

# A Model Driven-based Approach for Converting Feature Models of Software Product Lines to OWL Ontologies

Aissam Belghiat<sup>a</sup>, Mohamed Boubakir<sup>b</sup>, Ghada Chouikh and Djamilia Kemmache  
*LaRIA Laboratory, University of Jijel, 18000, Jijel, Algeria*

**Keywords:** Software Product Line, Feature Model, OWL, Ontology, Verification.

**Abstract:** Software product line engineering has gained recognition as a promising approach to developing families of software systems. A Software Product Line (SPL) is a set of software products that share and support a set of Features. The variabilities and commonalities of the features of a software product line are modeled by Feature models (FM). The lack of formal semantics for these models has hindered their analysis and verification, and consequently their correction and evolution. The use of Ontology Web Language (OWL) ontologies should solve the problem. They accurately allow capturing the interrelationships between features in a FM, and to proceed, thereafter, to the analysis and the verification of these models by using the formal semantics of the OWL which is based on the description logic. In this paper, we propose to convert Feature Models into OWL ontologies using Model Driven Engineering (MDE). We have firstly proposed numerous semantic rules to enable the transformation. After that, meta-modeling and model transformation are used to implement and automate the rules. Specialized MDE tools are used (e.g. Acceleo, Eclipse modeling framework). The Protégé tool is used for reasoning on the generated OWL ontology. A case study is given to show the effectiveness of our approach.

## 1 INTRODUCTION


Product Line Engineering (PLE) (Van der Linden, 2002) is an approach in software development that aims to enhance efficiency by reusing and managing both commonalities and variabilities across a line of products. By using this method, companies can create new products faster while keeping costs and time to market as low as possible.


An SPL (Software Product Line) (Clements, 2002) is a set of software products that share and support a set of Features. Feature models (FMs) are widely used to represent the commonalities and variabilities within a product line (She, 2011) (Rubin, 2012). Feature models offer a clear way to represent features and their relationships, which facilitates the configuration of various product variants (Acher, 2013). However, these diagrams, often used for specifying these features, have some shortcomings regarding their formal structure (Batory, 2005). In fact, because they lack strict formal semantics, conducting automated analysis and reasoning can be

difficult. This lack of formality may result in issues related to ensuring the consistency, completeness, and correctness of the feature model.

Several approaches have been proposed both by researchers and practitioners to overcome this limitation by formalizing Feature Models, such as using formal languages (Batory, 2005). These efforts are aimed to improve the reliability of Feature model and enable better whole complex system analysis and verification.

The formal language OWL (Ontology Web Language) (W3C, 2012) is a central component in the context of Semantic Web and ontologies. OWL provides a powerful and rich framework for representing and reasoning about knowledge, which facilitates machine-understandable representations and interoperability. Transforming Feature Models to OWL Ontologies is a promising path to help the practitioners of Software Product Lines creating formal ontologies, which capture the semantics of features, relationships, and variabilities within a product line.

<sup>a</sup>  <https://orcid.org/0000-0002-5968-609X>

<sup>b</sup>  <https://orcid.org/0000-0003-3890-4913>

By combining the intuitiveness of Feature Modeling, and the expressiveness and formal power of OWL ontologies, software engineers can achieve a higher level of precision and accuracy in capturing the variability and structure of software product lines. This integration enables automatic analysis, reasoning, and validation of product configurations, leading to improved quality assurance and decision-making processes within PLE projects.

In this study, we use the potential of Model Driven Engineering (Schmidt, 2006) to develop an approach for automatic converting of FMs to their corresponding OWL ontologies. The approach firstly proposes multiple transformation rules that semantically relate FMs to OWL. These rules are then implemented using a model-driven technique to generate a tool that fully automates the transformation process.

The Eclipse modeling project (Eclipse Modeling Project, 2024) is adopted to realize this approach. EMF (Budinsky, 2004) is used to meta-model FM models. This enables generating editors for such models. Afterwards, Acceleo (Acceleo, 2023) is used to implement the proposed semantics rules to allow the generation of OWL ontologies which are then uploaded in special tools for analysis such as Protégé (Protégé, 2023).

The rest of the paper is as follows. Section 2 presents the context of the work. Section 3 explains the methodology adopted to develop the approach. Section 4 applies the approach on a real example. Section 5 presents related work. Section 6 concludes the paper.

