Designing a set of Service-Oriented Systems as a Software Product Line

Flávio Mota Medeiros[1,3], Eduardo Santana de Almeida[2,3] and Silvio Romero de Lemos Meira[1,3]
Federal University of Pernambuco (UFPE)[1]
Federal University of Bahia (UFBA)[2]
Reuse in Software Engineering (RiSE)[3]
Email: {fmm2,srlm}@cin.ufpe.br esa@dcc.ufba.br

***Abstract*—Software reuse is crucial for organizations interested in productivity gains and software quality. In this context, Software Product Line (SPL) and Service-Oriented Architecture (SOA) are two reuse strategies that share common goals and can be used together with the purpose of increasing reuse and producing service-oriented systems, customizable to specific customers, faster and cheaper than creating individual systems. In this sense, this work investigates the problem of designing software product lines using service-oriented architectures, and presents a systematic approach to design product lines based on services. The approach provides guidance to identify, design and document components, services, service compositions and their associated communication flows. In addition, an initial experimental study performed with the intention of validating and refining the approach is also depicted demonstrating that the proposed solution can be viable.**

I. Introduction

Software reuse is a key factor for enterprises interested in productivity gains, reduced development costs, improved time-to-market, and increased software quality [1]. In the context of software reuse, SPL and SOA are two strategies that are getting attention in research and practice lately [2][3]. In addition, these strategies share common goals, i.e., they both encourage organizations to develop flexible and cost-effective software systems, and support the reuse of existing software and capabilities during the development of new systems [4].

In this way, SPL and SOA concepts can be used together with the purpose of increasing and systematizing reuse during the Service-Oriented Development (SOD) and producing service-oriented systems faster, cheaper and customizable to specific customers [5]. Moreover, service characteristics, e.g., dynamic discoverability and binding, can be used to support the development of flexible and dynamic product lines [6].

This work investigates the problem of designing software product lines using service-oriented architectures. This combination raises several challenges during the design, such as how to identify and design services and service compositions for the domain, decide the variation points to be considered in the context of SOD, identify service variability implementation mechanisms and define architectural views to represent the Service-Oriented Product Line Architecture (SO-PLA).

In this sense, a systematic design approach with a set of activities, with clearly defined inputs and outputs, and performed by a predefined set of roles is described in this work with the purpose of providing guidance to solve the problems of designing service-oriented product line architectures. In this work, a service-oriented product line is considered as a set of similar service-oriented systems that supports the business processes of a specific domain and can be developed from a common platform or set of core assets [2].

In order to define a SO-PLA, an approach is essential to provide guidance to the team, specify the artifacts to be produced, and associate activities with specific roles and the team as a whole. Without it, the development team may develop software in an ad-hoc manner, with success relying on the efforts of a few dedicated individual participants [7].

The remainder of this paper is organized as follows: Section II discusses its related work; Section III presents an overview of the proposed approach, and Section IV discusses its activities in detail; Section V presents an experimental study performed with the purpose of validating and refining the proposed solution, and finally, Section VI concludes the paper with some concluding remarks and directions for future work.

II. Related Work

Few work in the literature has considered the combination of SPL and SOA concepts. An approach for developing service-oriented product lines was presented in [6]. In this work, a method to identify services and service compositions from feature models is depicted. Moreover, the work proposes templates to document the services identified. In our work, we also provide methods to identify services candidates, not only from feature models, but also using business processes, use cases and quality attribute scenarios. We also suggest templates to document services, components and their communication flows. Some architectural views are also proposed to represent the interactions among architectural elements.

The concept of Business Process Line (BPL) is used in [8]. This work provides a process to develop service-oriented product lines based on business processes that contain variability and can be customized to specific customers. In this way, the business processes are adapted to a specific context according to the customer's requirements, and then, the target SOA system is developed. Our work considers variability in the business processes, but it also uses feature models to represent variability in an easy and exploitable way. We believe that representing too much variability information using only business process models may pollute these diagrams.

In [9], an initial process for service-oriented product lines is presented. This work discusses the characteristics of SOA and SPL processes, but does not provide a systematic process for service-oriented product lines. It also depicts the implementation of service variability using code transformation tools with an example on the *Web Store* domain.

The key difference of the work presented in this paper is the systematization of the design, which provides a set of sequential activities and sub-activities with clearly defined inputs and outputs. In addition, our work was validated and refined through an initial experimental study, different from the related work discussed.

