Feature Models as Service Contracts in Service Oriented Architecture

Akram Kamoun, Mohamed Hadj Kacem, Ahmed Hadj Kacem, Khalil Drira

Feature Models as Service Contracts in Service Oriented Architecture

Akram Kamoun[1], Mohamed Hadj Kacem[1], Ahmed Hadj Kacem[1], and Khalil Drira[2]

[1] *Laboratory of Development and Control of Distributed Applications (ReDCAD), National Engineering School of Sfax, Tunisia*
[2] *LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France*
akram.kamoun@redcad.tn, mohamed.hadjkacem@redcad.org, ahmed.hadjkacem@fsegs.rnu.tn, and khalil@laas.fr

Abstract

The service contract is one of the fundamental design principles in the Service Oriented Architecture (SOA). Its goal is to express the features (e.g., services and capabilities) of Service Providers (SPs) so Service Consumers (SCs) can identify them to communicate correctly. The two most known service contracts in the literature are: WSDL for SOAP and WADL for REST. We identify that these service contracts suffer from several problems (e.g., they only allow expressing a limited set of features). Also, we notice from the literature a lack of service contracts dedicated for SC. In order to overcome these problems, we propose two Features Models (FMs) based on SOA design patterns that can be considered as generic and formal service contracts for SP and SC. We propose extensions to the Feature MetaModel (FMM) and semantic constraints that ensure generating fully functional, valid, customized and consistent SPs and SCs from the proposed FMs.

Index terms— Service oriented architecture; Service contract; Service provider and service consumer; Software product line; Feature model; Design pattern.

1 Introduction

In recent years, the Service Oriented Architecture (SOA) has been widely used as an architectural model that represents a distributed computing platform in the software engineering. In the literature, many works have been proposed to facilitate the development of SOA applications [1], [2], [3], [4]. SOA uses services as the essential means through which Service Providers (SPs) and Service Consumers (SCs) can communicate. Services consist of a set of capabilities that are developed and exposed by an SP and can be invoked by different SCs. Each capability can offer different SP features such as the communication technologies (SOAP, REST and Messaging-Oriented Middleware "MOM"), the authentication support, publish/subscribe support, message acknowledgements and the support of synchronous and asynchronous communications. Service contracts

are documents including meta information used to describe these features. They can be used by SCs to identify these SP features in order to communicate correctly with the SP.

Erl classifies the service contract as one of the fundamental design principles in SOA [4]. The two most used traditional service contracts in SOA are: WSDL for SOAP and WADL for REST. We identify that these service contracts suffer from several problems: First, they only allow expressing a limited set of SP features. Thus, some features cannot be formally described in these service contracts to impose for example certain business logics or complex constraints (e.g., propositional constraints between features) or Service Level Agreements (SLAs) or non functional requirements that SCs must respect to communicate correctly with the SP. In this case, the SP developer needs to write informal documentations so that SC developers can identify the offered SP features. The problem is that, due to this informality, these documentations cannot be included in an automatic process to generate the artifacts of SCs and SPs. Second, service contracts are dependent to specific communication technologies. For example, the WSDL expresses only the features of SOAP and WADL defines only the features of REST. The problem is that SP developers have to use different syntaxes even to describe the same features (e.g., input and output data information) in order to develop the service contracts WSDL and WADL. This can lead to misinterpretation and difficulty in understanding these contracts. Third, some communication technologies (e.g., MOM) do not offer service contracts. This makes it difficult for SCs to identify the features of these communication technologies. Fourth, there is a lack in the literature of service contracts dedicated for SC that generate the artifacts of SCs. Fifth, developing many separated service contracts (e.g., WSDL and WADL) might be needed to identify the different features offered by the SP. This decreases the governance of the SP and makes it difficult for SC developers to identify these features. These identified problems can make the development of SPs and SCs costly, a repetitive task, time-consuming, error-prone and requires a solid core of expert knowledge.

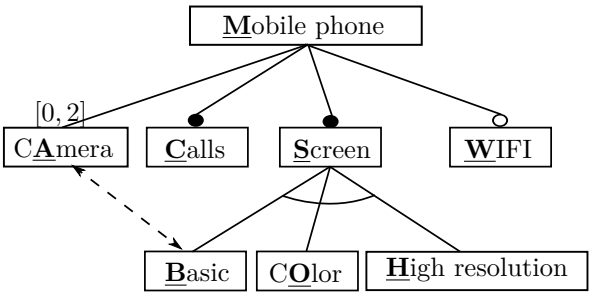
In order to overcome these problems, we propose designing new service contracts for SP and SC that should take into consideration the following challenges. They should be generic, formal and independent of the communication technologies. They should provide mechanisms to express features and complex constraints. They also should permitting to generate fully functional, valid, customized and consistent SPs and SCs. In this paper, our objective is to carry out an in-depth and rigorous study that addresses these challenges. In this context, we present an SOA approach based on the Software Product Line (SPL) [5]. The latter is a paradigm allowing the mass customization of applications. We particularly propose in our approach two Feature Models (FMs) [6] for SP and SC that can be considered as new service contracts which take into consideration the identified challenges. We note that the goal of FMs is to model variability in SPL. The variability consists of the ability of an artifact to be customized or configured in a particular context.

The rest of this paper is structured as follows. In Sect. 2, we provide a brief overview of the FM. In Sect. 3, we introduce our approach. In Sect. 4, we evaluate our approach. In Sect. 5, we present threats to validity. In Sect. 6, we discuss some related works. We summarize the main conclusions and suggest future work in Sect. 7.

2 A brief overview of the feature model

The Software Product Line (SPL) is a paradigm proposing advanced techniques for the mass customization of applications using as an essential means the concept of variability modeling. SPL developers often design a *Variability Model (VM)* [5] to model the customized applications, based on features, that the SPL offers to generate. Then, an end-user needs to derive an Application Model (AM) (also called configuration [7]) from this VM by selecting the required features. This derived AM is used to generate the artifacts of the required application.

The Feature Model (FM) [6] is one of the most used VMs in the literature that will be used in this work. The structure of the FM is a rooted tree of features. It is possible to define constraints between features in the FM in order to restrict the legal combination of features (i.e., to restrict the AMs that can be derived).



WIFI → COlor ∨ High resolution

(a) FM_1

$F_1 = \{M, A, C, S, W, B, O, H\}$

(b) Corresponding features

$[\![FM1]\!] = \{$
$\{M, O, S, W\},$
$\{M, H, C, A, S\},$
$\{M, O, S, C\},$
$\{M, A, S, C, W, H\},$
$\{M, W, A, S, O, C\},$
$\{M, C, H, S, W\},$
$\{M, C, S, H\},$
$\{M, C, B, S\},$
$\{M, S, O, C, A\}$
$\}$

(c) Corresponding AMs

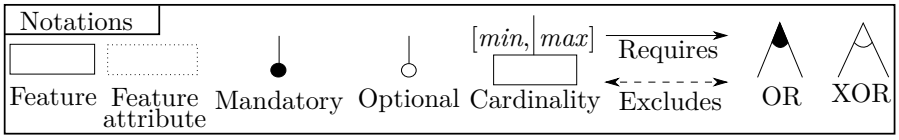


Figure 1: Feature model FM_1, corresponding features and AMs, and notations

The FM can be defined through different notations [6] (see Fig. 2). Features can be either *mandatory* (e.g., Screen) or *optional* (e.g., WIFI). The *feature attribute* allows to add an attribute value (e.g., integer) to specify extra-functional information for features (e.g., camera sensor size). The *cardinality-based feature* $[min, max]$ defines a lower and an upper bounds of instances of a given feature that can be expressed in an AM. The *"requires"* constraint is in the form of: if a feature A requires a feature B, then the selection of A in the AM requires the selection of B. The *"excludes"* constraint is in the form of: if a feature A excludes a feature B, then both features cannot be selected in the same AM. Two widely used types of a feature group can be identified: the *feature group XOR* $[1, 1]$ and the *feature group OR* $[1, n]$. A *feature group XOR* $[1, 1]$ allows

selecting exactly one out of its child features which can be called alternative exclusive features. A *feature group OR* $[1, n]$ allows selecting one or many of its child features which can be called alternative inclusive features. The FM allows to define *propositional constraints* between features (e.g., see WIFI $\rightarrow$ Color $\vee$ High resolution in Fig. 2).