## 2 CONTEXT

### 2.1 FM

In PLE, Feature models are widely used to model common and variable features of an SPL and relationships between them. They are originally proposed as part of FODA approach (Kang, 1990). A FM is a set of features hierarchically structured into multiple levels of detail (Batory, 2005). It represents all possible products of a software product line in a single model (Benavides, 2007). A FM is usually represented by a feature diagram and a set of constraints.

A feature diagram is a hierarchical tree diagram that shows features and the relationships between them. It has a root feature that describes the model and all its characteristics (Ghabach, 2018). This model is represented by tree structures where each node is a Feature and each edge can have four

possible values known as parental relationships that are (Benavides, 2007): Mandatory, Optional, alternative, and Or. Constraints are defined in the feature model to ensure that the products created are valid and correct. Constraints that are defined in FODA are (Kang, 1990): Requires and Excludes.

An example of a Feature Model is shown in figure 9.

### 2.2 OWL

The web ontology language (OWL) (W3C, 2012) is precisely an ontology language that adds more vocabulary to describe properties and classes unlike RDF and RDF-S that just provides classes and properties. OWL offers advanced semantic modeling capabilities, for example relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes (W3C, 2012). OWL provides three sub-languages with increasing expressiveness respectively: OWL Lite, OWL DL and OWL Full. We use OWL DL (Description Logic) because it provides more expressiveness while maintaining computational completeness and decidability.

An example of an OWL ontology is given in figure 11.

### 2.3 MDE

Model Driven Engineering (MDE) (Schmidt, 2006) is an approach for software development that relies on models. Models are expressed using modeling languages and they conform to meta-models. A meta-model is itself a model that defines the structure of modeling languages. A meta-model is composed of an abstract syntax which describes the elements of the models and their relationships, and a concrete syntax that describes the representation of the elements defined by the abstract syntax (it can be graphic, textual or mixed). Meta-models themselves must be conformed to a Meta-Meta-Model. It is a model that describes a meta-modeling language, i.e. the modeling elements required for modeling language definition.



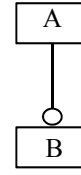
To realize a transformation between models using MDE, we need to define meta-models for the source and target models. Then a model transformation, which consists on defining corresponding semantics rules between those meta-models is developed to enable automatic mapping of input models to output models.

### 3 METHODOLOGY

#### 3.1 Converting Rules

Feature models and OWL ontologies have shown several similarities. To enable the transformation of Feature models into their corresponding OWL ontologies, we have proposed multiple rules. Indeed, we have studied the characteristics of both formalisms and we have come up with multiple semantics rules, which allow converting each Feature model into its equivalent OWL ontology. Table 1 summarizes the transformation rules.

Table 1: The proposed transformation rules.

Feature Diagram	OWL Ontology
<p><b>Feature</b></p> 	<p><b>Owl Class</b></p> <pre>&lt;owl:Class rdf:about="A" &gt; &lt;/owl:Class&gt;</pre>
<p>A FD feature is transformed into an OWL class. The name of the class will be the same as the name of the feature.</p>	
<p><b>Mandatory</b></p> 	<pre>&lt;owl:Class rdf:ID="B"&gt; &lt;owl:equivalentClass&gt; &lt;owl:Restriction&gt; &lt;owl:onProperty rdf:resource="#Has[A]"/&gt; &lt;owl:cardinality rdf:datatype="http://www.w3.org/2001 /XMLSchema#nonNegativeInteger"&gt; 1 &lt;/owl:cardinality&gt; &lt;owl:onClass rdf:resource="#[A]"/&gt; &lt;/owl:Restriction&gt; &lt;owl:equivalentClass&gt; &lt;owl:FunctionalProperty rdf:about="#HasB" /&gt;</pre>
<p>A mandatory feature in a feature diagram defines a restriction on the property "Has[B]" with a cardinality of 1. This indicates that instances of this class must have exactly one value for the property. We have specified "hasB" to be a functional property, which means that every "A" can have at most one "B".</p>	
<p><b>Optional</b></p> 	<pre>&lt;owl:Class rdf:ID="B"&gt; &lt;rdfs:subClassOf rdf:resource="B" /&gt; &lt;/rdfs:subClassOf&gt; &lt;owl:Restriction&gt; &lt;owl:onProperty rdf:resource="#HasB" /&gt; &lt;owl:maxCardinality rdf:datatype="http://www.w3.org/2001 /XMLSchema#nonNegativeInteger"&gt; 1 &lt;/owl:maxCardinality&gt; &lt;owl:Restriction&gt; &lt;/rdfs:subClassOf&gt;</pre>

An Optional feature is represented as a max cardinality restriction that specifies the maximum number of instances that can be connected to a property in a given class.

