Feature Model based on Design Pattern for the Service Provider in the Service Oriented Architecture

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

Feature Model based on Design Pattern 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), service contracts are widely used for designing and developing the features (e.g., services and capabilities) of Service Providers (SPs). Two of the most widely used traditional service contracts in SOA are: WSDL and WADL. We identify that these service contracts suffer from several problems, like: they only work for SOAP and REST communication technologies and do not rely on modeling SOA Design Patterns (DPs). One benefit of using SOA DPs is that they permit developing proven SPs for different platforms. In order to overcome these problems, we introduce a new DP-based Feature Model (FM), named FM_{SP}, as a service contract that models the variability of SP features including 15 SOA DPs (e.g., Event-driven messaging DP) and their corresponding constraints. This permits to easily identify and develop valid SOA compound DPs. We demonstrate, through a practical case study and a developed tool, that our FM_{SP} allows to automatically generate fully functional, valid, highly customized and DP-based SPs. We also show that our FM_{SP} reduces the required effort and time to develop SPs.

Index terms— Service oriented architecture; Service provider; Service contract; Design pattern; Feature model.

1 Introduction

Service Oriented Architecture (SOA) [1] is a widely used distributed computing platform. An SOA application is composed of Service Providers (SPs) and Service Consumers (SCs) that communicate through services. In this paper, we focus on designing and developing SPs. A SP can implement different features like: capabilities (i.e., operations), services and communication technologies (e.g., Simple Object Access Protocol "SOAP", Representational state transfer "REST" and Middleware Oriented Messaging "MOM"). In particular, a feature

can be designed to implement a Design Pattern (DP) [2] (e.g., `Event-driven messaging DP`).

The DP is an appropriate and proven design solution for a specific problem in a certain context. In the practice, it is frequent to use a compound DP to solve specific problems. A compound DP is a coarse-grained DP, who is composed of a set of finer-grained DPs. Erl [2] gathers in his book 78 SOA DPs (e.g., `Event-driven messaging DP`). In our work, we focus on studying 15 SOA DPs and their possible combinations (see Sect. 3).

In SOA, the contract-first is a widely used approach to develop Service Providers (SPs). It consists in developing service contracts that will be used to generate the artifacts of the corresponding SP. A service contract is a document (e.g., XML) of meta information through which the SP features are expressed.

The service contract represents a core part and one of the fundamental design principles in SOA [1]. Two of the most widely used SOA traditional service contracts are: Web Services Description Language (WSDL) [3] and Web Application Description Language (WADL) [4]. By studying these service contracts, we identify that they suffer from several problems, as follows:

P.1 they are dependent on specific communication technologies. The WSDL and WADL only work for the SOAP and REST communication technologies, respectively;

P.2 they do not use a standardized or common syntax to define SP features. They use different syntaxes even to describe the same SP features (e.g., input and output data features). This can lead to misinterpretation and difficulty to understand SP features;

P.3 they only allow to express a limited set of features. This prevents the development of complex SPs;

P.4 they do not rely on modeling SOA DPs and compound DPs [2]. This prevents the development of proven SPs. Developing SOA DPs and valid compound DPs is not a straightforward and easy task and requires a solid core of expert knowledge. For example, using the `Event-driven messaging DP` [2] requires the use of the `Service callback DP` [2]. In this case, if a given compound DP includes the `Event-driven messaging DP` but omits the `Service callback DP`, then this compound DP is invalid. Hence, the SP developer must manually develop valid SOA DPs and compound DPs in order to develop a valid SP;

P.5 some communication technologies (e.g., MOM) do not offer service contracts. In this case, using the first-contract approach to develop SPs cannot be possible.

In order to overcome the enumerated SOA traditional service contract problems, we propose in this paper, a DP-based Feature Model (FM), named FM_{SP}. The latter is designed as a service contract which models different SP features including SOA DPs and valid compound DPs. It is also designed to generate fully functional, valid and highly customized SPs for different communication technologies (SOAP, REST and MOM). In the literature, several FMs [5], [6], [7], [8] have been proposed to model the SP features. However, these FMs do not

permit to generate fully functional SPs and do not model DPs and compound DPs.

