

Decision Support for the Software Product Line Domain Engineering Lifecycle

EBRAHIM BAGHERI, FAEZEH ENSAN AND DRAGAN GASEVIC

Abstract. Software product line engineering is a paradigm that advocates the reusability of software engineering assets and the rapid development of new applications for a target domain. These objectives are achieved by capturing the commonalities and variabilities between the applications of the target domain and through the development of comprehensive and variability-covering feature models. The feature models developed within the software product line development process need to cover the relevant features and aspects of the target domain. In other words, the feature models should be elaborate representations of the feature space of that domain. Given that feature models, i.e., software product line feature models, are developed mostly by domain analysts by sifting through domain documentation, corporate records and transcribed interviews, the process is a cumbersome and error-prone one. In this paper, we propose a decision support platform that assists domain analysts throughout the domain engineering lifecycle by: 1) automatically performing natural language processing tasks over domain documents and identifying important information for the domain analysts such as the features and integrity constraints that exist in the domain documents; 2) providing a collaboration platform around the domain documents such that multiple domain analysts can collaborate with each other during the process using a Wiki; 3) formulating semantic links between domain terminology with external widely used ontologies such as WordNet in order to disambiguate the terms used in domain documents; and 4) developing traceability links between the unstructured information available in the domain documents and their formal counterparts within the formal feature model representations. Results obtained from our controlled experimentations show that the decision support platform is effective in increasing the performance of the domain analysts during the domain engineering lifecycle in terms of both the coverage and accuracy measures.

Keywords: Software Product Lines, Feature Models, Domain Engineering, NLP Model Inference

1. Introduction

Software product line engineering is a systematic software reusability approach that is based on the idea of explicitly capturing the commonalities and variabilities of the applications of a target domain within a unified domain representation [1]. Such an approach to domain modeling allows domain analysts to have a holistic view of the target domain and to be able to understand the similarities and differences that exist between the different applications of the same domain. One of the major advantages of product line engineering is that new products and applications can be rapidly developed by exploiting the available commonalities among them and other applications within the same family; hence, reducing time-to-market and development costs [2]. Many large industrial corporations have already successfully adopted the software product line engineering approach. For instance, Nokia was able to increase its production capacity for new cellular phone models from 5-10 to around 30 models per year by using product line engineering techniques [3, 4].

The software product line engineering paradigm consists of two major development lifecycles, namely *Domain Engineering* and *Application Engineering* [5]. The development of a software product line often starts with the domain engineering phase, through which the target domain is carefully analyzed, understood, and formally documented. One of the main artifact developed as a result of the domain engineering phase is a formal representation of the various aspects and features of the applications of the target domain. In other words, the domain engineering phase will need to consider and capture the inherent variabilities and commonalities, which exist between the applications of a family of products that are available in the same target domain, within a unique formal representation. Such domain models can be viewed as the *feature space* of the target domain, i.e., the relevant features that can be later incorporated into domain-specific applications related to that domain should have presence in the model.

The subsequent application engineering phase is undertaken with the explicit assumption that a domain model has been already developed within the domain engineering phase. The role of application engineering is to develop new domain-specific applications. For this reason, application engineers will constantly interact with end-users and stakeholders to identify their most desirable set of features to be incorporated into the final product. This set of desirable features are determined in light of the available features in the domain model. The feature space can be gradually restricted, using a process which is referred to as *staged configuration* [6], by selecting the necessary features and excluding the undesirable ones for the final product. The final artifact of the application engineering phase is a concrete representation of a domain-specific application built from the formal domain model.

The process for developing new applications within the application engineering lifecycle shows that the completeness and suitability of the products of a software product line rely significantly on the quality and comprehensiveness of the domain models [7]. The formal domain models developed in the domain engineering lifecycle are in most cases produced from the labor-intensive process of engaging with domain experts, reading corporate documentations, conducting interviews and processing interview transcriptions. As the complexity and size of the domain grows, the number of documents that need to be processed and taken into account also radically increases. This demands more time dedication and perseverance from the domain analysts.

Stenz and Ferson [8] have argued that *epistemic uncertainty* is an indispensable element of human judgments and that it increases as the cognitive and problem complexities grow. Therefore, as we have also observed in our controlled experiments, the growth of the problem domain, in terms of size and complexity, affects the quality of work and performance of the domain analysts [9]. This phenomenon has been shown to be independent of the available time at the disposal of the human experts. In other words, the increase in the *cognitive complexity* of the task assigned to the human experts impacts their performance even if enough time is provided to them [10]. It is important that suitable support mechanisms are developed that would assist the domain analysts in a way that their quality of work does not deteriorate with the increase in the complexity and size of the target domain.

In order to achieve this goal, we have developed a decision support platform to help the analysts throughout the domain engineering lifecycle. The decision support platform is mainly an structured information extraction tool that identifies important pieces of information from domain documents and as such can also be considered as a targeted information extraction tool for domain engineering. The main reason we refer to it as decision support is that it highlights information from the text and provides additional external information such as visualization and ontological representation that helps the domain analysts to make a better decision. The objective of our work is not to take over the domain modeling process from the domain analysts, but rather to act as a support system that would allow the analysts to make more informed and justified modeling decisions. From our point of view, given size and cognitive complexities and also the involvement of multiple analysts in a domain engineering process, the main characteristics that we aimed to provide through the decision support platform are the following:

- *Diligence*: As discussed above, the attentiveness and performance of human analysts decreases with the increase of the complexity and size of a problem. Our intention is to help the domain analysts maintain high quality standards in their work by reducing the *mental burden* that they have to carry during the modeling process;
- *Collaboration*: Large domains often cross multiple disciplines and require different domain experts to effectively communicate with each other during the domain modeling process. Previous research has shown that ad-hoc collaboration strategies can lead to inconsistent models [11]; hence, our goal is to provide a platform that supports for easy and effective collaboration between the domain analysts;
- *Knowledge Transfer*: Software product lines are often designed for the purpose of being used during long periods of time and for multiple generations of software applications; therefore, it is likely that the original domain analysts will no longer be available at a later stage of a project to explain and justify their modeling decisions. For this reason, the decision support platform needs to unambiguously capture the decisions of the domain modelers and efficiently document them for future reference;
- *Traceability*: As a domain evolves, the documents that were used earlier during the domain modeling process may gradually become obsolete. It is important for the domain modelers to be able to trace back the elements of their formal domain model representation to the sources from which they have originated. For our case, it is desirable that the domain analysts be able to identify the parts of the documents from which certain features of the domain model were extracted.

In this paper, we concentrate on *feature models*, which are a widely used graphical domain modeling notation that support variability and commonality representation. Our decision support platform consists of four main functionalities that

contribute to the aforementioned characteristics: 1) it employs natural language processing techniques to extract important feature modeling concepts from domain documentation and offers an annotated and visualized version of the documents to the domain analysts; hence, it aims to reduce the mental burden of manually processing large domain documents; 2) it cross-references the extracted important concepts with an external ontology, i.e., WordNet and provides a semantic orientation for the concepts in the document and hence reduces ambiguity; 3) it provides integration capabilities for the migration of annotated domain documents to an extended version of the MediaWiki system, so allowing for collaborative interaction between the domain analysts; and 4) it facilitates the correspondence of domain features extracted from the documents to the actual formal domain model elements represented as feature models (within our AUFM feature modeling suite); therefore, supports for traceability. We report our controlled experiments performed on various domains, which show that the decision support is effective in enhancing the performance of the domain analysts during the domain engineering lifecycle.

The rest of this paper is organized as follows: Software product lines and its related conceptual modeling formalism, feature modeling, is introduced in Section 2. Section 3 describes the overall picture of the proposed decision support platform. The details of the work are discussed in Section 4, which is followed by a discussion on methodology integration. The tooling support for our work is explained in Section 6. The setup and the structure of the experiments is presented in Section 7, which is followed by the explanation of the obtained results in Section 8. Before concluding the paper, a discussion including the threats to validity and the lessons learnt, and also a review of the related work are presented.

2. Software Product Line Concepts

Software product line engineering is concerned with capturing the commonalities, universal and shared attributes of a set of software-intensive applications for a specific problem domain [1]. It allows for the rapid development of variants of a domain specific application through various configurations of a common set of reusable assets often known as *core assets*, which support the management of commonality as well as variability. On the one hand, *commonality* is supported by providing domain analysts with both the ability and the required tools for capturing the shared conceptual information within the applications of a given domain. On the other hand, *variability* is addressed by allowing the domain analysts to include application-specific attributes and features within their unified model.

In the context of software product lines, *feature modeling* techniques are amongst the important techniques for performing domain modeling in the domain engineering lifecycle [2, 12]. Feature modeling is important in that it provides means for capturing variability in software product lines [13]. Given that feature models are becoming the *de facto* standard for domain modeling in software product lines, we will base our work on this formalization in this paper.

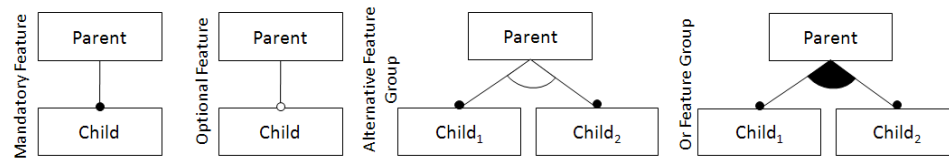


Figure 1. The main graphical notations employed in feature modeling.

2.1. Feature Models

Features are important distinguishing aspects, qualities, or characteristics of a family of systems [2, 14]. They are used for depicting the shared structure and behavior of a set of similar systems. To form a product line, the features of a set of similar/related systems are composed into a feature model. A feature model is a means for representing the possible configuration space of all the products of a system product line in terms of its features. For this reason, it is important that feature models be able to capture variability and commonality among the features of the different applications available in a given domain. In other words, feature modeling is a domain modeling approach that captures the variability and commonality among the features of the applications of a given target domain. As we will see in the following paragraphs, feature models provide suitable means for modeling commonality, by allowing the domain modelers to form a common feature model representation for multiple applications. It also captures variability by providing means to model competing features of different applications under one unified umbrella. An example of capturing commonality is when a similar feature, which exists in multiple applications is represented as a unique feature in the overall domain representation, while an example of variability is when one notion is viewed differently by separate applications and is therefore modeled using competing features.

Feature models can be represented both formally and graphically; however, the graphical notation depicted through a tree structure is more favored due to its visual appeal and easier understanding. More specifically, graphical feature models are in the form of a tree whose root node represents a domain concept, e.g., a domain application, and the other nodes and leaves illustrate the features. In this context, a feature is a concept property related to a user-visible functional or non-functional requirement modeled in a way to capture commonalities or possibly differentiate between product line variants [15].

In a feature model, features are hierarchically organized and can typically be classified as:

- *Mandatory*, the feature must be included in the description of its parent feature. This type of features is used for representing commonality;
- *Optional*, the feature may or may not be included in its parent description given the situation;

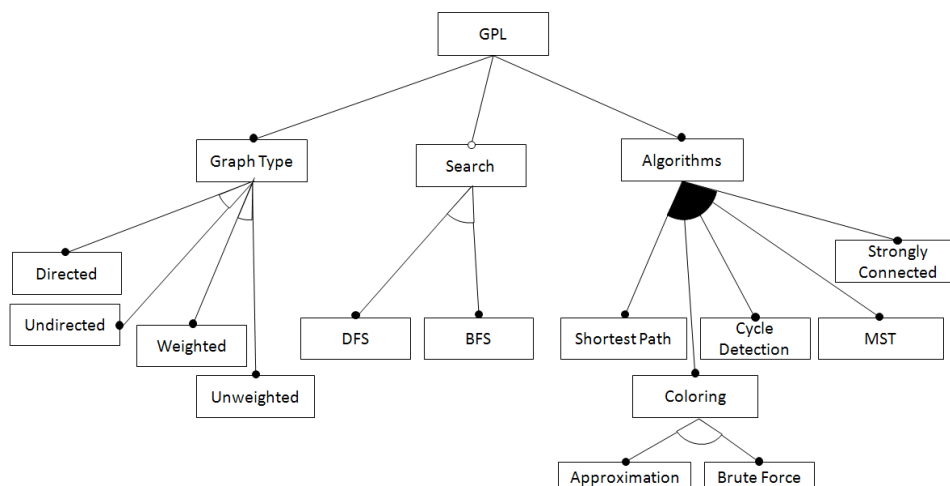


Figure 2. The graph product line feature model.

- *Alternative feature group*, one and only one of the features from the feature group can be included in the parent description;
- *Or feature group*, one or more features from the feature group can be included in the description of the parent feature.

