Feature Model as a Design-Pattern-based Service Contract for the Service Provider in the Service Oriented Architecture

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

Feature Model as a Design-Pattern-based Service Contract for the Service Provider in the 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

In Service Oriented Architecture (SOA), many feature modeling approaches of Service Provider (SP) have been proposed, notably: the two widely used service contracts WSDL and WADL. By studying these approaches, we found that they suffer from several problems, notably: they only work for specific communication technologies (e.g., SOAP or REST) and they do not explicitly model SOA Design Pattern (DPs) and their compounds. One major benefit of using a DP or a compound DP is to develop SPs with proven design solutions. In this paper, in order to overcome these problems, we propose an approach that integrates Software Product Line (SPL) techniques in the development of SPs. Essentially, we propose a Feature Model (FM), which is the defacto standard for variability modeling in SPL, for the feature modeling of SP. This FM, named FM_{SP}, is designed as a DP-based service contract for SP that models different features including 16 SOA DPs and their compounds that are related to the service messaging category. Its objective to enable developers to generate fully functional, valid, DP-based and highly customized SPs for different communication technologies. Through a practical case study and a developed tool, we validate our FM_{SP} and demonstrate that it reduces the development costs (effort and time) of SPs.

Index terms— Service oriented architecture, Service provider, Service contract, Design pattern, Feature model, Software product line

1 Introduction

Service Oriented Architecture (SOA) is an architectural model that represents a distributed computing platform by considering services as the essential means through which a solution logic is implemented [1]. A service consists of a set of capabilities (i.e., operations) that are implemented in a Service Provider (SP) and can be invoked by different Service Consumers (SCs). One of the main

objectives of SOA is to promote loose coupling, reusability and interoperability of SCs and SPs. The latter can be customized to implement different features. In this paper, we focus on modeling the SP features notably the ones of services, capabilities, SOA Design Patterns (DPs) [2], and the three communication technologies Simple Object Access Protocol (SOAP), REpresentational State Transfer (REST) and Middleware Oriented Messaging (MOM) [3], [4].

DPs are appropriate and proven design solutions that have been introduced by veteran problem solvers for specific problems in certain contexts. In the practice, it is frequent to implement a compound DP that represents a composition of a set of DPs that are applied together in order to solve a complex problem [2].

In the literature, many SP feature modeling approaches have been proposed [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16] notably the two widely used service contracts: Web Services Description Language (WSDL) [5] and Web Application Description Language (WADL) [6]. These service contracts are represented with XML documents, with different notations, through which the features of the communication technologies SOAP and REST are modeled, respectively. It is important to note that the service contract represents a core part and one of the fundamental design principles in SOA [1]. Erl [1] said: "*the service contract represents a core part of a service's architecture and is a focal point during the service design process to the extent that a principle is dedicated to its customization*". By studying these SP feature modeling approaches, we identify that they suffer from several problems, as follows:

P.1 explicitly modeling the SOA DPs and compound DPs is not considered. This makes difficult the development of DP-based and complex SPs with proven design solutions. It should be noted that developing valid SOA DPs and compound DPs is not a straightforward and easy task and requires a solid core of expert knowledge [17]. Schmidt *et al.* [17] said: "*combining several patterns into a heterogeneous structure is complicated*";

P.2 only a limited set of features has been modeled. This prevents the development of complex SPs;

P.3 there is a lack of solutions to generate fully functional SPs. The main reason is that, as reported by Parra and Joya [8] and Fantinato *et al.* [9], the features (e.g., input and output data) of capabilities and services are not modeled;

P.4 there is a lack of solutions that model SP features independently of communication technologies. This is important to be able to generate SPs that support different communication technologies. For example, the features modeled in the service contracts WSDL and WADL are dependent on the SOAP and REST communication technologies, respectively;

P.5 some communication technologies do not offer service contracts, notably the MOM. In this case, it would be not possible to benefit from the advantages of using service contracts;

P.6 developing many separated service contracts can be needed to develop a SP. For example, if a given service in the SP supports the SOAP and REST communication technologies, then this service should be implemented and

accompanied with the two different service contracts WSDL and WADL, respectively. This can decrease the governance of the SP and can make difficult for SP developers to implement SP features. Also, it can make difficult for SC developers to discover the features that are offered by the SP (e.g., to discover the supported communication technologies of SP);

P.7 each SP feature modeling approach (e.g., WSDL and WADL) uses its own notation even to model the same features (e.g., input and output data features). This can lead to misinterpretation and difficulty to understand SP features, and reduces the efficiency of the reusability design principle [1] in the SP feature modeling.

In order to overcome these problems, we introduce in this paper, an approach that uses Software Product Line (SPL) [18] techniques for the feature modeling and the mass-customization of SPs in SOA. Essentially, we propose a Feature Model (FM) [19], named FM_{SP}, that is designed as a DP-based service contract for the SP feature modeling. This FM expresses 72 features including 16 SOA DPs that are related to the service messaging category (see chapter "Service Messaging Patterns" in [2]). This category provides various techniques for processing and coordinating data exchanges between services. One of the main challenges tackled in this work is to design this FM_{SP} in a way that it ensures deriving valid compounds of these 16 SOA DPs and valid SPs accordingly. The objective of the proposed FM_{SP} is to enable developers to generate fully functional, valid, DP-based and highly customized SPs for different communication technologies (SOAP, REST and MOM).

The contribution proposed in this paper extends our earlier work [20], which has been published as a conference paper in *ICEIS'2017*. Principally, we extend the earlier proposed FM_{SP} with other 26 features, we introduce some revision and enhancements to it to be able to derive more complex SPs, and we discuss it in more details. The objectives of these newly added features are twofolds. First, they allow to generate more complex SPs. In particular, we add modeling the `Messaging metadata DP` [2] which is an essential DP for the other modeled DPs. Second, they take into consideration all the required information that must be discovered by SC developers in order to develop SCs which can communicate correctly with all the possible SPs that can be derived from FM_{SP}.

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 including our FM_{SP}. In Sect. 4, we evaluate our approach through a practical case study. In Sect. 5, we discuss some related works. This paper is concluded in Sect. 6.

2 A brief overview of the feature model

One increasing trend in application development is the need to develop multiple and customized applications instead of just a single individual application. The main reason is that, because of the cost and time constraints, it is not possible for developers to realize a new application from scratch for each new project, and so software reuse must be increased. The Software Product Line (SPL) [18] offers software reusing solutions to these not quite new, but increasingly challenging, problems to enable the mass-customization of applications. It relies on the variability modeling of the application artifacts (e.g., source code and design)

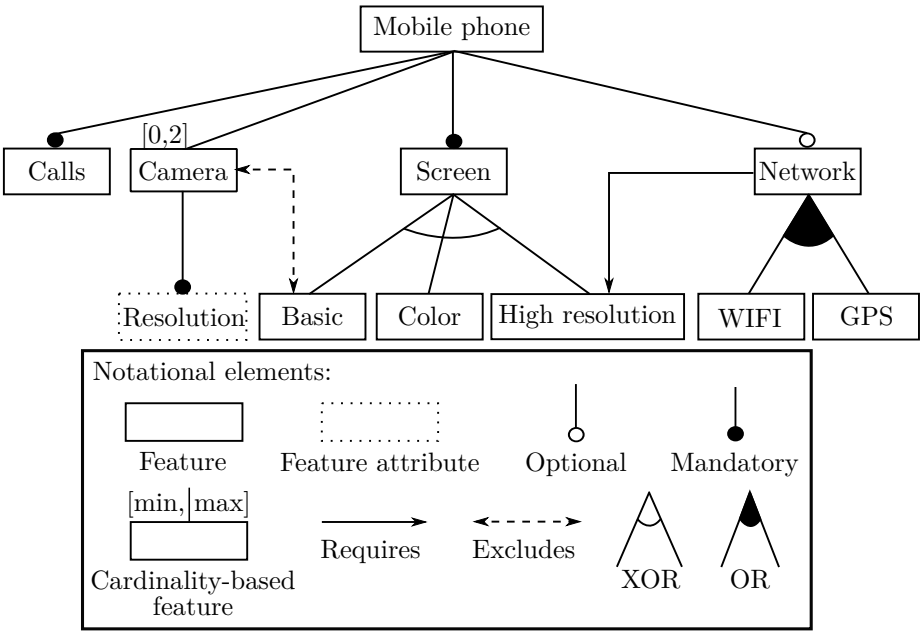


Figure 1: Example of a feature model for mobile phone

to be able to generate customized applications. The variability consists in the ability of an artifact to be customized or configured in a particular context. The Feature Model (FM) [19] is the defacto standard for variability modeling in SPL. Its objective is to model the legal combination of the SPL features to generate customized applications.

