Developing Families of Software Services: A Semantic Web Approach

Marko Bošković and Dragan Gašević
School of Computing and Information Systems, Athabasca University, 1 University Drive, Athabasca, AB T9S 3A3, Canada
School of Interactive Arts and Technology, Simon Fraser University Surrey, 13450 102 Ave., Surrey, BC V3T 5X3, Canada
{marko.boskovic, dragang}@athabascau.ca

Bardia Mohabbati, Mohsen Asadi and Marek Hatala
School of Interactive Arts and Technology, Simon Fraser University Surrey, 13450 102 Ave., Surrey, BC V3T 5X3, Canada
{mohabbati, masadi, mhatala}@sfu.ca

Nima Kaviani
Department of Computer Science, University of British Columbia, ICICS/CS Building, 201-2366 Main Mall, Vancouver, BC V6T 1Z4, Canada
nkaviani@cs.ubc.ca

Jeffrey Rusk
Q1Labs, 80 Bishop Drive, Fredericton, NB E3C 1B2, Canada
jeff.rusk@q1labs.com

Ebrahim Bagheri
School of Computing and Information Systems, Athabasca University, 1 University Drive, Athabasca, AB T9S 3A3, Canada
Electrical and Computer Engineering Department, University of British Columbia, Forest Sciences Centre Building, FSC 2623–2424 Main Mall, Vancouver, BC V6T 1Z4, Canada
ebagheri@athabascau.ca

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...
Developing Families of Software Services: A Semantic Web Approach

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.

ACM 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 para-
digm 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 (Papazoglou,
2004). Public availability and “openness” of services offer a large potential for the rapid develop-
ment 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) (Pohl et al., 2005), is an emerging approach for soft-
ware 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 (e.g. see Boffoli et al., 2009, Lee
et al., 2008, and Medeiros et al., 2009). 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 hypotheses of the
methodology are that ontologies as shared conceptualizations of the domain can:

• enhance semantics-based discovery during the domain engineering,
• be effective in reducing the time required for the configuration process and
• facilitate reuse-based development in application engineering.

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 Software Product Lines (Sec. 2.1), and Service-oriented Architectures and Semantic Web Services (Sec. 2.2).

2.1. Software Product Lines
The Software Product Line Engineering (SPLE) discipline (Pohl et al., 2005) 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 (Czarnecki and Kim, 2004). 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 (Lee et al., 2002). The main relations defined in feature models are Kang et al., 1990:

- **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. 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). An example of a feature model is given in Figure 1.

The relations and constraints define mandatory conditions that must always hold in every feature model configuration.

Since the initial feature modeling methodology introduced in FODA by Kang et al (1990), there was a need for verification of whether the feature configurations (aka the set of selected features) follow the constraints specified in feature models. Currently, several approaches for verification, mostly based on interpretation of feature models as propositional formulas exist, such as those presented by Mannion (2002) and Batory (2005). Wang et al (2007) 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 adopt the DL based representation of a feature model, because such a representation adds flexibility in specification of various additional requirements as parts of features.

2.2. Service-oriented Architectures and Semantic Web Services

Service-oriented Architecture (SOA) is a standardized set of methods for addressing the requirements of loosely coupled, standard based, and protocol independent distributed computing (Erl, 2005). 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 (Tsai, 2005). The most popular types of services are Web services, which are described using the Web Service Description Language (WSDL) (WSDL, 2011), accessed through Simple Object Access Protocol (SOAP) (SOAP, 2011), and published and discovered in Universal Description, Discovery and Integration registry (UDDI) (UDDI, 2011). This set of protocol is commonly adopted in the industry. However, WSDL specification of services facilitates discovery based on syntactic match of service interfaces.

In the scope of this paper, we employ 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 (McIlraith et al, 2001). Some well-known efforts in that context are WSMO (Roman et al, 2005), OWL-S (Martin et al, 2004), and WSDL-S (Patil et al, 2004).

