

# Developing Families of Software Services: a Semantic Web Approach

Marko Bošković, Bardia Mohabbati, Nima Kaviani, Mohsen Asadi, Jeff Rusk,  
Dragan Gašević, Ebrahim Bagheri, and Marek Hatala

**Abstract**—Current experience in the development of service-oriented systems indicates that tasks such as discovery, systematic reuse, and appropriate composition of services are difficult and error prone. For addressing these issues, the application of Software Product Line Engineering (SPLE) appears to be a promising approach. SPLE promotes systematic reuse in the development of systems with similar requirements. SPLE promotes reuse by development of comprehensive sets of systems (aka software families) as a whole (domain engineering), and later configuring them according to the functionality requested by the stakeholders (application engineering). Furthermore, the reuse of services that are part of a software product line is enhanced, since each time a member of SPL is instantiated and deployed, services that are used in the instantiated member are reused. The research community have recognized and proposed several promising solutions to the development of service-oriented systems based on the SPLE principles; however, there have been little results that report on concrete tools that automate and thus, reduce the amount of efforts needed for completing specific tasks of a development process. In this paper, we first introduce a process for developing service-oriented families. Then, we investigate the use of Semantic Web technologies and ontologies to support service discovery, context description, and verification in domain engineering; and staged configuration and service composition generation in application engineering of the introduced development process. We evaluate the effectiveness of our Semantic Web-based proposals in terms of optimizing the amount of efforts necessary for completing relevant tasks in the discussed development process.

**Categories and Subjects**—D.2.2. [Software Engineering] Design Tools and Techniques

**General Terms**— Design

**Keywords**— Families of Service-oriented Systems, Software Product Lines, Semantic Web, Ontologies, Description Logics



## 1 INTRODUCTION

The prevalence of the World Wide Web and its supporting applications has contributed to the development of exciting opportunities for Service Oriented Architectures (SOA) as a novel paradigm for distributed computing. SOA moves the development of software systems from an independent and isolated process where reuse is based on available local libraries and components, to the development of network applications that reuse independent and publicly available functional entities called services. Services are self-describing, platform independent, computational entities that can be described, published, discovered and programmed using standard protocols [56]. Public availability and “openness” of services offer a large potential for the rapid development of low-cost, distributed applications, with high level of reuse-based functionality. However, the current state of the art shows that there is still no systematic approach for reuse of available services.

Software Product Line Engineering (SPLE) [42], is an emerging approach for software reuse. SPLE promotes the development of sets of software systems that share many common features. They are developed as a whole, from shared reusable components and assets for a specific domain. The process of development of an SPL is performed through its *domain engineering* lifecycle. Typically, in this lifecycle, features of the system are captured with feature models. Later, particular software systems are developed in the *application engineering* lifecycle by selecting the desired product features from the feature model.

Recently, the integration of SPLE and SOA is being explored [7][30][34]. By applying SPLE paradigm to the development of SOA systems, available services can be grouped and systematically reused. However, there is still lack of support for systematic discovery and composition in the process of *domain engineering* and customization, and configuration and verification in the process of *application engineering*.

In this paper, we propose a method, concentrated on use of ontologies and Semantic Web technologies, to overcome previously mentioned shortcomings. The main hypothesis of the methodology is that ontologies as shared conceptualizations of the domain can enhance semantics-based discovery during the domain engineering, and reuse-based development in application engineering. Furthermore, we also hypothesize that the employment of a shared conceptualization of the context of use between domain engineers and application engineers can be effective in reducing the time required for the configuration process. Finally, we also propose that Description Logics (DL) as one of the essential elements of the Semantic Web can support verification of SPLs in the process of application engineering.

As a support for our method, we provide the following contributions:

- (a) Introduction of semantically annotated feature models for enrichment with domain specific semantics;
- (b) Discovery of services based on semantically enriched feature models;
- (c) Semi-automatic configuration of feature models

depending on context of use;

- (d) Verification of configured feature models using standard Description Logics (DL) reasoning mechanisms;
- (e) Transformation of configured feature models into executable service compositions.

The rest of the paper is organized as follows: the next section gives background on SOAs and SPLs. Section 3 introduces our proposed approach in detail. Section 4 describes contributions to support the proposed methodology. In Section 5, our approach has been thoroughly evaluated and the results have been analyzed and described. The paper is then concluded by comparative related works, and the discussion of future directions of research.

## 2 BACKGROUND

This Section gives an introduction to the main concepts of SPLs (Sec. 2.1), and SOAs (Sec. 2.2).

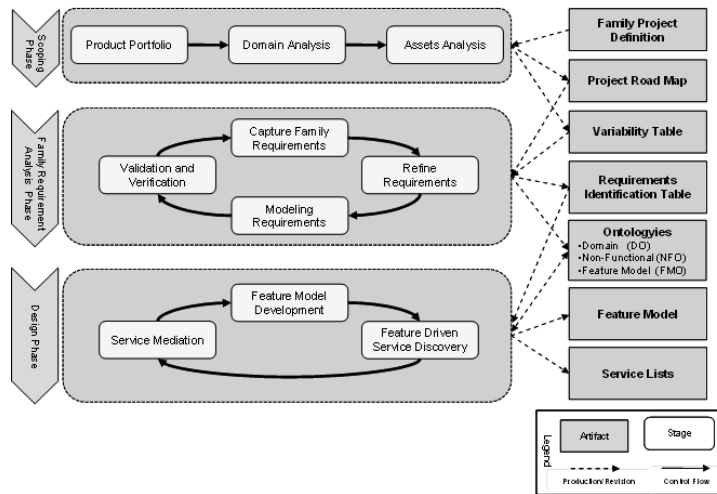


Figure 1. Domain engineering lifecycle for families of service-oriented architectures

### 2.1. Software Product Lines

The Software Product Line Engineering (SPLE) discipline [42] provides methods for managing variability and commonality of core software assets in order to facilitate the development of families of software-intensive products. In this context, software families are characterized by a set of features that are shared by all the individual products of a family [12]. SPLE empowers the derivation of different product family applications (aka, family members) by reusing the realized product family assets such as common models, architectures, and components. The development of software product families is performed by conducting the steps in the *domain engineering* lifecycle, which is followed by the *application engineering* lifecycle. Domain engineering identifies the commonalities and variability of the core assets, which are exploited during the *application engineering* phase where individual applications are assembled and customized. During application engineering, variability points get disambiguated by choosing the proper set of features and by combining them with the common software artifacts to derive the desired product family members matching the specifications of the

software family.

*Feature modeling*, as an important technique to model variability, is employed for specifying the problem space of a software product family. Furthermore, feature models have been widely adopted in the domain of requirements engineering to facilitate the process of constructing and managing the reference representation of a family of software systems and their corresponding implementations. Feature models depict both formally and graphically the expression of relations, constraints, and dependencies of the software products family attributes, aka features [31]. The main relations defined in feature models are [4]:

- **Mandatory** : a unary relation indicating that the involved feature must be present in the final configuration;
- **Optional**: a unary relation denoting an optional feature;
- **Or**: a relationship that affects two or more features and makes sure that at least one of the involved feature is selected in the final configuration;
- **Alternative**: a relationship over two or more features

where only one of the involved features can be selected.

Additionally, integrity constraints can be defined in feature models. The constraints define mandatory conditions that must always hold in every feature model configuration. Most commonly used integrity constraints are *includes* (presence of a feature A implies the presence of a feature B in a valid feature model configuration) and *excludes* (mutual exclusion between A and B).

### 2.2. Service-oriented Architectures

Service Oriented Architecture (SOA) is a standardized set of methods for addressing the requirements of *loosely coupled*, *standard based*, and *protocol independent* distributed computing [15]. In the context of SOA, a service is an *autonomous*, *platform-independent* functionality that can be *described*, *published*, *discovered*, *orchestrated*, and *programmed* using standard protocols [51]. The most popular types of services are Web services, which are described using the Web Service Description Language (WSDL), accessed through SOAP, and published and discovered in Universal Description, Discovery and Integration registry (UDDI). For the scope of this paper, of special interest are so-called Semantic Web services, which are defined as the

augmentation of Web service descriptions through ontology-based annotations in order to facilitate the higher automation of service discovery, composition, invocation, and monitoring on the Web. Some well-known efforts in that context are WSMO [43], OWL-S [33], and WSDL-S [39].

### 3. THE PROPOSED METHOD

The process model of a software product line comprises of domain engineering (i.e., the process through which the product line architecture, common assets, and variants are developed) and application engineering (process for developing the product line application instances) lifecycles. Employing SPLE in different domains requires the specialization of the domain engineering and application engineering lifecycles. In this paper, we propose a process model for developing families of software services. In the remainder of this section the main phases and activities as well as the product artifacts are described.

#### 3.1. Domain Engineering

Domain engineering aims at discovering, organizing and implementing the common assets of a product family. Moreover, specifying the scope of a family and describing the variability of the models is achieved during the domain engineering lifecycle. The reference architecture, reusable assets, variability models are the outputs of this lifecycle. Figure 1 illustrates the phases and stages of the domain engineering lifecycle.

Domain engineering starts with the *Scoping* phase, which determines which products (software services) will be a part of the family and what commonalities and variability will be considered. Scoping the family is performed in three stages [46]. First, *Product portfolio scoping* uses market inputs and determines the range of products that shall be supported; then *domain scoping* identifies the major functional areas (domains) which belong to the current product line by using the basis provided in the previous stage. Finally, *Asset analysis* precisely defines functionality components that should be supported by the family. The range of variability and the number of potential products are decided using techniques such as variability table – a table whose rows represent variables and the columns shows elements such as types of variability, possible variants, and status of variable (closed or opened) [61][62]. A product-line roadmap is produced as the output of this

phase.