Figure 1 depicts the graphical notation of the feature relationships. The tree structure of feature models falls short at fully representing the complete set of mutual interdependencies of features; therefore, additional constraints are often added to feature models and are referred to as *Integrity Constraints (IC)*. Two types of integrity constraints are:

- *Includes*, the presence of a given feature (set of features) requires the *existence* of another feature (set of features);
- *Excludes*, the presence of a given feature (set of features) requires the *elimination* of another feature (set of features).

2.2. The Graph Product Line Feature Model

Let us now further explain feature modeling concepts using the Graph Product Line (GPL) [16], which is designed with the ambition to be a standard benchmark for evaluating feature modeling techniques shown in Figure 2. The intention of GPL is to provide the capability to develop configurations that address different problems in the domain of graph manipulation. For instance, GPL can be configured to perform several graph search algorithms over a directed or undirected graph structure.

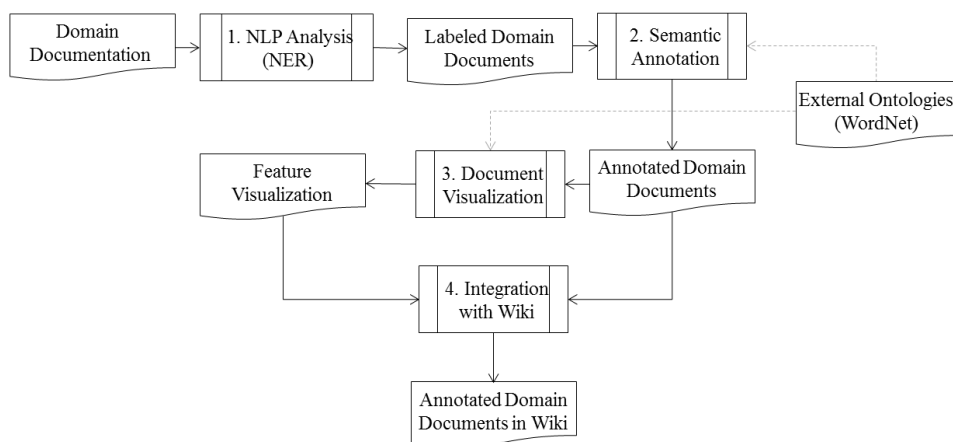


Figure 3. Overview of our approach.

As it can be seen, GPL consists of three main features: 1) Graph Type: features for defining the structural representation of a graph; 2) Search: traversal algorithms in the form of features that allow for the navigation of a graph; 3) Algorithms: other useful algorithms that manipulate or analyze a given graph. Clearly, not all possible configurations of the features of GPL produce valid graph programs. For instance, a configuration of GPL that checks if a graph is strongly connected cannot be implemented on an undirected graph structure. Such restrictions are expressed in the form of integrity constraints. Some examples of these constraints are:

$$IC = \begin{cases} \text{Cycle Detection EXCLUDES BFS (Breadth-First Search);} \\ \text{Cycle Detection INCLUDES DFS (Depth-First Search);} \\ \text{Strongly Connected INCLUDES DFS;} \\ \text{Strongly Connected INCLUDES Directed;} \end{cases}$$

These integrity constraints ensure the correct composition of features in the various final software products developed from this feature model.

3. Approach Overview

The overview of our approach is shown in Figure 3. Essentially, our work focuses on performing additional useful processing over domain documents in such a way that they facilitate the tasks of the domain analysts during the domain engineering lifecycle. The main attention of our work is given to the feature model design process within domain engineering. Therefore, the decision support platform exclusively supports the design of a feature model. However, as will be discussed in subsequent sections, it has clear benefits to the other steps of the domain engineering process

as well. Let us walk through our proposed approach based on a short scenario. We will discuss the technical details of each step in the next section.

Assume that a domain engineer is tasked to create a feature model that represents all of the possible applications related to graph processing operations (Figure 2). He/she is provided with the following short textual description of the graph processing systems domain (in reality, domain documents are much longer than this short example):

A graph is an abstract mathematical representation that is used for exemplifying inter-entity relationships. Graphs can have different types. They can be directed or undirected; weighted or unweighted. A graph cannot be both directed and undirected. A graph can either be weighted or unweighted. One of the most desirable operations on a graph is the search operation. Graphs can be searched using depth-first or breadth-first strategies...

A domain analyst will need to carefully read through this domain description and look for and identify two pieces of important information in order to be able to develop the formal domain representation in terms of the feature model: *the features* and *the integrity constraints*. Our decision support platform assists the domain analysts in this process:

Step 1. NLP Analysis (NER): The first step of our approach employs natural language processing techniques to identify potential features and integrity constraints in the domain document. This step will create a labeled version of the domain description document in which the identified features and integrity constraints are highlighted to be considered by the domain analyst. For instance, the graph description text is automatically labeled as follows:

A graph is an abstract mathematical representation that is used for exemplifying inter-entity relationships. Graphs can have different types. They can be directed or undirected; weighted or unweighted. A graph cannot be both directed and undirected. A graph can either be weighted or unweighted. One of the most desirable operations on a graph is the search operation. Graphs can be searched using depth-first or breadth-first strategies...

In the text above, text labeled with f and ic are feature and integrity constraint recommendations, respectively. The domain analysts can review the labeled suggestions and decide on their suitability, i.e., accept or reject the recommendations.

Step 2. Semantic Annotation¹: Once the features and integrity constraints are identified, the decision support platform will perform a cross-referencing process with an external ontology/terminology of the analysts' choice (in our experiments WordNet was used). This way, the expressions inside the domain documents will be interrelated with concepts from a widely used and well understood source. An example of a cross-reference is:

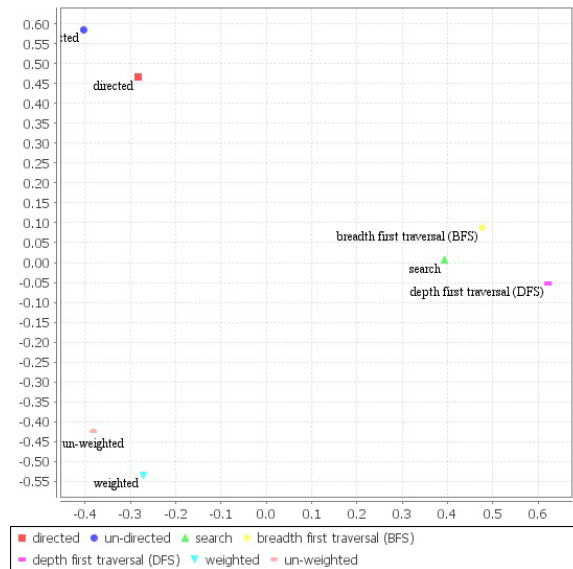


Figure 4. A sample feature distribution visualization.

[...] One of the most desirable operations on a graph is the search [\sim /wn-ns/
 f
synset-search-noun-4] operation.

Here, the identified feature is cross-referenced with the concept definitions provided in WordNet and a reference to the most appropriate match is found and provided. In this example, the reference to the most relevant WordNet concept is \sim /wn-ns/synset-search-noun-4, which is attached to the search feature. This can be extremely helpful for meaning clarification in domains where many unfamiliar terms are frequently employed.

Step 3. Document Visualization: One of the important tasks of the domain analysts is to find the relationships between the identified features and try to create the hierarchical tree structure of a feature model. To facilitate this process, the decision support platform employs the semantic annotations of the domain documents and creates a feature distribution graph visualization of the identified features. Within this visualization, the extracted features are distributed on a 2-dimensional space in such a way that the features that are similar to each other are placed closer to each other on the graph while the less relevant features to each other are placed as far as possible. The distribution of the features on this graph can help the domain analysts identify the most related features to each other and assist the domain analysts in the design of the feature model. For instance, Figure 4 shows a sample graph. In this graph, the Search, DFS, BFS features that are similar to each other are placed next to one another and further away from other less relevant features.

Table 1. Details of some extracted information for a Blackberry device.

Processed Text	Any application that makes use of certain restricted functionality must be digitally signed so that it can be associated to a developer account at RIM. This signing procedure guarantees the authorship of an application but does not guarantee the quality or security of the code. RIM provides tools for developing applications and themes for BlackBerry. Applications and themes can be loaded onto BlackBerry devices through either BlackBerry App World, Over The Air through the BlackBerry mobile browser, or through BlackBerry Desktop Manager.	
Extracted Features	Feature	WordNet Concept
	1. Digitally signed	S: (n) endorsement, indorsement (a signature that validates something)
	2. Tools for developing	S: (n) instrument (the means whereby some act is accomplished)
	3. BlackBerry App World	Proper noun
	4. BlackBerry mobile browser	Proper noun
Extracted Constraints	5. BlackBerry Desktop Manager	Proper noun
	1. Application that makes use of certain restricted functionality must be digitally signed.	
	2. Does not guarantee the quality or security of the code.	
3. Either BlackBerry App World, Over The Air through the BlackBerry mobile browser, or through BlackBerry Desktop Manager.		

This visualization of features is able to form clusters of features on the visualized graph and help domain analysts during the design of the domain model.

Step 4. Integration with Wiki: The final step of the decision support process consists of the transformation of the annotated domain documents and the document visualization graph into the MediaWiki format. We will discuss in the tool support section of the paper that we have developed several plug-ins for the MediaWiki environment that allow for the collaborative identification of features and integrity constraints by multiple domain analysts and also for the rendering and visualization of the semantically annotated domain documents.

As another example, Table 1 shows the details of the information extracted from a short text describing a Blackberry device. As can be seen, five features and three integrity constraints have been extracted that will be suggested to the domain analysts. For the features, they are cross-referenced with concepts from an external ontology (WordNet in this case). So for instance, **Digitally signed** is extracted as a feature representing the ability to provide a digital signature and has been marked as a potential feature for the domain analysts to evaluate. Furthermore, this extracted feature has been interrelated with a corresponding concept (and its description) from Wordnet. Furthermore, integrity constraints have also been extracted and recommended to the domain analysts, e.g., **Application that makes use of certain restricted functionality must be digitally signed** shows a restriction and hence an integrity constraint on certain applications.

In the following section, the techniques behind each of these four steps are introduced.

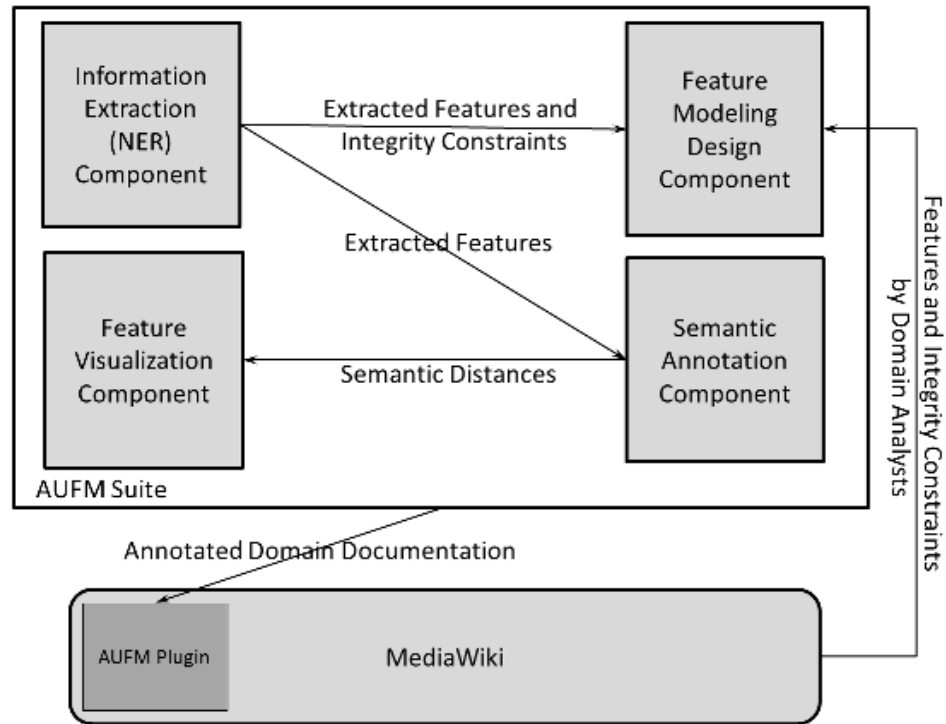


Figure 5. The component view of our decision support platform.

4. Approach Technicalities

We now present the technical underpinnings of each of the parts of our proposed approach. A component view of our decision support platform is shown in Figure 5, which describes how the different parts of the platform interact and what of information they exchange. Further details are given in the following.