Employment of ontology-based annotations for automated discovery of desired services enables use of several strategies to find and match a service against the requirements of a query, e.g., text-
Developing Families of Software Services: A Semantic Web Approach

based, structural, and logic-based. Typically, when deciding on the search strategy, there is a tradeoff between effort in specification of requirements, and decision in selecting the appropriate features. Logic-based and structural search strategy require a concrete specification of requirements in the terms of pre-conditions and post-conditions, etc. and allow for less flexibility in selection of services to be used, while text-based selection mainly relies on keywords or tags describing a feature in order to search and fetch the set of matching services, herewith putting more effort in selecting the features while allowing more flexibility in the design. In this paper, we employ text-based search based on ontological concepts. Our experiments show that this approach is highly applicable in realistic contexts.

For the purpose of our approach there is also a need to use an executable Semantic Web Service composition language to be able to make automated compositions of services. Two existing frameworks which support service composition are OWL-S and WSMO. Existing literature reviews regarding the comparison between WSMO and OWL-S 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 Web Service Modeling Ontology (WSMO) model for our work. WSMO is a Semantic Web Services specification metamodel where services and service compositions are encoded in the Web Service Modeling Language (WSML) a concrete syntax of WSMO. The WSMO metamodel enables description of all relevant aspects needed for semantic based discovery and composition of Semantic Web Services – ontologies (for shared conceptualization used for description of services), goals (for specification of discovery requirements), web services (for specification of web services for discovery), and mediators (for resolving interoperability issues between different WSMO elements). WSMO services are deployed onto WSMX, a framework for executing WSMO-based service compositions. Additional benefits of using WSMX is that it has a tooling support for providing feedback about syntactical and semantic completeness of their service compositions. 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 that some researchers would use the other language, such as discussion, but although very important, is outside of our research and this paper.

Finally, web services and web service compositions need to be organized in reusable building blocks for the product line. In this paper, we adopt the approach of Lee et al (2010), who asserts that the low level services are grouped in molecular services as building blocks for the family. Molecular services which are stateless and self-contained in terms of computation and control are later organized in orchestrating services representing the workflow of the system.

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

Journal of Research and Practice in Information Technology, Vol. 43, No. 3, August 2011 183
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. 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) (see Gomaa, 2004 and Kim *et al.*, 2005). 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 some required resources. 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 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 Wang et al. (2007) 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, (see Roman et al, 2005) 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 3.

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 (see Czarnecki et al., 2005) 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 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 (see Lau et al., 2006). 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 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 MMS, Fax, VoiceMail, or Email (MMS, 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

As previously mentioned, we have adopted the approach of Wang et al. (2007) for representation of feature models. In the following we give a short summary of this approach.

Let F_1, \ldots, F_n be a set of features in the feature model. And let F_1, \ldots, 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{align*}
F_i & \sqsubseteq T, \quad \text{has} F_i \sqsubseteq \text{ObjectProperty}, \\
\text{Rule} F_i & \sqsubseteq T, \quad T \sqsubseteq \exists \text{has} F_i. \text{Ont} F_i, \\
\text{RuleOnt} F_i \exists \text{has} F_i, & F_j, \quad \text{for } 1 \leq i \leq n, \\
F_k & \sqsubseteq \neg F_j, \quad \text{for } 1 \leq j, k \leq n
\end{align*}

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

\begin{align*}
\text{OrderManagement} & \sqsubseteq T, \\
\text{RuleOrderManagement} & \sqsubseteq T, \\
\text{hasOrderManagement} & \sqsubseteq \text{ObjectProperty}, \\
T & \sqsubseteq \exists \text{hasOrderManagement.OrderManagement}, \quad \\
\text{RuleOrderManagement} & \exists \text{hasOrderManagement.OrderManagement},
\end{align*}
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 inter-relations 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 (2007).

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.

For the purpose of having a more descriptive feature model for our product family we first need to find a domain ontology (e.g., using Swoogle (Swoogle, 2011)) that best describes the feature model, and then use the domain ontology to annotate the features from the feature model.

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 (Fornesca and Lewis, 2009). 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 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.
Developing Families of Software Services: A Semantic Web Approach

having in the range class Money_Order of the ecommerce domain ontology (ECommerce, 2011), 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 non-functional 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., non-functional properties). Hence, a strategy is needed to effectively convert these descriptive keywords into service selection queries.