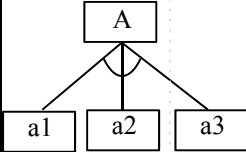
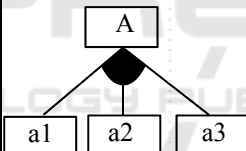
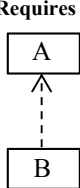
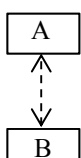
<p><b>Alternative</b></p> 	<pre>&lt;rdfs:subClassOf&gt; &lt;owl:Class&gt; &lt;owl:complementOf rdf:resource="#[a1]"/&gt; &lt;/owl:Class&gt; &lt;/rdfs:subClassOf&gt; &lt;rdfs:subClassOf&gt; &lt;owl:Class&gt; &lt;owl:complementOf rdf:resource="#[a2]"/&gt; &lt;/owl:Class&gt; &lt;/rdfs:subClassOf&gt; &lt;rdfs:subClassOf&gt; &lt;owl:Class&gt; &lt;owl:complementOf rdf:resource="#[a3]"/&gt; &lt;/owl:Class&gt; &lt;/rdfs:subClassOf&gt;</pre>
<p>An alternative feature is transformed to a complement of.</p>	
<p><b>Or</b></p> 	<pre>&lt;owl:Class rdf:about="A"&gt; &lt;owl:unionOf rdf:parseType="Collection"&gt; &lt;owl:Class rdf:about="#a1" /&gt; &lt;owl:Class rdf:about="#a2" /&gt; &lt;owl:Class rdf:about="#a3" /&gt; &lt;/owl:unionOf&gt; &lt;/owl:Class&gt; &lt;owl:Class rdf:about="a1"&gt; &lt;owl:disjointWith rdf:resource="#[a2]"/&gt; &lt;owl:disjointWith rdf:resource="#[a3]"/&gt; &lt;/owl:Class&gt;</pre>
<p>"unionOf" construct is utilized within the context of a parent feature connected through an "or" relation. For each individual sub-feature of this parent, a specification is made to ensure its disjointness from the other sub-features.</p>	
<p><b>Requires</b></p> 	<pre>&lt;owl:Class rdf:about="A"&gt; &lt;rdfs:subClassOf&gt; &lt;owl:Restriction&gt; &lt;owl:onProperty rdf:resource="#HasB"/&gt; &lt;owl:allValuesFrom rdf:resource="#B" /&gt; &lt;/owl:Restriction&gt; &lt;/rdfs:subClassOf&gt;</pre>
<p>Required feature is represented by AllValuesFrom restriction over hasB.</p>	

Table 1: The proposed transformation rules. (cont.)

<p><b>Excludes</b></p> 	<pre> &lt;rdfs:subClassOf&gt; &lt;owl:Class&gt; &lt;owl:complementOf&gt; &lt;owl:Class&gt; &lt;owl:intersectionOf rdf:parseType="Collection"&gt; &lt;owl:Restriction&gt; &lt;owl:onProperty rdf:resource="#Has[A]"/&gt; &lt;owl:allValuesFrom rdf:resource="#[A]"/&gt; &lt;/owl:Restriction&gt; &lt;owl:Restriction&gt; &lt;owl:onProperty rdf:resource="#Has[B]"/&gt; &lt;owl:allValuesFrom rdf:resource="#[B]"/&gt; &lt;/owl:Restriction&gt; &lt;/owl:intersectionOf&gt; &lt;/owl:Class&gt; &lt;owl:complementOf&gt; &lt;owl:Class&gt; &lt;/rdfs:subClassOf&gt;                 </pre>
<p>Exclude class relationship is transformed to a class that's the opposite of the combined intersection of two subclasses, indicating exclusion based on certain feature connections.</p>	