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 FM_{SP}. In Sect. 4, we evaluate our FM_{SP} 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

The Software Product Line (SPL) [9] is a paradigm whose goal is the mass-customization of applications that fit individual customer needs. In this context, it relies on the variability modeling of the application artifacts (e.g., source code and design) to develop customized applications. The variability consists in the ability of an artifact to be customized or configured in a particular context.

The FM [10] is the defacto standard for variability modeling in SPL. It permits to model the customized applications, based on features, that the SPL offers to generate. The FM expresses the legal combination of features to derive customized applications (e.g., see Fig. 1). An end user can derive from this FM, an Application Model (AM), by selecting the required features and deselecting the others, in line with the constraints of the FM (e.g., see Fig. 3). Based on the features of this AM, the SPL will generate the artifacts of the corresponding application.

The structure of the FM is a rooted tree of features that can be defined through different notations [10]. Many FM metamodels [11], [10] have been proposed in the literature that offer different notations. In this work, we rely on the FM metamodel of Czarnecki *et al.* [10] because its notations (see Fig. 1) allow to efficiently express the SP features.

We present in Fig. 1, these notations 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 (e.g., integer) to specify extra-functional information for features. The *feature* $[min, max]$ defines the lower and upper bounds of instances of a given feature. The *"requires"* constraint is in the form: 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: if a feature A excludes a feature B, then both features cannot be selected in the same AM. 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. The FM allows to define complex constraints between features (e.g., see the *propositional constraints* in Fig. 1).

3 Feature model for the service provider FM_{SP}

In this paper, we propose a new DP-based FM, named FM_{SP} (see Fig. 1), which allows to generate fully functional, valid and highly customized Service Providers (SPs) for different communication technologies (SOAP, REST and MOM). Our FM_{SP} is designed as a service contract, which models 46 SP features, that permits to overcome the SOA traditional service contract problems

(see Problems **P.1**, **P.2**, **P.3**, **P.4** and **P.5** in the introduction). We present in Table 1 the descriptions of these features.

We rely on 15 SOA DPs [2] to develop our FM_{SP}, which are: `Contract centralization DP`, `Service messaging DP`, `Direct authentication DP`, `Service agent DP`, `Intermediate routing DP`, `Service callback DP`, `Asynchronous queue DP`, `Event-driven messaging DP`, `Reliable messaging DP`, `Atomic service transaction DP`, `Dual protocols DP`, `State messaging DP`, `Stateful services DP`, `Service instance routing DP` and `State repository DP`. These DPs are described as features in our FM_{SP}.

Mathematically, there exist 2^{15} compound DPs that represent the existence or the non-existence of the 15 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 the `Service callback DP` (see Sect. 3.2). 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. 3.2). Erl presents several constraints between these DPs [2]. However, he illustrates them in many dispersed diagrams. This makes difficult to identify properly these constraints. Also, many constraints are missing.

3.1 Benefits

Our FM_{SP} is designed to overcome the SOA traditional service contract problems (see Problems **P.1**, **P.2**, **P.3**, **P.4** and **P.5** in the introduction). We present in the following the benefits of our FM_{SP} and which problems they consider:

1. it relies on the FM notations which permit to efficiently model the features and complex constraints (i.e., propositional constraints) of the SP. Also, the FM graphical presentation can be easily interpreted and facilitate deriving customized and valid SPs. In order to generate a valid SP, the SP developer only needs to derive an AM_{SP} (e.g., see Fig. 3), by selecting the required features and deselecting the others, from FM_{SP} (see Fig. 1). This facilitates the mass-customization of SPs. The constraints presented in FM_{SP} allow to guide the SP developer to derive valid AM_{SP} (considering Problems **P.1**, **P.2**, **P.3** and **P.5**);