The *Family Requirements Analysis* phase captures requirements and develops a requirements model containing unique and unambiguous definitions for each requirement. Customer viewpoints, project vision and documents, the product-line roadmap, and variability ranges serve as the inputs to this phase. First, the stakeholders' (functional and non-functional) needs are analyzed and documented in the *capture stakeholder requirements* stage. The non-functional requirements include not only stakeholders' requirements such as performance, reliability, and availability of software services but also final stakeholders' extra-functionality needs such as quality of service. Then, the *Refine Requirements* stage aggregates and decomposes the requirements in order to make their specification more understandable and clearer. Next, the *Requirement Modeling* stage applies requirements modeling techniques such as use-case modeling to develop the requirement model from requirements specifications. This model contains the functional and non-functional requirements and will be utilized in later phases. The consistency and completeness of the requirements model and matching with stakeholders' needs are checked in the *Validation and Verification* stage.

The *Family Design* phase aims at providing solution structures for the family. A solution structure includes a feature model as well as service-based implementation of the features (i.e., the services that satisfy the features functionality and non-functionality requirements). The *Developing Feature Model* stage manages the common and variable functionalities by representing them in the feature model structure. By analyzing the requirements model, the functionalities of the system (i.e., the features), their granularity level, and relations are identified and shown in the feature model.

Additionally, concepts from a non-functional property ontology corresponding to non-functional requirements are appended to features as annotation information. In this paper, we refer to various types of non-functional property ontologies. Such ontologies represent an explicit and formal representation of shared conceptualizations of various domains of non-functional properties. An example of such non-functional property ontologies is the delivery context ontology. This ontology is used to specify non-functional requirements of each feature. These non-functional requirements have to be satisfied by target deployment

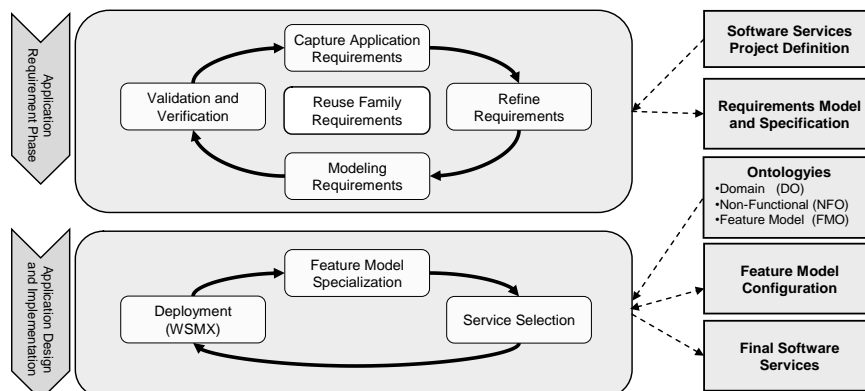


Figure 2. Application engineering lifecycle for families of service-oriented architectures

platforms in concrete applications. Moreover, developers can add concepts from a domain ontology to the features so as to enable the semantic discovery of services for each feature.. The use of both these ontologies is very important for the SOA systems, as they allow different collaborating parties to formally specify and agree upon terminologies, which will then describe software shared software artifacts. Thus, a hypothesis is that the level of mutual understanding will be increased. In this phase, the feature model is automatically transformed into an ontological representation, called the Feature Model Ontology (FMO), to provide reasoning support on the feature model (i.e., consistency checking as per [52] and automated staged configuration as described in Sect. 4.2) and service discovery (Sect. 4.3).

In this stage, we advocate reusability of services as a possible way to reduce the costs of the family implementation. In our methodology to increase the level of service reusability, our first option for implementation of the family assets is to try to discover appropriate services that satisfy the functional and non-functional requirements of the assets. The *Feature-driven Service Discovery* stage extracts the semantic information including non-functional property ontology concepts and domain ontology concepts (functional goals) from leaf features and constructs relevant queries for finding the right services that can implement these features. Then the queries are submitted to a service discovery engine (e.g., Seekda) in order to find services which fulfill functional and non-functional aspects of features. Since non-functional properties are represented as annotation properties, a feature may have various non-functional properties with different values. Therefore, more than one service with the same functionality and different non-functional properties may be retrieved for each feature. Since discovered services might have their descriptions in various formats and be annotated by different ontologies, the *Service Mediation* stage needs to be performed. In this stage, we fully rely on the state of the art in the area [43] and do not provide further discussion on this matter in the paper.

### 3.2. Application Engineering

The application engineering lifecycle aims at creating concrete software services for a target application (e.g., a member of the family) by utilizing the reusable assets created in the domain engineering lifecycle. The application engineering lifecycle is carried out every time a new product is developed. The lifecycle starts by a request for a concrete software system belonging to a family, and then the requirements of target software services are captured from the stakeholders' needs (e.g., application documents). Next, the reference architecture is adapted and common assets as well as the variants corresponding to the application requirements are selected. The application engineering process (i.e., main phases and artifacts) is illustrated in Figure 2.

The *Application Requirement Analysis* phase receives the definition of a target software application and develops its related requirements model. In order to produce the requirements model, it utilizes the family requirements

model developed in the domain engineering lifecycle. Similar to the domain engineering life cycle, the application requirements engineering phase starts by eliciting and documenting the software application requirements and then refining and further clarifying them. Finally, the application requirements model is developed and validated. The activities are similar to the activities in the family requirements analysis with the following differences: They concentrate on the one software application, so they do not deal with *variability* in the family. Moreover, the activities in this phase use the family requirements model as a reference model.

The *Application Design* phase aims at developing the application by selecting the most appropriate set of features from the feature model through a staged configuration process. The *staged configuration* process [10] starts from the feature model and carries out successive specializations (including selecting some of the functionalities; determining their non-functional properties as well as their values; and finally decides on the appropriate service(s) for each feature with respect to both functional and non-functional properties) to achieve the final products' representation. In essence, the staged configuration process would limit the space of the product family to the space most relevant for the current application that is being built. Finally, the feature model configuration is transformed into a service composition.

## 4. METHOD SUPPORTING CONTRIBUTIONS

In this section, we introduce a set of algorithms and conceptual solutions that we developed in order to support the proposed process from the previous section. Particularly, our contributions support family design where ontologies are used for semantic annotation and search of services, and application design and implementation where we introduce automated staged configuration, and transformation of the configured produces into services. Before diving into the technical details, we first introduce a running case study which is be used throughout the rest of the section to demonstrate the proposed conceptual solutions.

### 4.1. Running Case Study

In order to demonstrate the proposed method, the EShop case study depicted in Figure 3 is used. EShop is a web application for shopping over the web, which has been widely used as a standard problem in the SPLE community [27]. The main control flow of this application consists of three subprocesses: **OrderManagement**, **Payment**, and **Shipment**, all represented with their corresponding features in the feature diagram. Order management consists of the selection of an item from the **Catalog** and later computation of costs (**CostComputation**). After cost computation, payment takes place (**Payment**). A payment is performed with various payment methods (**PaymentMethod**). In this EShop application, it is predicted that the payment can be done with a Debit Card (**Debit**), Master and/or Visa Credit Cards (**Master**, **Visa**, **CreditCard**), or payment can be performed with cash (**Cash**). Alongside the payment, fraud detection is also

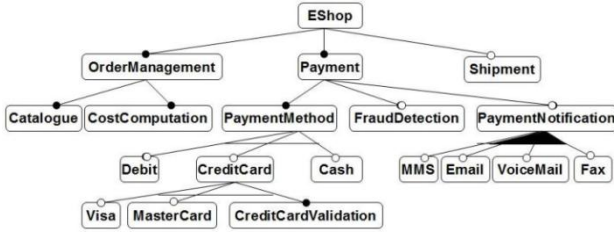


Figure 3. EShop feature model

carried out (**FraudDetection**). Also, when a credit card is used, it is validated (**CreditCardValidation**). After the payment, the customer always receives a payment notification (**PaymentNotification**). The payment notification can be performed either with SMS, Fax, VoiceMail, or Email (**SMS**, **Fax**, **VoiceMail**, **Email**). As the purchase order considers first payment at the time of the receipt of the item, it does not need the fraud detection. However, as one more security measure, fraud detection must be done when a credit or debit card is used. Therefore, there exist two include constraints: 1) between features **Debit** and **FraudDetection**, and 2) between **CreditCard** and **FraudDetection**.

## 4.2. Feature Modeling for Service System Families

Since the initial feature modeling methodology introduced in FODA [48], there was a need for verification of configured features. Currently, several approaches for verification, mostly based on interpretation of feature models as propositional formulas exists such as those presented in [4][32]. Wang et al. [52] introduced a Description Logic (DL) based representation of feature models and feature model configurations, in order to use automatic DL reasoners for feature configuration verification and debugging. In this paper, we refer to the DL based representation of a feature model as Feature Model Ontology (FMO). In their approach, feature models and configurations are represented as follows.