### 3.2 Automating the Conversion of FM into OWL

The approach proposed is to use the techniques of MDE technology to build a tool that transforms automatically Feature models into their corresponding OWL ontologies. Figure 1 shows the process of this mapping which consists on several steps:

- **Step 1:** We initially proposed a meta-model for the Feature models. We used Ecore as the meta-modeling language, which is part of the Eclipse Modeling Framework (EMF).
- **Step 2:** We utilize EMF to generate a hierarchical editor for the proposed Feature models. This editor enables us to describe our Feature models easily.
- **Step 3:** This process is a *Model2Text* type transformation. The objective here is to generate an OWL ontology from the Feature model established in the previous phase. For this phase, we used Acceleo as the transformation language.
- **Step 4:** this step consists of uploading the generated OWL specification into the Protege tool. Once the description is accepted, we can use all the services offered by this tool, especially simulation, verification, etc.

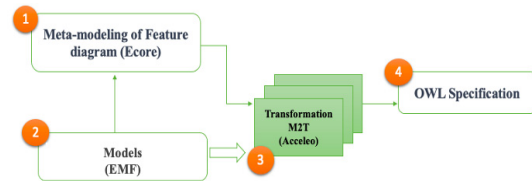


Figure 1: General framework of the proposed approach.

#### Step 1: Meta-Modeling

We have proposed a meta-model for Feature models. This meta-model allows for the creation of valid models conform to this type of diagram. Figure 2 illustrates the structure of this meta-model.

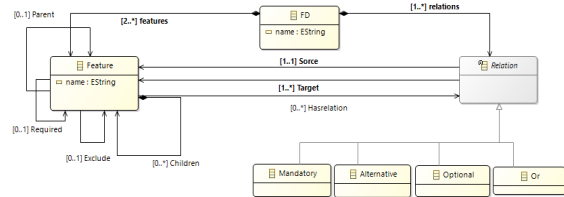


Figure 2: The meta-model of Feature diagram.

The proposed meta-model is composed of classes and relations. We have proposed the following classes for metamodeling Feature diagrams:

- *FeatureDiag*: This class represents a Feature diagram. It has a key attribute named "name".
- *Feature*: This class represents the main element of the diagram. It has a key attribute "name".
- *Relation*: This class represents the relations between features. It has four types *Mandatory*, *Optional*, *Or* and *Alternative*.
  - The "Or" relationship between a group of sub-features and the parent feature indicates that one or more features from the group can be selected in a product when the parent feature is selected.
  - The "Alternative" relationship between a parent feature and a group of sub-features indicates that only one sub-feature can be selected in a product when the parent feature is selected.
  - The "Optional" relationship between a parent feature and a sub-feature indicates that the sub-feature can be selected in a product or not when the parent feature is selected.
  - The "Mandatory" relationship between a parent feature and its sub-feature indicates that the presence of the parent feature necessitates the presence of the sub-feature in the product.

**Step 2: Generation of the Modeling Environment**

In this step, we generate a hierarchical editor for the defined meta-model using EMF as indicated in Figure 3. This editor is used to describe different Feature Models.

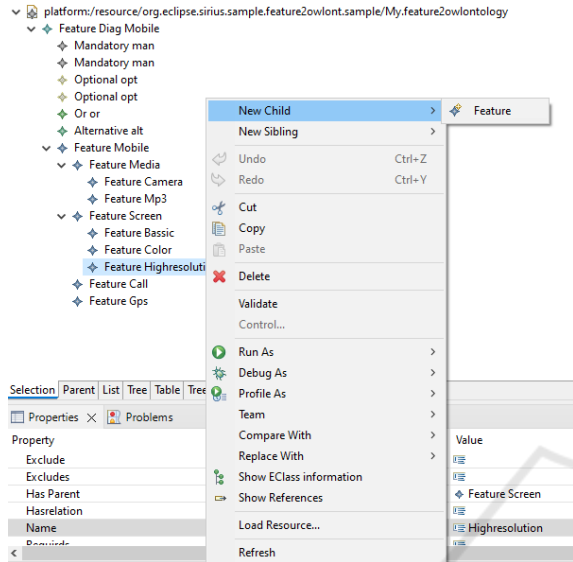


Figure 3: The generated editor for Feature models.