We present, in Fig. 2, an example of a FM that reflects the features of a mobile phone.

Definition 1 *(Feature model semantic)* $F = \{f_1, f_2, \cdots, f_n\}$ *defines the finite set of features of a given* FM*. The set of the features* F_1 *of* FM_1 *in Fig. 2a is depicted in Fig. 2b. The set of AMs that can be derived from a* FM *is denoted* $[\![FM]\!]$*. Each* $AM = \{f_1, f_2, \cdots, f_m\}$ *is composed of a set of features. The set of valid AMs of* FM_1 *in Fig. 2a is enumerated in Fig. 2c.*

Definition 2 *(Feature model specialization) Specialization is a transformation process that allows the elimination of some AMs from* FM *to produce* $FM_{specialize}$ *[6]. The specialization process requires that:* $[\![FM_{specialize}]\!] \subset [\![FM]\!]$*. Czarnecki et al. [6] introduce seven ways for specialization (e.g., an optional feature becomes either mandatory or it is omitted).*

3 Contribution

3.1 Approach overview

The contributions proposed in this paper are based on results obtained in our earlier work [8]. The latter proposes an approach named Multiple SPLs (MSPL) for Service Oriented Architecture (SOA) that allows developers to generate *customized*, *valid* and *consistent* Service Providers (SPs) and Service Consumers (SCs). This allows to increase the reusability principle, improve reliability, make it easier and gain time in the development of SPs and SCs. The MSPL [9] is a paradigm representing a set of related and dependent SPLs to contribute to a larger SPL. Consistent SP and SC means in our case that they can communicate correctly. For example, if a given capability in the SP implements the communication technologies SOAP and REST and requires an authentication, then the SC must use one of these communication technologies and provide credentials for authentication to invoke this capability. We present in Fig. 3 on overview of our approach which is ensured by three actors that are: MSPL developer, SP developer and SC developer. Our approach requires eleven main implementation steps. We define solid lines as the manual steps ensured by developers and the dotted lines as the automatic steps ensured by our approach.

In the first step, the MSPL developer implements an MSPL that is composed of two dependent SPLs for SPs and SCs, named SPL_{SP} and SPL_{SC}, respectively. The variability of these two SPLs is managed by two FMs, respectively named FM_{SP} (see Fig. 5) and FM_{SC} (see Fig. 4) which express the variability of the features of SP and SC. We consider these FMs as the service contracts of SP and SC. In the second step, we check the consistency of FM_{SC} and FM_{SP}. This allows to check that all the AMs that can be derived from these FMs permit to generate consistent SPs and SCs. This step is newly introduced in this paper compared with our earlier work in [8]. In this paper, we essentially study in details these two steps.

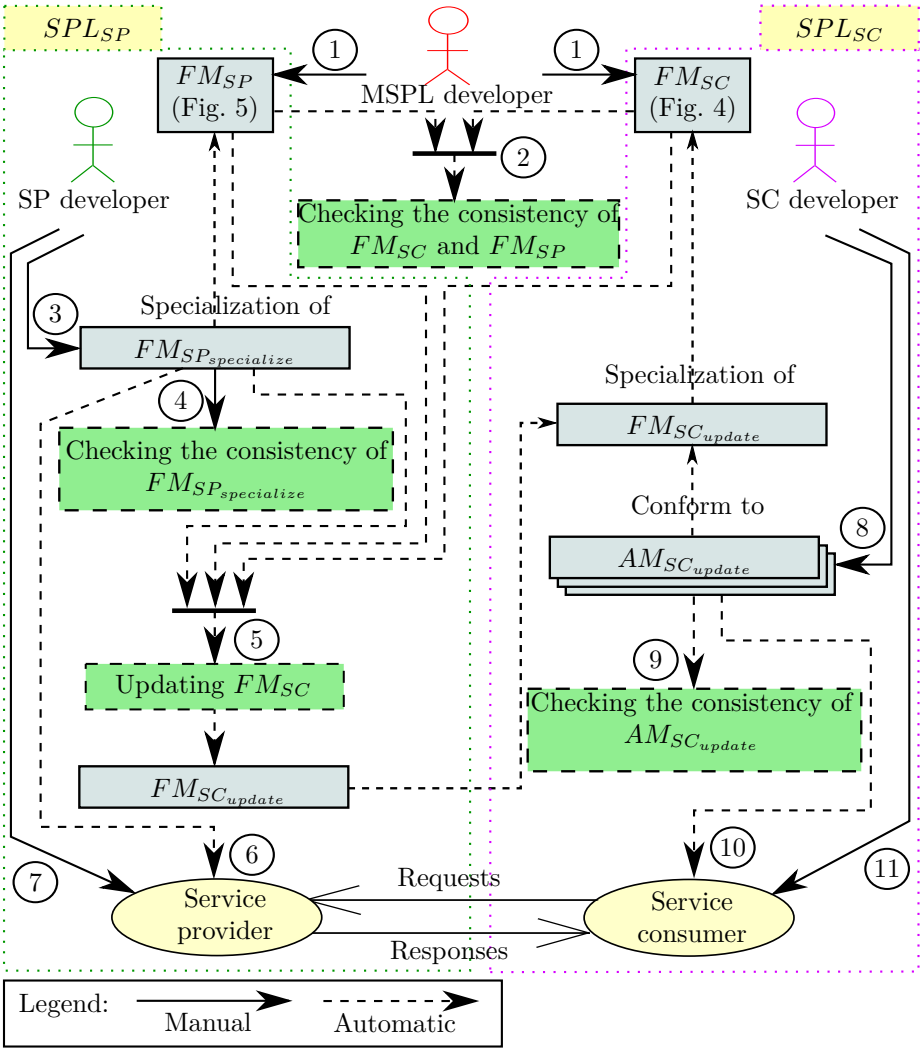


Figure 2: Approach overview

In the third step, the SP developer specializes a FM, named $FM_{SP_{specialize}}$, from FM_{SP}. This step has two goals. First, it defines the features that the SP developer wants to implement in the SP. Second, it helps to show the variability supported by the SP to the SC developers. In the fourth step, we check the consistency of the specialized FM, i.e., it must be a specialization of FM_{SP} (see Definition 2). In the fifth step, we use the update operator, which has been introduced in [8], to propagate the variability of the $FM_{SP_{specialize}}$ to FM_{SC}, i.e., updating the variability of FM_{SC} with that of $FM_{SP_{specialize}}$. The result of this operator is an updated FM_{SC}, named $FM_{SC_{update}}$. For example, if $FM_{SP_{specialize}}$ omits the feature `SOAP` (see Figs. 4 and 5), the $FM_{SC_{update}}$ must omit it as well. This allows the $FM_{SC_{update}}$ to derive customized and valid SCs which are *consistent* with the generated SP. In the sixth and seventh steps, a