In Fig. 1, we present an example of a FM for mobile phone. The structure of the FM is a rooted tree of features that can be defined through different notational elements. Many FM metamodels [21], [19] have been proposed in the literature that offer different notational elements. We rely on the FM metamodel of Czarnecki *et al.* [19] because its notational elements fit well and are necessary in our work. In Fig. 3, we present these notational elements that will be briefly presented in the following. The *feature* can be either *mandatory* or *optional*. The *feature attribute* allows to add an attribute value to specify extra-functional information for features. The *cardinality-based feature* $[min, max]$ defines the lower and upper bounds of instances of a given feature. In our example, a mobile phone can have from 0 to 2 instances of the feature `Camera`. Each instance has the mandatory feature attribute `Resolution` that must be valued by the developer in order to set the camera resolution. The feature constraints *"requires"* and *"excludes"* permit to define inclusion and exclusion constraints between features. The *feature group XOR* $[1, 1]$ allows selecting exactly one out of its child features which can be called as alternative exclusive features. The *feature group OR* $[1, n]$ allows selecting one or many of its child features which can be called as alternative inclusive features.

In order to derive and generate a customized application from a FM, a developer needs to derive a model that is a specialization of this FM. Specialization is a refinement process that allows the elimination of some variability information

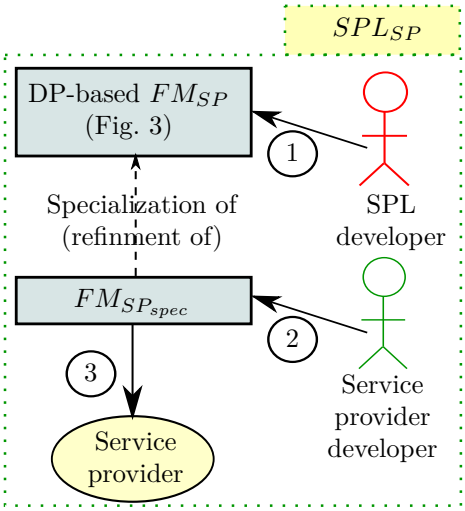


Figure 2: Approach overview

from a FM. Czarnecki *et al.* [19] introduce several specialization ways, such as: refining a cardinality-based feature $[min, max]$, selecting a required feature, deselecting an unwanted feature and assigning a feature attribute value. If all the variability of the derived model has been resolved by the developer (i.e., all of its features are mandatory), then this model is called Application Model (AM).

3 Contribution

In this section, we present our contribution. First, we present an overview of our approach. Second, we provide its benefits. Third, we lead a rigorous discussion of the features and constraints that have been modeled in the proposed FM_{SP}.

3.1 Approach overview

In order to overcome the problems that have been enumerated in the introduction **P.1**, **P.2**, **P.3**, **P.4**, **P.5**, **P.6** and **P.7**, we propose an approach that consists in developing an SPL, named SPL_{SP}, for the feature modeling and the mass-customization of SPs. In Fig. 2, we introduce an overview of our approach including its implementation steps that will be presented in the following.

In the **first step**, the SPL developer realizes a FM, named FM_{SP}, for the SP feature modeling. In this FM, we propose that anything that is required to develop SPs is modeled as a feature. For example, services, capabilities, communication technologies and Design Patterns (DPs) are modeled as features. In Fig. 3, we present the proposed FM_{SP} which will be discussed in detail in Sect. 3.3. This FM is design to be a DP-based service contract for SP. It models 72 SP features including 16 SOA DPs [2] and the three communication technologies SOAP, REST and MOM. In Table 1, the descriptions of all of these features are presented and the DPs are illustrated with a color. We recall that the 16 studied DPs are related to the service messaging category (see chapter

"Service Messaging Patterns" in [2]). The objective of the proposed FM_{SP} is to enable developers to generate fully functional, valid, DP-based and highly customized SPs. We recall that this FM_{SP} extends our earlier work [20] as mentioned in the introduction. Also, some contents of Table 1 have been reused from this same earlier work.

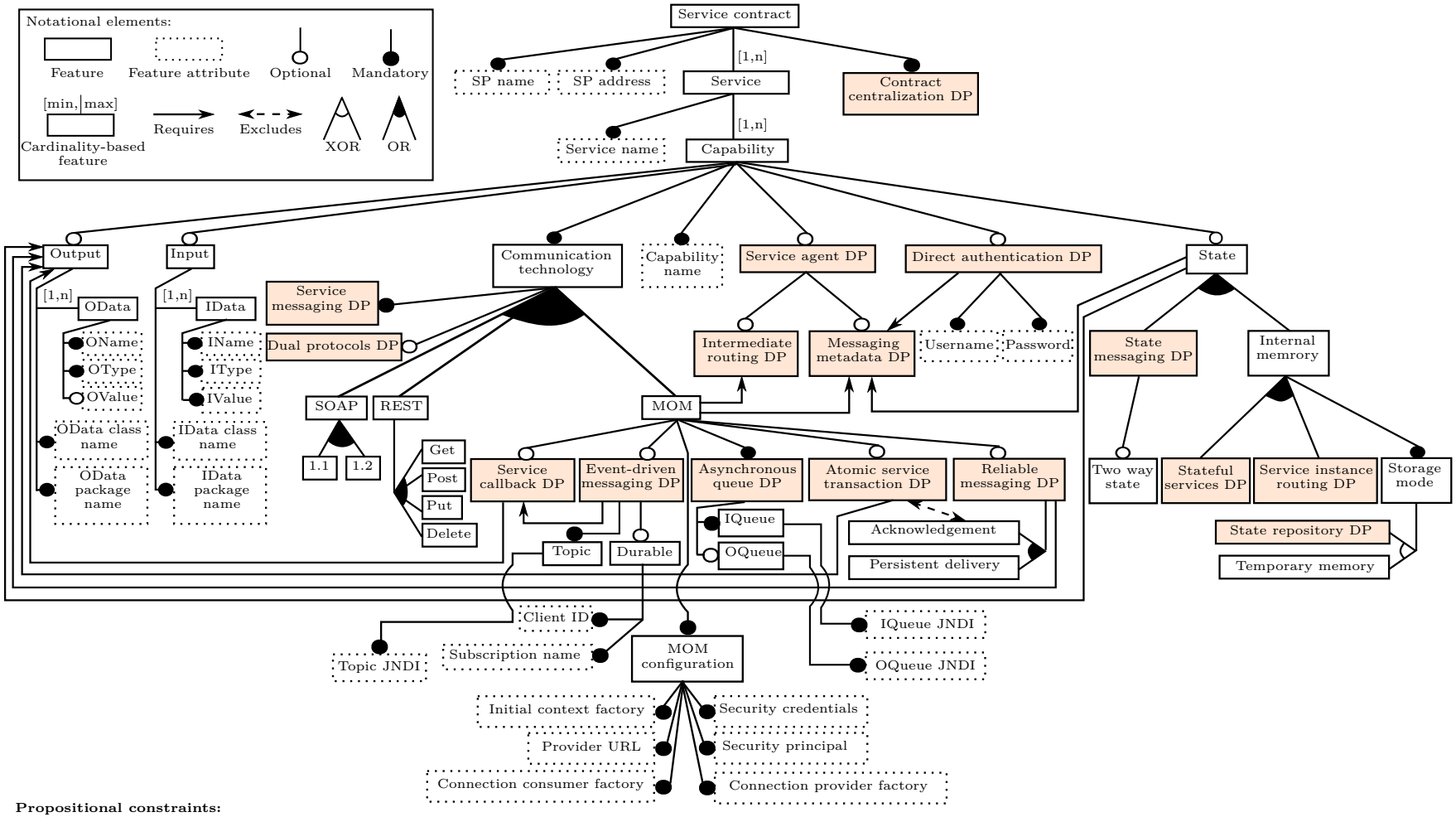
Developing SOA DPs and valid compound DPs is not a straightforward and easy task and requires a solid core of expert knowledge [17]. Mathematically, there exist 2^{16} compound DPs that represent the existence or the non-existence of the 16 studied DPs. However, not all of these compound DPs are valid. For example, in order to use the `Event-driven messaging DP`, it is necessary to use in conjunction the `Service callback DP` (see Table 1). Hence, if a given compound DP includes the `Event-driven messaging DP` but omits the `Service callback DP`, then this compound DP is invalid. One of the main challenges tackled in this work is to identify and model the valid compound DPs in our FM_{SP}. In this context, it is crucial to identify and model the constraints between DPs in FM_{SP} (see Sect. 3.3). We note that Erl [2] presents several relationships between these DPs. However, he illustrates them in many dispersed and not standardized diagrams. This makes difficult to properly identify their constraints.