Let  $F_1, \dots, F_n$  be a set of features in the feature model. And let  $F_i, \dots, F_n$  be their corresponding terms in a DL feature model representation. This convention, where representation of features in the feature model are **bold**, while DL features' representations are written in *italics* is used in the rest of the paper. Each feature of the feature model is represented in description logic as:

$$\begin{aligned}
 F_i &\sqsubseteq T, & hasF_i &\sqsubseteq \text{ObjectProperty}, \\
 RuleF_i &\sqsubseteq T, & T &\sqsubseteq \exists hasF_i. OntF_i, \\
 RuleOntF_i &\sqsubseteq \exists hasF_i. F_i, & & \text{for } 1 \leq i \leq n \\
 F_k &\sqsubseteq \neg F_j & & \text{for } 1 \leq j, k \leq n
 \end{aligned}$$

For example, the **OrderManagement** feature is represented in this FMO as:

$$\begin{aligned}
 OrderManagement &\sqsubseteq T, \\
 RuleOrderManagement &\sqsubseteq T, \\
 hasOrderManagement &\sqsubseteq \text{ObjectProperty}, \\
 T &\sqsubseteq \exists hasOrderManagement. OrderManagement, \\
 RuleOrderManagement &\sqsubseteq \exists hasOrderManagement. OrderManagement,
 \end{aligned}$$

It can be seen that each feature from the feature model is described with two classes in the FMO. One class is representing the feature itself. The other one is a "rule"

class, and it represents interrelations between that feature and the other features of that model. Both of these classes are depicted in Figure 4. Modeling of the interrelations is out of the scope of this paper. An interested reader is referred to Wang et al [52].

In order to enrich the feature model with additional, community shared and defined information, such as additional domain-specific (functional) descriptors or non-functional properties (e.g., execution platform characteristics or provision of some services), we use the power of OWL to import and use concepts of external ontologies. The approach for specifying this additional information is illustrated in Figure 4.

In the example in Figure 4, we have added information about the requests that some services satisfy and the execution platform characteristics and additional functional description for the purpose of search. The requests for some service provision and/or execution platform characteristics are features' hard requirements. Concepts for defining features' hard requirements are defined in the features' hard requirements ontology (HRO). HRO imports a publicly available ontology, in our example from Figure 4, the Delivery Context Ontology [17]. Features' hard requirements are represented by the addition of properties to their representatives in FMO. In our case, hard requirements of the **Debit** and **CreditCard** features for existence of X509 Certificate is specified by adding the property *HRPropX509Certificate* to the FMO class *CreditCard*. It is specified that the concept of interest for our

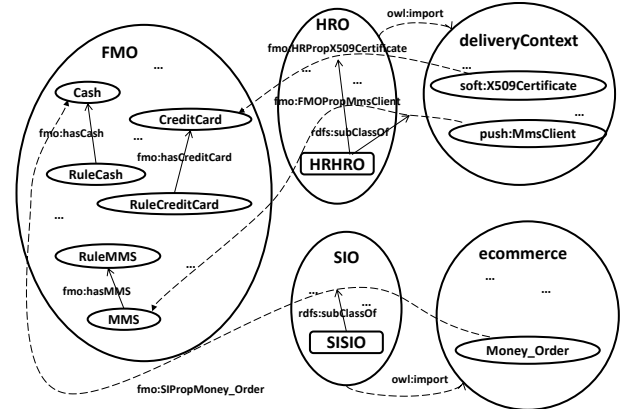


Figure 4. Integration of additional information to DL representation of the feature model

hard requirements ontology is X509 Certificate. Furthermore, in order to be able to send MMS notifications to the clients, the feature requires provision or existence in the execution environment a MMS push client. This is specified by adding the property *HRPropMMSClient* to the class *MMS* of the FMO. In the same way, additional information on functional properties, or semantics information, can be added. In Figure 4, the semantics of feature **Cash** is additionally described by adding the property *SIPropMoney\_Order*, having in the range class *Money\_Order* of the ecommerce domain ontology [14], to describe that when the item is paid by cash, it is paid through a money order.

## 4.3. Feature-driven Service Discovery

Modeling and annotation of features are followed by

searching and discovering services that can best fulfill the functionalities required by the specified features. The process of feature discovery is a first step in which feature abstractions start to turn into concrete software services and modules.

For feature models to get interpreted to concrete software modules, we treat each feature and its respective annotations as *service selection queries* during the discovery phase. Basically, features and their corresponding annotations provide us with a set of functional and nonfunctional descriptive keywords on what the desired software services should be, how they should behave (i.e., functional properties), and what criteria they should meet (i.e., nonfunctional properties). Hence, a strategy is needed to effectively convert these descriptive keywords into service selection queries. Several strategies can be adopted (e.g., text-based, structural, and logic-based [37][26]) in order to find and match a service against the requirements of a feature query. Even though logic-based and structural search and discovery of services demand for more concrete specification of feature requirements (in terms of the pre- and post-conditions, feature goals, etc.), a text-based service selection mainly relies on keywords or tags describing a feature in order to search and fetch the set of matching services.

Relying on conceptual annotation of feature models using ontologies, we can use the set of concepts (i.e., keywords) from the ontologies annotating the feature model in order to perform a text-based search and discovery of services. Each feature might be annotated with one or more concepts from the ontology. In case there is only one concept describing a feature, the search space is going to be too broad to be helpful of finding proper services when performing service discovery. However, in case of multiple concepts annotating a feature, the search space associated with each concept from the ontology might be in conflict with the search space associated with another concept from the ontology describing the same feature. Thus, a text-based search and discovery strategy will fail if the search spaces for discovering service are scattered or if the set of concepts describing a feature are semantically orthogonal. Here, we refer to the concepts that are not semantically inline such as concept *Agency with Phone Number* as its sub-concept. Consequently, there needs to be a strategy to unify or at least bring together the search spaces associated with each ontology concept to the search space associated with other concepts in order to take the maximum precision out of the discovery effort.

In order to have a more descriptive feature model for our product family we first need to find a domain ontology (e.g., using Swoogle<sup>1</sup>) that best describes the feature model, and then use the domain ontology to annotate the features from the feature model. Benefitting from the use of the domain ontology, we effectively relate the concepts of the features in the feature model to the concepts in the ontology. However, a single concept from the ontology still is not descriptive enough to be used as a keyword for service discovery. Hence, we try to expand on the search

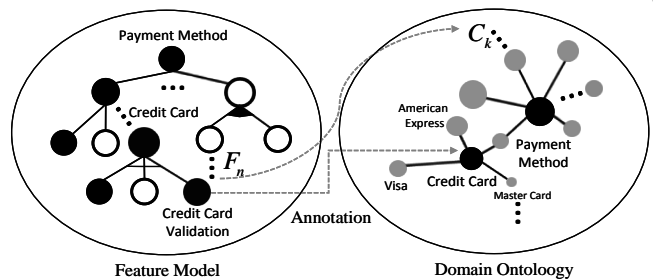


Figure 5. Annotating the feature model with the e-Commerce ontology

space on the search space for each concept from the ontology by not only taking the concept itself into account during the discovery phase, but also by looking into its set of immediate successors as potentially descriptive concepts to represent (at least partially) the intention behind the annotated feature. We limit ourselves only to immediate successors of a concept in the ontology, as ancestors deeper down in the ontological specification of a concept do not necessarily fall within the same semantic category as of their high-level predecessor. Not only does this strategy introduce a solution to dealing with situations where there are not enough concepts annotating a feature (by expanding on the number of keywords possibly describing a service), but also tries to find the possible intersection between the search spaces associated with each semantic concept from the ontology. However, not only does expanding on the set of concepts representing a feature help with expanding the search space for a feature, but on the contrary, it may reduce the search space if the keywords are not properly used to generate selection queries. Furthermore, even in the set of immediate successors of a concept from the ontology, there might be sub-concepts with diverging semantic meanings, either splitting the search space or making it improper to look into (see Figure 3). To overcome these problems, we use the following algorithm to produce proper search spaces during the service discovery and selection phase:

**Definition 1:** Let  $c$  be an ontology concept annotating a feature from the feature model and  $sc_1, sc_2, \dots, sc_n$  be the set of sub-concepts of  $c$ . We define the search queries,  $SQ$ , as the power-set of all combinations of these keywords, i.e.,

$$SQ = \{\{c\}, \{sc_1\}, \{sc_2\}, \dots, \{sc_1, sc_2\}, \dots, \{c, sc_1, \dots, sc_n\}\}$$

where

$$|SQ| = 2^{n+1} - 1$$

According to Definition 1, we consider potential search queries as combinations of a concept from the ontology annotating the feature model together with all its immediate sub-concepts. Nonetheless, when it comes to evaluating the returned services from each of the above queries, a service returned by more keywords should naturally be given a higher rank compared to a feature selected with fewer keywords. To reflect on this while evaluating the returned set of services, we give a weight to each of the query members from  $SQ$ .

**Definition 2:** Let  $m$  be a member of  $SQ$  with  $|m| = s$  and  $|SQ| = n$ . We define the weight for the result set returned by applying  $m$  as a query as follows:

$$w(m)_s = \frac{2 * s}{n * (n + 1)}$$

<sup>1</sup> <http://swoogle.umbc.edu>

which is equivalent to multiplying  $|m|$  by the inverse of the arithmetic progression of 1 to  $n$  for the size of concepts in  $SQ$ . The weighting algorithm is rather simple at this point, and more sophisticated functions (e.g., fuzzy function) could be used to obtain the weight of the queries. However, at this point our weighting strategy guarantees assigning more weight to services retrieved from a query with more keywords. Our weighting strategy is tailored towards the size of query sets in  $SQ$  such that,

$$w = \frac{1}{n} \sum_{i=1}^n u_i = 1$$

where  $u$  is a query set of the unique size in  $SQ$ . For each query  $m$ , its set of returned services is thus given the weight  $w(m)_s$ .

Once services returned from all queries are collected, we look into best hits for the set of discovered services associated to a feature. To do this, we go over all discovered services, identify the similar ones and sum up the weights for the similar services, which results in having weighted values for the query sets as follows:

**Definition 3:** Let  $SR_i$  be the set of weighted services returned by query set  $SQ_i$  with weight  $w_i$ ,  $p$  be the number of all sets of returned services by applying all queries, and  $f(s)$  equal to 1 if service  $s$  is in  $SR_i$  and 0 if  $s$  is not in  $SR_i$ . We calculate the overall weight for service  $s$  with respect to the set of all retrieved services, as follows:

$$f_i(s) * w_i$$

Of course, the best service choices are the ones with higher weights obtained from the above formula. The major value behind the above algorithm is that it makes the search space just wide enough for the most relevant services (from a text-based search point of view) to be discovered and given better ranks compared to services retrieved by less semantically related concepts from the ontology.