**Step 3: Transformation**

In this step, we implement the transformation rules proposed above. The automated process generates the "generatedFM.owl" file, including an OWL language ontology represented in RDF/XML format. This file is stored on hard drive, encapsulating structured knowledge comprising concepts, entities, and relationships from the OWL ontology. The Aceleo code is used to realize the mapping according to the rules in Table 1.

Due to space constraints, we only present a screenshot that illustrates the transformation.

Figure 4 depicts the transformation process using Aceleo of a feature of an FD into an OWL class, with special attention to exclusion constraints. Each feature is represented as an OWL class. In situations where a feature is subject to an exclusion constraint, it will be transformed into the negation intersection of classes.

The other rules in Table 1 are implemented similarly using templates in Aceleo. This latter has really helped us in realizing the correspondences between FMs meta-model and OWL ontology.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns="http://www.example.org/feature2owlontology"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  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#"
  [for ( F : Feature| aFeatureDiag_eAllContents(Feature)) separator('\n')]
  <owl:Class rdf:ID="[F.name]"/>
  [if (not F.hasParent.oclisUndefined())
  <rdfs:subClassOf rdf:resource="#[F.hasParent.name]"/>
  [/if]
  [if ( F.hasParent.oclisUndefined())
  [for (Feat : Feature| aFeatureDiag_eAllContents(Feature)) separator('\n')]
  [if (not Feat.Exclude.oclisUndefined())
  <rdfs:subClassOf>
  <owl:Class>
  <owl:complementOf>
  <owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Restriction>
  <owl:onProperty rdf:resource="#Has[Feat.Exclude.name]"/>
  <owl:someValuesFrom rdf:resource="#[Feat.Exclude.name]"/>
  </owl:Restriction>
  <owl:Restriction>
  <owl:onProperty rdf:resource="#Has[Feat.name]"/>
  <owl:someValuesFrom rdf:resource="#[Feat.name]"/>
  </owl:Restriction>
  </owl:intersectionOf>
  </owl:Class>
  </owl:complementOf>
  </owl:Class>
  </rdfs:subClassOf>
  [/if]
  [/for]
  ]
```

Figure 4: Extract of transformation code of a Feature model into OWL part 1.

**Step 4:** The derived OWL specification is automatically launched in Protégé to do analysis. Section 4.2 explains more this step.

**4 EXAMPLE**

**4.1 Mobile Phone System Description**

The mobile phone system outlined here is a well-known example frequently referenced in SPL literature. It is a flexible system that enables users to select various features according to their preferences. This system comprises multiple features and sub-features. The feature model for this mobile phone system is organized as follows (see Figure 5):

1. Root Feature: Mobile Phone - This feature represents the mobile phone itself, it is used as the foundation for all other features.
2. Sub-Feature: Calls (mandatory) - This represents essential functionalities related to calls like making and receiving phone calls.
3. Sub-Feature: Screen (mandatory) - Related to the display of the mobile phone, providing different screen options.
  - Sub-Feature: Basic - Represents a basic screen option.
  - Sub-Feature: Color - Represents a color screen option.

- Sub-Feature: High Resolution - Represents a high-resolution screen option.

Note: Following the alternative type relationship, only one screen option can be selected.

4. Sub-Feature: GPS (optional) - Represents the functionality of the Global Positioning System of the mobile phone.

5. Sub-Feature: Media (optional) - Represents multimedia features of the mobile phone.

- Sub-Feature: Camera - Represents the functionality of camera.
- Sub-Feature: MP3 Player - Represents the functionality of MP3 music player.

Note: Following the or-type relationship, if the Media sub-feature is selected, either one or both options (Camera and/or MP3 Player) can be selected.

The feature model has also two constraints:

- GPS excludes Basic - This means that a mobile phone cannot have both GPS and a basic screen.
- Camera requires High Resolution - This indicates that a mobile phone with a camera must also have a high-resolution screen.

To ensure a valid configuration, these constraints must be respected when selecting specific features. By adhering to the constraints and choosing the appropriate features, various configurations of mobile phones can be developed to meet the needs of users.

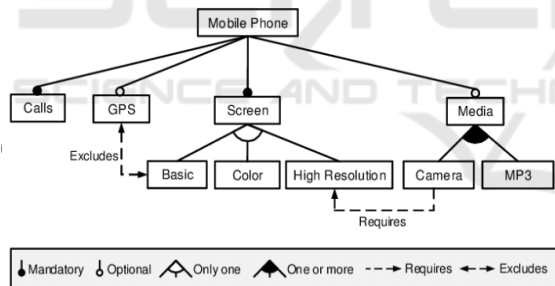


Figure 5: The Mobile Phone case study.

Using our approach, we specify this Feature model in our tool as shown in Figure 6.

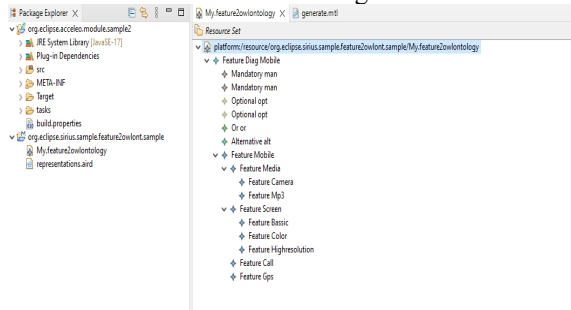


Figure 6: Feature model of the mobile phone system.

Afterwards, we use our Acceleo code "My.feature2owlontology" to generate the corresponding OWL ontology "owlontology" as illustrated in Figure 7.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns="http://www.example.org/feature2owlontology"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2001/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  <owl:Class rdf:ID="Mobile">
    <rdfs:subClassOf>
      <owl:Class>
        <owl:complementOf>
          <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
              <owl:Restriction>
                <owl:onProperty rdf:resource="#hasGps"/>
                <owl:someValuesFrom rdf:resource="#Gps" />
              </owl:Restriction>
              <owl:Restriction>
                <owl:onProperty rdf:resource="#hasBasic"/>
                <owl:someValuesFrom rdf:resource="#Basic"/>
              </owl:Restriction>
            </owl:intersectionOf>
          </owl:Class>
        </owl:complementOf>
      </owl:Class>
    </rdfs:subClassOf>
    <owl:equivalentClass>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasScreen"/>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </owl:equivalentClass>
  </owl:Class>
  <owl:equivalentClass>
    <owl:equivalentClass>
```