In the **second step**, in order to derive a SP, a developer needs to specialize (i.e., refine) a FM, named $FM_{SP_{spec}}$, from FM_{SP} (essentially by selecting and deselecting features). The $FM_{SP_{spec}}$ has two objectives. The *first objective* is allowing to define the features that the SP developer wants to implement in his/her SP. The *second objective* is permitting SC developers to discover the optional features offered by the SP. For example, let us suppose that the feature `Direct authentication DP` (see Table 1) is modeled as optional in $FM_{SP_{spec}}$ and in the SP accordingly. In this context, the SC developers can discover from $FM_{SP_{spec}}$, that they have the choice to provide or not credentials authentication, to use this feature, when developing their SCs to be able to communicate with this SP.

In our previous work [20], in order to derive a SP, we have proposed to instantiate an AM from FM_{SP}. This allows certainly to consider the *first objective*. However, the *second objective* cannot be respected because an AM can only contain mandatory features (i.e., cannot include optional features). This is why we propose, in this paper, to refine a FM ($FM_{SP_{spec}}$) from FM_{SP} in order to respect these two objectives.

In the **third step**, an automatic model to code transformation operation will transform $FM_{SP_{spec}}$ to the required SP. Afterwards, the SP developer can manually adapt the generated SP to implement his/her application requirements (e.g., defining the business logic of the SP).

In our approach, we propose, as a *principle*, that the $FM_{SP_{spec}}$ of the SP can be downloaded by the SC developers. The goal is to allow these developers to discover the supported features of this SP in order to implement SCs that can communicate correctly with it. In this context, our FM_{SP} is designed to include all the required features that must be discovered by SC developers so they can realize SCs which can communicate correctly with all the possible SPs that can be derived from this FM. To implement this *principle*, the FM_{SP} that has been proposed in [20] has been extended with other features which will be discussed in Sect. 3.3. We note that this *principle* is widely used by the service contracts WSDL [5] and WADL [6].



Propositional constraints:
1: (MOM ∧ Output) → (Atomic service transaction DP ∨ Acknowledgement)
2: OQueue ↔ (Service callback DP ∧¬ Event-driven messaging DP)
3: Dual protocols DP ↔ ((MOM ∧ REST) ∨ (MOM ∧ SOAP) ∨ (REST ∧ SOAP) ∨ (MOM ∧ REST ∧ SOAP))

Figure 3: Feature model for the service provider FM_{SP} (design pattern features are colored)

3.2 Benefits

Our FM_{SP} is designed to overcome the problems that have been enumerated in the introduction: **P.1**, **P.2**, **P.3**, **P.4**, **P.5**, **P.6** and **P.7**. We present in the following the benefits of our FM_{SP} and which problems they consider:

1. it relies on the FM notation which permits to efficiently model the features and complex constraints of the SP. Also, its graphical presentation can be easily interpreted to identify the features and constraints of SP. By using the FAMILIAR tool [22], we calculate that our FM_{SP} permits to derive a capability in the SP with 372904 different possible configurations. In order to derive a SP including its capabilities, a developer only needs to have basic knowledges about the features of FM_{SP} and to select the required ones in line with the constraints of this FM. This facilitates the mass-customization of SPs (considering Problems **P.2**, **P.3**, **P.4**, **P.5**, **P.6** and **P.7**);

2. it includes the required features and constraints to generate fully functional, valid, DP-based and highly customized SPs. This reduces the development costs (effort and time) of SPs (considering Problems **P.2**, **P.3**, **P.4**, **P.5** and **P.7**);

3. it is designed as a DP-based service contract for SP which is generic and independent of the communication technologies. It can be considered as a reference model [23] which reflects the variability of practical SP features (considering Problems **P.1**, **P.2**, **P.3**, **P.4**, **P.5**, **P.6** and **P.7**);

4. it models 16 DPs and their corresponding constraints. This permits to easily identify and derive only valid compound DPs. By using the FAMILIAR tool [22], we have calculated that FM_{SP} permits to derive 790 valid compound DPs from 2^{16} possible ones (considering Problem **P.1**);

5. many benefits can be enumerated when expressing DPs as features in our FM_{SP}. DPs are introduced by veteran problem solvers in order to provide appropriate and proven design solutions. A DP shows the right level of abstraction to describe a certain solution in a generic context, i.e., independently of the programming languages and platforms. It also has a major benefit of providing a common language which it is understandable by the developers instead of using terminologies related to a certain context. For example, simply saying the `Event-driven messaging DP` [2] is more efficient and easier than to explain it in details. Hence, integrating DPs in our FM_{SP} allows to ensure that the SPs that can be derived are based on proven solutions (considering Problems **P.1**, **P.2**, **P.4**, **P.5**, **P.6** and **P.7**).

3.3 Feature model for the service provider FM_{SP}

In Fig. 3 and Table 1, we present our FM_{SP} and descriptions of its features. In this table, we highlight the feature attributes which represent the values that need to be given by the developer to derive a SP. Also, we propose to classify the features of FM_{SP} into several categories that are presented in the first column of this table. This help to better understand and interpret the proposed FM_{SP}.

In the following subsections, we discuss in detail one-by-one the features and constraints of each category, notably the ones of the modeled 16 DPs. It is important to note that these features and constraints have been identified from theoretical and practical conducted studies that are essentially based on these works [2], [1], [24], [25], [4], [26].