The set of discovered and weighted services, however, are not the finalized list of services to be pushed to the next phase of software development. These services, even though reflect the functional properties of desired services, do not necessarily fit the nonfunctional requirements of the feature, as specified in the feature model and the associated non-functional property ontology. In order to finalize the list of selected services, our algorithm iterates through the set of discovered services and filters out those conflicting the non-functional properties specified in the feature model. Among these non-functional properties, one can list *service availability*, *service response time*, *service reliability*, etc. [44]. The process of service discovery and service selection can certainly be further augmented by applying some extra matching algorithms to the set of discovered services. A hybrid approach similar to SAWSDL-MX [26] can be applied to the set of remaining services by conducting structural or logic-based matching of discovered services with desired functionality from the feature. This, however, requires the specification of features to be richer (similar to how they are defined by Rosenberg et al. [45]), but feature modeling, in its current state, does not allow us to go beyond text-based service discovery and apply a hybrid

service matching approach to the list of selected services.

It is worth noting that the number of queries grows exponentially based on the number of sub-concepts that an ontology concept holds. In order not to make the search and creation of service queries exhaustive, we randomly select up to 5 of the immediate successors of an ontology concept to be included in the set of keywords to create the power-set for feature queries. This makes the maximum number of queries for each concept from the ontology equal to  $2^6 = 64$  queries. This way we save on the processing time required to search and discover services associated to one ontology concept and its corresponding search queries.

To exemplify the algorithm, we leverage the running example described earlier in Section 4.1. Let us consider the feature *CreditCardValidation* from the feature model. The feature is annotated with the concept *CreditCard* from our e-commerce ontology (i.e., domain ontology) (see Figure 5). Since only one concept from the e-commerce ontology is used to annotate the feature in the feature model, relying solely on this concept in order to retrieve proper services to address the *CreditCardValidation* feature in the feature model ontology, probably, is not going to be very successful. The list of discovered services will be very broad and it is very likely that the set of discovered services are not going to be relevant to the functional requirements of the feature. By applying the strategy from Definition 1, we extend the set of possible search queries to immediate successors of the *CreditCard* concept from the ontology, i.e., *American Express*, *Visa*, and *Master Card*. Selecting all concepts from the immediate successors of *CreditCard*, the set of potential keywords to be used in order to search and discover a payment method service is going to be five keywords, resulting in 31 different queries.

Applying the strategy from Definition 2 would assign all the queries with only one keyword (e.g.,  $m = \{Credit Card\}$ ) a weight of  $w=0.066$ , while queries with three keywords will receive a weight of  $w=0.20$  and the query with all keywords (i.e.,  $m = \{CreditCardValidation, Credit Card, Visa, Master Card, and American Express\}$ ), a weight of  $w=0.33$ . We then propagate through all retrieved services from the above queries. It is worth noting that, a wide range of services returned from a query like  $m = \{CreditCardValidation\}$  are not very likely to be repeatedly retrieved by other queries, and hence the overall weight for most of those services returned from  $m = \{CreditCardValidation\}$  are going to receive of a weight of 0.066 which make those services options for elimination.

When it comes to applying NFRs to the set of selected services, the *CreditCardValidation* feature gets annotated by concepts such as *availability*, and *security* in order to guarantee a secure and quick transfer of money. The appropriate values from the non-functional properties are feed in to the system as instances of the non-functional property ontology and help with filtering undesired services. Services that successfully pass the criteria specified by non-functional properties are then proposed to the developers in order to go through and select from, enabling the developers to perform an ontology-based staged configuration of these services.

#### 4.4. Ontology-based Staged Configuration

Staged configuration, as described previously, consists of a series of feature model specialization steps, where a set of unfeasible features are removed in each step, yielding a more specialized feature model. Herewith, with the series of specialization steps, the decision space in the final configuration step is reduced to only those features feasible by the provided environment, policies and other constraints.

In order to automate the specialization step, we introduce an algorithm, which takes as an input a (specialized) feature model and information on requirements of features, as described in Section 4.2., and provided execution environment, stored in the knowledge base (KB), and removes the features unfeasible in that environment. In this paper, we provide only an illustration of the algorithm while for its details and formalization, we refer an interested reader to [64]. In order to illustrate the execution of the algorithm, we use the running example introduced in Section 4.1. Let us assume that the environment in which the EShop application should be deployed does not have an X509 Certificate, but has an MMS agent. In that case, the ontology HRO is extended with an instance *mmsp* of type *push:MMSClient*<sup>2</sup> and set defining provided environment consists of only the *push:MMSClient* concept. The outcome of this algorithm is a specialized feature model (see Figure 6) having only feasible features in its structure, i.e., the initial feature model without features whose hard requirements are not satisfied by platform. Furthermore, features which depend on removed features are also removed.

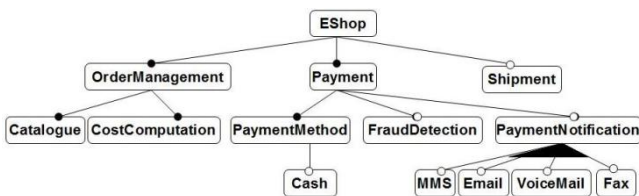


Figure 6. A specialization of the feature model from Figure 3 after applying the ontology-based algorithm

#### 4.5. Feature-driven Service Composition

Once the services are discovered (Section 4.3) and the configuration is derived (Section 4.4), the next step is to finalize the concrete application. In the context of service composition, the final step is to generate a software artifact which can coordinate the execution of the selected services in the configured feature model. Feature models are mostly used for capturing commonality and managing variability between members of a SPL. However, they can also be used for automatic generation of executable services compositions.

Here, we need to use an executable service composition language (c.f. Section 2.2 for examples). In our concrete case, we decided to use a semantic Web service language to be able to make use of our ontology-based approach. Two

existing frameworks which support service composition are OWL-S and WSMO. Existing literature reviews regarding the comparison between WSMO and OWL-S [49][28][3] show that WSMO is more powerful than OWL-S in both from service aspects (e.g. service description, discovery, orchestration and choreography) and the Semantic Web aspects (e.g. expressiveness, and incorporating Description Logic (DL) and higher order logics). With respect to our requirements and literature review results, we have chosen the WSMO model for our work<sup>3</sup>. WSMO is a semantic web services framework where service compositions are encoded in the WSML language. The conceptual model of WSMO describes all relevant aspects of web services – *ontologies*, *goals*, *web services*, and *mediators*. WSMO services are deployed onto WSMX, a framework for executing WSMO-based service compositions. WSMX has a tooling support for providing feedback about syntactical and semantic completeness of their service compositions.

In order to use a feature model for automatic generation of a choreography or orchestration of services, it is necessary to implement certain design rules within the feature model and resulting product configurations. To represent service composition within feature models, one possible approach is to introduce the conceptual ordering of services to the practice of feature modeling. Although the order of features on the same level within a feature model typically has no real meaning, in this case a certain style is implemented to ensure that the feature model is reflective of the general order of the potential flow of the business process. This order is, of course, reflected in greater detail within an executable service composition description, which is typically based on a formal model such as abstract state machines (ASM). The style incorporates the feature modeling methodology proposed by Montero *et al.* [38] as an initial base. Their basic composition rules are illustrated in Figure 7.

The figure relates the structure and order of business processes, in our case represented by service compositions, within a feature model to their transition rules in the ASM which form essential elements of formal choreography and orchestration descriptions in WSMO. In Figure 7, the first example shows that service A is invoked and changes the state of the ASM before service B can be invoked. In the second example, service A is invoked, changing the state of the ASM and service B may or may not be required or able to be invoked. In the third, service A is invoked and changes the state of the ASM and then both services B and C are invoked in parallel. In the fourth example, service A is invoked, changing the state of the ASM and then either B or C, but not both, are invoked. In the last, service A is invoked, changing the state of the ASM and then one or

<sup>2</sup> The namespace "push" <http://www.w3.org/2007/uwa/context/push.owl#> for the push technology terms

<sup>3</sup> It is not our intention in this paper to claim that WSMO is the most suitable and effective service composition language, even in the Semantic Web service arena. Our *only* intention was to have a solid service composition framework on which we can experiment with our transformation rules from configured feature models to service configurations. Moreover, the provided mapping rules between feature models and WSMO can easily be applied to any other service composition language, and especially the language such as OWL-S. We are absolutely aware of that some researchers would use the other language, but such as discussion, although very important, is outside of our research and this paper.

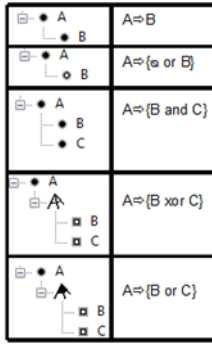


Figure 7. Composition Order for Feature Models

both of services B and C are invoked.

Lee *et al.* [29] assert that the building blocks of compositions are molecular services which are stateless and self-contained in terms of computation and control. In order to make these rules applicable to this work though, we have to consider that the leaves in a feature model refer to these molecular Web services. Composite services are services which accomplish more complex tasks through the use of these molecular services. The variability in potential compositions of Web services is primarily through the selection of services employed to accomplish a task. In the feature model, the abstract tasks and subtasks themselves represent the commonality while the potential services available represent the variability. The result of a product configuration serves as an orchestrating service.