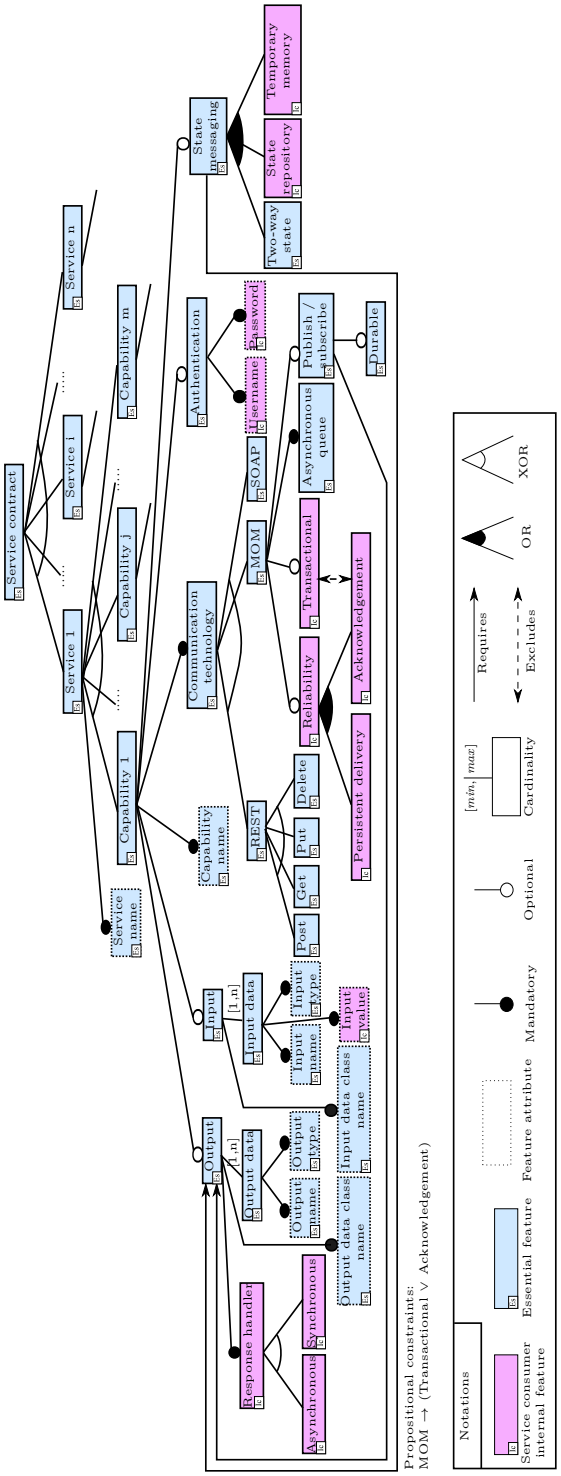


Figure 3: Feature model FM_{SC} (readers of the electronic version can zoom in)

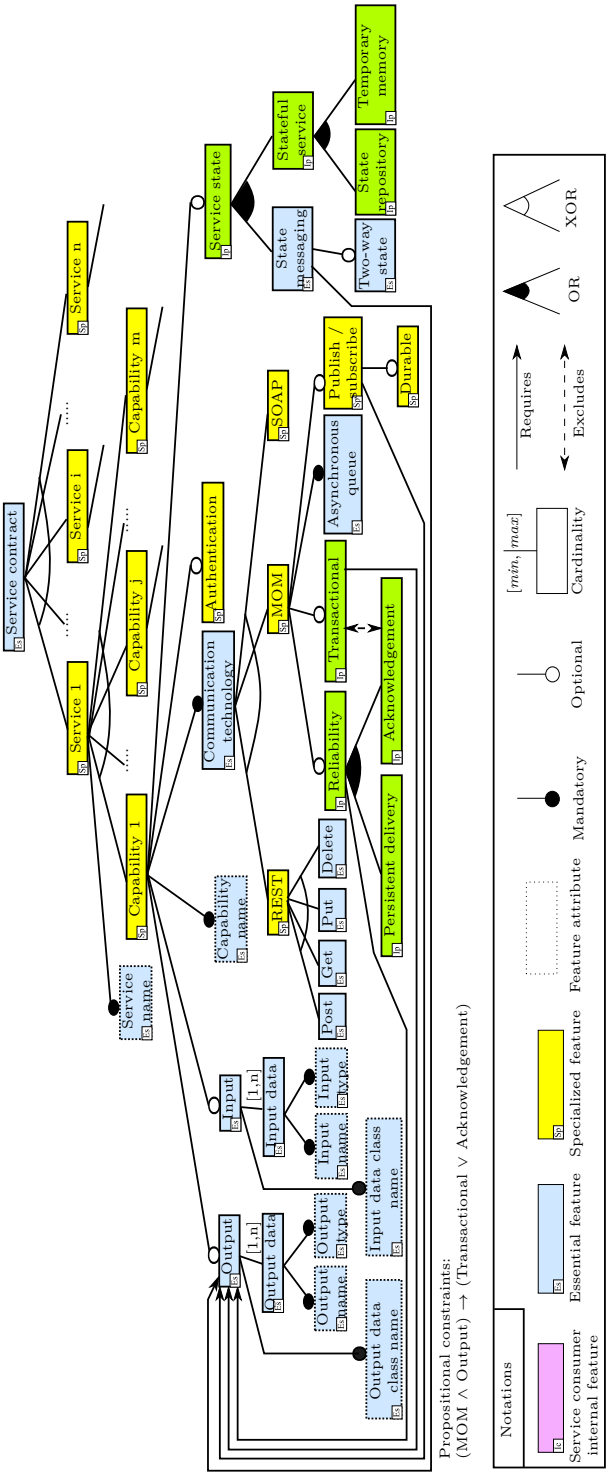


Figure 4: Feature model FM_{SP} (readers of the electronic version can zoom in)

fully functional, valid and customized SP is generated from $FM_{SP_{specialize}}$ which can be adapted if necessary by the SP developer.

In the eighth step, the SC developer derives one or many AMs, named $AM_{SC_{update}}$, from the $FM_{SC_{update}}$ according to his/her case study. In the ninth step, we check that these AMs are in conformity with the $FM_{SC_{update}}$. In the tenth and eleventh steps, a fully functional, valid and customized SC, which is consistent with the generated SP, is generated from these AMs which can be adapted if necessary by the SC developer. At this point and theoretically, we can ensure that the generated customized SP and SC are valid and consistent.

3.2 Extensions of the feature metamodel

The FM proposes notations to model variability for a general use (see Fig. 2). However, in order to take into consideration the particularities of the variability of the SP and SC features in respectively FM_{SP} and FM_{SC}, we need to extend the Feature MetaModel (FMM) with new notations. We propose adding to the FMM of Czarnecki *et al.* [6] these five UML classes `Feature type`, `Internal`, `Shared`, `Essential` and `Specialized` (see Fig. 6). We define the `Internal`, `Essential` and `Specialized` as concrete classes and the `Feature type` and `Shared` as abstract classes. Hence, the MSPL developer must specify for each feature in the FM if it is either `Internal`, `Essential` or `Specialized`. We propose in Fig. 6, the graphical notations that can be used in FM_{SP} and FM_{SP} to describe these three feature types.

