.8(2009).

[10] McGuinness, D. L. and F. van Harmelen (Eds.), "OWL Web Ontology Language Overview", http://www.w3.org/TR/2003/PR-owl-features-20031215/, 2003.

[11] Peng, X., W. Zhao, Y. Xue, and Y. Wu, "Ontology-Based Feature Modeling and Application-Oriented Tailoring", *In ICSR*, (2006), pp.87-100.

[12] Thiel, S. and A. Hein, "Systematic Integration of Variability into Product Line Architecture

Design", SPLC 2002, *LNCS*, Springer-Verlag, Vol.2379, pp.130-153.

[13] Trinidad, P., D. Benavides, A. Duran, A. Ruiz-Cortes, and M. Toro, "Automated Error Analysis of Feature Models", *Journal of Systems and Software*, 2008.

[14] von der Massen, T. and H. Lichter, "Deficiencies in feature models", *In T. Mannisto and J. Bosch, editors, Workshop on Software Variability Management for Product Derivation -Towards Tool Support*, 2004.

[15] Wang, H., L. Y. Fang, J. Sun, H. Zhang and J. Z. Pan, "A Semantic Web Approach to Feature Modeling and Verification", *In Proc. of the ISWC2005 Workshop on Semantic Web Enabled Software Engineering(SWESE),* 2005.

[16] Wang, H. H., Y. F. Li, J. Sun, H. Zhang, and J. Pan, "Verifying Feature Models Using OWL", *In Journal of Web Semantics : Science, Services and Agents on the World Wide Web*, Vol.5, No.2(2007), pp.117-129.

[17] Zaid, L. A., F. Kleinermann and O. D. Troyer, "Applying Semantic Web Technology to Feature Modeling", *ACM SAC(Symposium on Applied Computing),* 2009.

◈ 저 자 소 개 ◈

최 승 훈 (csh@duksung.ac.kr)

서울대학교 계산통계학과에서 학부를 졸업하고, 서울대학교 대학원 계산통계학과에서 석사 학위를 취득하였으며, 서울대학교 대학원 계산통계학과에서 박사학위를 취득하였습니다. 현재 덕성여자대학교 컴퓨터학과 교수로 재직 중이며, 2004년에는 방문 연구 교수로 George Mason University를 다녀왔습니다. 현재 온톨로지를 활용한 소프트웨어 프러덕트 라인 개발 방법과 코드 자동 생성 프로그래밍에 대해 연구를 수행 중입니다.