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. 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 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 \( sc_1, sc_2, \ldots, sc_n \) 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.,

\[
SQ = \{ \{c\}, \{sc_1\}, \{sc_2\}, \ldots, \{sc_1, sc_2\}, \ldots, \{c, sc_1, \ldots, sc_n\} \}
\]

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 \( SQ \).
Developing Families of Software Services: A Semantic Web Approach

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:

\[
w(m) = \frac{2 \times |m|}{n \times (n + 1)}
\]

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,

\[
\sum_{i=1}^{n} w(i) = 1
\]

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 \( w_i(m) \).

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:

\[
\sum_{i=1}^{p} f_i(s) \times w_i
\]

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 \textit{service availability}, \textit{service response time}, \textit{service reliability}, etc.

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 MasterCard. 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., \( m = \{ \text{Credit Card} \} \)) 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., \( m = \{ \text{CreditCardValidation}, \text{Credit Card}, \text{Visa, MasterCard, and American Express} \} \)), 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 \( m = \{ \text{CreditCardValidation} \} \) are not very likely to be repeatedly retrieved by other queries, and hence the overall weight for most of those services returned from \( m = \{ \text{CreditCardValidation} \} \) are going to receive 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 fed into 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 (The namespace “push” http://www.w3.org/2007/uwa/context/push.owl# for the push technology terms) 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.

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.

In order to use a feature model for automatic generation of a choreography or orchestration of Semantic Web Services in WSMO, 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 (2008) 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
Developing Families of Software Services: A Semantic Web Approach

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 both of services B and C are invoked.

As previously mentioned, Lee et al (2010) assert that the building blocks of compositions are molecular services that are organized into orchestrating services as major workflows. Applying these rules for this work, we consider that the leaves in a feature model refer to molecular Web 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 structure of the feature model to the WSMO Web Service description in WSML. Where services are selected in other groups, the
Developing Families of Software Services: A Semantic Web Approach

<table>
<thead>
<tr>
<th>Feature Model Element</th>
<th>Description</th>
<th>WSML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Feature Model</td>
<td>Root Project</td>
<td>Not Directly Mapped (Contains Feature Model, Metamodel, and Metametamodel elements of .fmp file)</td>
</tr>
<tr>
<td>Example Composition</td>
<td>Root of Feature Model</td>
<td>webService ExampleComposition nonFunctionalProperties dc:title hasValue ExampleComposition dc:description hasValue ExampleComposition endNonFunctionalProperties capability ExampleCompositionCapability interface ExampleCompositionInterface choreography ExampleCompositionChoreography stateSignature ExampleCompositionSig transitionRules ExampleCompositionRules</td>
</tr>
<tr>
<td>ServiceA ('GroundingServiceA': STRING)</td>
<td>Selected Feature (with In Grounding Attribute)</td>
<td>Inside WSMO State Signature in InfoDesignsRequest withGrounding &quot;<a href="http://groundingServiceA?wsdl">http://groundingServiceA?wsdl</a> #wsdl.interfaceMessageReference()&quot;</td>
</tr>
<tr>
<td>Category of Services 1</td>
<td>Selected Service (single group)</td>
<td>// 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[owasm#value hasValue owasm#InitialState] memberOf owasm#ControlState) do forall with() do // TODO: ServiceA Related Specific Transition Rules endForall delete( ?controlstate[owasm#value hasValue owasm#InitialState] memberOf owasm#ControlState) add( ?controlstate[owasm#value hasValue owasm#State1] memberOf owasm#ControlState) endForall</td>
</tr>
</tbody>
</table>

Table 1: Summary of Basic Mapping Rules

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.

Using the ATLAS Transformation Language (ATL) (ATL, 2011), we implemented a transformation (available at http://io.acad.athabascau.ca/~jeffr/webdoc.html) that takes the resulting product configuration from the feature model and maps it into the structure of a composite Web service description in WSML. 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 WSML description, specifically the capability and interface information. The capability is represented through preconditions and postconditions. The interface description of the composite
Developing Families of Software Services: A Semantic Web Approach

A 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, 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. In order to evaluate the hypotheses specified in the introduction of 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
For evaluation of the H1 hypothesis we have implemented the discovery algorithm proposed in Section 4.3 to work with Seekda™, a free search engine for discovering Web services and their providers with more than 25000 registered services (Seekda, 2011). 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. The evaluation of the H1 hypothesis consisted of 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. 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. Additionally, to calibrate the algorithm, and demonstrate that the algorithm is applicable for practical use, we have measured during our experiments the amount of time it took for each query to be executed.