2. it is designed as a service contract for SP which is generic and independent of the communication technologies. It can be considered as a reference model [12] which reflects the variability of practical SP features (considering Problems **P.1**, **P.2**, **P.3** and **P.5**);

3. it includes the required features to generate fully functional, valid and highly customized SPs. This reduces the required effort and time to develop SPs (considering Problems **P.1**, **P.2**, **P.3** and **P.5**);

4. it models 15 DPs and their corresponding constraints (see Sect. 3.2). This permits to easily identify and derive the valid compound DPs. By using the FAMILIAR tool [13], we identify, from the DPs constraints modeled in FM_{SP}, that there are 406 valid compound DPs from 2^{15} possible ones (considering Problem **P.4**);

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 and can be used to generate the artifacts for different contexts (considering Problems **P.1**, **P.2**, **P.3**, **P.4** and **P.5**);

3.2 Constraints of the design patterns

We discuss here the constraints between the 15 DPs modeled in our FM_{SP} (see Fig. 1). These constraints have been identified from theoretical and practical studies that we have led which are essentially based on these works [2], [14], [15], [16].

We express the feature `Contract centralization DP` as mandatory because our FM_{SP} is designed as a service contract that models the SP features, like the `SOAP`, `REST` and `MOM` communication technology features. In the introduction, we mention that the `MOM` does not offer a service contract (see Problem **P.5**). Because our FM_{SP} models the `MOM` features, then it can be considered as a service contract for `MOM`. Hence, it would be possible to use the first-contract approach to develop SPs based on the `MOM` features.

3.2.1 Constraints between the design patterns with the communication types

The SC needs to send a request message to the SP in order to invoke a capability. This communication type is called one-way. If this capability returns a response message, then the communication type is called two-way. In FM_{SP}, the feature `Output` represents the two-way communication type. If this feature is omitted when deriving an AM_{SP}, then the result type of the capability is void and the communication type is considered as one-way.

The features `Service callback DP`, `Atomic service transaction DP`, `Reliable messaging DP` and `State 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`.

3.2.2 Constraints of the communication design patterns

We express the feature `Service messaging DP` as mandatory because the three communication technologies (`SOAP`, `REST` and `MOM`) expressed in the FM_{SP} are messaging-based.

The `Dual protocols DP` requires that a given capability must support two or more communication technologies and vice-versa. The first propositional constraint defined in FM_{SP} implements this requirement.