III. Approach Overview

The approach is called SOPLE-DE, which means design in the context of service-oriented product line engineering. It is a top-down approach for the systematic identification, design and documentation of service-oriented core assets supporting the non-systematic reuse of service-oriented environments. The SOPLE-DE is divided in two life cycles as the software product line engineering [10]. The core asset development aims to provide guidelines and steps to identify, design and document generic architectural elements with variability. During the product development cycle, the architectural elements are specialized to a particular context according to specific customer requirements [2].

The SOPLE-DE receives the domain feature model, the business process models and the quality attribute scenarios of the domain as mandatory inputs. Moreover, the domain use cases are also considered as optional inputs. The use cases are not normally used in SOD because the focus is on business processes rather than use cases [11]. However, SOPLE-DE considers the use cases as optional inputs because they can provide detailed (additional) information that may not be presented in the feature model and business processes.

The output of the SOPLE-DE is an architecture document describing the SO-PLA, with traceability links among the architectural elements, i.e., components and services, and the models used to identify them, e.g., domain feature model and business process models. We recommend the following main sections in the architecture document:

- *Introduction:* it describes an overview of the architecture document;
- *Glossary:* it lists the main words and acronyms with their meaning in the architecture document;
- *Overview:* depicts a description about the service-oriented product line project, its purpose and systems;
- *Technology description:* it lists the technologies that will be used in the project, e.g., programming languages;
- *Architectural views:* presents the architectural views produced to represent the SO-PLA;
- *Architectural elements:* describes the architectural elements identified including UML models to represent their behavior.

The SOPLE-DE considers the architectural style shown in Figure 1. This architectural style presents the layers that are commonly used in SOA [12][13]. Thus, SOPLE-DE provides guidelines to identify, design and document architectural elements for these layers. We use this architectural style because we believe that these layers are essential for any SOA solution.

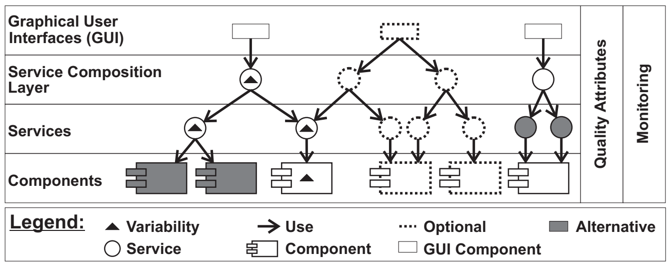

Fig. 1. SOPLE-DE Layered Architectural Style

The interface layer is composed of Graphical User Interfaces (GUI) components. Only service-oriented product lines that require visual interfaces to interact with the services may use this layer. In addition, the visual interface components may be specific for each system, in this way, they will not be considered as core assets in some product lines, i.e., they will be developed specifically for each new system.

The service composition layer consists of composite services, which implement coarse-grained business activities, or even an entire business process, that need the participation and interaction of several fine-grained services. The service layer is composed of self-contained and business-aligned services, which implement fine-grained business activities.

The component layer consists of a set of components that provide functionality for the services and maintain their Quality of Service (QoS) [12]. The quality attribute layer consists of additional architectural elements responsible for satisfying specific quality attributes, e.g., performance and security. Finally, the monitoring layer, which is responsible for monitoring the health of the SOA solution, e.g., monitoring the number of service calls, the response time of services, number of service errors and Service Level Agreements (SLA).

It is important to note that the architectural elements of these layers are developed taking variability into account, and they can be mandatory, optional or alternative. For instance, different metrics can be monitored in specific systems and each system of the product line may be customized to satisfy specific quality attributes.

The SOPLE-DE considers that service-oriented product line architectures support two variability levels [8]: *Configuration variability,* in which architectural elements are selected from the core assets in order to obtain the target system, i.e., optional and alternative architectural elements are selected or excluded from the architecture; *Customization variability,* in which architectural elements already selected for a system are customized according to the requirements of the specific system, i.e., architectural elements with variability are customized internally.