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 1. 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 (Figure 1).

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.

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 behaviour 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

<table>
<thead>
<tr>
<th>Developer 1</th>
<th>Initial Search Query (Sec.)</th>
<th>Inspection of Web Service Spec. (Sec.)</th>
<th>Relevance (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>63</td>
<td>98</td>
<td>123</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total (Sec.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finalized Required Time (Sec.)</td>
<td>913</td>
<td>649</td>
<td>463</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Developer 2</th>
<th>Initial Search Query (Sec.)</th>
<th>Inspection of Web Service Spec. (Sec.)</th>
<th>Relevance (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>132</td>
<td>243</td>
<td>280</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total (Sec.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finalized Required Time (Sec.)</td>
<td>780</td>
<td>920</td>
<td>692</td>
</tr>
</tbody>
</table>

Table 2: The results of developer experiments for manual discovery of services
Developing Families of Software Services: A Semantic Web Approach

<table>
<thead>
<tr>
<th>Automated Service Discovery</th>
<th>Query Generation Time (ms.)</th>
<th>Querying Seekda (ms.)</th>
<th>Service Ranking Time (ms.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.0</td>
<td>13081.0</td>
<td>6.0</td>
</tr>
</tbody>
</table>

Table 3: Automated Service Discovery and Selection

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™.

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

<table>
<thead>
<tr>
<th>Developers’ Inspection of Returned Services</th>
<th>Initial Developers Selection Hit</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>D2</td>
</tr>
<tr>
<td>Service Discovery Results</td>
<td></td>
</tr>
<tr>
<td>63%</td>
<td>56%</td>
</tr>
<tr>
<td>D1</td>
<td>D2</td>
</tr>
<tr>
<td>62.5%</td>
<td>100%</td>
</tr>
</tbody>
</table>

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
Developing Families of Software Services: A Semantic Web Approach

to have to manually go through the list of all potential services to find their desired services. Therefore, our experiment results support the H1 hypothesis, i.e., shared conceptualizations (ontologies) do enhance semantics based discovery during the domain engineering. Additionally, low response times of the algorithm show its high applicability in realistic environments.

5.2. Specialization Algorithm Performance

Each stage of the ontology-based configuration automatically discards all unfeasible features of that stage. The algorithm for feature removal traverses features of the feature model, and removes all 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 in a stage of staged configuration is called Unfeasible Features Set (UFS). In order to demonstrate the benefits of automated staged configuration, i.e., benefits of automating the removal of UFSs, and herewith to evaluate the H2 hypothesis of our approach, we have measured the mean size of the UFS for various feature models. Mean size of UFS is computed by computing UFS of every feature in the feature model, and dividing it with the number of features.

In order to perform these experiments, we have used the modified version of the automated feature model generator introduced by White et al. (2008) is used for the generation of feature models. The modification was performed just to exclude the usage of Java Choco Constraint Solver (Choco, 2011), given that we used the DL-based verification and implemented our own DL-based specialization 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 by White et al. (2008). 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

<table>
<thead>
<tr>
<th>Crosstree conns</th>
<th>0</th>
<th>0.25n</th>
<th>0.5n</th>
<th>0.75n</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM Size (n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2.0</td>
<td>2.4</td>
<td>2.6</td>
<td>2.8</td>
</tr>
<tr>
<td>50</td>
<td>9.1</td>
<td>10.5</td>
<td>12.7</td>
<td>13.5</td>
</tr>
<tr>
<td>100</td>
<td>19.4</td>
<td>21.6</td>
<td>21.9</td>
<td>27.9</td>
</tr>
<tr>
<td>500</td>
<td>86.4</td>
<td>102.1</td>
<td>123.4</td>
<td>138.0</td>
</tr>
<tr>
<td>1000</td>
<td>172.1</td>
<td>198.4</td>
<td>223.6</td>
<td>385.5</td>
</tr>
</tbody>
</table>

Table 5: Mean size of Unfeasible Feature Set in specializations
Developing Families of Software Services: A Semantic Web Approach

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. These results approve the H2 hypothesis, as they clearly show benefits of ontology-based staged configuration.