Figure 7: An extract of the generated OWL ontology.

## 4.2 Analysis of the Ontology

We consider here two illustrative configurations. The first configuration adheres to all the constraints outlined in the feature model showed in the previous diagram. In contrast, the second configuration disregards the alternative relationship, deviating from the established constraints within the model.

C1 = {Mobile Phone, Calls, Screen, High Resolution, Media, Camera, MP3}

C2 = {Mobile Phone, Calls, Screen, Basic, High Resolution, Media, Camera}

For the first configuration, Figure 8 show the lack of response of the reasoner due to the absence of inconsistencies.

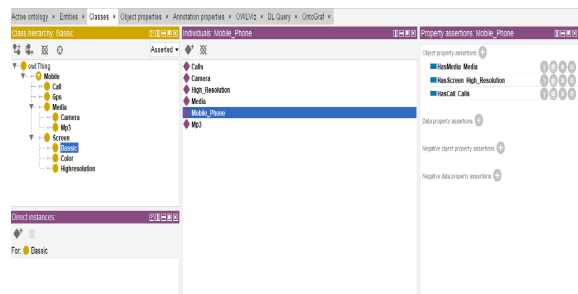


Figure 8: First configuration absence of inconsistencies.

In contrast, the second configuration, Figure 9 clearly illustrate how the responsiveness of reasoner is affected by the presence of inconsistencies.

The inconsistency in the reasoning process is attributed to the exclusion rule that was not respected. Specifically, in this scenario, the "high-resolution" class was intended to exclude the "basic" class, indicating that a device cannot have both high-resolution and basic classes simultaneously. However, in practice, both classes were assigned to the same device, leading to a violation of this exclusion rule.