4.1. NLP Analysis (NER)

The first step of our approach is to automatically detect and identify the relevant features and integrity constraints that are available in the domain documents using NLP technology. The most appropriate technique for this purpose is called Named Entity Recognition (NER) [17]. The main objective of NER methods is that given a set of predefined categories, to find sequences of words within a natural language document that belong to one of those categories. The most widely used applications of NER is the identification of person names, postal addresses, email addresses, and protein and gene names within textual documents. For our problem domain, we are interested in two categories: features and integrity constraints. The intention is to

employ NER techniques to browse through domain documents and label sequence of words that are potential matches for one of these two categories.

In the NLP literature, researchers have already proposed three main classes of NER techniques, namely dictionary-based, rule-based and statistical machine learning-based approaches [18]. In the dictionary-based approaches, a set of words or expressions are gradually collected by human experts and are labeled. Once this dictionary of labeled words is gathered, then new documents are searched to find matches between their content and the content of the dictionary. As an example, `westlaw.com` contains the names of all lawyers in the United States, which can be used to find lawyer names in legal documents. The approach used by rule-based techniques is more complex in that they use a set of rules developed by experts to extract word sequences and their category labels. Examples of these rules include the use of regular expressions for finding email addresses.

The third and the most complex class of methods for NER employ statistical machine learning-based techniques to label sequence of words with appropriate categories [19, 20, 21]. These techniques are initially trained based on an already labeled corpus, based on which they will create a learnt model. The learnt model is then used to label any subsequent text documents that are provided to them; therefore, the use of these techniques requires first a *training step*, where a statistical model is built based on textual features (e.g. n-grams, word lengths, part-of-speech tags). The learnt model is later used in the *labeling step* to annotate new textual documents with category information. In our work, we employ the variant of NER that implements a linear chain Conditional Random Field (CRF) sequence model along with text feature extraction algorithms [22].

A conditional random field is a discriminative undirected probabilistic graphical model [23]. A node in this graphical model represents a random variable whose probability distribution needs to be computed. The edges between the nodes represent the dependencies between the random variables. CRF is an alternative graphical model to Hidden Markov Models (HMM), which have been previously used by software engineers for the purpose of inferring specifications from natural language API documentation [24]. The major difference between HMM and CRF in terms of their application for NER is related to their dependency assumption; hence speed. HMM models make the strong independence assumption based on which all the random variables are assumed to be independent. CRF models relax this assumption; resulting in a slower speed in practice but at the same time more accurate probability distributions and more accurate results. For this reason, we employ the Stanford named entity recognizer model [22], which is based on CRF in our domain document labeling process.

The conditional random field-based NER method takes as input an already labeled domain document, i.e., a document whose features and integrity constraints have been tagged by a human expert, extracts textual features from the labeled document and learns a statistical model based on these textual features and their occurrence in the input document. Once the model is learnt, we employ it to label the features and integrity constraints of other unlabeled domain documents through label probability estimation. This process allows us to automatically label any new

domain documentation based on the learnt NER model and make recommendations to the domain analysts.

4.2. Semantic Annotation

Once the features and integrity constraints of a domain document have been identified, it is important that they are cross-referenced with appropriate concepts from a widely used ontology. Ontologies are a standard form for representing the concepts within a domain, and the relationships between those concepts [25]. Many domain-specific ontologies have already been created, with the purpose of being used as a *shared vocabulary* to model a target domain. It is important that the features identified in Step 1 are interrelated with ontological concepts, since:

1. As the target domain becomes more complex, many complicated terms and expressions are used in the domain documents that can be hard for the analysts to comprehend. The connection between the domain document terminology and the concepts in the ontology can increase domain *understandability*;
2. Some terms may have multiple meanings, i.e., be *homonyms*. For such cases, it would be difficult for the analysts to figure out the precise meaning of the terms. The links to ontology concepts can remove such ambiguity by creating a one-to-one correspondence between a term and an ontology concept; hence, specifically grounding the meaning of the term.

Our decision support platform provides the means to automatically map the identified features (hence the integrity constraints, as they build on top of features) onto the concepts of one or more ontologies. The decision about which ontologies to use is up to the analysts; however, we decided to use WordNet [26] in our experiments. Wordnet is a lexical database for the English language, which represents the various semantic relations between words that are available in its database. It can be viewed as a general-purpose lexicon of words and their semantic relationships. We selected Wordnet in our work as the default ontology because of its domain-independent nature. Precisely for this reason, Wordnet might not have an ideal performance in domain-specific settings and hence we have provided the capacity in the decision support platform to replace Wordnet with other suitable domain specific ontologies when they are available to the domain analysts.

In order to automatically annotate domain document features with ontological concepts, we need to find the best concept match for each feature within the ontology. Ontologies are generally represented using one of the variants of the Web Ontology Language (OWL) [27]. This language has its roots in Description Logics and provides formal and clear semantics for the definition of concepts and their relationships. OWL ontologies are often serialized using an RDF/XML representation, which is a triple format that models information using triples in the form of subject-predicate-object expressions. The information represented in RDF format (e.g., OWL ontologies) can be queried using a standard RDF query language called SPARQL [28], which is an SQL-like language. Appropriate SPARQL queries can

be formulated to extract meaningful and pertinent pieces of information from an RDF data source. A SPARQL query of interest to our decision support platform would be one that finds the best matching concept within an ontology for a given input feature. Such a query would enable us to find a relevant concept from the ontology for each of the features and hence would provide the means for annotating features with the matching ontological concepts.

Similar to SQL, one of the limitations of using SPARQL queries is that only perfect string matches are taken into consideration for satisfying the query. For instance, slight dictation differences between the spelling of features in a domain document and those in the ontology will result in an unsuccessful query. Given that spelling errors and dictation variants (e.g., British compared to American spelling) are inevitable in large domain documentation, we need to provide a degree of flexibility in the matching process. For this purpose, we employ *imprecise SPARQL (iSPARQL)* [29] to allow for such flexibility. The iSPARQL language and its supportive query engine allow for partial string matches through the use of various string matching functions such as Levensthein edit distance, Jaccard similarity and others. The advantage of iSPARQL is that it returns a confidence value in $[0,1]$, which indicates the degree of match between the input term and the found information. Researchers have already shown that iSparql is very effective for the purpose of mining software repositories [30]. Our decision support platform automatically generates iSPARQL queries for each of the discovered features from the *NLP Analysis* step and selects the ontological concept with the highest confidence value as the best match.

In our experiments, we employed the OWL representation of WordNet as the reference ontology; therefore, the features extracted from the domain documents are automatically annotated with concepts from the WordNet ontology. In summary, the *Semantic Annotation* step will formulate iSPARQL queries over the WordNet ontology for each of the identified features. The concept with the highest confidence value within the results of the query will be used to annotate that feature. This way, each feature will be cross-referenced with a concept from within the ontological frame of reference. The following is an example of the iSPARQL query that is executed for identifying WordNet concepts:

```
PREFIX isparql: <java:ch.unizh.ifi.isparql.query.property.>
PREFIX wn20instances: <http://www.w3.org/2006/03/wn/wn20/instances/>
PREFIX wn20schema: <http://www.w3.org/2006/03/wn/wn20/schema/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?description ?thing ?siml
WHERE
{
    $thing wn20schema:gloss ?description .
    ?strategy1 isparql:name "LevenshteinStrCmp" .
    ?strategy1 isparql:arguments (?description input) .
    ?strategy1 isparql:similarity ?siml .
}
```

```

    FILTER (?sim1 >= 0.7)
  }
ORDER BY DESC(?sim1)

```

This iSPARQL query shows that we use the Levenshtein distance measure for performing an inexact string match over WordNet concepts to find the best match. We only consider concepts that are at least 70% similar in the retrieved results.

4.3. Document Visualization

Besides the proper identification of features and integrity constraints, one of the important tasks of the domain analysts is to understand the relationships that exist between the identified features. A feature model allows the representation and capturing of such relationships through the use of a tree-like structure, but it is the responsibility of the domain analysts to identify such relations. In order to be able to characterize the relationships between the features, the domain analysts need to understand the purpose and meaning of each feature independently and subsequently find related features and gradually develop the hierarchical relations between the features as required by feature models. The decision support platform assists the domain analysts in this process by creating a visualization of the identified features. A visualization of the features on 2-D space, as shown in Figure 4, allows the domain analysts to understand the possible similarities between the features and to help them in the process of building the feature model.

In essence, the relationships between the features of a feature model depend on the conceptual similarity between the features. For instance, DFS and BFS are siblings in the feature model shown in Figure 2 since they are both algorithms for graph traversal and search (they are also placed close to each other in the visualization shown in Figure 4). For this reason, our visualization technique works with the conceptual representation of each feature. In order to be able to visualize the identified features, we will need to define two important mechanisms: 1) a distance measure between the identified features; 2) a mechanism for converting feature distances into 2-D coordinates.

For the purpose of calculating the distance between two features, we employ its ontological concept annotations that were found in Step 2. In other words, the distance between two features is the distance of their ontological concept annotations in the ontology hierarchy. Two features are close to each other if their ontological concept annotations are close to each other in the ontological hierarchy and they are considered to be distant otherwise. In order to quantify the distance between two features, we employ the ontological concept distance metric proposed by Lin [31]; therefore, the distance between two features is Lin's distance between the ontological concept annotations of the two features. Lin asserts that *'the similarity [hence distance] between [two concepts such as] c_1 and c_2 is measured by the ratio between the amount of information needed to state the commonality of c_1 and c_2 and the information needed to fully describe what c_1 and c_2 are'* [31]. This is

formally defined as:

$$dist_{Lin}(c_1, c_2) = 1 - \frac{2 \times sim_{res}(c_1, c_2)}{IC(c_1) + IC(c_2)}. \quad (1)$$

where Information Content (IC) is $IC(c) = -\log(p(c))$; $p(c)$ being the probability of encountering c in a given corpus and $sim_{res}(c_1, c_2)$ is Resnik's concept similarity measure [32] defined as follows:

$$sim_{res}(c_1, c_2) = MAX_{c \in S(c_1, c_2)} IC(c). \quad (2)$$

where $S(c_1, c_2)$ is the set of all concepts that subsume both c_1 and c_2 in the ontology. Based on these definitions, the distance between two features f_1 and f_2 that have been annotated with c_1 and c_2 , respectively is $dist(f_1, f_2) = dist_{Lin}(c_1, c_2)$. Now, assuming that n features have been identified in Step 1, we can compute a distance matrix $[DM]_{n \times n}$ between all n features such that $DM[i, j] = dist(f_i, f_j) = dist_{Lin}(c_i, c_j)$.

Once the distance matrix is computed, it needs to be converted into coordinates on a 2-D space to be suitable for visualization. We employ *Classical Multi-Dimensional Scaling (CMDS)* [33] for this purpose. The CMDS method, aka Torgerson Scaling, is a process that takes as input a matrix with item-item distances and assigns a location to each item on a k -dimensional space ($k=2$ for our case). Formally said, for a distance matrix such as $[DM]_{n \times n}$, CDMS will generate a vector of k -dimensional points like $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,k})$ such that $\|\mathbf{x}_i - \mathbf{x}_j\| = DM[i, j]$, where $\|\cdot\|$ is a vector norm. In other words, this mechanism generates n k -dimensional points based on an input $n \times n$ distance matrix.

The decision support platform employs the Torgerson scaling method to convert the feature-feature distance matrix into a set of 2-D points that can be drawn on a graph. Once these points are generated, each will represent a corresponding feature from the domain documents. Given that distance is defined based on the ontological concept annotations of each feature; therefore, similar features will be placed closer to each other and the less related ones will be farther apart. This can be seen in Figure 4 where similar features are next to each other and they form clusters. This visualization of the features can help the domain analysts identify the relationships that could exist between the features and assist in the process of forming the feature model hierarchy.

4.4. Integration with Wiki

The final step is more of a systems engineering process that involves the packaging and migration of the produced artifacts in the previous steps to a MediaWiki environment [34]. This step converts the domain documents, their semantic annotations and the document visualizations into a MediaWiki acceptable format, exports them into that environment and provides the appropriate references to the exported material on the Wiki to the domain analysts. The main purpose of this step is to provide the following facilities for the analysts:

1. Wiki sites allow for the easy creation and editing of a set of interrelated documents. Their main power lies in their support for collaborative editing. Within our decision support platform, the integration of the domain documents and the related produced information into a Wiki allows several analysts to collaboratively work on the same set of domain documents. Therefore, the employment of a Wiki as the integration point enhances the means for *collaboration* between the analysts;
2. Many organizations are now using a Wiki environment as their document management system, since it can be easily searched and maintained [35]. The export of domain documentation and the artifacts of our decision support platform to a Wiki will increase the chances of *domain knowledge preservation* for later reference and also *knowledge transfer*.

In Section 6, we show how the MediaWiki environment has been extended to facilitate the integration of annotated and visualized domain documents and to support domain analysts' collaboration.