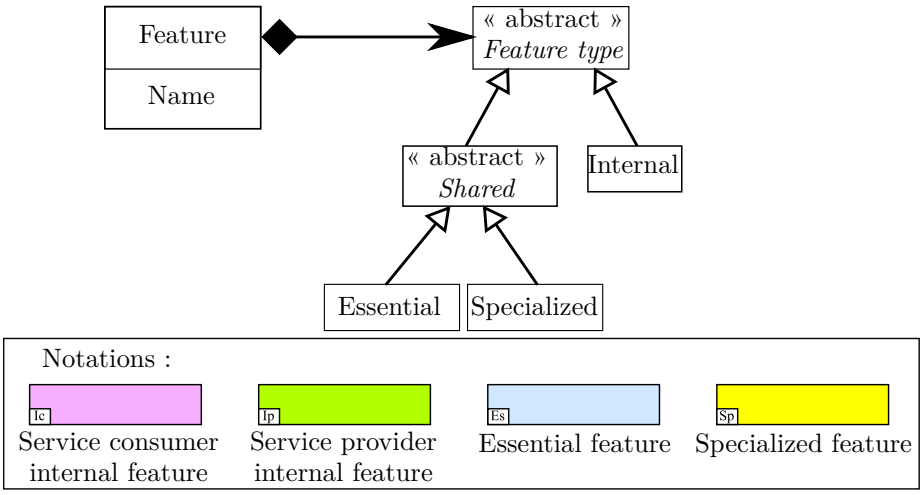


Figure 5: Extensions of the feature metamodel

3.2.1 Internal features

An internal feature is used to express the internal variability of SP and SC in FM_{SP} and FM_{SC} respectively. We propose that the variability of the internal features of FM_{SP} (respectively, FM_{SC}) are not visible to SC developers (respectively, SP developers). For example, the features `Reliability` and

`Transactional` which allow to configure the `MOM` communication technology, are expressed as internal features in FM_{SP} and FM_{SC} (see Figs. 4 and 5). The SC developer (respectively, SP developer) will not have the knowledge about how the variability of these features is resolved in the SP (respectively, SC). Thus, we propose that the variability of the SP internal features of $FM_{SP_{specialize}}$ will not be propagated to FM_{SC} in the fifth step of our approach (see Fig. 3). In other words, the $FM_{SC_{update}}$, which is an update of FM_{SC} that has been generated from this step, will not present any information about the variability of SP internal features of $FM_{SP_{specialize}}$. In this context, internal features can be useful to hide some sensitive SP variability (e.g., the security policies of the SP) from the SC developers. This allows to protect the variability of the SP from malicious SCs. Acher *et al.* [10] and Metzger and Klaus [11] demonstrate that exposing certain variability of an SPL can lead to severe consequences and should then be protected. In fact, since the SP internal features are not considered by the SC, then we propose that their variability must be resolved (i.e., become mandatory or should be omitted) when specializing $FM_{SP_{specialize}}$ (see step 3 in Fig. 3).

3.2.2 Shared features

A shared feature is used to manage and interrelate the variability which exists in both FM_{SP} and FM_{SC}. Its goal is to inform SC developers about the features supported by the SP (e.g., the supported communication technologies, see Figs. 5 and 4) that he/she can use to invoke capabilities. In contrast with internal features, the variability of shared features in $FM_{SP_{specialize}}$ is propagated to FM_{SC} in the fifth step of our approach (see Fig. 3). In other words, this variability will be used, in this step, to update that of the FM_{SC} and then to generate $FM_{SC_{update}}$. For example, if a shared feature is mandatory (e.g., `SOAP`) in $FM_{SP_{specialize}}$, then it becomes mandatory in $FM_{SC_{update}}$. If it is omitted in $FM_{SP_{specialize}}$, then it is omitted in $FM_{SC_{update}}$. It is useful to define certain features as shared in order to impose certain business logics, complex constraints, SLAs or non functional requirements that both the SP and SC must respect. **Throughout in this paper, we denote by F^{sh} the set of shared features of a given FM.** In fact, we propose that a shared feature must be either a specialized or an essential feature.

3.2.3 Specialized features

A specialized feature is the one which can be specialized, when specializing $FM_{SP_{specialize}}$ from FM_{SP}. For example, if a specialized feature is optional in FM_{SP}, then it can become a mandatory feature or remain an optional feature (i.e., it preserves its variability) or is omitted in $FM_{SP_{specialize}}$. This feature type allows to show to the SC developer the variability of certain features offered by the SP such as the supported communication technologies (see Figs. 5 and 4).

3.2.4 Essential features

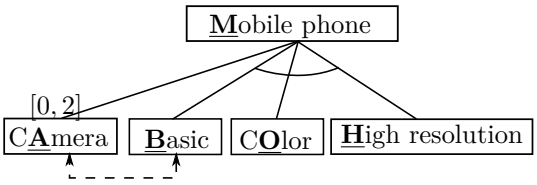
An essential feature is the one whose variability must be resolved (i.e., becomes either mandatory or it is omitted) by the SP developer when specializing $FM_{SP_{specialize}}$ from FM_{SP}. In contrast with specialized features, an essential

feature cannot be specialized. The goal of this feature type is to express features whose variability must be resolved in $FM_{SP_{specialize}}$ and propagated to FM_{SC} like the input and output data of capabilities (see Figs. 5 and 4).

3.3 Checking the consistency of FM_{SC} and FM_{SP} (step 2)

Developing consistent FM_{SC} and FM_{SP} is crucial to derive AM_{SC} and AM_{SP} that generate consistent SCs and SPs. Hence, the MSPL developer should design these two FMs while taking into consideration their consistency in mind.

We present in the following the automated analysis slice operator [7] that we particularly rely on in our work. This operator is a unary operation on FM, denoted $\Pi_{F_{slice}}(FM)$, where $F_{slice} \subseteq F$ represents a set of features to slice from FM. The overall idea behind this operator is to compute from a FM, a FM_{slice} that considers only F_{slice}. The latter can be seen as a projection of the relational algebra on FM when the other features are discarded. We present in Fig. 7 an example of the slice operator: $FM_{1_{slice}} = \Pi_{M,A,B,H,O} FM_1$ (see Fig. 2).



(a) $FM_{1_{slice}} = \Pi_{M,A,B,H,O} FM_1$

$F_{1_{slice}} = \{M, A, B, H, O\}$

(b) Corresponding features

$[\![FM_{1_{slice}}]\!] = \{$
$\{M, B\},$
$\{M, O\},$
$\{M, A, O\},$
$\{M, H\},$
$\{M, A, H\}$
$\}$

(c) Corresponding AMs

Figure 6: Feature model, corresponding features and AMs of $FM_{1_{slice}} = \Pi_{M,A,B,H,O} FM_1$ (see Fig. 2)

We identify three semantic constraints that should be considered to develop consistent FM_{SP} and FM_{SC}, which are:

- First, the following instruction should be true: $F^{sh}_{SP} = F^{sh}_{SC}$. It ensures that all features defined by the shared notation (i.e., specialized or essential) in FM_{SP} exist and defined by the shared notation in FM_{SC} and vice-versa.
- Second, the FM_{SP} should be developed in a way that each of its AM_{SP} is consistent (i.e., can communicate) at least with an AM_{SC} of FM_{SC}, formally: $\forall AM_{SP} \in [\![FM_{SP}]\!], \exists AM_{SC} \in [\![FM_{SC}]\!]$ such that AM_{SC} is consistent with AM_{SP}. The objective is to ensure that all the SPs that can be generated from AM_{SP} can be consumed by SCs that are generated from AM_{SC}. For example, if the protocol SOAP is expressed in FM_{SP}, then the FM_{SC} must support it. This constraint can be formulated as: $[\![\Pi_{F^{sh}_{SP}}(FM_{SP})]\!] \subseteq [\![\Pi_{F^{sh}_{SC}}(FM_{SC})]\!]$.
- Third, like to the second constraint, the FM_{SC} should be designed in a way that each of its AM_{SC} is consistent at least with an AM_{SP} of FM_{SP}, formally: $\forall AM_{SC} \in [\![FM_{SC}]\!], \exists AM_{SP} \in [\![FM_{SP}]\!]$ such that AM_{SP} and AM_{SC} are consistent. The goal is to guarantee that all the SCs that

can be generated from AM_{SC} are supported by SPs that are generated from AM_{SP}. For example, if the FM_{SC} supports the publish/subscriber communication, then the FM_{SP} must support it as well. This constraint can be formulated, as follows: $[\![\Pi_{F^{sh}_{SC}}(FM_{SC})]\!] \subseteq [\![\Pi_{F^{sh}_{SP}}(FM_{SP})]\!]$.