Table 1: Descriptions of the features of FM_{SP} (design pattern features are colored)

Category	Feature name	Feature attribute	Description
SP information	Service contract	-	Root feature
	Contract centralization DP	-	Gathering the SP features within the service contract so it will be used by the SC as the sole entry point to communicate with the SP
	SP name	+	Name of the SP
	SP address	+	Address of the SP
Service and capability	Service $[1,n]$	-	Services of the SP
	Service name	+	Name of a given service
	Capability $[1,n]$	-	Capabilities of a given service
	Capability name	+	Name of a given capability
	Input	-	Input data of a given capability
	IData $[1,n]$	-	Gathering the input data features
	IName	+	Name of a given input data
	IType	+	Type of a given input data (e.g., String)
	IValue	+	An input value that will be given by the SC to invoke a given capability
	IData class name	+	Name of the class that encapsulates the input data names and types
	IData package name	+	Name of the package that includes the input data classes
	Output	-	Output data of a given capability
	OData $[1,n]$	-	Gathering the output data features
	OName	+	Name of a given output data
	OType	+	Type of a given output data
	OValue	+	Output value of a given capability that will be returned to the SC

	OData class name	+	Name of the class that encapsulates the output data names and types
	OData package name	+	Name of the package that includes the output data classes
	Service messaging DP	-	Using a messaging-based communication between the SC and SP to remove the need of persistent connections (e.g., remote procedure call binary connections) and to reduce coupling requirements
	Dual protocols DP	-	Configuring a given capability to support two or more communication technologies. This allows to accommodate different application requirements
	Communication technology	-	Gathering communication technologies
	REST	-	REST communication technology
	Get	-	HTTP get method for REST
	Post	-	HTTP post method for REST
	Put	-	HTTP put method for REST
	Delete	-	HTTP delete method for REST
	SOAP	-	SOAP communication technology
	1.1	-	SOAP version 1.1
	1.2	-	SOAP version 1.2
	MOM	-	Middleware Oriented Messaging (MOM) communication technology
	MOM configuration	-	Gathering the MOM configuration information
Communication	Initial context factory	+	Initial context that is required to access to the MOM via a Java Naming and Directory Interface (JNDI)
	Provider URL	+	MOM address that is required to access to this MOM
	Connection provider factory	+	Object name that encapsulates a set of MOM connection configuration parameters that will be used by the SP to access to this MOM
	Connection consumer factory	+	Object name that encapsulates a set of MOM connection configuration parameters that will be used by the SC to access to this MOM
	Security credentials	+	Username that is required to access to the MOM
	Security principal	+	Password that is required to access to the MOM

	Asynchronous queue DP	-	Deploying an intermediary MOM allowing the SP and SC to asynchronously communicate and to independently process messages by remaining temporally decoupled
	IQueue	-	Input queue that implements the `Asynchronous queue DP` to handle the asynchronous incoming SC messages
	OQueue	-	Output queue that implements the `Asynchronous queue DP` to handle the asynchronous outgoing SP messages
	IQueue JNDI	+	A JNDI that allows to discover, look up and communicate with `IQueue`
	OQueue JNDI	+	A JNDI that allows to discover, look up and communicate with `OQueue`
	Event-driven messaging DP	-	Asynchronously sending the response messages of the publisher (i.e., SP), when ready, to its corresponding subscribers (i.e., SCs) through the MOM
	Service callback DP	-	Redirecting the SP response messages to a callback address that can be different of the requester SC address
	Topic	-	Topic that implements the `Event-driven messaging DP`
	Topic JNDI	+	A JNDI that allows to discover, look up and communicate with `Topic`
Security and reliability	Atomic service transaction DP	-	Treating a group of the SP response messages as a single work unit. The latter is wrapped in a transaction with a rollback feature that resets all actions and changes if the exchanging messages fails
	Reliable messaging DP	-	Adding a reliability mechanism to the SP response messages in order to ensure message delivery. This mechanism relies on acknowledging the SP messages and persisting them in a data store
	Persistent delivery	-	Persisting the SP messages in a data store so they are not lost if the MOM fails. Therefore, we ensure that the SP messages are delivered to the SC
	Acknowledgement	-	The MOM acknowledges the SP about its messages that have been received by the SC
	Durable	-	A durable MOM stores the messages of the publisher (i.e., SP) for the subscribers (i.e., SCs) if the latter disconnect. Hence, we ensure that, upon reconnecting, the subscribers will receive all these messages
	Client ID	+	An identifiant that must be given by the SC to use the feature `Durable` of MOM
	Subscription name	+	A subscription name that must be given by the SC to use the feature `Durable` of MOM

	Direct authentication DP	-	Requiring that the SCs must provide authentication credentials (username and password) to invoke a capability
	Username	+	A username that must be given by the SC for authentication
	Password	+	A password that must be given by the SC for authentication
Agent	Service agent DP	-	Deferring some logic (e.g., logging messages) from services to event-driven programs to reduce the size and performance strain of services
	Intermediate routing DP	-	Dynamically routing messages through a service agent that relies on an intermediary routing logic
	Messaging metadata DP	-	Supplementing the messages headers, through service agents, with metadata in order to share activity-specific logic between SC and SP.
State	State	-	Gathering techniques that handle the state data of capabilities
	Internal memory	-	Storing the state data in the SP internal memory
	Stateful services DP	-	Managing and storing state data by intentionally stateful utility services
	Service instance routing DP	-	Allowing a given SC to communicate with the same instance of a given service to retain its state
	Storage mode	-	Gathering the modes of how the state data are stored
	Temporary memory	-	Storing the state data in a temporary memory in the SP
	State repository DP	-	Deferring storing state data from a temporary memory to a state repository in the SP. The objective is to alleviate services from having to unnecessarily retain state data in memory for extended periods
	State messaging DP	-	Delegating the storage of state data to the SP response messages instead to the SP internal memory. The objective is the same as the `State repository DP`
	Two way state	-	Configuring both the SP and SC to delegate the storage of state data in their outgoing messages

3.3.1 Service provider information features

The feature `Service contract` represents the root of FM_{SP}. The feature attributes `SP name` and `SP address` need to be valued by the SP developer to specify the name and address of his/her SP. The values of these feature attributes will allow the SC developers to realize SCs that can communicate correctly with this SP.

We express the feature `Contract centralization DP` as mandatory for two

reasons. First, the FM_{SP} is designed as a centralized service contract that models the SP features. Second, this FM will be used by SCs as the sole entry point to discover the features supported by the SP in order to correctly communicate with it. Implementing this DP permits to avoid developing different and separated service contracts to model the SP features which can be problematic as shown in Problems **P.6** and **P.7** in the introduction.

3.3.2 Services and capabilities features

The cardinality-based feature `Service` $[1,n]$ allows the developer to implement 1 to n instances of services in his/her SP. Each instance has a name that is modeled by its child feature attribute `Service name`. It also has a cardinality-based feature `Capability` $[1,n]$. The latter allows the developer to implement 1 to n instances of capabilities in each service instance. The other features of FM_{SP} are modeled as child features to each capability instance. Hence, it would be possible to configure the variability of each capability instance differently and independently. In FM_{SP}, each capability can be configured with 372904 different possible configurations, in particular, with different 790 valid compounds of the 16 modeled DPs.

Each capability has a name that is modeled by the feature attribute `Capability name`. The optional features `Input` and `Output` and their children permit to model the information about the input and output data of each capability. If these features have been omitted by the developer when deriving a given capability in his/her SP, then this capability will not accept any input data from SCs and will not return any result. In this case, the signature of this capability will be: `void capabilityName()`.

The cardinality-based features `IData` $[1,n]$ and `OData` $[1,n]$ specify the count of the input and output data of each capability. These features contain in particular the feature attributes `IValue` and `OValue`, respectively. The sole objective of the mandatory feature attribute `IValue` is to inform the SCs that they must provide an input value in the body of their request messages to invoke a given capability. The optional feature attribute `OValue` gives the choice to the SP developer to specify a static result of capabilities. In fact, capabilities often have dynamic results that are defined in the business logic of capabilities. In this case, the feature attribute `OValue` can be omitted by the developer when deriving his/her SP. However, this feature attribute can be useful to test if a given capability, with a given configuration, can be invoked correctly by SCs.