5. Methodology Integration

The domain engineering lifecycle is comprised of a set of systematic phases for the development of a common core architecture for the applications of a given target domain of interest. The main focus of this lifecycle is to optimize *reuse*, i.e., it is meant to support *engineering-for-reuse*. Within the literature, the terms domain engineering and domain analysis have been loosely used to refer to the same lifecycle and employed interchangeably; a practice that we have also followed in this paper. Domain Engineering is often broken down into four main phases [36], namely: 1) domain requirements engineering; 2) variability modeling; 3) domain design; and 4) domain realization and testing. In the following, we will briefly introduce each of these four phases and discuss how our decision support platform is able to support them.

5.1. Phase 1. Domain requirements engineering

This first phase involves the definition of the scope of the domain based on which a comprehensive process is undertaken to gather information regarding the various aspects of the applications within that domain. In the context of product line engineering, the purpose of this phase is not only to identify the requirements of each individual application of that domain, but also to find commonalities between the requirements of the individual applications in order to facilitate commonality modeling and enhance reuse engineering. To this end, the main activity of this phase is to find the most relevant legacy application documentations, interview stakeholders and review corporate documents. Hence, the classification and management of domain requirements information is one of the most important tasks of the domain analysts in this phase.

The decision support platform provides the means to convert, categorize and export the domain requirements documents into an appropriate Wiki environment. Therefore, the domain analysts are able to manage and gradually classify the requirements information that they gather through this phase within a Wiki environment. The arrangement and organization of this information in the Wiki enhances the *transfer of knowledge* from this phase to the next phases. It also facilitates the *collaboration* between multiple domain analysts as they accumulate and analyze the domain requirements information.

5.2. Phase 2. Variability modeling

During the variability modeling phase, the domain analysts review the gathered domain requirements documents in order to identify domain-specific features. The aim is to pinpoint the features that are common between all of the applications of the domain (commonalities) and also the features that are unique only to a certain group of applications (variabilities). The identification of these features allows the domain analysts to further create hierarchical relations between these features and develop domain representative *feature models*. In order to complement the feature model hierarchies, the domain analysts will also need to find and formulate the integrity constraints between the features. This phase of the domain engineering lifecycle has a very high importance as its degree of completeness impacts the later applications that will be developed from the product line based on this feature model.

The main contributions of the decision support platform can be attributed to this phase of the lifecycle. The platform automatically processes domain documents and identifies potential features and integrity constraints and suggests them to the domain analysts; hence, facilitates the process of commonality and variability specification. Furthermore, it provides a semantics-based visualization of the domain-specific features that can be useful for inferring the hierarchical relations of the features. These two components support the desirable *diligence* characteristic. Furthermore, the semantic annotations of the identified features provides a meaningful platform for sharing the extracted knowledge with the domain analysts that will join the domain engineering process later; therefore, facilitate *knowledge transfer*. Similar to the previous phase, all of the developed artifacts and information of this phase are stored in a Wiki environment and hence increase the potential for *collaboration* between the domain analysts.

5.3. Phase 3. Domain design

This third phase of the domain engineering lifecycle revolves around the development of two main product line artifacts: the common core architecture, and the production plan. The design of the common core architecture consists of the selection of the most suitable architectural style and also the decision regarding the inclusion or exclusion of variable system components within the core architecture. In addition, the product plan will specify how concrete applications can be derived

from the common core architecture based on the features of the feature model and reusable assets, a process which can be either manual or automatic.

One of the important issues within the design of both the common core architecture and the production plan is to preserve and maintain a continuous connection with the artifacts of the previous phases. In other words, it is important for the designed architecture and the production plan to be aware of the features that they benefit from and where these features originate. This is specifically important for the purpose of change management. The decision support platform maintains a link between the features identified in the domain requirements documents, the features represented in the feature model, the common core architecture components and the production plan. Therefore, the domain analysts will be able to know where the roots of a feature are (within the domain documents) and what impact they have on the product line architecture and the production plan. Such linkage facilitates *traceability* between the artifacts of these three phases within the domain engineering lifecycle.

5.4. Phase 4. Domain realization

The domain realization phase, aka domain implementation, is responsible for providing actual implementation of the common core architecture, components and the features defined in the previous phases. The purpose of this phase is to provide implemented fragments that can be later integrated and incorporated into a uniform structure to build an actual product during the application engineering lifecycle. This phase consists of writing documentation and implementing domain-specific languages and model/code generators for the product line architecture and feature model.

In comparison with the previous three phases, the decision support platform provides the least support for this phase. Its main impact is on the provisioning of means for *traceability*. Similar to Phase 3, the implemented artifacts of domain realization are linked to the artifacts of the previous phases within the decision support platform; therefore, the domain analysts can track back each of the implemented components to its source within the realm of the domain requirements. These links serve as a suitable mechanism for *round-trip traceability* during the domain engineering lifecycle: from domain requirements to implementations and back. In addition to traceability, the documentation of the implementations and realizations of the artifacts in this phase take place in the Wiki environment. These documents can be later referenced from within the application engineering lifecycle. Given that the support for implementation documentation is not very systematic in this phase, it only provides weak support for the *knowledge transfer* characteristic between this phase and the application engineering lifecycle.

6. Tooling Support

AUFM Suite is our in-house Eclipse-based plug-in for feature modeling. The tooling support provided for the proposed decision support platform is developed within

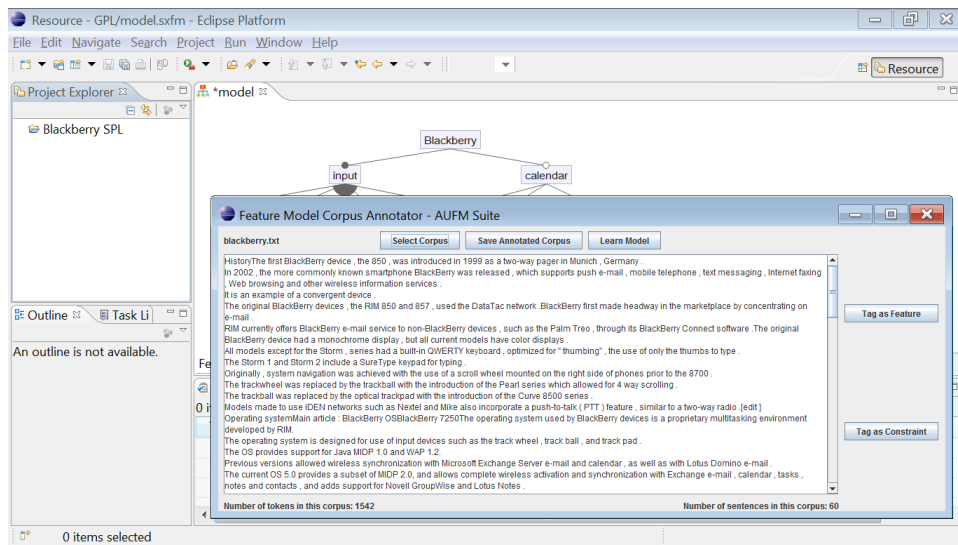


Figure 6. AUFM Suite tool for manual corpus annotation and model learning.

the general architecture of the AUFM Suite². There are four major components that form the basis of our tools.

The first component (Box 1 in Figure 3) mainly handles the NLP NER model learning step. This component provides the means to manually annotate an input corpus (domain document), store the annotated corpus, learn a NER model based on the annotated corpus and save the learnt model for future use. As shown in Figure 6, this component allows the domain analysts to select a corpus and tag its relevant terms or expressions as features or integrity constraints. Once the annotation of the input corpus is completed, the domain analysts can store the annotated document, learn a NER model based on it and save the learnt model. This component is responsible for building a NER model that can be later reused to annotate any input domain document.

The second component (covering Boxes 2 and 3 in Figure 3) builds on the assumption that a NER model has already been developed using the first component described above. This second component takes the learnt NER model and a domain document as input and automatically annotates the domain document by identifying its features and integrity constraints. The component also creates the semantic annotations of the features and draws the feature distribution graph visualization. Once the processing of the input domain document is completed, the component will convert it into appropriate Wiki format and exports it to the Wiki environment to be further processed by the domain analysts. The dialog box shown in Figure 7 is the front-end for this component and is responsible for performing automatic

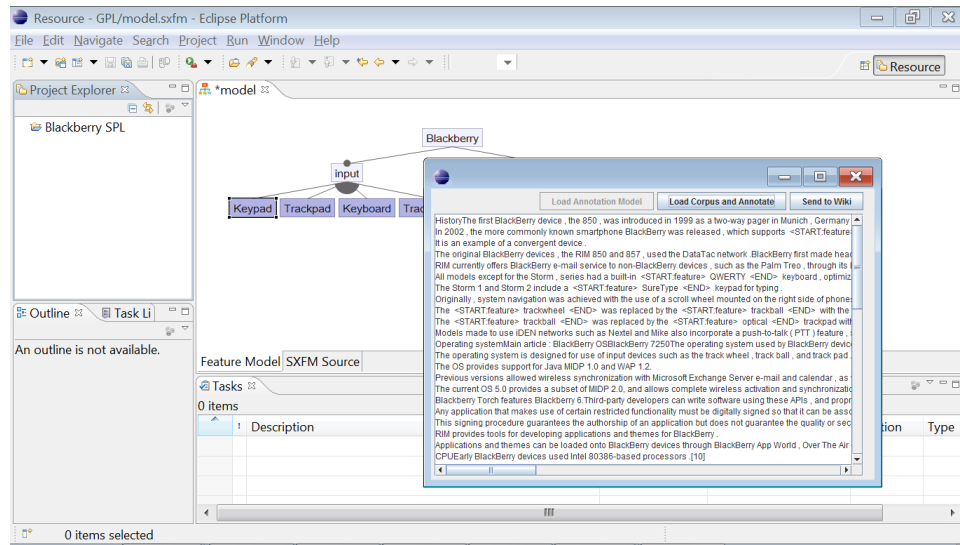


Figure 7. AUFSM Suite tool for automatic corpus annotation.

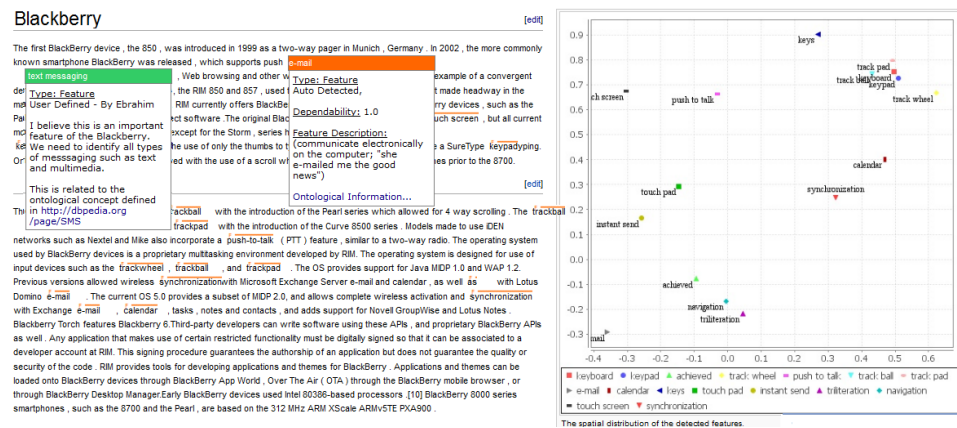


Figure 8. The view of the annotated Wiki page.

domain document annotation, finding semantic linkages, developing document visualization and exporting the processed domain document to a Wiki environment.

The third and fourth components (Box 4 in Figure 3) are extensions to the MediaWiki environment. These extensions are designed to enable the viewing and editing of the annotated domain documents. Figure 8 partially shows how an an-

7.1. Research Questions

The approach that we have taken towards the evaluation of our work is to investigate the impact of the decision support platform on the performance of a group of domain analysts. The *hypothesis* in our experiments is that the decision support platform is able to increase the accuracy and coverage of the domain analysts tasks and at the same time reduce the time taken to perform them. To evaluate this hypothesis, we formulate four interesting research questions and investigate how the decision support platform addresses each of these questions. These four questions are as follows:

- RQ1.** How accurately does the decision support platform perform in terms of automatically identifying features and integrity constraints from an input domain document?
- RQ2.** How does the execution time of the decision support platform scale as the size of the input domain documents increases?
- RQ3.** Would the employment of the decision support platform result in the identification of a more comprehensive set of features and integrity constraints by the domain analysts?
- RQ4.** How effective is the decision support platform in reducing the time spent by the domain analysts?