5.3 Automated Generation of Service Compositions
To evaluate the H3 hypothesis, i.e., that our approach facilitates reuse-based development in application engineering, we have 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 (2007), 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 Asadi et al (2009). Nevertheless, even in this stage of development, results in Table 1 are in favour of the H3 hypothesis. Particularly, having in mind that the generated WSMO compositions are formed from already existing service specifications discovered in the domain engineering phase.

<table>
<thead>
<tr>
<th>WSML Component</th>
<th>Percentage of Automatically Generated WSML</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS Description (Total)</td>
<td>76 %</td>
</tr>
<tr>
<td>Capability</td>
<td>47%</td>
</tr>
<tr>
<td>Interface</td>
<td>82%</td>
</tr>
<tr>
<td>Goal Description</td>
<td>74%</td>
</tr>
</tbody>
</table>

Table 6: Average percentages of automatically generated WSML

6. RELATED WORK
Given the comprehensiveness of the proposed approach, there are several major groups of related work as described in the remainder of the section.

6.1 Software Services and Product Lines
Helferich et al (2007) have 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 (2007) 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 (2008) and Lee et al (2010) considered adapting feature-oriented product line engineering. They first developed an infrastructure consisting of generic services for the particular family. Then, they constructed and evolved 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 the 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 (2009) 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 consists 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 (2009). 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.

All of previously mentioned approaches exploit the synergy between SPL and SOAs. However, to our knowledge, we are the first to exploit benefits of Semantic Web in engineering of families of service-oriented systems.

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 (2002) has 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 in graphical representation such as feature diagram is not convenient. Czarnecki et al (2004) 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) (Czarnecki et al, 2005). Batory (2005) 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 (see Battory, 2005). LTMS mostly use the Satisfiability problem, Constraint Satisfaction Problem (CSP) (see Benavides et al, 2005), and Binary Decision Diagrams (Mendonca et al, 2008) to verify feature
Developing Families of Software Services: A Semantic Web Approach

models. Wang et al (2007) 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 (2006) 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.

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 (see Keller et al., 2006), 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 (see Keller et al., 2005a). So-called lightweight set-based discovery (see Keller et al., 2005b) 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 (Keller et al., 2006) 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 the near future.

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 (2007) 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 (2008). 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.
Developing Families of Software Services: A Semantic Web Approach

As mentioned in Section 5.3, the amount of manual intervention after our transformations is comparable to the state-of-the-art solutions (see Brambilla et al., 2007). 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. DISCUSSION

As our previous sections demonstrated, the above mentioned research approaches for combining SPLs and SOAs 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, we are 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 (see Blake, 2007), this is the first attempt in the context of families of SOAs.

Also different approaches for configuration and verification of feature models have 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.

It will be incorrect to claim that our contribution w.r.t. search 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. Of course, 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 (see Klutsch et al., 2009) 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. (2006), 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).

8. 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.

ACKNOWLEDGEMENT
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
Developing Families of Software Services: A Semantic Web Approach


Developing Families of Software Services: A Semantic Web Approach


Developing Families of Software Services: A Semantic Web Approach

BIOGRAPHICAL NOTES

Marko Bošković (http://mboskovic.athabascau.ca/) is a graduate from the Military Academy, Belgrade, and holds a PhD from the University of Oldenburg, Germany. Currently he is a postdoc fellow in the School of Computing and Information Systems at Athabasca University, and School of Interactive Arts and Technology at Simon Fraser University, Canada. He is researching in the area of integrating semantic web technologies in the process of development of software product lines, as well as integration of non-functional properties assessment techniques in the development of service oriented software product lines. His research interests are integration of non-functional system requirements estimation and evaluation in domain specific modeling languages, software product lines and model-driven engineering in general. He served as reviewer for several journals and international conferences and workshops. He is on the editorial board of one electronic journal.