To implement a SP, it is important to permit the SP capabilities to take and return objects (classes instances) as input and output data. In this context, the classes that encapsulate these data need to be implemented as serializable so their objects can be included in the SP and SC messages. We note that the use of serializable classes is widely supported by the programming languages, like Java and C#. In the other hand, to implement a SC that can communicate correctly with this SP, these classes must be also implemented in this SC by using the same names and packages that have been defined in the SP [27]. In this context, as a requirement, the class and package names that are defined in the SP must be discovered by the SC developers. This requirement has been taking into consideration by modeling the mandatory feature attributes `IData`

`class name`, `OData class name`, `IData package name` and `OData package name` in FM_{SP}. The values of these feature attributes must be defined by the SP developer to derive a valid SP and to allow SC developers to discover them.

3.3.3 Communication features

The SC needs to send a request message to the SP in order to invoke a capability. If this capability returns a response message, then the communication type is called two-way. Otherwise, it is called one-way. In FM_{SP}, selecting or omitting the feature `Output` when deriving a SP will induce using the two-way or one-way communication types, respectively.

In FM_{SP}, we model the feature `Service messaging DP` as mandatory because the three modeled communication technologies `SOAP`, `REST` and `MOM` are messaging-based, and the modeled DPs are related to the service messaging category (see chapter "Service Messaging Patterns" in [2]). Erl [2] reports that this DP is one of the most fundamental DPs because it permits to promote the loose-coupling and interoperability design principles [1] by reducing the coupling requirements between the SC and SP.

The feature `Dual protocols DP` requires that a given capability must support two or more communication technologies and vice-versa. The third propositional constraint defined in FM_{SP} implements this requirement. Our FM allows to implement a capability that can support the communication technology `SOAP` with versions `1.1` or `1.2` or both. Also, a capability can be implemented to support the communication technology `REST` with one or more HTTP methods. This allows to accommodate different application requirements for each capability.

The features `SOAP` and `REST` rely on a synchronous communication for the message exchanging between the SC and SP. The problem of the synchronous communication is that it forces processing overhead in SC and SP because they must wait and continue to consume resources (e.g., memory) until they finish the message exchanging [2]. To overcome this problem, the asynchronous communication is used as a solution which is implemented through the feature `MOM` in our work. In this context, because the DP features `Service callback DP`, `Asynchronous queue DP` and `Event-driven messaging DP` are dedicated for an asynchronous communication, we define them as child features of the feature `MOM`.

In the `MOM`, the SC messages are always carried on by an asynchronous queue that reflected by the feature `IQueue` which is an implementation of the feature `Asynchronous queue DP` [4]. This is the reason why we define these features `IQueue` and `Asynchronous queue DP` as mandatory in FM_{SP}. The feature `Asynchronous queue DP` can be configured with two different ways when deriving a SP. The first way consists in selecting the feature `Asynchronous queue DP` in conjunction with the feature `Service callback DP` and omitting the feature `Event-driven messaging DP`. In this case, the SC and SP messages will be handled by two different asynchronous queues that are modeled respectively by the features `IQueue` and `OQueue`. The second way consists in selecting the feature `Asynchronous queue DP` and omitting the features `Service callback DP` and `Event-driven messaging DP`. In this case, all SC and SP messages are han-

dled by the same asynchronous queue that is modeled with the feature `IQueue`. These two ways are taken into account in our work by modeling `OQueue` as an optional feature, by defining the second propositional constraint in FM_{SP} and by modeling a "requires" constraint from the feature `Service callback DP` to the feature `Output`.

In order to implement the feature `Event-driven messaging DP`, we rely on the asynchronous topic [4] which is reflected by the feature `Topic`. The latter is used to asynchronously redirect the response messages of the SP to its SC callback addresses. To permit this redirection, we define, in FM_{SP}, a "requires" constraint from the feature `Event-driven messaging DP` to the feature `Service callback DP`.

We model the child features attributes of the feature `MOM configuration`, and the feature attributes `IQueue JNDI`, `OQueue JNDI` and `Topic JNDI` as mandatory in FM_{SP} for two reasons. First, they must be valued to implement a `MOM` in the SP. Second, they represent all the required information that must be discovered by SCs so they can communicate with the `MOM` of the SP.

3.3.4 Security and reliability features

It is common to use the features `Atomic service transaction DP` and `Reliable messaging DP` in conjunction with the `MOM` to implement a reliable asynchronous communication [2]. Hence, we express these features as children of the feature `MOM`. Since the `MOM` communication technology ensures a loosely coupled and an asynchronous communication between the SC and SP, then it should inform the SC and SP if their outgoing messages have been successfully received. In this context, the `MOM` should use either the features `Acknowledgement` or `Atomic service transaction DP` [2], [4]. This requirement is considered in our work by defining these two features as mutually exclusive and by defining the first propositional constraint in FM_{SP}.

The features `Atomic service transaction DP` and `Reliable messaging DP` can be only applied for the SP response messages, i.e., for the two-way communication type (see Table 1). Thus, we define "requires" constraints from these features to the feature `Output`.

In FM_{SP}, the feature `MOM` can be configured to support the feature `Durable`. The latter has the `Client ID` and `Subscription name` as mandatory child feature attributes. Their sole objective is to inform the SCs that they must value them in their request messages to use the feature `Durable`.

The objective of feature attributes `Username` and `Password` is to inform the SCs that they must provide a username and a password, as metadata, in their request messages to invoke a given capability when using the `Direct authentication DP`. In this context, we define a "requires" constraint from the feature `Direct authentication DP` to the feature `Messaging metadata DP` in FM_{SP}.

3.3.5 Agent features

The DP features `Intermediate routing DP` and `Messaging metadata DP` are implemented through a service agent as shown in Table 1. Thus, we define them as optional child features of the feature `Service agent DP` in FM_{SP}.

In contrast of `SOAP` and `REST`, the SC request messages which are dedicated to `MOM` do not explicitly contain information about the capability and service that the SC wants to invoke. As a consequence, it would be not possible to invoke the required capability and service. As a solution, we propose, to implement the feature `Intermediate routing DP` which exploits the metadata (`Messaging metadata DP`) presented in the SC request messages to dynamically routing these messages to the required capability and service. To support this dynamic routing, we define, in our FM_{SP}, "requires" constraints from the feature `MOM` to the features `Intermediate routing DP` and `Messaging metadata DP`.

From our study on SOA DPs [2], we notice that the four features `Service callback DP`, `Event-driven messaging DP`, `Atomic service transaction DP` and `Reliable messaging DP` require the feature `Messaging metadata DP`. This requirement is already considered in our FM_{SP} because (1) we have already defined these four features as optional children to the feature `MOM` and we have already defined a "requires" constraint from the feature `MOM` to the feature `Messaging metadata DP`.

3.3.6 State features

Erl [2] reports that the service state DPs `State messaging DP`, `Service instance routing DP`, `Stateful services DP` and `State repository DP` can be implemented in conjunction in the SP. This requirement is implemented in our FM_{SP} by defining these DPs as alternative inclusive features.

In one hand, the feature `State messaging DP` works by delegating the storage of state data to the SP messages, as *metadata*. In the other hand, the features `Stateful services DP` and `Service instance routing DP` work by supplementing the SP messages with a specific identifier (session or service instance identifiers) for each SC, as *metadata*. These identifiers need to be incorporated in the SC messages so the SP can use them to manage correctly the state of their corresponding SCs. In this context, as a requirement, the three features `State messaging DP`, `Stateful services DP` and `Service instance routing DP` need to rely on the features `Messaging metadata DP` and `Output` to be able to supplement the SP messages with *metadata*. In order to implement this requirement, given that these three features are modeled as alternative inclusive children of the feature `State` in FM_{SP}, we define "requires" constraints from the feature `State` to the features `Messaging metadata DP` and `Output`.