The goal of the two slice operations $\Pi_{F^{sh}_{SP}}(FM_{SP})$ and $\Pi_{F^{sh}_{SC}}(FM_{SC})$ is to slice the shared features respectively from FM_{SP} and FM_{SC}. This is important to compare the AM_{SC} and AM_{SP} of FM_{SC} and FM_{SP} based on their shared features. In order to respect these three constraints, we propose the following rule:

$$[\![\Pi_{F^{sh}_{SP}}(FM_{SP})]\!] = [\![\Pi_{F^{sh}_{SC}}(FM_{SC})]\!] \quad (1)$$

If Equation 1 is ensured when developing the FM_{SP} and FM_{SC}, we can ensure that these FMs allow to always generate consistent SPs and SCs. Otherwise, three possible cases can be identified, as follows:

- if $[\![\Pi_{F^{sh}_{SC}}(FM_{SC})]\!] \subsetneq [\![\Pi_{F^{sh}_{SP}}(FM_{SP})]\!]$, then $[\![\Pi_{F^{sh}_{SP}}(FM_{SP})]\!] \setminus [\![\Pi_{F^{sh}_{SC}}(FM_{SC})]\!]$ are AMs offered by SPs but cannot be used by SCs. Despite this disadvantage, this case still ensures that all the AMs of FM_{SC} derive SCs which are consistent with any generated SP from FM_{SP}.
- if $[\![\Pi_{F^{sh}_{SP}}(FM_{SP})]\!] \subsetneq [\![\Pi_{F^{sh}_{SC}}(FM_{SC})]\!]$, then $[\![\Pi_{F^{sh}_{SC}}(FM_{SC})]\!] \setminus [\![\Pi_{F^{sh}_{SP}}(FM_{SP})]\!]$ are AMs offered by SCs but cannot be supported by SPs. In this case, these AMs will derive SCs that are inconsistent with any generated SP from FM_{SP}.
- if $[\![\Pi_{F^{sh}_{SP}}(FM_{SP})]\!] \cap [\![\Pi_{F^{sh}_{SC}}(FM_{SC})]\!] = \varnothing$, then the generated SP and SC will always be inconsistent and cannot communicate accordingly.

3.4 Developing FM_{SC} and FM_{SP} (step 1)

We demonstrate in the introduction that the traditional service contracts (WSDL and WADL) suffer from many problems (e.g., inability to express complex constraints). In order to overcome these problems, we propose in this paper, integrating the variability concept to model formally the features and constraints of SP and SC. In this context, we propose FM_{SC} and FM_{SP} (Figs. 4 and 5) as service contracts for SC and SP, respectively. These FMs are designed to be formal, generic, independent of the communication technologies and can express the features and constraints of SC and SP. They are also designed to be able to generate fully functional, customized, valid and consistent SCs and SPs. These FMs contain mandatory (e.g., information about services and capabilities) and optional (e.g., service state) features that model the variability of SC and SP. These FMs can also be considered as reference models [12] which reflect the variability of practical features of SC and SP.

We rely on SOA Design Patterns (DPs) [3] to define the features of FM_{SC} and FM_{SP}. Many advantages can be enumerated when expressing DPs as features in the FM_{SC} and FM_{SP}. DPs are introduced by veteran problem solvers in order to provide an appropriate and proven design solutions for specific problems in certain contexts. Hence, integrating DPs in our SPL_{SC} and SPL_{SP} and in particular in our FM_{SC} and FM_{SP} allows to ensure that the applications

that can be derived are based on proven solutions. DPs show the right level of abstraction to describe a certain solution. They have also a major benefit of providing a common language, which is understandable by the developers instead of using terminologies related only to a certain technology. For example, by simply saying reliable messaging DP [3] is more efficient and easier than explaining it in details.

3.4.1 Shared features of FM_{SC} and FM_{SP}

We present in this section the shared (i.e., essential and specialized) features presented in FM_{SC} and FM_{SP} (see Figs. 4 and 5). We define the roots of these FMs as essential features named `Service contract`. The latter are composed of the alternative exclusive features $\texttt{Service}_1, \cdots, \texttt{Service}_n$ that reflect the services offered by the SP which can be consumed by SCs. Each feature `Service` is composed of a mandatory feature attribute `Service name` that refers to the service name and alternative exclusive features $\texttt{Capability}_1, \cdots, \texttt{Capability}_m$.

The mandatory feature attribute `Capability name` refers to the name of the capability. The optional feature `Input` expresses the input information of the capability. It is composed of the cardinality-based feature $[1, n]$ `Input data` and the mandatory feature attribute `Input data class name`. The feature `Input data` is composed of two mandatory feature attributes `Input name` and `Input type` allowing to define the signature of the capability. The feature attribute `Input data class name` refers to the class name that regroups the input data information. Similarly to the feature `Input`, we define the feature `Output` to specify the output information of the capability. If the feature `Output` is omitted when deriving $FM_{SP_{specialize}}$ then the type of the result of the capability is void and the communication type is considered as one-way.

The mandatory feature `Communication technology` is composed of three alternative exclusive communication technology features: `SOAP`, `REST` and `MOM`. The feature `REST` is composed of four alternative exclusive features that represent the four HTTP methods most used by REST: `Post`, `Get`, `Put` and `Delete`. The feature `MOM` is composed of the mandatory feature `Asynchronous queue` and the optional feature `Publish/subscribe`. The `Asynchronous queue` is a mandatory feature that implements the asynchronous queue DP [3]. It allows to exchange messages between SP and SC via an intermediary buffer in order to ensure an asynchronous communication. The `Publish/subscribe` is a feature which implements the event driven messaging DP [3] and proceeds as follows. The SC subscribes itself to a capability. When the result of this capability is ready, it is returned to this SC and any of its subscribers (i.e., SCs). This feature is used only to send messages from the SP to the SC. Thus, we define a "requires" constraint from it to the feature `Output`. We note that if the type of communication is two-way and if the feature `Publish/subscribe` is not selected, then the exchanged data between SP and SC are carried on by the feature `Asynchronous queue`. The feature `Publish/subscribe` is composed of the optional feature `Durable`. The latter allows the SP to store messages for the SCs if the latter disconnect. Upon reconnecting, these SCs will receive these messages from the SP.

The optional feature `Authentication` implements the direct authentication DP [3]. The `State messaging` is a feature that implements the state messaging DP [3]. Its goal is to temporarily delegate the storage of state data of the SP

to messages instead of the memory. This feature can be used only in a two-way communication type because the state data is delegated to the messages sent from the SP to the SC. Thus, a "requires" constraint is defined from this feature to the feature `Output`. The feature `State messaging` is composed of an optional feature `Two-way state`. The latter's goal is to impose the SC on delegating the received state from the SP in its future outgoing messages.

In fact, it is the responsibility of the SP developer to choose if the feature `State messaging` will be integrated or omitted in the SP. The SC cannot choose if this feature will be implemented or not in the SP because it is related to the business logic of the SP and not to that of the SC. Thus, this feature cannot be defined as specialized. At this point, we can define this feature as internal in FM_{SP}. However, in order to use this feature, it must be supported by the SC [3] to work correctly. In this case, defining the feature `State messaging` as an essential feature in FM_{SP} and FM_{SC} is required. Also, the features describing the arguments of capabilities must be defined as essential because such variability must be resolved by SP and not by SC.