These research questions are evaluated based on our experiments that observe the behavior and outcome of the actions of a group of domain analysts. We use several metrics to evaluate the defined research questions.

7.2. Environment Setup

7.2.1. Objects of Study The main objects used in our experiments are domain-specific documentations. We have gathered these documents from different target domains of interest and have carefully processed these documents manually in order to be able to use them as *gold standards*. In other words, we have analyzed these documents, identified their features and integrity constraints such that they can be used as reference models. The processed documents are specifically employed for learning the NER model that can be later used for annotating other domain-specific documents.

In order to build and train the statistical NER model, we collected and manually labeled domain documents from five different target domains. For the labeling process, we carefully reviewed each of the domain documents and identified the features and integrity constraints that were available in these documents. We have used the first component of our AUFM tooling support, which was introduced earlier, to manually annotate these domain documents. The details of

Table 2. The details of the domain documents³ used for training the NER model.

Domains	#Sub-domains	#Tokens	#Sentences	#Pages
Browsers	5	26,207	908	72
Bio-Medical Software	7	3,338	111	11
Operating Systems	6	13,548	467	31
Security Software	6	5,922	202	19
Handheld Devices	9	22,168	815	60
Total	33	71,183	2,503	193

Table 3. The specific sub-domains that were considered in the domains of Table 2.

Browsers	BioMed. Soft.	Operating Syst.	Security Soft.	Handheld Devices
Chrome	BioClipse	FreeBSD	ClamWin	Blackberry
Firefox	BioMOBY	Haiku	GnuPG	Nintendo DS
IE	FreeMed	Inferno	IPTables	HTC-EVO
Opera	GIMIAS	NetBSD	PeerGuardian	iPad
Safari	MeshLab	OpenBSD	SmoothWall	iPhone
	Sirius	Windows 7	Winpooch	Samsung Galaxy
	THIRRA			HP iPaq
				Tablets
				Smart Phones

these documents are shown in Table 2. This table shows the number of sub-domains covered by each target domain, the number of tokens (words or complex multi-segment phrases), and the number of sentences in each of the documents. The calculations are done using the Stanford NLP tools available at <http://nlp.stanford.edu/links/statnlp.html#Tools>. It also provides the length of the documents based on an 11 point Times New Roman font. The sub-domains covered by each of the target domains in Table 2 are mentioned in Table 3. As it can be seen a total of 33 sub-domains, consisting of 71,183 tokens and 193 pages of domain documents have been processed for this purpose.

One of the important notes that need to be mentioned is that we use a *ten-fold cross-validation* approach [37], which is a method for estimating the performance of a predictive model, in order to evaluate the performance of our decision support platform. For this purpose, each of the domain documents are split into two segments with a size ratio of 9 to 1: the training segment (9/10) and a testing segment (1/10). The training segments of the documents are used in the NER learning stage, while the testing segments are used in our experiments with the domain experts. The reason for this is that if both the training and testing documents are equivalent, then the accuracy of the results is not reliable because it cannot be inferred whether the results of the experiments performed using our decision support platform is generalizable to unforeseen documents or not. The cross-validation approach allows for such generalization.

7.2.2. Interaction with Subjects The main objective of our experiments was to determine how effective our decision support platform is in assisting the domain analysts. In other words, would the domain analysts perform their tasks more ef-

Table 4. The specification of the two documents used in the experiments.

Document	Domain	#Tokens	#Pages	#Features	#Integrity Constraints
\mathcal{D}_1	Tablets	1,214	4	42	11
\mathcal{D}_2	Windows 7	1,216	4	31	9

ficiently in the presence of the decision support platform or not. Therefore, we performed a set of experiments with human subjects to evaluate the impact of the decision support platform. In our experiments, fifteen volunteers actively participated in the evaluations. The participants were all graduate students of Computer Science with the age range between 23-29 years. The participants all had a good knowledge of software engineering principles and were given a mini-tutorial on aspects of software product lines and feature models prior to the experimentation period. Different phases of the domain engineering lifecycle were also explicitly explained and the aim and artifacts of each phase were detailed for them.

Before the start of the interaction with the participants, the experimenters used the domain document training set (9/10) to learn a NER model. This NER model was then incorporated into our decision support platform and used by the participants throughout the experiments. During the experiments, each participant was provided with two domain documents (The details of the two documents, \mathcal{D}_1 and \mathcal{D}_2 , are shown in Table 4) from the test documents. The same two documents were provided to all of the participants. The task of the participants was to individually read the two documents, understand the content and then design a feature model based on the content of each document. So, the artifacts produced by each participant after the end of the experiment were two feature models, one for each domain document. Now, in order to evaluate the impact of the decision support platform, each of the participants were asked to perform the process for one document using the decision support platform and the other one without the intervention of the decision support platform.

In order to form a fair distribution, eight of the participants were asked to process \mathcal{D}_1 with the help of the decision support system and \mathcal{D}_2 without it, while the other seven participants were asked to perform otherwise. In addition, the first group of eight participants processed \mathcal{D}_1 first and then processed \mathcal{D}_2 ; therefore, they interacted with the decision support system in their first domain engineering experiment, whereas the second set of participants first performed the domain engineering process without the help of the decision support platform on \mathcal{D}_1 and were later able to use it while working on \mathcal{D}_2 . This experiment setting was used in order to minimize the impact of the documents, their order, and the order of the use of the decision support platform on the performance of the participants and hence on the obtained observations.

8. Observations and Evaluation

The goal of our evaluations is to validate our hypothesis, i.e., the decision support platform has a positive impact on the performance of the domain analysts. For this

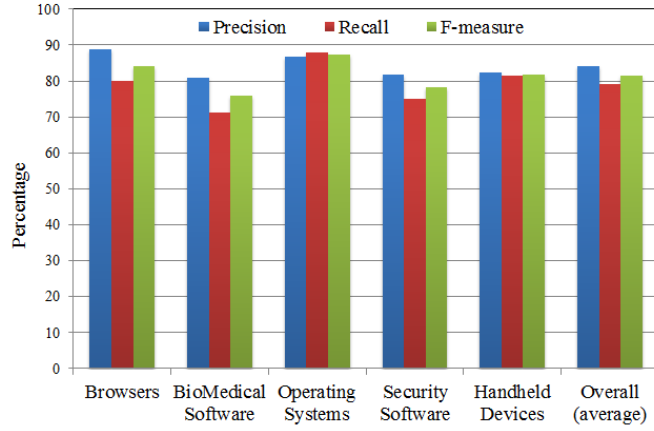


Figure 10. The accuracy and coverage of the decision support platform.

purpose, we report and analyze our observations with regards to the four related research questions. In the following, the reported execution times are based on the performance of our decision support platform on a dedicated 2.8 GHz Core i7 Personal Computer with 8GB of memory, running a Windows 7 operating system along with Java 6.0.

8.1. RQ1. accuracy and coverage

The first research question intends to analyze the accuracy and coverage of our decision support platform. Given the fact that the platform automatically processes input domain documents and suggests possible features and integrity constraints to the domain analysts, it is important to evaluate two characteristics of the platform: 1) *accuracy*: what ratio of the recommended features and integrity constraints are actually correct recommendations; and 2) *coverage*: how well can the platform identify all of the available features and integrity constraints from the domain documents. In order to evaluate accuracy and coverage, we employ the *precision* and *recall* metrics [38]:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \quad (3)$$

Precision is the ratio of retrieved information that are relevant for the intention; therefore, it is a good indication of the accuracy of the platform. In other words, precision shows the percentage of correct features and integrity constraints from among all of the identified ones.

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \quad (4)$$

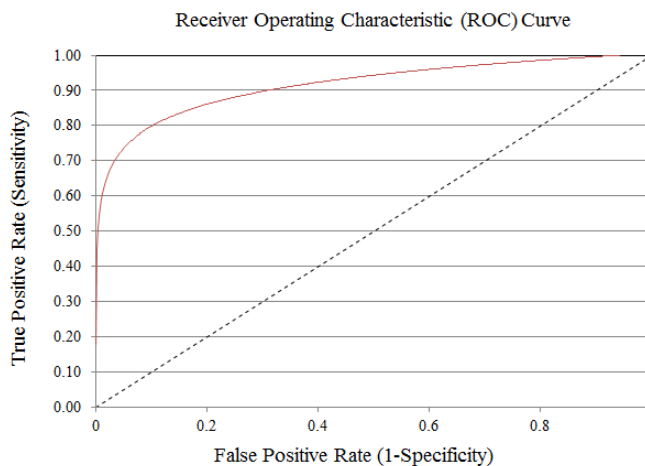


Figure 11. The ROC curve for the decision support platform.

The recall metric is the fraction of the relevant information that have been successfully retrieved; hence, it is an indication of coverage. In our work, recall represents the ratio of correct features and integrity constraints that were retrieved over all of the available ones within the domain documents.

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall}. \quad (5)$$

Traditionally, the precision and recall metrics have been combined through a harmonic mean and represented through the F-measure [38]. In order to calculate precision and recall, we have performed *ten-fold cross validation* over the objects of study that have been already manually annotated introduced in Table 2. The results are shown in Figure 10. As shown in the figure, the precision and recall of the decision support platform vary between 73-88% based on the sub-domain, but are on average above 75% across all of the domain documents (last column). This means that the platform is able to identify eighty percent of the available features and integrity constraints of a given domain document and also that its recommendations are 80% reliable, which is quite accurate for our purpose as a decision support platform.

In addition, Figure 11 shows the Receiver Operating Characteristics (ROC) curve [39] for the decision support platform. The ROC curve is plotted based on true positives (*sensitivity*) in function of the false positives (*1-specificity*). For a perfect system with no errors and complete coverage, the ROC curve passes through the upper left corner; therefore, the closer an ROC curve is to the upper left corner, the better it performs. In other words, a system with an ROC curve closer to the upper left corner can provide more true positives in the face of less false positives. On the other hand, an ROC curve closer to the identity line is very weak. Based on

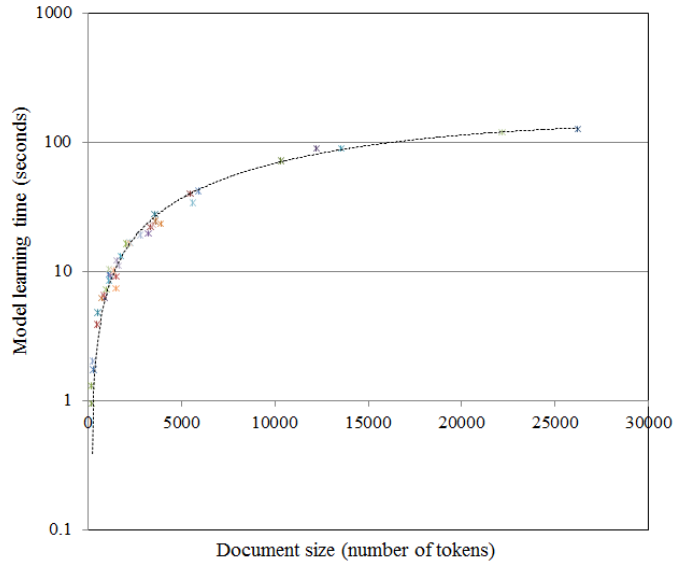


Figure 12. The execution time of the *learning step*.

the true positives and the false positives of the decision support platform depicted in Figure 11, its ROC curve leans towards the upper left corner and hence represents an acceptable system, as per interpretation given in [39]. In summary, based on the precision, recall and F-measure metrics and also the ROC curve, there is a solid evidence that the decision support platform has both satisfactory accuracy and coverage and can hence provide acceptable recommendations in terms of suitable features and integrity constraints from the domain documents to the domain analysts and support them in the process of identifying many of the features and integrity constraints. So, the decision support platform is able to perform positively wrt RQ1.

8.2. RQ2. execution time scalability

The scalability of the decision support platform in terms of execution time is quite important as it can delay the work of or discourage the domain analysts and hence reduce the chances of its use in practice. The evaluation of the execution time scalability was performed for both the model learning stage, where an already annotated model is provided to the NER technique for building a reusable model and also for the automatic document annotation stage where a domain document is automatically processed and annotated based on the reusable NER model.