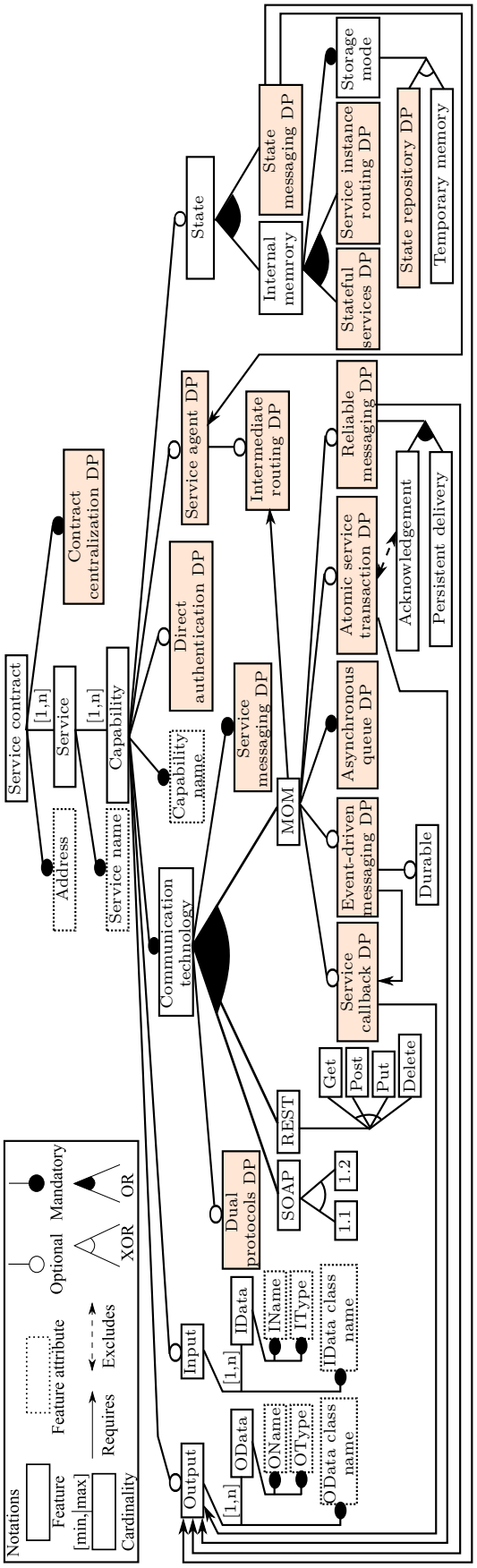

Figure 1: Feature model FM_{SP} (design pattern features are colored)

Table 1: Descriptions of the features of FM_{SP}

Feature name	Description
Service contract	Root feature
Address	Address of the SP
Service $[1, n]$	Services of the SP
Service name	Service name
Capability $[1, n]$	Capabilities of the SP
Capability name	Capability name
Input	Input data of a given capability
IData $[1, n]$	Gathering input data features
IName	Input data name
IType	Input data type
IData class name	Input data class name (e.g., Java class name)
Output	Output data of a given capability
OData $[1, n]$	Gathering output data features
OName	Output data name
OType	Output data type
OData class name	Output data class name (e.g., Java class name)
Communication technology	Gathering the communication technologies
SOAP	SOAP communication technology
1.1	SOAP version 1.1
1.2	SOAP version 1.2
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
MOM	Middleware oriented messaging communication technology
Durable	The MOM stores the messages of the publisher (i.e., SP) for the subscribers (i.e., SCs) if the latter disconnect. Upon reconnecting, the subscribers will receive all these messages
Acknowledgement	The MOM acknowledges the SP about the received messages
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
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
Direct authentication DP	Requiring that the SCs must provide authentication credentials to invoke a capability

Service messaging DP	Using a messaging-based communication technology which removes the need for persistent connections and reduces coupling requirements between the SP and SC
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 the use of a service agent
Service callback DP	Redirecting the SP response messages to a callback address that can be different of the requester SC address
Asynchronous queue DP	Deploying an intermediary MOM allowing the SP and SC to asynchronously communicate and to independently process messages by remaining temporally decoupled
Event-driven messaging DP	Asynchronously sending the response messages of the publisher (i.e., SP), when ready, to its corresponding subscribers (i.e., SCs)
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
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
Dual protocols DP	The capability is designed to support two or more communication technologies
State	Gathering the techniques that handle the state data of capabilities
Stateful services DP	Managing and storing state data by intentionally stateful utility services
Service instance routing DP	Supplementing the exchanged messages with an instance identifier, given by the SP as messaging metadata, so the SC uses it to communicate with the same instance of a given capability
State messaging DP	Delegating the storage of state data to the SP response messages instead to the SP internal memory
State repository DP	Deferring storing state data from a temporary memory to a state repository
Internal memory	Storing the state data in the SP internal memory
Storage mode	Gathering the modes of how the state data are stored
Temporary memory	Storing and managing the state data in a temporary memory

The SOAP and REST rely on a synchronous communication for the message exchanging between the SP and SC. The problem of the synchronous communication is that it forces processing overhead because the SP and SC must wait and continue to consume resources (e.g., memory) until they finish the message exchanging. To overcome this problem, the asynchronous communication is used as a solution. Also, in certain cases, it is necessary to implement (i.e., develop the artifacts of) the features Atomic service transaction DP and Reliable messaging DP.