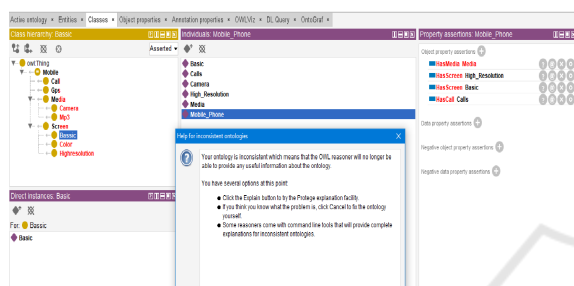


Figure 9: Second configuration inconsistencies.

## 5 RELATED WORK

Several studies in the literature have focused on modeling with FMs and formalizing them for verification purpose. This section reviews essential research contributions.

The authors in (Benavides, 2005a) have proposed a theoretical framework for reasoning on Feature Models. They extended FMs to capture some non-functional properties and they use Constraint Satisfaction Problems (CSP) to reason on them in (Benavides, 2005b). In (Trinidad, 2023), the same authors improve their work by proposing an automation of the framework to analyzing stateful feature models. The main idea behind the work is that they viewed the issue as an automated reasoning problem solvable through CSP. Using CSP is great, but it needs to add more information to enable correct analysis operations, which become computationally expensive and even impossible for complex models. In (Zhang, 2013), the authors proposed not to use CSP because of exponential complexity and instead use contradictory feature relationships behind the errors. In (Karatas, 2013), the authors introduce a mapping from extended feature models to constraint logic programming over finite domains.

In (Santoro, 2012), an approach has been proposed to construct and manage consistent feature

models, crucial for managing valid software product lines using OWL. The approach is specific to multimedia domain. The research in (Duran-Limon, 2015) introduces OntoAD, a new system for automatically creating product architectures from a base design. Unlike existing methods focused on features, OntoAD tackles the challenge of automating architecture customization. It uses clever reasoning to generate transformation rules based on selected features, in the purpose of simplifying the process and reduce errors. The study in (Bhushan, 2018) proposes a method to find and explain errors (inconsistencies) in software product designs (feature models). While other methods can detect these errors, they often don't explain why they happen. The authors' approach translates feature models into a FOL predicate-based ontology and uses rules to pinpoint both the errors and their causes in clear language. The authors in (Wang, 2007) present an approach to modeling and verifying feature diagrams using OWL ontologies and the FaCT++ reasoner, however they do not provide automated support which hinders their real-word utilization.

Other works can be cited since they tried to use MDE to automatic generation of OWL ontologies from conceptual models for the purpose of reasoning, such as the work done in (Belghiat and Bourahla, 2012).

In contrast to these contributions, we have aimed to develop a complete framework based on model driven engineering to transform automatically FMs to OWL ontologies and launching the verification process immediately. The MDE allows advanced complex semantics transformations based on meta-models. In our work, we have focused on features and their relationships, and we have used OWL DL to allow automatic analysis while maintaining decidability using the well-known Protégé tool.

## 6 CONCLUSION

In this paper, we have proposed a model-driven based approach for transforming Feature Models into their corresponding OWL ontologies. We have started by proposing a meta-model for FMs to generate an adequate environment for their modeling. Next, we have formulated a set of transformation rules. These rules are designed to bridge the gap between FMs meta-model and OWL ontologies. Then, we have implemented these transformation rules to enable automatic generation of an output OWL ontology from an input valid Feature model. To realize the mapping, we have used a bench of tools. EMF and

Ecore have been used in meta-modeling FMs, and Aceleo to generate the corresponding OWL code in RDF/XML format.

To guarantee the quality and consistency of the generated ontologies, we thoroughly checked them using the Protégé tool. This step is very important to ensure that these OWL ontologies are accurate and coherent. Furthermore, Protégé is used for more advanced reasoning on those ontologies.

In future work, we plan to expand our transformation capabilities to cover more advanced aspects of Feature models, for example to include non-functional properties. This expansion will enable us to tackle more complex challenges, enhancing the utility of our approach.