As opposed to the previous experiment, the ten-fold cross validation approach was not used in this step anymore. The reason for this was that the performance of the models in terms of accuracy and coverage were not subjects of the study for this

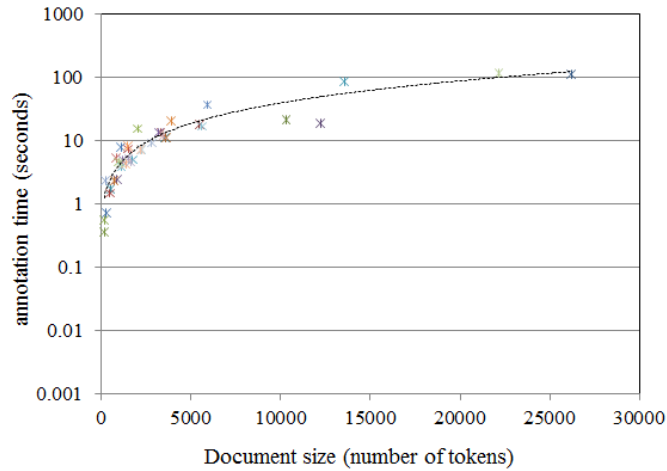


Figure 13. The execution time of the labeling step.

research question. The execution time of the training step and the labeling step were of significance in this study; therefore, the complete domain documents were used in both steps in order to allow us to study scalability with larger documents. The measurement of the execution time of the learning step was achieved by supplying the various domain documents from different sub-domains, shown in Table 3, to the conditional random field-based NER method. The time taken to learn a model based on the input documents was measured. Figure 12 depicts the execution time of the learning step in seconds. It can be seen that for relatively smaller domain documents with less than 10,000 tokens (20 page documents), the decision support platform takes around 65 seconds to learn the model. For larger documents with over 26,000 tokens (over 70 page documents), the execution time is close to 120 seconds. The figure shows that the execution time of the learning step does not radically increase with the size of the domain documents and remains within acceptable ranges. For instance, based on the figure and a simple regression, we can anticipate that for documents of around 1,000 pages (with size 11 Times New Roman font) that the learning step can take less than 20 minutes. This execution time is reasonable in light of the fact that the learning step is only performed once and its results are repeatedly used in the labeling step.

The execution time of the labeling step was also measured, the results of which are reported in Figure 13. The labeling step performs similarly to the learning step. This execution time can be problematic if the domain analysts insert all of their domain documents into the decision support platform at once. For instance, there will be a maximum processing time of around one hour for domain documents with up to 3,000 pages. However, it is very unlikely that the domain analysts would start processing all of the domain documents at once, since the documents may not be always available to them at the same time or the domain analysts do not

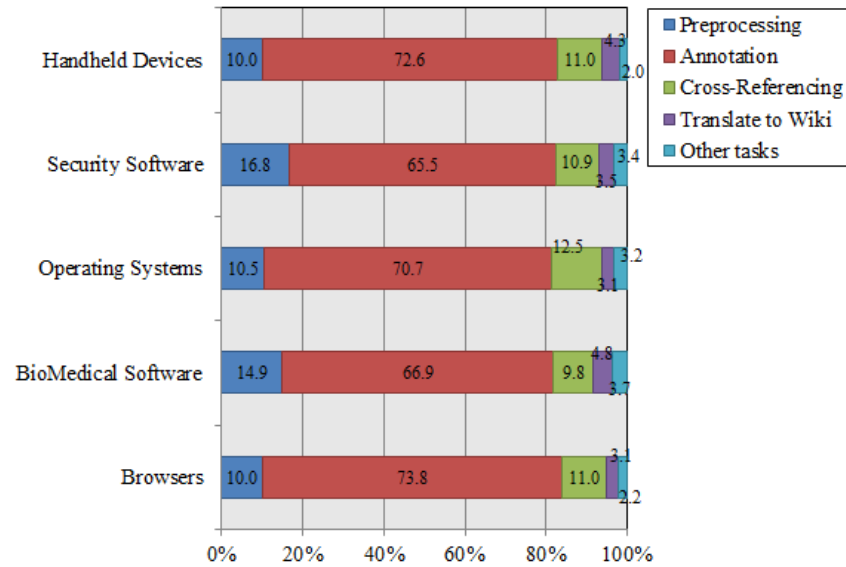


Figure 14. The execution time breakdown for the labeling step.

have the resources to process all of the documents simultaneously. They are most likely to go through the domain documents one at a time, which will distribute the execution time of the labeling process over multiple times; therefore, the domain analysts will only have to wait for a processing time of less than 2-3 minutes for documents of around 100 pages.

Figure 14 further shows the breakdown of the labeling step execution time. As can be seen in the figure, besides the automatic labeling and annotation of the domain documents, the execution time also consists of time needed for identifying semantic annotations (cross-referencing with ontologies), document visualization and export to the Wiki environment. However, a significant portion of time is allocated to the NLP Analysis (NER) step. Overall, the execution times of all steps seem to be manageable and hence the platform can be scaled to domains with large domain descriptions.

8.3. RQ3. domain analysts performance

One of the indicators for the performance of the domain analysts is the measure of how successful they have been in terms of identifying the correct features and integrity constraints for a given target domain based on the available documentations. To measure this indicator, we employ a similar measurement approach to that used for RQ1, where coverage and accuracy were assessed. As explained earlier

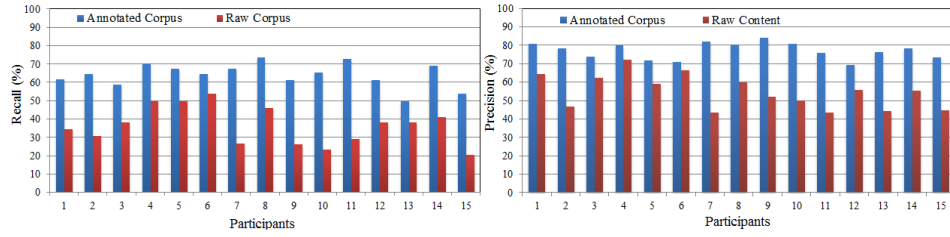


Figure 15. The precision and recall of the domain analysts.

for both this research question and the next, fifteen participants were involved in the experiments.

The participants benefited from the capabilities of the decision support platform while working on one of the domain documents; hence, received automatic feature and integrity constraint recommendations from the platform, viewed the cross-referenced domain document with WordNet and also had a view of the feature distribution graph visualization. However, for the other domain document, they only had access to the AUFM feature modeling tool (without the decision support capabilities, i.e., only the feature model designer GUI). Once the participants completed their work on each of the documents, the experimenters compared the work of each participant with the gold standard and computed the precision, recall and F-measure for each document. The results are depicted in Figure 15. In this figure, annotated corpus refers to the domain document that was processed with the help of the decision support system, whereas the raw corpus is the one without access.

The figure clearly shows that all of the participants reached higher coverage and accuracy with the help of the decision support platform. For cases such as Participant 7 who had more difficulty with the domain engineering process, the decision support was able to increase the performance of the participant by as much as +40% in terms of both precision and recall. Figure 16 visualizes the F-measure for the performance of the participants. This figure is a clear indication that the employment of the decision support platform is a positive contributing factor to the better performance of all fifteen participants in terms of coverage and accuracy. An interesting observation of this study based on the performance of the participants on both documents is that the decision support platform has not only been able to improve the work of the participants who usually perform quite well even without the decision support platform (such as participants 4, 5 and 6), but has also succeeded in significantly helping the weaker performing domain analysts (e.g., participants 9, 10 and 11) to improve their work and bring it up to par with the work of the better performing participants.

It is important to mention that for all of the three measurements, i.e., precision, recall and F-measure, a *paired t-test* yields a *two-tailed P value* of less than 0.0001 (10^{-4}); therefore, the improvement made based on the use of the decision support platform can be considered to be *extremely statistically significant* for all these three measurements. In other words, the impact of the decision support platform on the

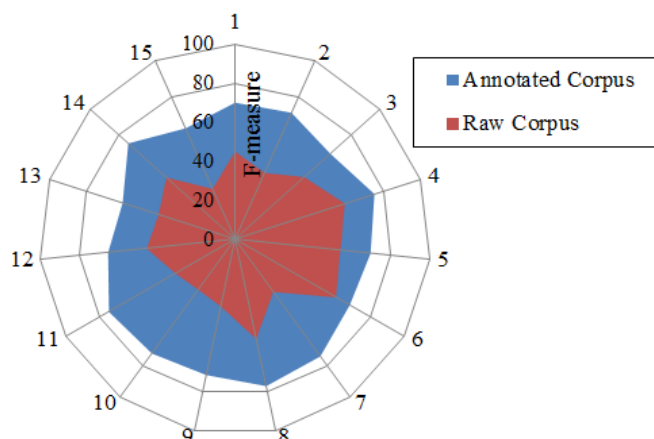


Figure 16. The overall performance of the domain analysts measured by F-measure.

improvement of precision, recall and F-measure has been substantial and influential on the performance of the participants.

Figure 17 provides a comparative analysis of the performance of the participants based on the ROC curve. Simply said, this ROC curve compares the work of the participants in terms of the true positives and false positives in their results. As can be seen, the participants perform much better (the ROC curve is closer to the upper left corner) when they are equipped with the decision support platform compared to the situation when they do not use the platform. Given that we have strived to keep all circumstances of the experiments (annotated corpus vs raw corpus) the same other than the role of the decision support platform, we believe the significant improvement in the performance of the participants can be attributed to the effectiveness of the decision support platform.

8.4. RQ4. time spent by domain analysts

Depending on the size and complexity of a target domain of interest, the size and number of domain documents that need to be processed vary accordingly. A large domain would require the domain analysts to go through and understand a larger number of documents and information; therefore, the larger the domain is, the more time the domain analysts need to spend. For this purpose, our final research question involves the observation of the impact of the decision support platform on the time spent for processing domain documents by the domain analysts. In other words, the intention was to study whether the decision support platform had any effect, either positive or negative (ideally positive), on the reduction of the time needed by the analysts for processing domain documents.

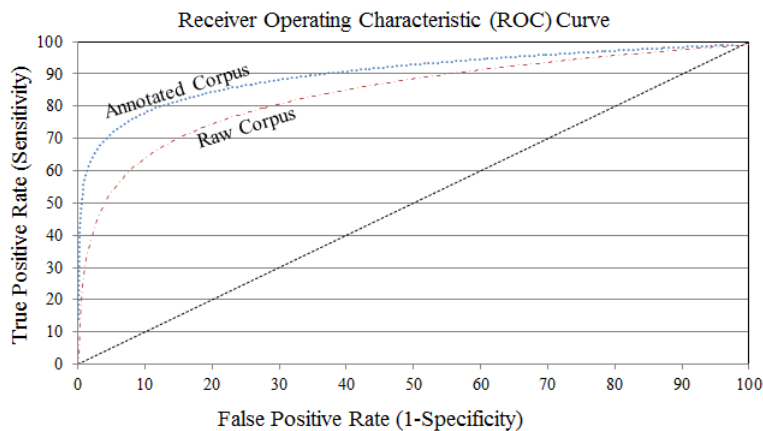


Figure 17. The ROC curve showing the performance of the participating domain analysts.

Before the study, the impression was mixed; the decision support platform might delay and negatively effect the time spent by the analysts because it provides them with additional pieces of information including semantic annotations and feature distribution graphs that they have to take into account and additionally process. On the other hand, given the fact that it provides suggestions on appropriate features and integrity constraints, it could possibly reduce the workload of the domain analysts and hence decrease the time spent. In order to investigate the impact of the decision support platform on this research question, the amount of time spent for the processing of each of the two domain documents (\mathcal{D}_1 and \mathcal{D}_2) by the participants was measured.

The measured time is shown in Table 5. Each cell in the table shows the average time spent by the participants along with the standard deviation. The first column of the reported times shows the time spent by the participants on each of the two documents with the help of the decision support platform; whereas the second column shows the time spent without that assistance. As can be seen from the table, the decision support platform does not impact the average time spent on the analysis of the domain documents. Based on our close observation of the behavior of the participants, we believe that the reason for this lack of impact can be attributed to the way domain analysts process domain documents. In our experiments, the participants tended to carefully read through all of the domain documents and took notes in both cases (with and without the decision support platform). In the case where they received the suggestions, they still read through the whole document and did not just rely on the platform for its recommendations; therefore, the average time they spent was not significantly different. However, an additional observation was that some of the participants had a higher confidence in the decision support platform than the others. This group quickly accepted the feature and integrity constraint recommendations, while another group of participants explored

Table 5. The time in minutes spent by the participants (average time \pm standard deviation).

Document	Avg Time - with support	Avg Time - without support
\mathcal{D}_1	23.125 \pm 6.05	23.285 \pm 4.38
\mathcal{D}_2	23.857 \pm 6.84	23.875 \pm 4.29

the recommendations in more depth and spent more time on them, which is the underlying reason for the higher standard deviation for the time spent when the decision support platform was used.