In this context, it is possible to configure the SOAP and REST to implement the features Atomic service transaction DP and Reliable messaging DP and to support the asynchronous communication by implementing the feature Service callback DP. However, this requires significant costs associated with necessary infrastructure upgrades in the SP and also in the SC [2]. In the other hand, the MOM offers advanced functionalities to implement these features in both the SP and SC. Thus, we propose implementing these features with the feature MOM. Hence, we express the three asynchronous communication DPs (Service callback DP, Asynchronous queue DP and Event-driven messaging DP), the Atomic service transaction DP and the Reliable messaging DP as child features of the feature MOM.

In the MOM, the SC request messages are always carried on by an asynchronous queue [15] which it is an implementation of the feature Asynchronous queue DP. This is the reason why we define this feature as mandatory. In order to implement the feature Event-driven messaging DP, we use an asynchronous topic [15]. The latter is used to send asynchronously the SP response messages as notifications to its subscribers (i.e., SCs). The addresses of the asynchronous queue and topic are different. In this context, we define a "requires" constraint from the feature Event-driven messaging DP to the feature Service callback DP because one need redirecting the SP response messages to the asynchronous topic address.

In the case where the features Asynchronous queue DP and Service callback DP are selected and the feature Event-driven messaging DP is omitted when deriving an AM_{SP}, then the SC request and SP response messages are handled by two different asynchronous queues with different addresses. If the Service callback DP is omitted, then all messages are handled by a single asynchronous queue.

Because the MOM communication technology ensures a loosely coupled and an asynchronous communication between the SP and SC, then it should inform the SP and SC 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], [15]. This requirement is considered in our FM_{SP} by defining these two features as mutually exclusive and by defining the second propositional constraint in Fig. 1.

3.2.3 Constraints of the service agent design patterns

The feature Intermediate routing DP is implemented through a service agent (see Table 1). Thus, we define the Intermediate routing DP as an optional child feature of the feature Service agent DP.

In contrast of SOAP and REST, the SC request messages which are dedicated to MOM do not explicitly contain information about which capability the SC wants

to invoke. As a consequence, it would be not possible to invoke the required capability. As a solution, we propose implementing the feature `Intermediate routing DP` that dynamically routing the SC request messages to the required capability. Hence, we define, in our FM_{SP}, a "requires" constraint from the feature `MOM` to the feature `Intermediate routing DP`.

3.2.4 Constraints of the service state design patterns

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

The goal of the `State messaging DP` is delegating the storage of state data to the SP response messages. In this context, we propose implementing the `Service agent DP` to automatically perform this delegation. Hence, we define, in our FM_{SP}, a "requires" constraint from the feature `State messaging DP` to the feature `Service agent DP`.

4 Evaluation

In order to show the merits and evaluate our FM_{SP} (see Fig. 1) in practice, we propose to use the case study of the Integrated Air Defense (IAD) (see Fig. 2).