We propose in particular that $\texttt{Service}_1, \cdots, \texttt{Service}_n$, $\texttt{Capability}_1, \cdots, \texttt{Capability}_m$ and the communication technologies (`SOAP`, `REST` and `MOM`) should be alternative exclusive features and defined as specialized. The objective is to ensure that each $AM_{SC_{update}}$ (see Fig. 3), derived by the SC developer from $FM_{SC_{update}}$, permits to invoke only one service and only one of its capabilities using a given communication technology. Hence, each $AM_{SC_{update}}$ can be considered as a configuration file that includes the features used to invoke a given capability, such as the used communication protocol, the synchronous or asynchronous communication types, and the authentication credentials. This will increase modularity in the different $AM_{SC_{update}}$, thus allowing these AMs to be more flexible to adapt and to edit.

We note that we do not need specializing FM_{SC} in our approach (see Fig 3). Thus, we define the specialized features in FM_{SP} as essential in FM_{SC}.

3.4.2 Internal features of FM_{SC}

We add to the essential feature `Input data`, the mandatory internal feature attribute `Input value`. The latter allows to specify a value to the input data in order to invoke a capability. We also add to the essential feature `Output`, the mandatory internal feature `Response handler`. The latter is composed of two alternative exclusive internal features that are `Synchronous` and `Asynchronous`. These features are useful to specify whether the SC should synchronously or asynchronously handle the incoming messages from the SP.

The shared feature `MOM` regroups four internal features: `Transactional`, `Reliability`, `Persistent delivery` and `Acknowledgement`. The optional feature `Transactional` implements the atomic service transaction DP [3]. It allows to establish a transactional communication between the SC and the SP. The optional feature `Reliability` implements the reliable messaging DP [3]. This feature regroups two alternative inclusive internal features, which are: `Persistent delivery` and `Acknowledgement`. The feature `Persistent delivery` is used to persist the received messages so that they are not lost if the `MOM` fails. Therefore, we ensure that the messages are delivered to the receiver (SP or SC). The feature `Acknowledgement` allows the messages to be acknowledged as soon as the receiver gets it. We define the features `Acknowledgement` and `Transactional`

as mutually exclusive [3], [13]. In fact, a `MOM` requires the use of one of these two features [13]. Thus, we define the following propositional constraint: `MOM` $\rightarrow$ (`Transactional` $\vee$ `Acknowledgement`).

The shared feature `State messaging` is composed of the essential feature `Two-way state` and the two internal features `State repository` and `Temporary memory`. Erl [3] presents two approaches describing how the SC can implement the feature `State messaging`. The first approach proposes that the SC retains the state data in memory and only the SP benefits from delegating the state data to messages. In this context, the state data can be stored in a state repository (represented by the feature `State repository`) or in a temporary memory (represented by the feature `Temporary memory`) of the SC. We note that the feature `State repository` implements the state repository DP [3] which proposes deferring state data to a state repository. The goal of this DP is to avoid the limits of storing state data in a temporary memory [3]. The second approach proposes that both the SC and SP temporarily delegate the storage of state data in the exchanged messages. The essential feature `Two-way state` implements this approach. Erl reports that these two approaches require an extra custom development effort to delegate and retrieve state data by the SP and SC [3]. In our approach, this effort is handled automatically by our SPL_{SC} and SPL_{SP}.

3.4.3 Internal features of FM_{SP}

The internal features `Transactional`, `Reliability`, `Persistent delivery`, `Acknowledgement`, `State repository` and `Temporary memory` of FM_{SC} are also expressed as internal features in FM_{SP}. Their objectives is the same in both FM_{SP} and FM_{SC}. However, they are expressed differently in these two FMs (see Figs. 5 and 4) and also implemented differently in SPL_{SP} and SPL_{SC}. In FM_{SP}, the features `Transactional`, `Reliability`, `Persistent delivery`, `Acknowledgement` require the selection of the shared feature `Output`. Thus, we define "requires" constraints from these features to the shared feature `Output`. These constraints are not defined in FM_{SC}. The reason is that these features can be applied only to the outgoing messages either from the SP or from the SC. Since, the SC needs at least to send a message to invoke a capability, then these features can always be used without constraints. We also define the following propositional constraint which is required by `MOM` [13]: (`MOM` $\wedge$ `Output`) $\rightarrow$ (`Transactional` $\vee$ `Acknowledgement`).

Also, the internal features `State repository` and `Temporary memory` are expressed differently in both FMs. In fact, the FM_{SP} integrates the optional internal feature `Service state` that regroups two approaches of handling state data which are reflected by the two inclusive features `State messaging` and `Stateful service`. In contrast with FM_{SC}, the feature `State messaging` in FM_{SP} requires that the storage of state data of SP is by default temporarily delegated to messages [3]. The `Stateful service` feature is an implementation of the stateful services DP [3] and regroups the two alternative exclusive internal features `State repository` and `Temporary memory`. Since the SC does not participate in storing state data when the SP uses the feature `Stateful service`, then, in contrast with the feature `State messaging`, it is not presented in FM_{SC}. Erl reports that service state approaches can be implemented in conjunction in the SP [3]. This constraint is considered in FM_{SP} (see Fig. 5) by defining an alternative inclusive constraint between the state data features.

4 Evaluation

We propose using the case study of an Integrated Air Defense (IAD) system to show the merits of our approach in practice (see Fig. 8). The IAD is a command and control compound of geographically dispersed force elements already in peace time as well as in crisis and war. The force elements are grouped into three main forces: ground force (command and control system, radars, anti-aircrafts and infantry), air force (drones, helicopters and jet aircrafts) and maritime force (aircraft carriers and submarines). These elements communicate through services to achieve their missions. Each element can be either an SP or an SC or both. Also, each element is responsible for implementing its own requirements.

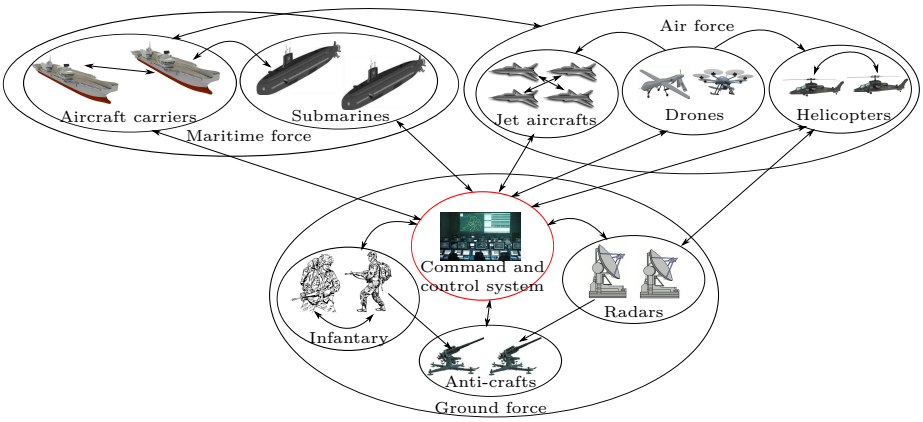


Figure 7: Case study of integrated air defense

The command and control system is composed of both an SC and an SP. Its goal is to coordinate the force elements to achieve common missions. It implements a service that broadcasts periodically a report about the current situations of the forces. In this context, it should integrate a MOM in all the capabilities of this service and this MOM should be configured to support the publish/subscribe feature (Event-driven messaging pattern [3]). Also, it should make sure to store the messages in a database for the force elements if the latter disconnect (e.g., in case of an error). For security and reliability matters, some capabilities need an authentication process and a transactional communication support. Furthermore, we must ensure that all the messages sent from/to the command and control system are persisted to a database so they are not lost in case of an error. Some air forces (drones) should implement a MOM to communicate asynchronously with the other forces. This allows to overcome the resource locking problem of synchronous communications [3]. Also, it is necessary that each message sent from this MOM should be acknowledged (i.e., via ACK messages) by the receivers as soon as they get it. The capabilities of SPs should be designed to support many communication technologies (SOAP, REST and MOM) in order to ensure their availability in case of an error. Many other requirements can be added.