Overall, it seems that on average and among a group of collaborating domain analysts, the platform will not impact the time spent on the domain engineering process; however, on an individual per participant basis, the impact on the spent time is dependent on the degree of confidence of that individual on the performance of the decision support platform. So, if the domain analyst trusts the platform with its recommendations, the time taken to browse through the domain documents can be reduced; while the opposite situation is also possible. Given our observation of the behavior of the participants in the experiments, it is likely that the time spent processing the documents could be correlated with the length of the documents; therefore, as future work, we are interested in exploring whether text summarization techniques [40] would be helpful to reduce the time spent on domain documents while preserving acceptable coverage and accuracy.

8.5. Summary

The hypothesis in our experiments was that the decision support platform is able to increase the accuracy and coverage of the work performed by the domain analysts while reducing the time taken to perform the related tasks. Our observations have shown that the accuracy and coverage obtained as a result of using the decision support platform have significantly increased, which is an indication of its impact and usefulness. However, the decision support platform has not had any meaningful effect for either increasing or decreasing the total time spent by the domain analysts.

9. Discussions

In this section, we provide our analysis of the threats to the validity of our experiments and will also provide some insight into the lessons learned from the experiments.

9.1. Threats to Validity

Empirical evaluation is always subject to different threats that can influence the validity of the results. Here, we will discuss the threats to conclusion, construct, internal and external validity. We will specifically refer to the aspects of our experiments that may have been affected by these threats.

Conclusion Validity: Conclusion validity is the extent to which the conclusions about the presence of a statistically significant relationship between the treatments and the outcomes are valid. In our experiments, a limited number of data points were collected due to our restricted access to appropriate participants that had enough knowledge of software engineering within the area of software product lines. In addition, we only collected useful publicly available domain documents from the Internet from five main domains. Although these numbers are comparable to similar studies in the area of empirical software engineering [41, 42] and can provide a basis for understanding the behavior of the decision support platform, they may pose threats to the drawn conclusions. For instance, the use of a larger number of domain documents from a wider set of domains could impact the coverage and accuracy of our decision support platform. We are currently working on the collection of a larger set of domain documents and training of qualified participants to conduct further replication studies.

Construct Validity: Construct validity is concerned with the extent that the independent and dependent variables provide accurate measurements of what they are intended to measure. In our experiments and based on the fundamental hypothesis, we have been interested in measuring and analyzing three main characteristics of the decision support platform, i.e., its impact on the accuracy, coverage and time of the domain analysts. The time spent by the domain analysts has been directly measured by the amount of time that they have spent working on domain documents within an experiment session. This is a straightforward and direct measurement mechanism and hence poses the least threats to construct validity. One of the threats that could be attributed to this measurement approach is that some domain analysts that participated in our experiments might have prior knowledge of a domain and that would indirectly influence the time they spent on the experiments. We have tried to control this parameter by making sure that the participants had a similar background. The other two characteristics, coverage and accuracy, have been measured based on information retrieval metrics, which rely on a gold standard for comparison. Our gold standard has been the set of domain documents that we have manually processed; therefore, it can be argued that since domain engineering has some aspects of subjectivity that direct comparison with a gold standard designed by the experimenters might impact measurement and pose threat. Despite this threat, researchers have been using this measurement mechanism in various domains such as ontology learning [43] and text analysis [44].

Internal Validity: Internal validity is the degree that conclusions can be made about the causal effect of independent variables on dependent variables. An internally invalid experiment can lead to results that are not necessarily inferred from a causal relationship. For this reason, we investigate the following issues:

- *Difference between subjects:* In the study reported in this paper, there was no significant difference between the experience of the participants in using software product line feature models, due to the fact that all of them were Computer Science graduate students that had taken at least one advanced software engineering course. Therefore, error variance from difference between subjects is kept to a possible minimum.

- *Knowledge of the universe of discourse:* Both of the employed domain documents (\mathcal{D}_1 and \mathcal{D}_2) used in the experiments were simple enough to be understandable by the participants. Hence, knowledge of the domain was not a significant impacting factor in the study.
- *Maturation effects:* The participants did not become involved in any other training exercises in the area of software engineering during the experiments in order to minimize the maturation/learning effect.
- *Fatigue effects:* The sessions held by the experimenters with each individual participant was limited to 90 minutes. Since the participants are all Computer Science students and a usual lecture time is 90 minutes, the fatigue effect is not significant in this study.
- *Persistence effects:* The study was performed on subjects that had no prior experience in participating in similar studies. Therefore, persistence effects were avoided.
- *Learning effect:* The participants were each asked to perform the task once using the decision support platform and once without. Therefore, it can be argued that the participants became more comfortable with the design process in the second experiment and hence the learning effect. However, we have randomized the order of executing the experiments to minimize this effect.
- *Subject motivation:* The participants were all graduate students and were therefore quite aware of the importance and impact of empirical evaluations for our research. Furthermore, the experiments were done on a volunteer basis and it is fair to assume that the participants were quite motivated to take part in the study.

Plagiarism and influence were controlled by explicitly asking the subjects not share any information about the study with each other until after the experiments were concluded. Furthermore, the evaluation sessions were held individually for each participant with the presence of an experimenter.

External Validity: External validity is the extent that the obtained results of a study can be generalized to the setting under study and other relevant research scenarios. The results of a completely externally valid study can be generalized and applied safely to the software engineering practice and be recommended as blueprints. The following two issues were considered for external validity:

- *Materials and tasks used:* In our experiments, we tried to gather a set of high quality domain documents and manually verified their quality. However, the domain documents were restricted to the five domains shown in Table 3. Therefore, further empirical studies need to be conducted on more comprehensive domain documents.
- *Subjects:* Due to the difficulty of getting professional domain engineers' participation in our experiments, we used Computer Science graduate students in our

studies. In our experiments, the participants do not particularly need a high level of industrial experience to be able to complete the experiment. Furthermore, authors such as Basili et al. [45] believe that in many cases, the use of students in experiments has little impact on the validity of the results. However, further experiments involving industrial professional subjects are needed to ensure the external validity of our experiments.

Another threat to validity may be related the usability aspects of our decision support platform. Since the platform has been designed for research purposes, it might not be as comfortable as industrial level standard tools and might be seen as a threat; however, given the fact that the platform was used by all of the participants in a similar manner and under the same conditions, we believe that it possibly effected all of the participants in the same way. Unfortunately the effect of this threat could not have been controlled in our experiments.

9.2. *Lessons Learnt*

The process of the design and execution of the research work required for this paper provided us with important insight into various aspects of performing empirical investigation within the domain engineering lifecycle for software product lines.

The main challenge that we faced was with regards to access to a set of industry-scale domain documentations. Understandably, many industrial corporations do not support the idea of publicly releasing the artifacts of their domain engineering practices. This is also true for non-product line approaches, i.e., traditional requirement engineering processes where it is hard to obtain real world requirement engineering documents for research purposes. Therefore, it is important to build a collection of domain documents within the research community that could be used for this purpose. The SPLOT repository [46] is an example of such effort that focuses on the collection of suitable feature models that can be used as benchmarks by researchers in their experiments. It seems that a similar repository for requirements and domain documents would help replicable empirical research within the software product line area.

The other challenge was the measurement of the performance of the decision support platform. Before starting our experiments, we considered two different design approaches for the experimental setting, namely the *holistic assessment* and *component-based assessment* settings. The design of an experiment based on the holistic assessment setting would involve the evaluation of the decision support platform as a whole. In other words, the evaluation observes the overall impact of the decision support platform on the performance of the domain analysts. On the other hand, the component-based assessment setting would consider the impact of each individual component within the decision support platform on the domain analysts. For the experiments, we decided that the holistic assessment setting would serve our intentions more closely; therefore, the experiments presented in this paper are based on the goal of evaluating the influence of the decision support platform as one solid entity. The post-mortem analysis of our evaluation strategy suggests

that a hybrid approach to experimental setting would have been able to reveal the impact of the decision support platform and also point to weaknesses in each of the individual components. The design of this form of experiment is non-trivial and we are working on it as a part of our future work.

The third issue that was observed as a result of our empirical experiments was the role of *trust* in the performance of the decision support platform. By *trust*, we mean the confidence of the participants in the decision support platform. We were able to identify two extreme groups within the participants that were either fully confident in the workings of the decision support platform or who were completely skeptical. For both of these groups, we observed higher variance in the activity time and performance (accuracy and coverage). As a result of this observation, we believe that it is important to carefully explain both the limitations and capabilities of a system to the participants of an empirical study in order to avoid over/under confidence in the system, which can lead to undesirable impacts to the validity of the obtained results and the experiments as a whole.

10. Related Work

As software product lines have been gaining attention due to their usefulness for reducing time-to-market and development costs of new products, several researchers have become interested in automating various aspects of the software product line development process. Until recently, the main focus has been on the development of automatic algorithms for the application engineering lifecycle. More specifically most of these techniques have been formulated to address the automated configuration of software product line feature models based on stakeholders' functional and non-functional requirements [47, 48, 49, 50, 51]. Feature model configurations have been often derived, validated and verified using Logic Truth Maintenance Systems (LTMS). Three of the most widely used methods in this context are Constraint Satisfaction Problem (CSP) solvers [52], propositional SATisfiability problem (SAT) solvers [49, 53] and Binary Decision Diagrams (BDD) [54]. Furthermore, Description Logic reasoners have also been employed to validate the structural correctness of feature model configurations [55, 50].

Considering the fact that the usefulness of a software product line depends heavily on the artifacts and products of the domain engineering lifecycle, researchers have recently started to explore the development of tools for quality evaluation and automated support for this lifecycle as well. For instance, in our previous work, we have provided a metric suite and quality prediction framework [9] for understanding the maintainability aspects of feature models early in the product line development process. There are several works in the literature that are close to the theme of the current paper. Most closely to our work, Dumitru et al. [56] have considered the automation of the feature model design process through an information retrieval recommendation process. In their work, these authors employ the diffusive clustering algorithm and the k -Nearest-Neighbor machine learning strategy to provide potential feature recommendations to feature model designers during the domain engineering lifecycle. Their work is different from ours from several points

of view: 1) the work by Dumitru et al. only provides automation support for Phase 2 (variability modeling) of the domain engineering lifecycle whereas our approach provides a framework that assists the domain analysts throughout the domain engineering lifecycle; 2) the work by these authors only provides recommendations to domain engineers wrt potential features and does not provide any suggestions about potential integrity constraints that are a pivotal element of a feature model; however, our decision support platform provides both forms of feature and integrity constraint recommendation; 3) these authors have worked with textual documents under certain structural constraints, e.g., a bullet-point list format, whereas our platform identifies features and constraints in any textual document. Neither of the two approaches provide mechanisms for helping the domain analyst in creating the feature model hierarchical structure.

The work by Weston et al. [57] also considers natural language documents for extracting feature models. In their work, the authors build on top of EA-Miner [58], which is a tool for finding candidate crosscutting concerns in textual documents. ArborCraft is the tooling framework that employ the artifacts of the EA-Miner and identifies variant features through a lexicon-based natural-language approach. The authors employ an Hierarchical Agglomerative Clustering (HAC) technique to create the hierarchical structure of a feature model. This work provides a promising framework for extracting feature models but the authors have not reported their empirical evaluation of the work other than on a sample Smart Home feature model. The work by Niu and Easterbrook [59] tackles the problem of domain analysis by proposing an extractive approach for identifying software product line requirements. The authors extract functional requirements profiles (FRPs) through lexical affinities that bear a *verb-direct object* relation. This work is different from ours in that it does not consider the development of a feature model per se.

More closely related to our work is the approach proposed by John [60]. In her work, John rationalizes that many of the information used in product line engineering are coming from legacy documentation. Therefore, an information extraction metamodel is proposed that includes user documentation, product line artifact and variability model packages. Then the metamodel is used as a basis for extracting information from legacy documentation through *extraction patterns*, which are simple templates representing information structure. In contrast to this work, our approach is not based on predefined extraction patterns and uses intelligent information extraction techniques for identifying the important concepts of a feature model.

The automated derivation of feature models has also been explored from a reverse engineering perspective. She et al. [61] have proposed a set of heuristics to process program source code and identify potential features and also extract constructs such as feature groups and mandatory features. Their work is most suitable for domains that already have many legacy applications developed; therefore, the domain engineering process can consider existing application source code along with domain documentation. Some other related work within the requirements engineering community exist that consider the use of natural language processing and semantic annotation technology to assist requirements engineers [62, 63, 64, 65].

For instance, Kof [66] employs a similar approach to ours in that he uses NER techniques to identify entities from requirements documents that are subsequently used for building message sequence charts from requirements documents. Despite the fact that these works do not consider variability and family modeling, they can be considered as important foundational predecessors to our work.