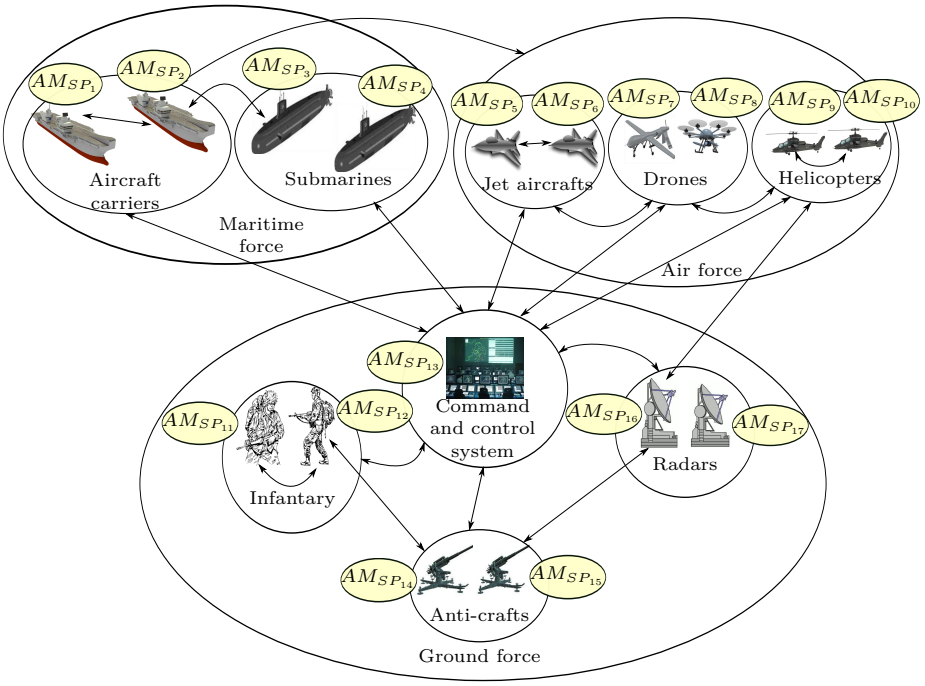

Figure 2: Case study of the integrated air defense

The IAD is a command and control compound of geographically dispersed force elements already in peace time as well as in crisis. In Fig. 2, we illus-

trate 17 force elements which 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 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, named SP_i, which it is responsible to implement its own features (see Fig. 1).*

As illustrated in the introduction (see Problems **P.1**, **P.2**, **P.3**, **P.4** and **P.5**), the SOA traditional service contracts WSDL [3] and WADL [4] suffer from several problems to develop SPs which prevent to efficiently realize the **Requirement 1**. In order to overcome these problems and to efficiently realize the **Requirement 1**, we propose deriving for each SP_i of the 17 IAD force elements a specific AM_{SP_i} (e.g., see Fig. 3) from our FM_{SP} (see Fig. 1). For example, in Fig. 2, the $AM_{SP_{13}}$ includes the SP features of the command and control system force element.

Using our FM_{SP} has several benefits as mentioned in Sect. 3.1 like it facilitates the mass-customization of SPs. This is important especially when we have an important SP count to develop which it is the case of our IAD case study (17 SPs to develop). Pohl *et al.* [9] show, from empirical investigations, that a given SPL is necessary and efficient if there are more than three or four systems to develop which it is our case.

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. 2 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 contribute to develop the DoD's business IT.

We present in Fig. 3 an example of a derived AM_{SP} that contains 40 SP features. We note that this AM_{SP} implements a valid compound DP that is composed of 15 DPs. It is possible that an AM_{SP} contains different services and capabilities (see Fig. 1). For the sake of simplicity, we illustrate in Fig. 3, an AM_{SP} that contains a single service, named "Personal", which it is composed of a single capability, named "login". The signature of this capability has a single input data, named "id", with a String type and included in the class "Session" (Java class). It also has a single output data, named "ok", with a Boolean type and included in the class "SessionResponse" (Java class). The goal of this capability is to enable a user to login to his/her account. We define the SP address as "http://localhost:8080/SP".

We developed a tool [17], [18] that relies on the Apache Velocity[1] tool (a "model to code" template engine) in order to transform a given AM_{SP} to the artifacts of the corresponding SP. Our tool generates SPs based on the Enterprise Service Bus (ESB) Switchyard [16]. Switchyard is a recent free software ESB that includes different technologies, such as the Service Component Architecture (SCA), HornetQ (a MOM implementation) [15], SOAP, REST, Spring