4 Evaluation

In order to show the merits and evaluate our approach including our FM_{SP} (see Figs. 2 and 3) in practice, we propose to use the case study of the Integrated Air Defense (IAD) (see Fig. 4 [20]). The IAD is a command and control compound of geographically dispersed force elements already in peace time as well as in crisis. In Fig. 4, we illustrate 17 force elements which are grouped into three

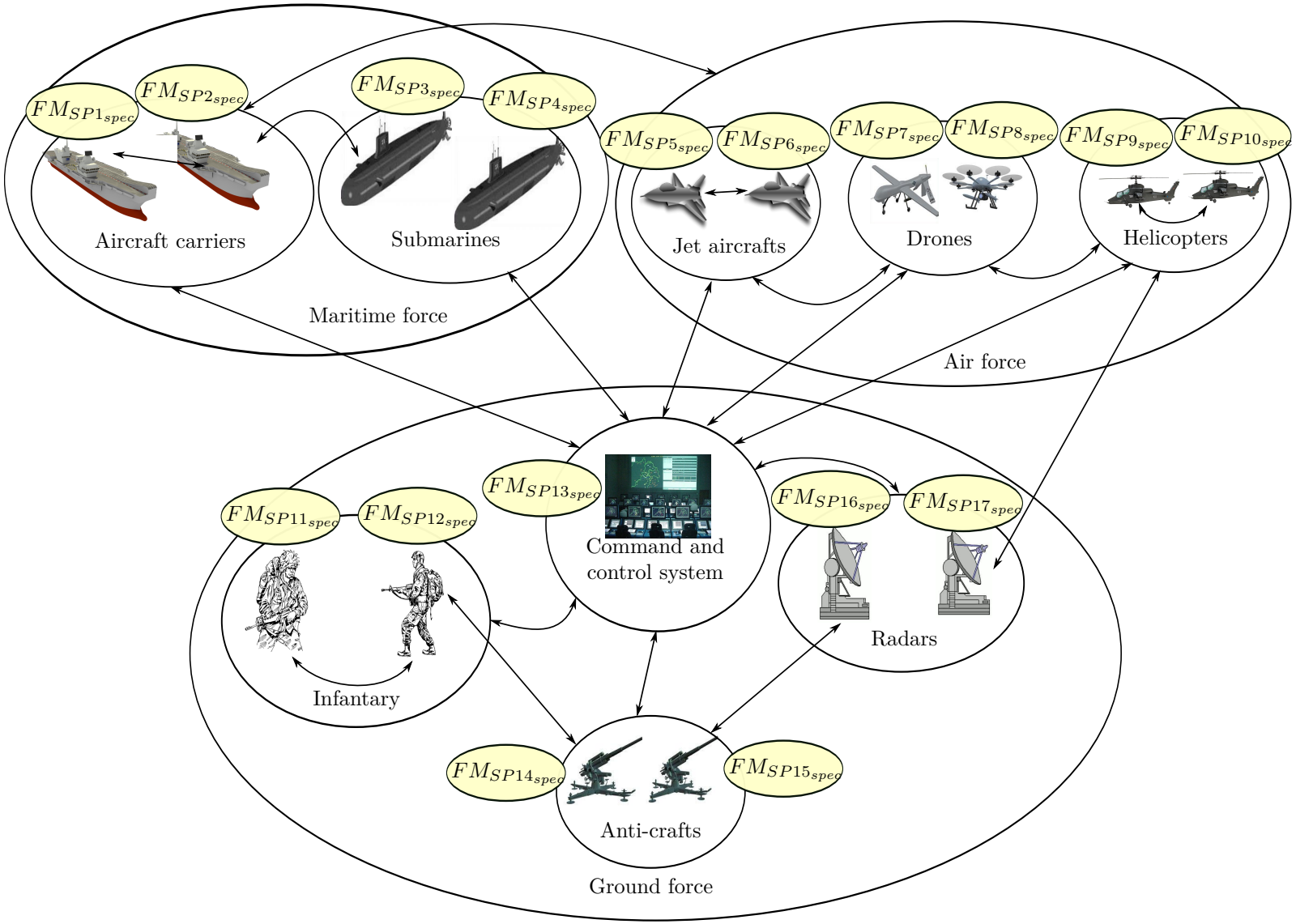


Figure 4: Case study of the integrated air defense

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 force elements communicate with services to achieve their missions. One main requirement must be satisfied to realize this IAD case study:

Requirement 1 *Each of the 17 force elements illustrated in the IAD case study is a SP that is responsible to implement its own features.*

As illustrated in the introduction (see Problems **P.1**, **P.2**, **P.3**, **P.4**, **P.5**, **P.6** and **P.7**), the SP feature modeling approaches that have been proposed in the literature [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16] notably the SOA traditional service contracts WSDL [5] and WADL [6] suffer from several problems to develop SPs. This prevents to efficiently realize **IAD Requirement**. In order to overcome these problems and to efficiently realize **IAD Requirement**, we propose deriving for each SP of the 17 IAD force elements a specific $FM_{SP_{spec}}$ (e.g., see Fig. 5) from our FM_{SP} (see Fig. 3). For example, in Fig. 4, the $FM_{SP13_{spec}}$ includes the SP features of the command and control system force element.

Using our FM_{SP} has several benefits as mentioned in Sect. 3.2 notably it facilitates the mass-customization of SPs. This is important especially when we have numerous SPs to develop which it is the case of our IAD case study (17 SPs to develop). Pohl *et al.* [18] show, from empirical investigations, that developing a SPL allows to reduce the development costs of systems if there are more than three or four systems to develop which it is our case (17 systems).

In fact, Erl [2] reports that the U.S. Department of Defense (DoD) has decided to plan and manage its business IT (Information Technology) via an architectural approach based upon SOA. The IAD system presented in Fig. 4 is a part of the DoD's business IT. He also reports that due to the scale, complexity and diversity of the DoD's business IT, the DoD developed a strategy with guiding principles which relies on the SOA DPs [2]. In this context, because our FM_{SP} relies on the SOA DPs, it can help to contribute to develop this DoD's business IT.

In Fig. 5, we present an example of a derived $FM_{SP_{spec}}$ from FM_{SP} for the command and system IAD force element (see Fig. 4). It is possible that this $FM_{SP_{spec}}$ contains different services and capabilities (see Fig. 3). For the sake of simplicity, we derive this $FM_{SP_{spec}}$ to contain a single `Service` which it is composed of a single `Capability`. The latter has a single `Input` data and a single `Output` data. Overall, it contains 66 features, notably:

- features that forms a valid compound of the modeled 16 DPs;
- features that define the information about the SP and its `Service` and `Capability`;
- features that define the variability of the communication technologies that can be used by SCs to invoke the `Capability`. We note that the `SOAP` version `1.2`, `REST` with the HTTP methods `Get` and `Post`, and `MOM` have been selected;
- optional features like the `Direct authentication DP`, `Service callback DP` and `Event-driven messaging DP` that can be used or not by SCs to invoke the `Capability`.