11. Concluding Remarks

Software product line engineering is a software reuse paradigm that aims at the rapid development of new domain-specific applications at a lower cost. Software product lines focus on developing formal representations of a family of applications for a domain; therefore, require a rigorous and in-depth analysis of a domain through their domain engineering lifecycle. This lifecycle will need to carefully sift through all of the available information related to the legacy and future applications of the domain and capture both variable and common aspects/features of these applications; a process which involves the consideration of a multitude of domain documents. In this paper, we have proposed a decision support platform that assists the domain analysts within the domain engineering lifecycle.

The decision support platform automatically processes the domain documents and provides useful information at the disposal of the domain analysts that can help them in their decision making process. Our empirical investigations show that the proposed platform is able to enhance the performance of the domain analysts in terms of accuracy and coverage. This means that the employment of the decision support system is able to positively influence the quality of the domain engineering lifecycle artifacts. It should be noted that our experiments did not show any meaningful impact, either positive or negative, on the time spent by the domain analysts in the domain engineering lifecycle. Furthermore, we have discussed how the decision support platform provides support for the different steps of the domain engineering lifecycle and hence can be integrated within that lifecycle and be employed by practitioners. We have also described how our decision support platform provides support for four main characteristics, namely *diligence*, *collaboration*, *knowledge transfer* and *traceability*.

As future work, we are interested in extending the work presented in this paper in several directions: 1) Section 5 shows how each of the processes within the domain engineering lifecycle are supported by the decision support platform. It can be seen that the last two processes receive less support than the other processes. We would like to consider augmenting the current support with means to assist domain analysts with the process of actually designing the common core architecture; 2) Based on the empirical evaluations, the decision support platform does not reduce the time spent by the domain analysts. We are interested in exploring additional means such as text summarization to reduce the amount of time spent by the domain analysts without negatively impacting coverage and accuracy; 3) Our current empirical work is based on a holistic assessment setting that evaluates the decision support platform as a whole. We would like to devise a hybrid assessment approach that would allow the evaluation of the platform as a whole and also reveal the weak

points of each of the individual components for the purpose of further evaluating and refining our work; 4) the degree of recommendation to domain analysts provided by the current form of the decision support platform is that it only provides a list of feature and integrity constraints and also a feature graph visualization that resembles the closeness of the features, but it does not provide any suggestions to the domain analysts wrt the hierarchical structure of the feature model. We are interested in exploring the possibility of creating a technique that would recommend relationships between features such as sibling and parent relations to the domain analysts.

Notes

1. For background information on semantic annotation, readers are encouraged to see [67].
2. The complete source code for the application introduced in this paper is accessible at <http://ebagheri.athabascau.ca/spl/ASE/>.
3. these documents have all been taken from relevant Wikipedia entries on this subject matter.

References

1. Pohl, K., Böckle, G., van der Linden, F.: *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer (2005)
2. Lee, K., Kang, K., Lee, J.: Concepts and guidelines of feature modeling for product line software engineering. *Lecture Notes in Computer Science* **2319** (2002) 62–77
3. Weiss, D.M., Clements, P.C., Kang, K., Krueger, C.: Software product line hall of fame. In: *SPLC '06: Proceedings of the 10th International on Software Product Line Conference*, Washington, DC, USA, IEEE Computer Society (2006) 237
4. McGregor, J.D., Muthig, D., Yoshimura, K., Jensen, P.: Guest editors' introduction: Successful software product line practices. *IEEE Software* **27** (2010) 16–21
5. Kang, K., Sugumaran, V., Park, S.: *Applied software product-line engineering*. Auerbach Publications (2009)
6. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration using feature models. *International Conference on Software Product Lines (SPLC)* (2004) 266–283
7. Czarnecki, K., Eisenecker, U.: *Generative programming*. Springer (2000)
8. Sentz, K., Ferson, S.: Combination of evidence in Dempster-Shafer theory. *Sandia National Laboratories - SAND 2002-0835* (2002)
9. Bagheri, E., Gasevic, D.: Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal* **19** (2011) 579–612
10. Henderson-Sellers, B.: *Object-oriented metrics: measures of complexity*. Prentice Hall (1996)
11. Bagheri, E., Ghorbani, A.A.: An exploratory classification of applications in the realm of collaborative modeling and design. *Inf. Syst. E-Business Management* **8** (2010) 257–286
12. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice* **10** (2005) 7–29
13. Babar, M., Chen, L., Shull, F.: Managing variability in software product lines. *Software, IEEE* **27** (2010) 89–91
14. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: *Feature-oriented domain analysis (FODA) feasibility study*. Carnegie Mellon University, Software Engineering Institute (1990)
15. Tessier, P., Gérard, S., Terrier, F., Geib, J.: Using variation propagation for model-driven management of a system family. *International Conference on Software Product Lines* (2005) 222–233
16. Lopez-Herrejon, R., Batory, D.: A standard problem for evaluating product-line methodologies. *International Conference on Generative and Component-Based Software Engineering* (2001) 10–24

17. Tjong Kim Sang, E., De Meulder, F.: Introduction to the conll-2003 shared task: Language-independent named entity recognition. In: Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4, Association for Computational Linguistics (2003) 142–147
18. Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. *Linguisticae Investigationes* **30** (2007) 3–26
19. Zhou, G., Su, J.: Named entity recognition using an hmm-based chunk tagger. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics (2002) 473–480
20. McCallum, A., Li, W.: Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In: Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4, Association for Computational Linguistics (2003) 188–191
21. Florian, R., Ittycheriah, A., Jing, H., Zhang, T.: Named entity recognition through classifier combination. In: Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4, Association for Computational Linguistics (2003) 168–171
22. Finkel, J., Grenager, T., Manning, C.: Incorporating non-local information into information extraction systems by gibbs sampling. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics (2005) 363–370
23. Finkel, J., Kleeman, A., Manning, C.: Efficient, feature-based, conditional random field parsing. *Proceedings of ACL-08: HLT* (2008) 959–967
24. Zhong, H., Zhang, L., Xie, T., Mei, H.: Inferring resource specifications from natural language api documentation. In: Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, IEEE Computer Society (2009) 307–318
25. Uschold, M., Gruninger, M.: Ontologies: Principles, methods and applications. *The Knowledge Engineering Review* **11** (1996) 93–136
26. Miller, G.: Wordnet: a lexical database for english. *Communications of the ACM* **38** (1995) 39–41
27. Antoniou, G., Harmelen, F.: Web ontology language: OWL. *Handbook on ontologies* (2009) 91–110
28. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. *The Semantic Web-ISWC 2006* (2006) 30–43
29. Kiefer, C., Bernstein, A., Stocker, M.: The fundamentals of isparql: A virtual triple approach for similarity-based semantic web tasks. *The Semantic Web* (2007) 295–309
30. Tappolet, J., Kiefer, C., Bernstein, A.: Semantic web enabled software analysis. *Web Semantics: Science, Services and Agents on the World Wide Web* **8** (2010) 225–240
31. Lin, D.: An information-theoretic definition of similarity. In: Proceedings of the 15th international conference on machine learning. Volume 1., Citeseer (1998) 296–304
32. Resnik, P.: Using information content to evaluate semantic similarity in a taxonomy. In: Proceedings of IJCAI-95. (1995)
33. Cox, M., Cox, T.: Multidimensional scaling. *Handbook of data visualization* (2008) 315–347
34. Barrett, D.: *MediaWiki*. O’Reilly Media (2008)
35. Grace, T.: Wikis as a knowledge management tool. *Journal of Knowledge Management* **13** (2009) 64–74
36. Harsu, M.: A survey on domain engineering. Tampere University of Technology (2002)
37. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: International joint Conference on artificial intelligence. Volume 14., Citeseer (1995) 1137–1145
38. Davis, J., Goadrich, M.: The relationship between precision-recall and roc curves. In: Proceedings of the 23rd international conference on Machine learning, ACM (2006) 233–240
39. Bradley, A.: The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition* **30** (1997) 1145–1159
40. Hovy, E., Marcu, D.: Automated text summarization. *The oxford handbook of computational linguistics* (2005) 583–598
41. Genero, M., Poels, G., Piattini, M.: Defining and validating metrics for assessing the understandability of entity-relationship diagrams. *Data Knowl. Eng.* **64** (2008) 534–557

42. Serrano, M.A., Calero, C., Sahraoui, H.A., Piattini, M.: Empirical studies to assess the understandability of data warehouse schemas using structural metrics. *Software Quality Journal* **16** (2008) 79–106
43. Buitelaar, P., Cimiano, P., Magnini, B.: Ontology learning from text: An overview. *Ontology learning from text: Methods, evaluation and applications* **123** (2005) 3–12
44. Weiss, S.: *Text mining: predictive methods for analyzing unstructured information*. Springer-Verlag New York Inc (2005)
45. Basili, V., Shull, F., Lanubile, F.: Building knowledge through families of experiments. *IEEE Transactions on Software Engineering* **25** (1999) 456–473
46. Mendonca, M., Branco, M., Cowan, D.: S.p.l.o.t.: software product lines online tools. In: *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, New York, NY, USA, ACM (2009) 761–762
47. Janota, M., Kiniry, J.: Reasoning about feature models in higher-order logic. In: *Software Product Line Conference, 2007. SPLC 2007. 11th International*. (2007) 13–22
48. Benavides, D., Trinidad, P., Ruiz-Cortes, A.: Automated reasoning on feature models. In: *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005. Volume 3520.*, Springer (2005) 491–503
49. Batory, D.: Feature models, grammars, and propositional formulas. *Software Product Lines* (2005) 7–20
50. Boskovic, M., Bagheri, E., Gasevic, D., Mohabbati, B., Kaviani, N., Hatala, M.: Automated staged configuration with semantic web technologies. *Int. J. Soft. Eng. Knowl. Eng.* **20** (2010) 459–484
51. Bagheri, E., Noia, T.D., Ragone, A., Gasevic, D.: Configuring software product line feature models based on stakeholders' soft and hard requirements. In: *The 14th International Software Product Line Conference*, Springer (2010)
52. Benavides, D., Segura, S., Trinidad, P., Ruiz-Cortes, A.: FAMA: Tooling a framework for the automated analysis of feature models. In: *Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*. (2007) 129–134
53. Bagheri, E., Noia, T., Gasevic, D., Ragone, A.: Formalizing interactive staged feature model configuration. (*Journal of Software Maintenance and Evolution: Research and Practice*, (in press))
54. Mendonca, M., Wasowski, A., Czarnecki, K., Cowan, D.: Efficient compilation techniques for large scale feature models. In: *Proceedings of the 7th international conference on Generative programming and component engineering*, ACM New York, NY, USA (2008) 13–22
55. Wang, H., Li, Y., Sun, J., Zhang, H., Pan, J.: Verifying feature models using OWL. *Web Semantics: Science, Services and Agents on the World Wide Web* **5** (2007) 117–129
56. Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., Mirakhorli, M.: On-demand feature recommendations derived from mining public product descriptions. In: *Proceeding of the 33rd international conference on Software engineering*, ACM (2011) 181–190
57. Weston, N., Chitchyan, R., Rashid, A.: A framework for constructing semantically composable feature models from natural language requirements. In: *Proceedings of the 13th International Software Product Line Conference*, Carnegie Mellon University (2009) 211–220
58. Sampaio, A., Rashid, A., Chitchyan, R., Rayson, P.: EA-miner: towards automation in aspect-oriented requirements engineering. *Transactions on aspect-oriented software development III* (2007) 4–39
59. Niu, N., Easterbrook, S.M.: Extracting and modeling product line functional requirements. In: *RE*. (2008) 155–164
60. John, I.: Capturing product line information from legacy user documentation. *International Conference on Software Product Lines - SPLC 2006* (2006) 127–159
61. She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K.: Reverse engineering feature models. In: *Proceeding of the 33rd international conference on Software engineering. ICSE '11*, New York, NY, USA, ACM (2011) 461–470
62. Ryan, K.: The role of natural language in requirements engineering. In: *Requirements Engineering, 1993.*, Proceedings of IEEE International Symposium on, IEEE (1992) 240–242

63. Gervasi, V., Nuseibeh, B.: Lightweight validation of natural language requirements. *Software: Practice and Experience* **32** (2002) 113–133
64. Fabbrini, F., Fusani, M., Gnesi, S., Lami, G.: The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In: *Software Engineering Workshop, 2001. Proceedings. 26th Annual NASA Goddard, IEEE* (2001) 97–105
65. Gervasi, V., Zowghi, D.: Reasoning about inconsistencies in natural language requirements. *ACM Transactions on Software Engineering and Methodology (TOSEM)* **14** (2005) 277–330
66. Kof, L.: Translation of textual specifications to automata by means of discourse context modeling. *Requirements Engineering: Foundation for Software Quality* (2009) 197–211
67. Uren, V., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E., Ciravegna, F.: Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Web Semantics: Science, Services and Agents on the World Wide Web* **4** (2006) 14–28