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

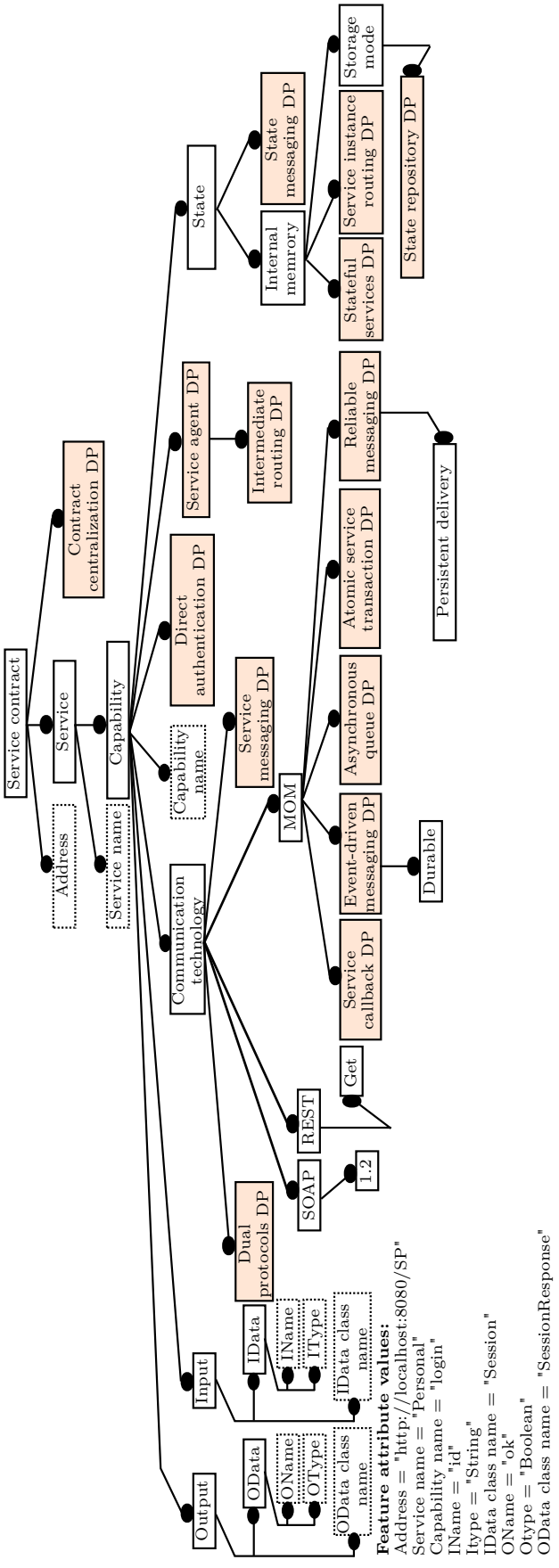

Figure 3: Application model AM_{SP} (design pattern features are colored)

and Apache Camel [14]. These technologies are integrated on demand in the generated SP depending on the features of AM_{SP}. Our tool also relies on the FAMILIAR tool [13] to develop and manage the FM_{SP} and AM_{SP} (e.g., to check that AM_{SP} is in conformity with FM_{SP}).

From the AM_{SP} illustrated in Fig. 3, our tool succeeds to automatically generate a valid and fully functional 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 can manually adapt the generated SP to his/her application requirements (e.g., defining the business logic of the SP). The generated SP is composed of 333 Java code instructions and five XMLs[2]. These XMLs permit to configure the SOAP and MOM technologies and to configure the ESB Switchyard. The time required to derive the AM_{SP} (see Fig. 3) and generate its corresponding SP is several seconds. By using the SOA traditional approach (i.e., using the SOA traditional service contracts and relying on the tools 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} (see Fig. 1) reduces the required effort and time to develop valid and fully functional SPs.

5 Related work

Many works have been proposed to model the SP features [7], [6], [8], [5], [19], [20]. In this section, we discuss these works and compare them with our FM_{SP} (see Table 2).