## REFERENCES

- Aceleo. <https://eclipse.dev/aceleo/>. Visited 09-2023.
- Acher, M., Baudry, B., Heymans, P., Cleve, A., Hainaut, J. L. (2013). Support for reverse engineering and maintaining feature models. In *VaMoS'13 Proceedings of the seventh international workshop on variability modelling of software-intensive systems*.
- Batory, D. (2005). Feature models, grammars, and propositional formulas. In *SPLC'05 Proceedings of the 9th international conference on software product lines*, Springer, pp.7–20.
- Belghiat, A., & Bourahla, M. (2012, March). Transformation of UML models towards OWL ontologies. In 2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT) (pp. 840-846). IEEE.
- Belghiat, A., & Bourahla, M. (2012). An approach based AToM3 for the generation of OWL ontologies from UML diagrams. *International journal of computer applications*, 41(3).
- Belghiat, A., & Bourahla, M. (2012). From UML Class Diagrams to OWL Ontologies: A Graph Transformation Based Approach. In *ICWIT* (pp. 330-335).
- Benavides, D., Trinidad, P., Ruiz-Cortés, A. (2005). Automated reasoning on feature models. In *International Conference on Advanced Information Systems Engineering* (pp. 491-503). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Benavides, D., Trinidad, P., Ruiz-Cortés, A. (2005). Using Constraint Programming to Reason on Feature Models. In *SEKE* (Vol. 5, pp. 677-682).
- Benavides, D. (2007). *On the automated analysis of software product lines using feature models: A framework for developing automated tool support*. PhD thesis, Universidad de Sevilla.
- Bhushan, M., Goel, S., Kaur, K. (2018). Analyzing inconsistencies in software product lines using an ontological rule-based approach. *Journal of Systems and Software*, 137, 605-617.
- Budinsky, F. (2004). *Eclipse modeling framework: a developer's guide*, Addison-Wesley Professional.
- Clements, P., Northrop, L. (2002). *Software product lines*. Addison-Wesley. Boston.
- Duran-Limon, H. A., Garcia-Rios, C. A., Castillo-Barrera, F. E., Capilla, R. (2015). An ontology-based product architecture derivation approach. *IEEE Transactions on Software Engineering*, 41(12), 1153-1168.
- Eclipse Modeling Project, <https://eclipse.dev/modeling/>. Visited 04-2024.
- Ghabach, E. (2018). *Supporting Clone-and-Own in software product line*. PhD thesis, COMUE Université Côte d'Azur.
- Kang, C. K., Cohen, S., Hess, J. A., Novak, W. E., Peterson, A. S. (1990). Feature Oriented Domain Analysis (FODA) Feasibility Study, *Software engineering institute*.CMU/SEI-90-TR-021.
- Karatas, A. S., Oğuztüzün, H., Doğru, A. From extended feature models to constraint logic programming. *Science of Computer Programming*, vol. 78, no. 12, pp. 2295–2312.
- Protégé. <https://protege.stanford.edu/>. Visited 09-2023.
- Rubin, J., Chechik, M. (2012). Combining related products into product lines. In *FASE'12 Proceedings of the 15th international conference on fundamental approaches to software engineering*, ACM, pp.285–300.
- Santoro, G., Pino, C., Spampinato, C. (2012). A feature model configuration for multimedia applications by an OWL-based approach. In *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)* (pp. 263-268). IEEE.
- Schmidt, D. C. (2006). Model-driven engineering. *Computer-IEEE Computer Society*, 39(2), 25.
- She, S., Lotufo, R., Berger, T., Wąsowski, A., Czarnecki, K. (2011). Reverse engineering feature models. In *ICSE'11 Proceedings of the 33<sup>rd</sup> international conference on software engineering*, ACM, pp.461–470.
- Trinidad, P., Ruiz-Cortés, A., Benavides, D. (2013). Automated analysis of stateful feature models. *Seminal Contributions to Information Systems Engineering: 25 Years of CAiSE*, 375-380.
- Van der Linden, F. (2002). Software product families in europe: the esaps & café projects. *IEEE software*, vol. 19, no. 4, pp. 41–49.
- Wang, H. H., Li, Y. F., Sun, J., Zhang, H., Pan, J. (2007). Verifying feature models using owl, *Journal of web semantics*, vol. 5, no. 2, pp. 117–129.
- W3C OWL Working Group. (2012). *OWL 2 Web ontology language document overview*, W3C Recommendation 11 December, <https://www.w3.org/TR/owl2-overview/>, 2<sup>nd</sup> edition.
- Zhang, G., Ye, H., Lin, Y. (2013). An approach for validating feature models in software product lines. *Journal of Software Engineering*, 7(1), 1-29.