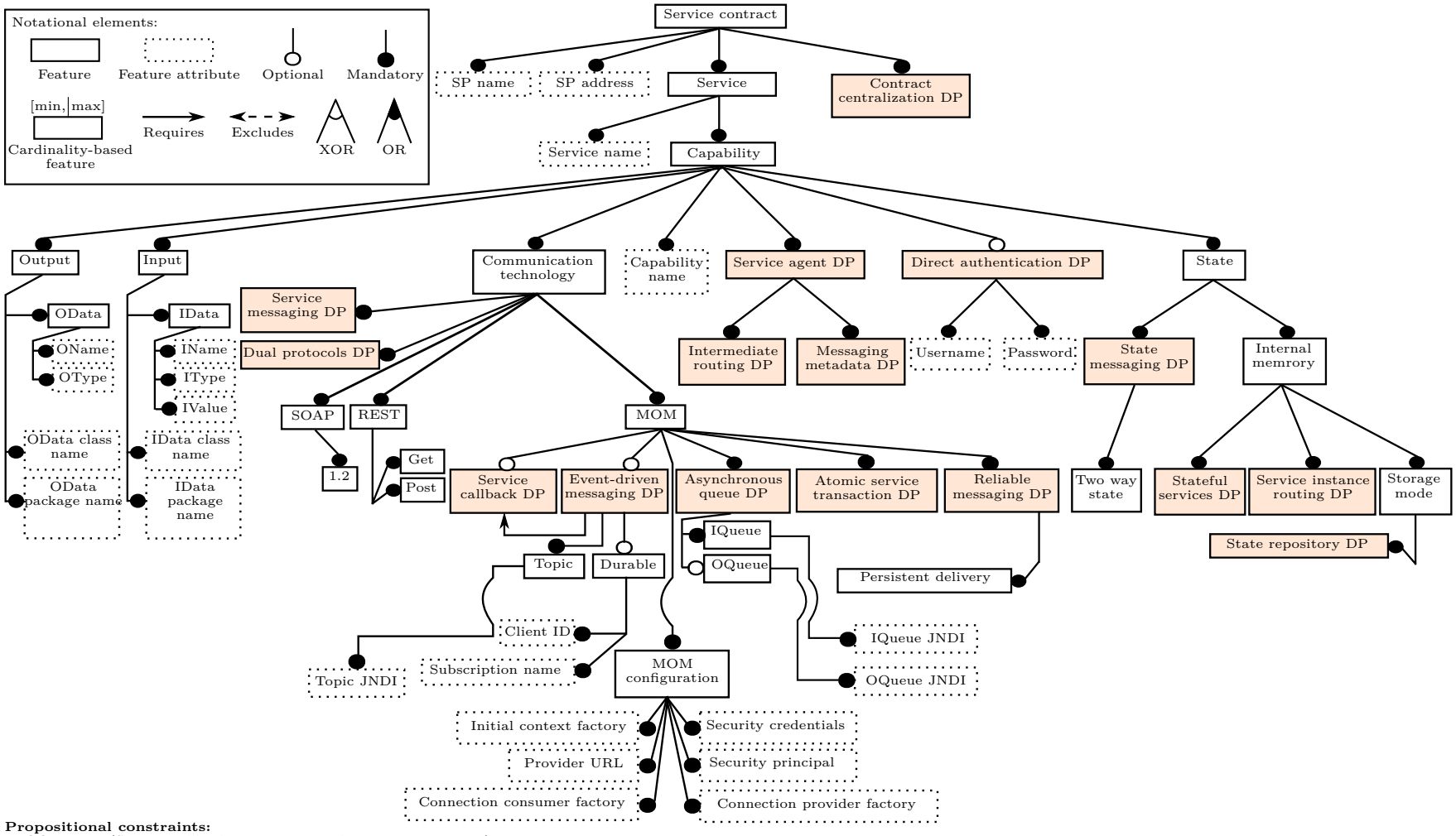


Figure 5: Example of a derived feature model of a service provider $FM_{SP_{spec}}$ from FM_{SP} (design pattern features are colored)

The values of the feature attributes defined in $FM_{SP_{spec}}$ are given in Table 2. Some feature attributes allow to define information about the SP, its capability and service, and the MOM configuration, like `SP address`, `Capability name` and `Provider URL`. Some others are dedicated to be valued by SCs in order to invoke the SP capabilities like `IValue`.

Table 2: The values of the feature attributes of $FM_{SP_{spec}}$ that is illustrated in Fig. 5

	Feature attribute	Value
SP information, service and capability	SP name	SP_command
	SP address	http://localhost:8080/SP_command
	Service name	Personal
	Capability name	login
	IName	id
	IType	String
	IData class name	Session
	IData package name	SP_command.input
	OName	isLogged
	OType	Boolean
	OData class name	SessionResponse
	OData package name	SP_command.output
MOM configuration	Initial context factory	org.jboss.naming.remote.client.InitialContextFactory
	Provider URL	remote://localhost:4447
	Connection provider factory	ConnectionFactory
	Connection consumer factory	RemoteConnectionFactory
	Security principal	guest
	Security credentials	guest_PassWorD
	IQueue JNDI	SP_command_in_queue_Personal_login
	OQueue JNDI	SP_command_out_queue_Personal_login
	Topic JNDI	SP_command_topic_Personal_login
SC	IValue	It needs to be valued in the SC
	Username	It needs to be valued in the SC
	Password	It needs to be valued in the SC
	Client ID	It needs to be valued in the SC
	Subscription name	It needs to be valued in the SC

We developed a tool [28] that relies on the Apache Velocity[1] tool (a model-to-code template engine) in order to transform a given $FM_{SP_{spec}}$ to the artifacts of the corresponding SP. Our tool relies on Java EE technologies. It generates SPs based on the Enterprise Service Bus (ESB) Switchyard [26]. The latter is a recent free software ESB that relies on the service component architecture [29] which is a technology-neutral assembly capability allowing the composition of services in SPs. It includes different technologies, notably the HornetQ [4] to implement the feature `MOM`, Apache CXF [3] to implement the features `SOAP` and `REST`, and Apache Camel [24] to implement DPs. These technologies are integrated on demand in the generated SP depending on the features of $FM_{SP_{spec}}$. Our tool also relies on the SPL tool FAMILIAR [22] to develop and manage the FM_{SP} and $FM_{SP_{spec}}$ (e.g., to check that $FM_{SP_{spec}}$ is a specialization of FM_{SP}).

From the $FM_{SP_{spec}}$ illustrated in Fig. 5, our tool succeeds to automatically generate a fully functional, valid, DP-based and highly customized SP. This generated SP has been successfully deployed in the JBoss Java server without any further manual interventions. It should be noted that the SP developer needs to manually adapt the generated SP to implement his/her application requirements (e.g., defining the business logic of the SP). The generated SP is composed of 334 Java code instructions and five XMLs[2]. These XMLs permit to configure the SOAP and MOM communication technologies and to configure the ESB Switchyard. The time required to derive the $FM_{SP_{spec}}$ (see Fig. 5) and generate its corresponding SP is one minute. By using the SOA traditional service contracts WSDL and WADL, and by relying on the tools that are offered by the ESB Switchyard, we require more than 20 minutes to develop the same SP and we need many manual interventions. Hence, we can say that using our FM_{SP} reduces the development costs (effort and time) of SPs.

In order to ensure that all the possible SPs, including the possible 790 compound DPs, that can be derived from our FM_{SP} are valid and fully functional, we implement in our tool [28] an automatic test functionality that generates all the possible SPs from FM_{SP}. These generated SPs have been successfully deployed in the JBoss Java server without errors and any further manual interventions. Hence, we can assume that these SPs are valid and fully functional.

5 Related work

In the literature, many works have been proposed to model the features and variability of SP and DPs [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. In Table 3, we present these works and compare them with our approach notably with our FM_{SP} (see Fig. 3) by relying on the following criteria:

- which feature modeling approach is used?
- do the SP features are modeled?
- do the modeled features include the required information that need to be discovered by the SC developers in order to realize SCs that can communicate correctly with a given SP?

[1] http://velocity.apache.org

[2] https://github.com/MSPL4SOA/MSPL4SOA-tool/tree/master/generated_SPs_SCs/conf/sp

- which communication technologies are modeled?
- does the variability of the communication technology features is modeled?
- do the DP features are modeled?
- does modeling compound DPs is considered?
- which DP type is modeled?
- does the code generation is supported. If it is the case, does the generated code is semi or fully functional?