Table 2: Comparing our FM_{SP} with related work (FGC: Functionality of the Generated Code, CT: Communication Technology; CDP: Compound DP)

Approach	Tool	FGC	CT	DP	CDP
WSDL	XML	Fully	SOAP	-	-
WADL	XML	Fully	REST	-	-
Wada *et al.,* 2007	FM	Semi	n/a	-	-
Fantinato *et al.,* 2008	FM	Semi	SOAP	-	-
Ed-douibi *et al.,* 2016	EMF	Semi	REST	-	-
Parra and Joya, 2015	FM	Semi	Generic	-	-
Kajsa and Návrat, 2012	FM	Semi	-	+	-
Our FM_{SP}	**FM**	**Fully**	**Generic**	+	+

Wada *et al.* [7] propose a FM that expresses the variability of non-functional aspects of the SP. Although their FM includes communication features, it does not explicitly specify which communication technologies it supports. That is why we put the symbol "n/a" in Table 2. Their FM can be used to extend

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

our FM_{SP} (see Fig. 1) in order to express the variability of SP non-functional aspects. In contrast, our FM_{SP} permits to generate fully functional SPs and models DPs and compound DPs.

Fantinato *et al.* [6] elaborate a FM that models the variability of WSDL service contract of SOAP. Our FM_{SP} is more complete than theirs in a way that it contains the required features (e.g., the features of the input and output data) to generate fully functional SPs. As reported by Fantinato *et al.*, the input and output data features are absent in their FM. Another advantage of our FM_{SP} is that it expresses the variability of different communication technologies (SOAP, REST and MOM) and models DPs and compound DPs.

Ed-douibi *et al.* [8] introduce a MDE-based approach, called EMF-REST, that takes EMF data models as input to generate their corresponding REST services. In our work, we propose FM_{SP} to model the REST features. Their approach supports generating more REST features (e.g., security features) than ours. In contrast, our FM_{SP} permits to generate services with different communication technologies (SOAP, REST and MOM) and models SOA DPs and compound DPs.

Parra and Joya [5] propose an approach that generates a FM (using reverse engineering techniques) from current JEE artifacts. The latter, as affirmed by the authors, can support only limited features (e.g., the input and output data features and the MOM features are missing). The advantage of our FM_{SP} is that it permits to generate fully functional and highly customized SPs and models SOA DPs and compound DPs. Parra and Joya report that their approach should be extended to generate fully functional SPs.

Kajsa and Návrat [19] introduce a FM to handle the variability of the object oriented DPs. The goal is to support the instantiation and the evolution which occur to these DPs. Their work helps to generate the artifacts of a specific DP. The advantage of our FM_{SP} is that it allows to generate the artifacts of DPs and also of compound DPs.

6 Conclusion

In Service Oriented Architecture (SOA), the service contract is one of the fundamental design principles used to develop Service Providers (SPs). In this paper, we have proposed, based on SOA Design Patterns (DPs), a Feature Model (FM) for SP, named FM_{SP}. The latter is designed as a service contract for SP that is generic and independent of the communication technologies. We have modeled in our FM_{SP} 15 SOA DPs (e.g., Event-driven messaging) and their corresponding constraints. This allows to easily identify and generate the possible valid compound DPs which permits to develop valid SPs, accordingly. Based on our FM_{SP}, we have identified that there are 406 valid compound DPs from 2^{15} possible ones. We have demonstrated through a practical case study and based on a developed tool, that our FM_{SP} permits to automatically generate valid, highly customized, fully functional, proven and DP-based SPs. We have shown that using our FM_{SP} reduces the required effort and time to develop SPs. We have also demonstrated the efficiency of our FM_{SP} compared with related works notably the SOA traditional service contracts WSDL and WADL.

For future research, we plan to extend our FM_{SP} by other features, including other DPs, in order to generate more complex SPs.

References

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

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

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

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

[5] 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.

[6] 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.

[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] 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.

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

[10] 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.

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

[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] 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.

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

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

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

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

[18] 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.

[19] 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.

[20] 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.