We can notice that the variability information in these requirements is different and can be complex. Traditional service contracts cannot describe this

variability information as demonstrated in the introduction. Thus, developing the SCs and SPs of the IAD case study, especially if they are numerous, without taking into consideration the variability modeling in mind can be costly, error-prone, a repetitive task, time-consuming and requires a solid core of expert knowledge. The FM_{SP} and FM_{SC} allow to express the variability information of our case study.

We demonstrate in Table 1 that integrating the concept of variability in service contracts, by using FMs, is efficient, useful and allows to overcome the problems of traditional contracts.

Table 1: Demonstrating how our FM_{SP} and FM_{SC} allow to overcome the problems of the traditional service contracts (WSDL and WADL)

Traditional service contracts problems	FM_{SP} and FM_{SC} solutions
-Traditional services contracts allow to express only a limited set of features. -They do not offer formal mechanisms to express complex variability (e.g., propositional constraints) of features of SP and SC. -They are SP-centric. -Some communication technologies of SP (e.g., MOM) do not offer service contracts, which makes it difficult to identify its features for SC developers.	-The FM_{SP} and FM_{SC} allow expressing formally any features (e.g., for SOAP, REST and MOM) and complex constraints of SP and SC thanks to the semantic and notations (e.g., shared and internal feature types) of the FM.
-Traditional service contracts are dependent of specific communication technologies (e.g., WSDL is only for SOAP). They use different syntaxes even to describe the same features (e.g., input and output data information) of SP and SC. This can lead to misinterpretation and difficulty in understanding these contracts.	-The FM_{SP} and FM_{SC} use a set of formal notations, which are independent of the technology domain, to express the features of SP and SC.
-An SP might require to expose many separated service contracts to expose its offered features. This decreases the governance of the SP and makes it difficult for SC developers to identify these features.	-All the features of SP and SC are regrouped in two FMs: FM_{SP} and FM_{SC}.
-The SP developer needs to write informal documentations to describe features like SLA and non functional constraints so SC developers can identify them. The problem is that, due to this informality, these documentations cannot be included in an automatic process to generate an SC.	-The formal nature of FM_{SP} and FM_{SC} permits to be used in an automatic process to generate fully functional, valid, customized and consistent SP and SC.

We develop a tool, named MSPL4SOA (available at our website [14]), which implements all the steps of our approach (see Fig. 3) notably the proposed FM_{SP} and FM_{SC}. It is based essentially on Java EE technologies and SPL tools (e.g., the FAMILIAR [15] tool). We note that we use the Java Messaging Server (JMS) HornetQ [13] to implement the feature `MOM`. This tool allows to concretely implement the variability of the case study IAD (see Fig. 8). For example, it allows to generate a fully functional, valid and customized SP of

a command and control system based on the Switchyard 2 Enterprise Service Bus (ESB) [16]. This SP is composed of 5186 Java lines of codes like classes, interceptors and composers of messages [16]. It is also composed of 13 XML configuration files, such as XMLs for configuring the asynchronous queues, publish/subscribe features, reliable and transactional communications, and other XMLs to configure the Switchyard 2 ESB [14]. We note that this SP has been generated in less than one minute [14].

5 Threats to validity

Threats to external validity are the factors and limits within our ability that reduce the generality of our results to industrial practice. Our first concern is that the FM_{SP} and FM_{SC} express a limited number of features (see Sect. 3.4). Thus, many other features (e.g., non functional requirements) in the industry are not defined to model the variability of SP and SC. However, the features and the constraints expressed in these FMs allow to generate fully functional, customized, valid and consistent SPs and SCs. Our second concern is whether the feature types (i.e., internal, specialized and essential) proposed in Sect. 3.2 are enough, in industrial practice, to express the variability of all features of SP and SC. We demonstrate that these feature types have successfully expressed all the features presented in FM_{SP} and FM_{SC} (see Sect. 3.4).

Threats to internal validity are concerned with the degree of control of the study design and its implementation correctness. Our concern is that whether FM_{SP} and FM_{SC} generate fully functional, valid, customized and consistent SPs and SCs. We implement in our MSPL4SOA tool [14] an automatic test that checks the correctness of these FMs. In this context, this test generates an SP and an SC that include respectively all the variants that can be derived from FM_{SP} and FM_{SC}. The generated SP and SC have communicate successfully and no errors have appeared [14].

6 Related work

Managing the variability of SCs and ensuring the consistency of the variability of SCs with that of SPs are issues that have not been intensively studied. We highlight this fact essentially from the literature and in particular from the systematic mapping study of combining SOA and SPL that has been proposed by Mohabbati *et al.* [17]. Holl *et al.* [9] introduce a systematic review study about many MSPL approaches. They reveal that, in a general context, there is a lack of work handling inconsistencies between FMs and their SPLs, accordingly. The same results have been obtained in the mapping study of strategies for consistency checking on FMs presented by Santos *et al.* [18]. These issues have been considered as the main challenges in our approach. In this context, we propose two FMs, named FM_{SC} and FM_{SP} (see Sect. 3.4), in order to manage the variability of SCs and SPs. We also introduce extensions of the FMM and semantic constraints to ensure the consistency of FM_{SC} and FM_{SP} (see Sects. 3.2 and 3.3). We note that our FM_{SC} and FM_{SP} are based on SOA DPs [3] and are designed to be generic (i.e., independent of a communication technology). The communication technologies supported by our FM_{SC} and

FM_{SP} are: SOAP, REST and MOM. They are also designed to generate fully functional, customized, valid and consistent SCs and SPs.

In the literature, several works [19], [20], [21], [22], [23], [24], [25] have been proposed to manage the variability of SP. In the following, we discuss these works and compare them with our FM_{SP} (see Table 2). This comparison also includes the traditional service contracts (WSDL and WADL).

Table 2: Comparing our FM_{SP} with related work

Approach	Tool	Functional	Communication technology	Design pattern
Ed-douibi *et al.* [19]	EMF	Semi	REST	-
Parra and Joya [20]	FM	Semi	Generic	-
Kamoun *et al.* [21]	FM	-	-	+
Kajsa and Navrat [22]	FM	Semi	-	+
Abumaatar and Gomaa [23]	FM	Semi	Generic	-
Fantinato *et al.* [24]	FM	Semi	SOAP	-
Wada *et al.* [25]	FM	Semi	n/a	-
WSDL	XML	Fully	SOAP	-
WADL	XML	Fully	REST	-
Our FM_{SP}	FM	Fully	Generic	+

Ed-douibi *et al.* [19] introduce EMF data models that express many features of the REST communication technology. In this paper, we elaborate a FM_{SP} to model the REST features as well. Their EMF data models support generating more REST features (e.g., features for security) than our FM_{SP}. In contrast, our FM_{SP} relies on SOA DPs (which are proven solutions [3]) and permits to generate fully functional SPs with different communication technologies (SOAP, REST and MOM).

Parra and Joya [20] propose a FM that expresses several SP features (e.g., REST and SOAP features). The advantage of our FM_{SP} is that it permits to generate fully functional SPs and considers SOA DPs. Parra and Joya report that their FM should be extended (by modeling the input and output data features and the MOM features) to generate fully functional SPs.