A further extension of the basic rules is to incorporate an additional level of abstraction by taking advantage of the two of the primary types of elements available within feature models – Features and Feature Groups. Presumably most, if not all, variability within a service-oriented architecture can be represented by sets of grouped features within a hierarchy, leaving solitary features outside of these

groups to represent higher levels of abstraction. Features which represent actual service compositions will therefore always be contained within feature groups representing the appropriate cardinality information. In addition to facilitating the higher level of abstraction, these assumptions also allow the transformation developed to better navigate the feature model. Parent features of a feature group represent categories of services. The child features of a feature group represent actual services which can comprise the composition. The constraints, *includes* and *excludes*, function as they would normally function. These constraints only affect what can be done during product configuration but, once resolved, there is no need to preserve that information beyond the valid product configuration to the WSMO environment.

Finally, this approach applies attributes to the various features. This extension to the basic feature model metamodel is very useful in that it can allow the easy inclusion, at design time, of information which may be more difficult to derive through other means. For the purposes of this approach, the attributes included with the services represented as features are the real-world groundings of the input the actual Web service expects.

These are actual Uniform Resource Identifiers (URI) which tend to be Web addresses and are used to invoke a Web service. URIs of actual web services are specified according to available services discovered in domain engineering (Sect. 4.3), and requirements for that particular application collected in application engineering. Output groundings are not necessary as these can actually be handled by the execution environment. In annotating the feature model with these groundings a higher level of automation can be achieved by transformation. The basic mapping rules are applied as shown in Table 1. These mapping rules relate the





Feature Model Element	Description	WSML
 Example Feature Model	Root Project	<i>Not Directly Mapped (Contains Feature Model, Metamodel, and Metametamodel elements of .fmp file)</i>
 Example Composition	Root of Feature Model	<pre> webService ExampleComposition nonFunctionalProperties   dc:title hasValue ExampleComposition   dc:description hasValue ExampleComposition endNonFunctionalProperties capability ExampleCompositionCapability interface ExampleCompositionInterface choreography ExampleCompositionChoreography stateSignature ExampleCompositionSig transitionRules ExampleCompositionTrules </pre>
<input checked="" type="checkbox"/> ServiceA ('GroundingServiceA' : STRING)	Selected Feature (with In Grounding Attribute)	
<input checked="" type="checkbox"/> Category of Services 1  ServiceA  ServiceB	Selected Service (single group)	<pre> // Pre and Post Conditions Inside WSMO Capability precondition definedBy.// TODO: Preconditions of ServiceB postcondition definedBy.// TODO: Postcondition of ServiceB // Out Mode Container in State Signature out ServiceBResponse //with optional grounding ... // ServiceB Web Service Transition forall {?controlstate} with( ?controlstate[oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState) do forall with() do // TODO: ServiceA Related Specific Transition Rules endforall delete( ?controlstate[oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState) add(?controlstate[oasm#value hasValue oasm#State1] memberOf oasm#ControlState) endforall </pre>

Table 1. Summary of Basic Mapping Rules

structure of the feature model to the WSMO Web Service description in WSMML. Where services are selected in other groups, the last rule is applied recursively to expand the transition rules of the ASM in the Web service interface choreography description. Although this business process information is potentially better represented within an orchestration description, unfortunately there is currently no finalized specification or tool support for WSMO orchestration. Therefore, we see in implementations such as [8] and [18] a necessary misuse of the choreography description to achieve the same effect as orchestration.

Using the ATLAS Transformation Language<sup>4</sup> (ATL), we implemented a transformation<sup>5</sup> that takes the resulting product configuration from the feature model and maps it in to the structure of a composite Web service description in WSMML. The heuristics of the transformation involve considering the order of the selected features in the product configuration to populate certain sections of the composite service's WSMML description, specifically the capability and interface information. The capability is represented through preconditions and postconditions. The interface description of the composite service is comprised of choreography and orchestration descriptions. Both choreography and orchestration descriptions require state signatures which define the state of the ASM through a set of mode containers. These containers have different modes including *static*, *in*, *out*, *shared*, and *controlled*. Generally, within this design, containers with an *in* mode represent input to the composite web service (and therefore input to its molecular services) while those with an *out* mode represent output from the composite web service (and therefore output from its molecular services). The transformation migrates the appropriate *in* and *out* mode containers to the composite Web services. Other mode containers, such as those with a *controlled* mode are used to control flow within the ASM and can either be assumed for the purpose of the transformation or managed through manual intervention. Choreography descriptions also require transitions rules. There is a generally accepted pattern, represented by the Ontologized Abstract State Machine [20], which is known to the WSMX choreography engine, to manage some of the control from within the ASM that can be assumed by the transformation.

## 5. Evaluation

The previous section has introduced algorithms and approaches which aim at increasing the level of service reuse and increasing the level of automation in different stages of our proposed methodology. The evaluation results of the effectiveness of automatic verification of feature models (Sect. 4.2) based on OWL-DL are reported in [52]. In this section, we report on the evaluation results obtained by experimenting with prototypes for service discovery, automated staged configuration, and service compositions.

### 5.1. Service Discovery

The discovery algorithm was implemented to work with Seekda™, a free search engine for discovering Web services and their providers with more than 25000 registered services [53]. We developed a discovery module that sends each of the individual queries to the Seekda search engine, retrieves the list of returned services, and stores the returned services back into a local repository. The repository was then used in order to perform evaluations on the results of service discovery and selection. To evaluate our discovery algorithm, as proposed in Section 4.3, we conducted several tests on the precision of the described algorithms, as well as their effectiveness in reducing the labor time and increasing the efficiency. For precision, since the set of all available services related to our search queries are not known, we measured *top-k precision* ( $p_k$ ) which evaluates the precision for the *top-k* returned services. Also, the amount of time it took for each query to be executed was measured during our experiments. Since the set of all relevant services available on Seekda were not known, we were unable to provide proper measurements on the recall of the proposed selection algorithm.

In order to measure ease-of-use and reducing labor effort, we recruited two developers to collect services related to the features in the feature model of Figure 3. We provided our developers with five major keywords, i.e., *credit card validation*, *credit card*, *master card*, *visa*, and *American express* and gave the developers the freedom to choose four combinations of the above keywords that they thought would help them find their desired set of services. The keywords, which we provided to our developers, were equal to the ontology concepts used to annotate the *CreditCardValidation* concept from our e-commerce ontology as well as the name of the feature from the feature model, i.e., the term *credit card validation* (see Figure 5).

The developers used the Web-based front-end for service discovery offered by Seekda in order to perform their service search and selection. We asked our two developers to always select unique services which they did not already pick during their earlier attempts in service selection. Since inspecting a large number of services seemed infeasible, we asked our developers to pick five of the returned Web services for each search query that they thought were closely related to what they were looking for in order to do the manual inspection. During the experiments, we measured the amount of time it took for them to inspect the WSDL document for each of their chosen services. Once the inspection phase was done, we asked our developers to give each of their selected services a percentage rank on how relevant the service has been to what they have been looking for. Table 2 shows the results of collected information during conducting the experiment. The columns in Table 2 represent separate queries provided by the developers to query Seekda with, and the rows represent the five services that the developers chose after each individual query.

Table 2. The results of developer experiments for manual discovery of services

<sup>4</sup> <http://www.eclipse.org/m2m/atl/>

<sup>5</sup> The transformation along with examples is available at <http://io.acad.athabasca.ca/~jeffr/webdoc.html>

Developer 1	Web Services	Initial Search Query (Sec.)				Inspection of Web Service Spec. (Sec.)				Relevance (%)			
		A1	A2	A3	A4	A1	A2	A3	A4	A1	A2	A3	A4
		1					185	98	65	58	45	55	10
2					163	110	55	66	30	25	15	75	
3	63	98	123	174	143	124	78	68	75	15	85	65	
4					115	112	89	74	75	15	15	50	
5					244	107	53	61	85	85	15	75	
Total (Sec.)					850	551	340	327					
Finalized Required Time (Sec.)					913	649	463	501					

Developer 2	Web Services	Initial Search Query (Sec.)				Inspection of Web Service Spec. (Sec.)				Relevance (%)			
		A1	A2	A3	A4	A1	A2	A3	A4	A1	A2	A3	A4
		1					98	132	45	65	30	75	20
2					120	127	58	54	30	80	20	15	
3	132	243	280	136	117	185	120	37	85	65	50	40	
4					128	46	102	74	55	65	35	70	
5					185	187	87	69	55	60	45	85	
Total (Sec.)					648	677	412	299					
Finalized Required Time (Sec.)					780	920	692	435					

Based on the results shown in Table 2, the average time it took the developers to find their desired services (obtained from the last row in Table 2), was 631.5 sec for the first developer and 706.75 for the second developer.

To analyze the behavior of our automated service discovery approach, we conducted experiments on a Dell E6500 Laptop with 4GB of RAM and a 2.53 GHz Core-Duo Intel processor connected to the internet using WiFi with a maximum throughput of 300mbps, running Windows Vista Home Edition. During the automated service discovery process, we injected the five keywords that we had already given to our developers to our service discovery and selection prototype by pooling them from our e-commerce ontology (cf. Section 4.3). Unlike in case of our developers, where they only used four combinations of the keywords to generate their queries, our approach generates all possible combinations for the queries, i.e., 31 queries, to query Seekda. The minimum number of services we found for a search-query was six services and the maximum number of services retrieved for a query was 2086. Out of the total number of returned services for each search query, we took the top 100 services (if available) with the highest availability ranking (as provided by Seekda) and applied our ranking algorithm. At the end, we ended up with 225 ranked services for all 31 possible queries. Table 3 lists the overall time it took for our automated service discovery prototype to generate the power-set of the queries, to query Seekda with all possible queries, and perform the ranking strategy described in Section 4.3. The query generation and ranking algorithms were very fast for our list of returned services, less than 10 seconds each. The most time consuming part of the algorithm, taking over 2 minutes, was to retrieve the list of services from Seekda.