Bardia Mohabbati is currently a PhD student at Simon Fraser University. His research interests include the areas of software engineering and service-oriented computing with a particular focus on engineering service-oriented software product lines, quality-aware software configuration and optimization. His research goal is to make enterprise service-based applications and business processes more adaptable and flexible to changing users’ requirements. He received his MSc degree in computer science from Amirkabir University of Technology. He has served as program committee member of several international conferences and reviewer for international journals.

Nima Kaviani is a PhD student at the university of British Columbia working in the area of application partitioning, distribution and deployment to the cloud. Nima’s research mainly is focused on facilitating resource utilization for applications deployed in a hybrid cloud to minimize cost and performance implications of a cloud deployment and to preserve security and privacy requirements of enterprises. Further to this, Nima has worked on utilizing the concepts of software product line engineering towards developing automated software as a service platform for ubiquitous environments. Nima has authored more than two dozen research papers and has served as the reviewer and program commity member for various reputable conferences in the area of cloud computing, product line engineering, and middleware.

Mohsen Asadi is a research assistant and PhD candidate in Simon Fraser University (SFU) working on “Model-Driven Development of Families of Semantically-enabled Service-oriented Architectures” project. He obtained his MSc in computer science (2009) from Sharif University of Technology (SUT), Iran. His main research interests are: Software product line engineering, service oriented architecture, requirements engineering, and model-driven development.
Developing Families of Software Services: A Semantic Web Approach

Jeff Rusk is a software engineer with Q1 Labs Inc. and an M.Sc (Information Systems) graduate from Athabasca University. His current research efforts are focused on service oriented architecture strategies and practices for integrating vulnerability management software with security intelligence platforms.

Dragan Gašević (http://dgasevic.athabascau.ca) is a Canada Research Chair in Semantic Technologies and an associate professor in the School of Computing and Information Systems at Athabasca University. He is also an adjunct professor in the School of Interactive Arts and Technology at Simon Fraser University, an IBM CAS Faculty Fellow, and an associated research member of the GOOD OLD AI Research Network at the University of Belgrade. His research interests include semantic technologies, software language engineering, technology-enhanced learning, and service-oriented architectures. Being a principal investigator in several national and international research projects, he is a recipient of Alberta Innovates Technology Futures’ 2008 New Faculty Award for his research on families of semantically-enhanced business processes and service-oriented systems. A co-author of numerous research papers and often keynote speaker, he is the lead author of the book monograph Model Driven Engineering and Ontology Development published in two editions by Springer in 2006 and 2009. Committed to the development of international research community and the concept of cross-community engineering, Dragan has co-founded several new events (e.g., SLE and LAK), has served as a chair and member of numerous steering and program committees, and edited several journal special issues and books.

Ebrahim Bagheri (http://ebagheri.athabascau.ca/) is currently an assistant professor at the AU School of Computing and Information Systems, a visiting professor with the University of British Columbia, honorary research associate at the University of New Brunswick and an IBM CAS faculty fellow. Previously, he was affiliated with the Institute for Information Technology at the National Research Council Canada. Ebrahim completed his PhD degree at the Faculty of Computer Science, University of New Brunswick, and specializes in topics ranging from the meta-modeling of complex interconnected systems to collaborative information systems design. Currently, his research focuses on two areas namely, quality engineering for software product lines and social informatics for healthcare. His work on collaborative modeling is one of a kind in providing tools and techniques for collaborative risk management and quality engineering. He has extensively published over 80 papers in top-tier international journals and conferences including the IEEE Transaction on Systems, Man, and Cybernetics Part A, Elsevier journal of Systems and Software, Springer’s Software Quality Journal, Wiley’s Journal of Software Maintenance and Evolution: Research and Practice, Springer’s Information Systems Frontiers Journal, Springer’s Knowledge and Information Systems Journal among others.
Developing Families of Software Services: A Semantic Web Approach

Marek Hatala (http://www.sfu.ca/~mhatala) is an associate professor and a graduate chair at the School of Interactive Arts and Technology at Simon Fraser University. He received his PhD degree in cybernetics and artificial intelligence from the Technical University of Kosice. His research interests are in the areas of knowledge management, artificial intelligence, distributed systems, user modeling, interoperability, security and trust policies, e-learning and collaborative systems.