We have proposed in our earlier work [21] a FM based on SOA DP for SP. The goal is to model the possible combinations between DPs. The problem of this FM is that many SP features are not modeled (mostly the input and output data features and the communication technology features) which prevents generating a fully functional and valid SP. Our FM_{SP} allows to overcome this problem.

Kajsa and Návrat [22] introduce a FM to manage the variability of the object oriented DPs. Their work helps to generate the source code of a specific DP. The advantage of our FM_{SP} is that it allows to generate both the source code of a specific DP and combinations of DPs.

Abumaatar and Gomaa [23] propose designing SOA systems as service families using SPL concepts. They introduce a FM that manages the variability of the SOA views (service contract, business process and service interface views). The advantage of our FM_{SP} is that it permits to generate fully functional SPs and considers SOA DPs. Also, our FM_{SP} models more communication technology features (see the MOM features in FM_{SP}).

Fantinato *et al.* [24] introduce a FM that models some features of the service contract WSDL of SOAP. In contrast with our FM_{SP}, as reported by the authors, many features are absent (e.g., the features of the input and output data) in their FM which prevents deriving fully functional SPs. Another advantage of our FM_{SP} is that it models the features of different communication technologies and considers SOA DPs.

Wada *et al.* [25] propose a FM that expresses the variability of non functional aspects of the SP. Although their FM includes communication features, it does not explicitly model which communication technologies it supports. Their FM can be used to extend our FM_{SP} in order to express the variability of SP non functional aspects. That is why we put the symbol "n/a" in Table 2. In contrast, our FM_{SP} relies on SOA DPs and permits to generate fully functional SPs for different communication technologies.

7 Conclusion

The service contract is one the fundamental design principles in the Service Oriented Architecture (SOA). Two of the most service contracts used in the literature are: WSDL for SOAP and WADL for REST. However, despite their usefulness, we have demonstrated in this paper that they suffer from several problems (e.g, they only allow to express a limited set of features and they cannot express complex constraints). Also, we have noticed from the literature a lack of formal service contracts dedicated for SCs. In order to overcome these problems, we have proposed a model-based, top-down, formal and end-to-end approach based on the Software Product Line (SPL). We have particularly proposed two practical Feature Models (FMs), named FM_{SP} and FM_{SC}, that integrate, based on SOA design patterns, features and constraints (e.g., functional requirements) of SP and SC, respectively. In order to be able to model the variability of these features, we have extended the feature metamodel. We have defined semantic constraints that should be respected when developing FM_{SP} and FM_{SC} to ensure their consistency. We have designed these FMs in order to be able to generate fully functional, valid, highly customized and consistent SPs and SCs. We have developed a tool that implements our approach, including the proposed two FMs, and we have demonstrated its efficiency and usefulness through a practical case study.

For future research, we plan to extend the FM_{SP} and FM_{SC} by other features in order to reflect more SOA features and constraints. Another work would be to extend our approach to be able to generate SPs and SCs with different programming languages (e.g., Java EE and .Net).

References

[1] Alshaimaa Mustafa, Hany F. ElYamany, Mahmoud Elarabawy, Nashwa M. Yhiea, and Hossam M. Faheem. An intelligent-ranking framework for web services selection process. *International Journal of Services Technology and Management*, 20(1/2/3):85–107, 2014.

[2] Xiaolin Zheng, Zhen Lin, Jianyue Wang, and Yijun Bei. Rule-based service charging method for composite services. *International Journal of Services Technology and Management*, 16(3/4):337–355, 2011.

[3] Thomas Erl. *SOA Design Patterns*. Prentice Hall, 2009.

[4] Thomas Erl. *SOA Principles of Service Design*. Prentice Hall, 2007.

[5] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. *Software Product Line Engineering*. Springer, 2005.

[6] Krzysztof Czarnecki, Simon Helsen, and Eisenecker Ulrich. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[7] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. Slicing feature models. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'2011)*, pages 424–427, Lawrence, Kansas, USA, November 2011.

[8] Akram Kamoun, Mohamed Hadj Kacem, and Ahmed Hadj Kacem. Multiple software product lines for software oriented architecture. In *Proceedings of the 25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'2016)*, pages 56–61, Paris, France, June 2016.

[9] Gerald Holl, Paul Grünbacher, and Rick Rabiser. A systematic review and an expert survey on capabilities supporting multi product lines. *Information and Software Technology*, 54(8):828–852, 2012.

[10] Mathieu Acher, Guillaume Bécan, Benoit Combemale, Benoit Baudry, and Jean-Marc Jézéquel. Product lines can jeopardize their trade secrets. In *Proceedings of the 10th International Symposium on Foundations of Software Engineering (FSE'2015)*, pages 930–933, New York, USA, August 2015.

[11] Andreas Metzger and Klaus Pohl. Software product line engineering and variability management: achievements and challenges. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'2014)*, pages 70–84, Hyderabad, India, May 2014.

[12] Matthias Galster, Paris Avgeriou, and Dan Tofan. Constraints for the design of variability-intensive service-oriented reference architectures – an industrial case study. *Information and Software Technology*, 55(2):428–441, 2013.

[13] Piero Giacomelli. *HornetQ Messaging Developer's Guide*. Packt Publishing, 2012.

[14] MSPL4SOA tool. https://mspl4soa.github.io, 2017.

[15] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. FAMILIAR: a domain-specific language for large scale management of feature models. *Science of Computer Programming*, 78(6):657–681, 2013.

[16] Switchyard tool. http://switchyard.jboss.org.

[17] Bardia Mohabbati, Mohsen Asadi, Dragan Gašević, Marek Hatala, and Hausi A. Müller. Combining service-orientation and software product line engineering: a systematic mapping study. *Information and Software Technology*, 55(11):1845–1859, 2013.

[18] Alcemir Rodrigues Santos, Raphael Pereira de Oliveira, and Eduardo Santana de Almeida. Strategies for consistency checking on software product lines: a mapping study. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering (EASE'2015)*, pages 5:1–5:14, Nanjing, China, April 2015.

[19] Hamza Ed-douibi, Javier Luis Cánovas Izquierdo, Abel Gómez, Massimo Tisi, and Jordi Cabot. EMF-REST: generation of RESTful APIs from models. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC'2016)*, pages 1446–1453, Pisa, Italy, 2016.

[20] Carlos Parra and Diego Joya. SPLIT: an automated approach for enterprise product line adoption through SOA. *Internet Services and Information Security*, 5(1):29–52, 2015.

[21] Akram Kamoun, Mohamed Hadj Kacem, and Ahmed Hadj Kacem. Feature model for modeling compound SOA design patterns. In *Proceedings of the 11th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'2014)*, pages 381–388, Doha, Qatar, November 2014.

[22] Peter Kajsa and Pavol Návrat. Design pattern support based on the source code annotations and feature models. In *Proceedings of the 38th International Conference on Current Trends in Theory and Practice of Computer Science on SOFtware SEMinar (SOFSEM'2012)*, pages 467–478, Špindlerův Mlýn, Czech Republic, January 2012.

[23] Mohammad Abu Matar and Hassan Gomaa. Feature-based variability meta-modeling for service-oriented product lines. In *Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems (MoDELS'2011)*, pages 68–82, Wellington, New Zealand, May 2011.

[24] Marcelo Fantinato, De Toledo Maria Beatriz Felgar, and De Souza Gimenes Itana Maria. WS-contract establishment with QOS: an approach based on feature modeling. *Cooperative Information Systems*, 17(03):373–407, 2008.

[25] Hiroshi Wada, Junichi Suzuki, and Katsuya Oba. A feature modeling support for non-functional constraints in service oriented architecture. In *Proceedings of the 4th IEEE International Conference on Services Computing (SCC'2007)*, pages 187–195, Salt Lake City, Utah, USA, July 2007.