Table 3. Automated Service Discovery and Selection

Automated Service Discovery	Query Generation Time (ms.)	Querying Seekda (ms.)	Service Ranking Time (ms.)
	7.0	130081.0	6.0

We then chose the top 30 services returned by our service selection and discovery prototype and let our developers look into them and tell us how many of these services they found useful for what they were looking for (i.e., calculating top-k precision). In another effort, we tried to see how many of these services had already been ranked by the developers (during their manual inspection phase) as highly valid services. We extracted those services that were ranked greater than 70% by the developers in the first part of the study (8 services for Developer 1 and 5 services for Developer 2) and checked the list of our returned queries to see whether their services were returned by our selection prototype. As for calculating top-30 precision, Developer 1 found 19 of the top 30 services relevant, giving us a precision 63%, while Developer 2 found 17 of the services relevant, giving us a precision 56%. Interestingly, 10 out of 13 services that were previously found helpful by our developers during their manual search and inspection were among the first 30 services returned by our ranking algorithm (5 services from the 8 services for Developer 1 and all services for Developer 2). To calculate the precision for our developers' manual search, we considered the ratio of services ranked above 70% by our developers to all their inspected services (i.e., 20 services) which would give us a precision 40% for Developer 1 and 25% for Developer 2. Comparing the results, the automated discovery and selection algorithm proves to be more reliable compared to the manual search and discovery of services. Table 4 shows some of the collected results during our experiments. The developers seemed to be a lot more satisfied with the offered search and selection prototype than to have to manually go through the list of all potential services to find their desired services.

Table 4: Developers' inspection of returned services, both only based on the automatically returned services, and their individual selections from the first phase of study

	Developers' Inspection of Returned Services		Initial Developers Selection Hit	
	D1	D2	D1	D2
Service Discovery Results	63%	56%	62.5%	100%

## 5.2. Specialization Algorithm Performance

Automated staged configuration removes all unfeasible features of that stage. The algorithm traverses features of the feature model, and removes unfeasible features, i.e., features whose hard requirements are not satisfied. Furthermore, it removes also features whose selection in the final configuration requires the selection of unfeasible features. The set of all removed features is called Unfeasible Features Set (UFS). In order to demonstrate the benefits of automated staged configuration, i.e., benefits of automating the removal of unfeasible feature sets, we have measured the mean size of the UFS for various feature models. Mean size of UFS is computed by computing UFC of every feature in the feature model, and dividing it with the number of

features.

Table 5. Mean number of removed features in specializations

Crosstree conns FM Size (n)	0	0.25n	0.5n	0.75n
10	2.0	2.4	2.6	2.8
50	9.1	10.5	12.7	13.5
100	19.4	21.6	21.9	27.9
500	86.4	102.1	123.4	138
1000	172.1	198.4	223.6	385.5

The modified version of the automated feature model introduced in [55] is used for the generation of feature models. The modification was performed just to exclude the usage of Java Choco Constraint Solver [21], given that we used the DL-based verification and implemented our own DL-based configuration algorithm (c.f. Section 4.4). We have measured mean UFS sizes of generated feature models containing 10, 50, 100, 500, and 1000 features. The branching factor of the experiment was limited to at most five sub-features per feature. Also, the probability of feature groups being generated was 1/3 at each feature with children, similarly to experiments performed in [55]. Furthermore, we have also measured mean UFS sizes for various crosstree connections. Crosstree connections are actually *includes* and *excludes* integrity constraints. Existence of crosstree connections particularly *includes constraints* increase the average size of UFS. We have also assumed that the probability of each feature being unfeasible is equal, and that each feature can be unfeasible in the given environment. For each size and each number of crosstree connections, five feature models have been generated, their mean UFS sizes computed, and the average of obtained results computed. The results can be seen in Table 5.

The results demonstrated in Table 5 show the effectiveness of the DL-based representation of feature models, which are semantically annotated with concepts from ontologies specifying non-functional properties. In order to show the benefits for employing the DL-based automated staged configuration independent of the size of the feature model, we have calculated the ratio between the average numbers of removed features and size of the feature models. We have decided to use a pessimistic approach and considered the minimum ratio as a representative for a feature model having a certain number of crosstree connections. The results show that, depending on the number of the crosstree connections even discovering one unfeasible feature can reduce the size of the feature model from 17.2% when there are no crosstree connections (for feature models of sizes 500 and 100), up to 27% when the number of crosstree connections is 75% (for feature models of size 50) of the number of features in feature model. The relatively high value of mean UFS and large potential in using automated staged configuration lies in high values of UFS for features, which are higher in the hierarchy of a feature model.

### 5.3 Automated Generation of Service Compositions

To evaluate the transformation, we measured the level of

completeness of the WSMO compositions obtained by our ATL transformation explained in Section 4.5. The initial transformation does not result in a completely executable composite Web service description. Feature models being less expressive than WSML Web service descriptions generally, it is necessary to conduct some manual intervention to complete elements of the Web service description, including non-functional properties, to ensure functionality within an execution environment. The average achievable percentages of automatically generated WSML are shown in Table 6. Comparing these results, with the state of the art in model-driven development of semantic service-oriented systems developed by Brambilla *et al.* [8], we conclude that our obtained results are at the same level as the mentioned components. This is also a good indicator for even more promising results, once our feature modeling approach is combined with business process modeling language, as we envisioned in [1].

Table 6. Average Percentages of Automatically Generated WSML

WSML Component	Percentage of Automatically Generated WSML
WS Description (Total)	76 %
<i>Capability</i>	47%
<i>Interface</i>	82%
Goal Description	74%

## 6. RELATED WORK

Given the comprehensiveness of the proposed approach, there are several major groups of related work as described in the reminder of the section.

### 6.1 Software Services and Product Lines

Helferich *et al.* [19] explored the relation between SOA and SPL and concluded that they are not necessarily mutually exclusive, but that they share a number of characteristics. They proposed that the knowledge gained in the research on SOA can be beneficial for implementing features of a software product line, because a software development organization may consider not developing all the features by itself. Moreover, their results show that the shortcoming of SOA and SPL can be complemented by the other.

Trujillo *et al.* [50] pursue the idea of integrating SOA and SPL by using SOA in a scenario where different products of different product lines are used to create a more elaborated product of a third product line. They propose the Service-oriented Product Line (SOPL) notion that considers the product lines as services and uses the service oriented standards to compose the software products supplied from different product lines with only little human intervention. The concern of viewing SPLs as services is the level of granularity that service should be considered.

Lee *et al.* [30] considered adapting feature-oriented product line engineering. They first developed an infrastructure consisting of generic services for the particular family. Then, they construct and evolve the family members. That is, the method organizes the product family features into a feature model, and then identifies binding units within

feature model and their corresponding service categories as well as their binding times. A service category contains a molecular service - composed of low level atomic services and considered as building block of family products, and orchestrating service - workflows and their constituting tasks. Finally, the services are provided for user by integrating and parameterizing of the molecular services at run-time.

Boffoli et al. [7] proposed an approach which considers SOA applications as software product lines in order to be able to adapt SOA applications according to stakeholders' needs in changeable environments. The method consist of: i) *Business Process Lines* phase which by using SPL principles creates a *process variant* - a process convenient for specific customer needs, as well as a *variability model* - a variability selection table; and ii) *Process Oriented Development* phase which transforms a process variant into a SOA system.

One more research effort for integration of SOA and SPL has been done by Medeiro et al [34]. They have used a top-down approach to identify and implement service-oriented core assets, constituting the *core asset lifecycle*. This is followed by the process of specializing the core assets for a particular target context according to the target customers' requirements, which is referred to as the *product development lifecycle*.

As our previous sections demonstrated, the abovementioned research approaches significantly motivated our research. Our citations throughout the paper acknowledge specific parts where the cited work was very important for us to make some relevant decisions. Although all these works proposed very significant research results, there we are being the first to make use of Semantic Web technologies to integrate various stages in the lifecycles of service families. This is a crucial prerequisite for an advanced reconciliation of the "closed" SPL with "open" SOA worlds. To the best of our knowledge, this is the first work that acknowledges the importance of service discovery in engineering of families of service-oriented systems. While this has been recognized in general in the SOA lifecycles [63], this is the first attempt in the context of families of SOAs.

## 6.2 Configuration and Verification of Feature Models

Several approaches for configuration and verification of feature models have been proposed in the domain of SPL. Mannion [32] proposed the use of first order logic for representation of feature models and verification of feature configurations. However, representing a feature model as a logic-based formula to practitioners who are more interested to graphical representation such as feature diagram is not convenient. Czarnecki et al. [12] use the graphical model whose syntax is defined by a context free. But, integrity constraints cannot be represented by just using context free grammars. So, in order to support integrity constraints, they employ Object Constraint Language (OCL) [11]. Batory [4] connects the grammars and propositional formulas of feature models to allow for defining constraints among features.

Logic Truth Maintenance Systems (LTMS) can be used in

both propositional formula and grammar representation for the verification and debugging of feature models [47]. LTMS mostly use the SATisfiability problem [4], Constraint Satisfaction Problem (CSP) [6], and Binary Decision Diagrams [35] to verify feature models. Wang et al. [52] proposed the semantic based approach to verifying feature models. They provide a set of transformation rules that develops ontological representation of feature model in OWL. Their approach can perform reasoning over consistency of created ontology and consequently helps with verifying the validity of feature model and configurations. We actually leveraged this approach as a foundation of our staged-configuration algorithm.