Table 3: Comparing our approach notably our FM_{SP} with related works (Generic: modeling more than two communication technologies; OOP: Object Oriented Pattern; DREP: Distributed Real-time and Embedded Pattern)

Work	Feature modeling approach	SOA				Design pattern			Generated code
		SP	SC	Communication tech.		Single	Compound	Type	
				Features	Variability				
WSDL [5]	XML	+	+	SOAP	+	-	-	-	Fully
WADL [6]	XML	+	+	REST	+	-	-	-	Fully
Wada *et al.* [7]	FM	+	-	+ (n/a)	+	-	-	-	Semi
Fantinato *et al.* [9]	FM	+	-	SOAP	+	-	-	-	Semi
Ed-douibi *et al.* [12]	EMF	+	-	REST	+	-	-	-	Semi
Parra and Joya [8]	FM	+	-	Generic	-	-	-	-	Semi
Kajsa and Návrat [13]	FM	-	-	-	-	+	-	OOP	Semi
Street Fant *et al.* [14]	FM	-	-	-	-	+	+	DREP	-
Seinturier *et al.* [10], [11]	FM	+	-	Generic	+	-	-	-	Semi
Our approach	FM	+	+	Generic	+	+	+	SOA	Fully

From this table, we can notice that, in contrast with the related works, our FM_{SP} supports all the criteria. To summarize, it permits to generate fully functional, valid, DP-based and highly customized SPs for different communication technologies (i.e., generic) while modeling the required features that need to be discovered by the SCs to communicate correctly with these SPs. In the following, we discuss the related works presented in this table by highlighting their major contributions and showing how they can be used to extend and enhance our FM_{SP} for potential future works.

WSDL [5] and WADL [6] are two widely used service contracts that model the SP features of the communication technologies SOAP and REST, respectively. Our FM_{SP} is designed to extend these service contracts so it would be possible to generate DP-based SPs for different communication technologies.

Wada *et al.* [7] propose a FM that models SP non-functional features. Although their FM includes communication features, it does not explicitly specify which communication technologies are supported. This is why we put the symbol "+ (n/a)" in Table 3. Their FM can be used to extend our FM_{SP} in order to support SP non-functional features.

Parra and Joya [8] propose a FM that expresses the SOAP, REST and EJB communication technology features. However, in contrast with our FM_{SP}, their FM does not model the variability of these communication technology features, i.e., it does not model their possible configurations. As an instance, the HTTP methods (post, get, put and delete) that can be used by REST and the versions of SOAP are not modeled. Fantinato *et al.* [9] elaborate a FM that models the features of the communication technology SOAP. These two FMs [8], [9], as reported by their authors, need to be extended with the features (e.g., input and output data) of capabilities and services of SP so they can generate fully functional SPs. This has been considered in our FM_{SP}.

Seinturier *et al.* [10], [11] propose a FM to model the features of their FraSCAti tool. The latter is a component framework and an implementation technology providing runtime support for the service component architecture [29] in SOA. Our FM_{SP} has a higher level of abstraction than their FM because it is based on generic SOA DPs that are independent to the implementation technologies. Their FM can be used in conjunction with ours to be able to generate highly customized SPs dedicated for specific implementation technologies.

Ed-douibi *et al.* [12] introduce EMF data models that are dedicated to express the features of the communication technology REST. These models can be used to extend our FM_{SP} with more REST features (e.g., security features).

Kajsa and Návrat [13] introduce a FM that models the features of the object oriented DPs [30]. The goal is to enable generating, on demand, the code of a specific DP. The advantage of our FM_{SP} is that it allows to generate the code of DPs and also of compound DPs.

Street Fant *et al.* [14] elaborate a FM that expresses the features of a set of distributed real-time and embedded DPs. For each DP, they elaborate UML diagrams (collaboration, interaction, component and state machine diagrams) to identify its variability and behavior. The objective is to generate customized DPs and compound DPs with customized diagrams. In our work, we focus on generating the code of SOA DPs rather than generating their UML diagrams. Their work can be useful in our approach to generate customized diagrams for SOA DPs.

6 Conclusion

In this paper, we have proposed an approach that consists in developing a Software Product Line (SPL) for the mass-customization of Service Provider (SP) in Service Oriented Architecture (SOA). Essentially, we have introduced a Feature Model (FM), named FM_{SP}, for the SP feature modeling. Its objective is to enable developers to generate fully functional, valid, DP-based and highly customized SPs for different communication technologies while modeling the required features that need to be discovered by the SCs to communicate correctly with these SPs. Our FM_{SP} is designed as a DP-based service contract for SP that models 72 features. In particular, it includes the features of the three communication technologies Simple Object Access Protocol (SOAP), REpresentational State Transfer (REST) and Middleware Oriented Messaging (MOM). It also includes the features of 16 SOA DPs that are related to the service messaging category. It is important to note that our FM_{SP} allows to derive only valid compounds from these DPs which is crucial to drive valid SPs. Based on the

features and constraints that are modeled in FM_{SP}, we have calculated that it permits to derive 790 valid compound DPs from 2^{16} possible ones. Also, we have calculated that our FM_{SP} expresses 372904 possible configurations that can be used to derive a highly customized capability in the SP.

We have demonstrated through a practical case study and a developed tool, that our FM_{SP} is valid and permits to reduce the development costs (effort and time) of SPs. We have also shown the efficiency of our FM_{SP} compared with some SP feature modeling approaches that have been proposed in the literature, notably the two widely used service contracts Web Services Description Language (WSDL) and Web Application Description Language (WADL).

In future research, we plan to extend our FM_{SP} by other features, especially with security DPs, in order to generate more complex and secured SPs. Currently, we are working on developing an SPL, including a FM, that is dedicated for Service Consumer (SC). The objective is to enable SC developers to generate fully functional, valid, DP-based and highly customized SCs that can communicate correctly with all the possible SPs that can be derived from our FM_{SP}.

References

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

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

[3] Naveen Balani and Rajeev Hathi. *Apache CXF Web Service Development*. Packt Publishing, 2009.

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

[5] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. WSDL 2.0. https://www.w3.org/TR/wsdl20, 2007.

[6] Marc Hadley and Sun Microsystems. WADL. https://www.w3.org/Submission/wadl, 2009.

[7] 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.

[8] 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.

[9] 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.

[10] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, and Jean-Bernard Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. *Software: Practice and Experience*, 42(5):559–583, 2012.

[11] Feature model of FraSCAti. http://frascati.ow2.org/doc/1.4/ch12s02.html.

[12] 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.

[13] 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.

[14] Hassan Gomaa, Julie Street Fant, and Robert G Pettit IV. A pattern-based modeling approach for software product line engineering. In *Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS'2013)*, pages 4985–4994, Maui, Hawaii, USA, Jan 2013.

[15] 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.

[16] 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.

[17] Douglas C. Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, Kevin Henney, Regine Meunier, Peter Sommerlad, and Michael Kircher. *Pattern-Oriented Software Architecture (POSA)*, volume 1-5. Wiley, 1996-2007.

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

[19] 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.

[20] Akram Kamoun, Mohamed Hadj Kacem, Ahmed Hadj Kacem, and Khalil Drira. Feature model based on design pattern for the service provider in the service oriented architecture. In *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS'2017)*, pages 111–120, Porto, Portugal, April 2017.

[21] Kyo C. Kang and Hyesun Lee. Systems and Software Variability Management: Concepts, Tools and Experiences. chapter 2: Variability Modeling, pages 25–42. Springer, 2013.

[22] 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.

[23] 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.

[24] Claus Ibsen and Jonathan Anstey. *Camel in Action*. Manning Publications Corporation, 2011.

[25] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2004.

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

[27] Antonio Goncalves. *Begining Java EE 7*. Apress, 2013.

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

[29] Simon Laws, Mark Combellack, Raymond Feng, Haleh Mahbod, and Simon Nash. *Tuscany SCA in Action*. Manning Publications Corporation, 2011.

[30] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 1995.