Peng et al. [41] proposes ontology-based feature modeling method which defines an OWL-based metamodel containing the OWL definition of different constructs (i.e. relations, and features) of feature models. The integrity constraints on the feature model are converted to rule defined over ontology-based representation of feature models. The validation of feature models regarding integrity constraints is conducted by ontology inference.

All these above works significantly inspired our research. A discussion about the level of expressivity of underlying formalisms used in the cited work and their comparison with our DL-based approach goes beyond the scope of this paper. Certainly, DL is less expressive than first-order logic, given an open-world assumption of the used DL underlying OWL-DL. However, the open world assumption is exactly tailored to support reasoning on the Web, which is suitable for SOAs. Moreover, the use of ontologies for representation of feature models facilitates enrichment of feature models with domain specific concepts. Herewith, the use of ontologies to formally share meaning of features for domain and non-functional property conceptualizations is another very important contribution of our work. Finally, up until now, we are the first to give quantitative characterization on benefits of staged configuration. Most of current work only declaratively expresses benefits of it. Of course, in the future work, we would also like to make use of non-functional properties coming directly from discovered services in the process of staged configuration. Thus, staged configuration will then include the selection of both features and services.

## 6.3 Service Discovery

Different levels of abstraction could be adopted for the description of Web services providing the functionality of feature and satisfying the feature goals. As a consequence, various Web service discovery approaches might be leveraged with respect to the levels of service abstraction. There are three main discovery approaches which could be accounted in general, in the entire service domain, and in particular, in the WSMO context. A keyword-based discovery approach, supported in WSMO [23], was built on the simple notion of term matching using standard information retrieval methods. In spite of the fact that these techniques are limited due to the ambiguities of natural language and the lack of semantic understanding of natural language descriptions, using keyword-based discovery

approach has two major advantages. This approach is able to be scaled up easily to a large number of services and can utilize mature keyword matching technologies and information retrieval methods. Furthermore, the required annotations in the terms of keywords are already present through available service descriptions [16]. So-called *lightweight* set-based discovery [23] was defined by using Description Logic and Logic Programming. The approach uses service descriptions that describe the output of a service in an abstract way, considering merely the post-conditions and effects of services and goals, and not taking into account the service preconditions and assumptions. A more *heavyweight* discovery [25] is based on richer semantic descriptions by taking into account relationships between preconditions, post-conditions, assumptions, and effects. The heavyweight discovery enables a service developer to specify constraints on the service input, whereas the goal description can describe the relationship desired between the input data he will provide and the output the service should provide. Discovery engines which implement these approaches will appear in a near future.

It will be incorrect to claim that our contribution is yet another discovery engine. On the contrary, our research fully aims to make use of the state of the art in the area of service discovery and currently available discovery engines. Our contribution is in integrating of currently available discovery engines with software methodologies for developing families of service-oriented systems. Through the use of ontologies and integration of service discovery with feature modeling, our contribution can be represented as follows: i) more systematic approach to the reusability of publicly shared services; and ii) reduced amount of efforts of service engineers in process of discovery of relevant services publicly available and developed by third parties.

#### 6.4 Transformations, Product Lines, and Services

Regarding the transformation between feature model and business process models, some approaches have been proposed. Bae et al. [2] describe a methodology for developing feature model from family of business processes in the divide and conquer approach. The method identifies use cases from business process and then producing feature model from use-cases. Generating business process model represented in Business Process Modeling Notation (BPMN) from a feature model has been introduced by Montero et al. [38]. They use feature model in a way parent features are considered as complex processes, and their sub-features represents their sub-processes. They produce the initial business process from feature model and then complete the produced business process. The defined transformations between feature models and WSMO compositions in our method are inspired by their work.

As mentioned in Section 5.3, the amount of manual intervention after our transformations is comparable to the state of the art solutions [8]. However, our approach for services generation is in the context of SPLs and uses enriched feature models for generation of service oriented systems based on these specifications.

## 7. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a methodology for building families of software services and deriving a particular service-oriented application out of this family following the principles of SPLs. The proposed methodology extensively benefits from the use of Semantic Web technologies for context specific service discovery, configuration, and use of Description Logics for verification of SPL configurations. In order to evaluate the benefits of the proposed methodology, we have conducted a series of experiments which show that (a) our *lightweight* semantic based search of publicly available services based on shared ontologies is very useful, even when applied in currently available search engines, such as Seekda, which do not support semantic based service descriptions; (b) automated staged configuration can significantly reduce the size of the feature model, herewith, reducing the time and effort in application engineering; and (c) configured models can be used to automatically generate a large amount of executable service compositions, herewith reducing the efforts for manual development.

In our future work, we are going to concentrate on extending our methodology in order to reduce the manual intervention needed in final services development. One promising direction is employment of elements of Business Process Management, and usage of benefits offered by BPMN. BPMN can increase the features choreography and orchestration description to the level where a minimal manual intervention is needed. Additionally, employing BPMN in the process of designing families of software service development can improve the communication between business administration and software engineering, additionally reducing time-to market and cost of software service development. However, this increase in expressivity will require us to provide additional services for verification of families of business processes, that is, that every configuration of a service-oriented software family is valid w.r.t. both the process and feature models. Here, besides a need for weaving of BPMN and feature modeling languages, the use of approaches such as critical pair analysis is our next destination. Our future work will also include an integration logic-based matching to complement our current combined ontology and text-based service.

## ACKNOWLEDGMENT

The research results present in this paper are at least in part supported by the Alberta Innovates Technology Futures through the New Faculty Award program.

## REFERENCES

- [1] Asadi, M., Mohabbati, B., Kaviani, N. Gašević, D., Bošković, M., Hatala, M., "Model-Driven Development of Families of Service-Oriented Architectures," In *Proc. of the 1st Int. WS on Feature-Oriented Software Development*, 2009, pp. 95-102
- [2] Bae, J. and Kang, S. A., "Method to Generate a Feature Model from a Business Process Model for Business Applications," In *Proceedings of the 7th IEEE Int. Conf. on Computer and Information Techn. CIT. IEEE Computer*, 2007 pp. 879-884.

- [3] Balzer, S., Liebig, T., Wagner, M., "Pitfalls of OWL-S – A practical Semantic Web use case," *In Proc. of the 2nd Int. Conf. on Service Oriented Computing*, pp. 289- 298, 2004.
- [4] Batory, D. "Feature Models, Grammars, and Propositional Formulas," *In Proc. of the 9th Int. Conf. on SPLs*, 2005, pp. 7-20.
- [5] Batory, D., "Feature models, Grammars, and Propositional formulas," In H. Obbink and K. Pohl (Eds.): *Proc. of the 9th Int. Conf. on SPLs, SPLC 2005, LNCS 3714*, Springer, pp. 7 20. 2005.
- [6] Benavides, D., Trinidad, P. and Ruiz-Corts, A., "Automated Reasoning on Feature Models," *In Proceedings of the Conference on Advanced Information Systems Engineering 2005 (CAISE2005)*. LNCS 3520, Springer. pp. 491-503. 2005.
- [7] Boffoli, N., Cimitile, M., Maggi, F.M., Visaggio, G., Managing SOA System Variation through Business Process Lines and Process Oriented Development, *Workshop on Service-Oriented Architectures and Software Product Lines (SOAPL)*, 2009.
- [8] Brambilla, M., Ceri, S., Facca, F.M., Celino, I., Cerizza, D., Cefriel, E.D.V., " Model-driven design and development of Semantic Web Service Applications," *ACM Transactions on Internet Technology*, 8(1): 1-31, 2007.
- [9] Classen, A., Hubaux, A., and Heymans, P., "A Formal Semantics for Multilevel Staged Configuration," *In Proc. of the 3rd WS on Variability Modelling of Software-intensive Systems*, 2009.
- [10] Czarnecki, K. , Helsen, S., and Eisenecker, U., "Staged Configuration through Specialization and Multi-level Configuration of Feature Models." *Software Process Impr. and Practice*, 10(2), John Wiley & Sons. pp 143-169. 2005.
- [11] Czarnecki, K. and Kim, C. H. P., "Cardinality-Based Feature Modeling and Constraints: A Progress Report," *In Proceedings of the OOPSLA05 Int. Workshop on Software Factories*. 2005.
- [12] Czarnecki, K., Helsen, S., Eisenecker, U., "Staged Configuration Using Feature Models," *Proceedings of the 2004 Software Product Line Conference*, LNCS 3145, pp. 266-283.
- [13] D. Batory. Feature Models, Grammars, and Propositional Formulas. *In Proc. Int'l SPLC*, 2005.
- [14] eCommerce Ontology Project, Development of Web Ontologies as Data Exchange and Decision Support Tools, New Jersey Institute of Technology: <http://web.njit.edu/~kh8/project/ecommerce.owl>
- [15] Erl, T., *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005.
- [16] Fensel, D., Kerrigan, M., and Zaremba, M., "Implementing Semantic Web Services: The SESA Framework," Springer, 2008.
- [17] Fonseca, J.M.C., Lewis, R., Delivery Context Ontology, W3C Working Draft 16 June 2009, 2009, <http://www.w3.org/TR/dcontology/>
- [18] Gone, M. and Schade, S., "Towards semantic composition of geospatial web services: Using WSMO in comparison to BPEL," *Int. J. of Spatial Data Infrastructures Research*, vol. 3, 2008.
- [19] Helferich, A., Herzwurm, G., Jesse, S., Mikusz, M., "Software product lines, service-oriented architecture and frameworks: Worlds apart or ideal partners," *In Trends in Ent. Application Architecture*, 2007, pp. 187-201). Berlin, Germany: Springer.
- [20] Herold, M., WSMX Documentation. Version 1.0. Digital Enterprise Research Institute Galway, Ireland. <http://www.wsmo.org/TR/d22/v0.2/>.
- [21] <http://www.emn.fr/x-info/choco-solver/doku.php?id=introduction>. Accessed: April 2010.
- [22] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Person, A.S., "Feature-oriented domain analysis (FODA) feasibility study," Technical Report CMU/SEI-90-TR-21, Software Engineering Institute (SEI), Carnegie Mellon University, 1990.
- [23] Keller, U., Lara, R., Lausen, H., Polleres, A., and Fensel, D., "Automatic location of services," *In Proc. of 2nd European Semantic Web Conf. (ESWC)*, 2005.
- [24] Keller, U., Lara, R., Polleres, A., Toma, I., Kifer, M., and Fensel, D., D5.1v0.1: WSMO Web Service Discovery". Technical report, DERI Innsbruck, 2005.
- [25] Keller, U., Lausen, H., and Stollberg, M., "On the semantics of functional descriptions of web services," *In Proc. of 3rd European Semantic Web Conf. (ESWC)*, 2006.
- [26] Klusch, M., Kapahnke, P., and Zinnikus, I., "Hybrid Adaptive Web Service Selection with SAWSDL-MX and WSDL-Analyzer," *In Proceedings of the 6th European Semantic Web Conference on the Semantic Web*, pp. 550-564.
- [27] Lau, S. Q., "Domain analysis of e-commerce systems using feature-based model templates," Master's Thesis, University of Waterloo, Canada, 2006. Available [Online]: <http://gp.uwaterloo.ca/files/2006-lau-masc-thesis.pdf>
- [28] Lautenbacher, F., Bauer, B., "Creating a metamodel for Semantic Web Service standards," *In Proceedings of WEBIST 2007*. Accessed April, 2008.
- [29] Lee, J., Muthig, D., Naab, M. "A feature-oriented approach for developing reusable product line assets of service-based systems," *Journal of Systems and Software* 83(7): 1123-1136 (2010).
- [30] Lee, J., Muthig, D., Naab, M., "An approach for developing service oriented product lines," *In Proc of the 12th Int'l Software Product Lines Conference*, pp. 275-284. 2008.
- [31] Lee, K., Kang, K. C., and Lee, J., "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering," *In Proc. of the 7th Int. Conf. on Software*, pp 62-77, 2002.
- [32] Mannion, M., "Using First-Order Logic for Product Line Model Validation," *In Proc. of the 2nd Int. Conf. on SP*. LNCS, vol. 2379. Springer-Verlag, London, 176-187. 2002.
- [33] Martin, D. et al. (2004) OWL-S: Semantic markup for web services, version 1.1 available at <http://www.daml.org/services/owl-s/1.1/>. Member submission, W3C. Available from: <http://www.w3.org/Submission/OWL-S/>
- [34] Medeiros, F.M., de Almeida, E.S., de Lemos Meira, S.R. "Towards an Approach for Service-Oriented Product Line Architectures," *Workshop on Service-Oriented Architectures and Software Product Lines (SOAPL)*, 2009.
- [35] Mendonca, M., Wasowski, A., Czarnecki, K., Cowan, D., "Efficient Compilation Techniques for Large Scale Feature Models," *In Proc. of the 7th Int. Conf. on Gen. Programming and Component Engineering*, ACM Press, pp. 1322. 2008.
- [36] Mohabbati, B., Kaviani, N., Gašević, D., 2009. Semantic Variability Modeling for Multi-staged Service Composition, *In Proc. of the 13th SPLC, Vol. 2 (3rd Int. WS on SOA&SPLs)*, 2009.
- [37] Mohabbati, B., Kaviani, N., Lea, R., Gašević, D., Hatala, M., Blackstock, M., "ReCoIn: A Framework for Dynamic Integration of Remote Services in a Service-Oriented Component Model," *In Proc. of the 2009 IEEE Asia-Pacific Serv. Comp. Conf.*, 2009.
- [38] Montero, I., Pena, J., Ruiz-Cortes, A., " From Feature Models to Business Processes," in *Proceedings of the IEEE International Conference on Services Computing Vol. 2*, 2008., pp.605-608.
- [39] Patil, A, Oundhakar, S., Sheth, A., Verma, K. (2004) Semantic Webservices: Meteor-S Web service annotation framework. *In: 13th Int. Conf. on WWW. Appl Ontol* 1(1):77-106
- [40] Peltz, C. "Web Services Orchestration and Choreography," *Computer*, vol. 36, no. 10, pp. 46-52, 2003.
- [41] Peng, X., Zhao, W., Xue, Y., Wu, Y., "Ontology-Based Feature Modeling and Application-Oriented Tailoring," *Proc. of the 9th Int. Conf. on Software Reuse (ICSR2006)*, LNCS 4039.
- [42] Pohl, K., van der Linden, F., Bockle, G., *Software Product Line Engineering: Foundations, Principles, and Techniques*, Springer, 2005.
- [43] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D., "Web service modeling ontology, *Applied Ontology*, vol. 1, no. 1, pp. 77-106.

- [44] Rosenberg, F., Celikovic, P., Michlmayr, A., Leitner, P., Dustdar, S., "An End-to-End Approach for QoS-Aware Service Composition," *In Proceedings of the 13th IEEE Int. Enterprise Distributed Object Computing Conference*, pp.151-160, 2009.
- [45] Rosenberg, F., Platzer, C., and Dustdar, S., "Bootstrapping Performance and Dependability Attributes of Web Services," *In Proc. of the IEEE Int. Conf. on Web Services* pp, 205-212, 2006.
- [46] Schmid, K., "Scoping Software Product Lines - An Analysis of an Emerging Technology. Software Product Lines: Experience and Research Directions," *In Proc. of the 1st SPLC*, pp. 513-532., 2000
- [47] Schnieders, A., and Puhmann, F., "Variability mechanisms in e-business process families," *9th International Conference on Business Information Systems (BIS 2006)*, 2006.
- [48] Schobbens, P., Heymans, P., and Trigaux, J., "Feature Diagrams: A Survey and a Formal Semantics," *In Proc. of the 14th IEEE Int. RE Conf. IEEE Computer* , pp. 136-145. 2006.
- [49] Shafiq, O., Moran, M., Cimpian, E., Mocan, A., Zaremba, M., Fensel, D., "Investigating Semantic Web Service execution environments: A comparisons between WSMX and OWL-S," *In Proc. of 2nd Int. Conf. on Internet and Web Applications and Services*, 2007, pp. 31-36.
- [50] Trujillo, S., Kastnes, C., Apel, S. (2007). Product lines that supply other product lines: A Service oriented approach. In SPLC'07 Workshop: Service-Oriented Architectures and Product Lines – What is the Connection? Kyoto, Japan, September 2007. Retrieved November 12, 2007 from <http://www.struji.com/files/2007splcsoapl.pdf>
- [51] Tsai, W., "Service-oriented system engineering: a new paradigm," *In Proceedings of the IEEE International Workshop of Service-Oriented System Engineering*, 2005, pp. 3-6.
- [52] Wang, H. H., Li, Y. F., Sun, J., Zhang, H., and Pan, J. "Verifying feature models using OWL," *Web Semant.* 5, 2, 2007, 117-129.
- [53] Web Service Search Engine @ seekda.com, Available [Online]: <http://webservices.seekda.com/>, 2011.
- [54] White, J., Benavides, D., Dougherty, B., Schmidt, D. C., "Automated Reasoning for Multi-step Software Product-line Configuration Problems," *In Proceedings of the 13th Software Product-lines Conference (SPLC2009)*. Volume 1. 2009.
- [55] White, J., Schmidt, D.C., Benavides, D., Trinidad, P., Ruiz-Cortes, A., "Automated Diagnosis of Product-Line Configuration Errors in Feature Models," *Software Product Line Conference, 2008. SPLC '08. 12th Int.* , vol., no., pp.225-234.
- [56] Papazoglou, M.P., "Service-Oriented Computing: Concepts, Characteristics and Directions". *In Proc. of the 4th Int. Conference on Web Information Systems Engineering*, p. 3.
- [57] Parnas, D.L. , "On the Design and Development of Program Families," *IEEE Trans. Softw. Eng.*, vol. 2, no. 1, 1976, pp. 1-9.
- [58] OWL Web Ontology Language Guide <http://www.w3.org/TR/owl-guide/>  
Accessed: April 2011.
- [59] Web Service Modeling Language (WSML). <http://www.w3.org/Submission/WSML/> Accessed: April 2010.
- [60] Haller, A., Cimpian, E., Mocan, A., Oren, E., and Bussler, C., "WSMX - A Semantic Service-Oriented Architecture," *In Proc. of the IEEE Int. Conf. on Web Services*, pp. 321-328, 2005.
- [61] Kim, S., Min, H.G., Her, J.S., Chang, S.H., "DREAM: A practical product line engineering using model driven architecture," *In: ICITA 2005, Australia*, pp. 70--75 (2005).
- [62] Gomaa, H. 2004 *Designing Software Product Lines with Uml: from Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing Co., Inc.
- [63] Blake, M.D., "Decomposing Composition: Service-Oriented Software Engineers," *IEEE Softw.*, vol. 24, no. 6, pp. 68-77 2007
- [64] Boskovic, M., Bagheri, E., Mohhabati, B., Kaviani, N., Hatala, M., "Automated Staged Configuration with Semantic Web Technologies",
- Int. Journal of Softw. Eng. and Know. Eng.*, vol. 20, no. 4, pp. 459-484, 2010.