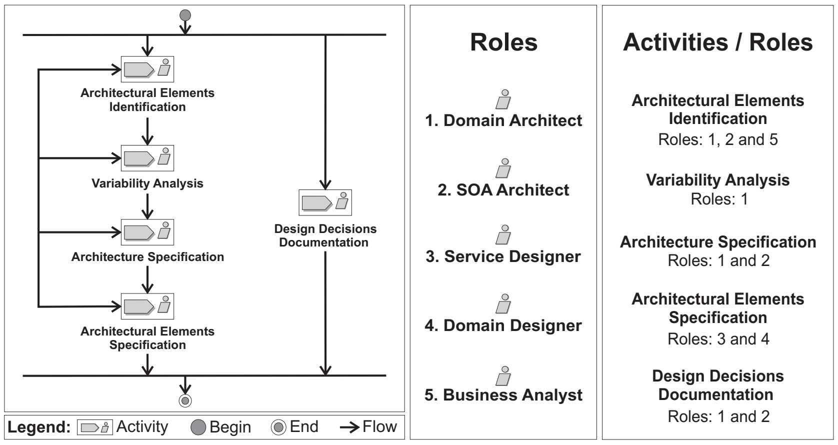
Architectural Elements Identification
Variability Analysis
Architecture Specification
Architectural Elements Specification
Design Decisions Documentation
Legend: Activity Begin End Flow
Roles
1. Domain Architect
2. SOA Architect
3. Service Designer
4. Domain Designer
5. Business Analyst
Activities / Roles
Architectural Elements Identification Roles: 1, 2 and 5
Variability Analysis Roles: 1
Architecture Specification Roles: 1 and 2
Architectural Elements Specification Roles: 3 and 4
Design Decisions Documentation Roles: 1 and 2

Fig. 2. SOPLE-DE Activities and Roles

In addition, SOPLE-DE considers variability in the communication among the architectural elements, e.g., different protocols can be used for communication, and the messages exchanged can be sent in a synchronous or asynchronous way.

IV. SOPLE-DE

In this section, the activities and tasks of the SOPLE-DE are discussed in detail. The activities and roles of the SOPLE-DE are presented in Figure 2. As it can be seen, its activities are executed in an interactive way.

A. Architectural Elements Identification Activity

The identification of service candidates is a challenging and crucial task of SOD [6]. In the context of SOPLE, it is even harder due to concerns with commonality and variability. In this sense, SOPLE-DE combines complimentary techniques that use different sources to identify services.

The SOA architect should perform the service identification activity. However, the business analyst should review the service candidates to ensure an accurate representation of the business logic [14]. The service identification techniques provided by SOPLE-DE are described next.

1) Defining Services from Business Processes: The steps that should be performed to define the architectural services and service compositions from the business processes are: *identify automatic business activities* and *analyze business activities interactions*. In the first step, automatic and partially automatic business process activities will be identified from the business process models. Automatic business process activities are performed entirely by a system with no manual interference, while partially automatic activities are executed manually, but supported by a system [15]. Initially, each automatic and partially automatic business process activity is considered as a service candidate.

The first step towards the identification of service compositions is the analysis of interactions among business process activities. In this step, the list of automatic and partially automatic business activities produced previously will be analyzed to identify related activities that need to be executed in special conditions, such as the interaction patterns listed next that usually require service compositions to control and execute the related business process activities correctly [16][17]:

- *Sequential:* business process activities that must be executed in a pre-specified sequential order;
- *Concurrent:* activities that must be executed concurrently in order to perform a functionality correctly;
- *Exclusive:* business process activities that cannot be activated during the execution of other business activities;
- *Subordinate:* activities that can be activated only if another business activity is being executed;
- *Loop:* business process activities that must be executed until a condition becomes false, or executed a predefined number of times;
- *Optional:* a business process activity that will be executed according to conditions;
- *Alternative:* a set of business process activities in which only one will be executed depending on conditions.

For instance, Figure 3 depicts a concurrent interaction pattern. Considering that activities A,B and C are automatic or partially automatic, service candidates can be identified to implement each of these activities. Additionally, a service composition can be used to control and execute these services concurrently. In addition, a business process may contain variability, thus, the services should be classified as mandatory, optional and alternatives depending on the business activities they implement, e.g., service B, which implements business process activity B, is optional (dashed line) in the example.

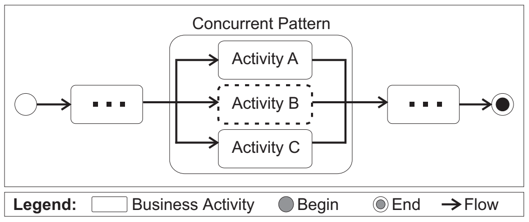

Fig. 3. A Concurrent Interaction Pattern

2) Defining Service and Component Candidates from Use Cases: In this step, the key business entities of the domain should be identified using the use cases. These entities are directly manipulated by several services and need specific services to implement their life-cycle operations, e.g., create, delete, update and retrieve [3][14]. As the use cases are optional inputs, if they were not provided, this identification should be realized from the business processes and feature model, which may turn this step more difficult.

The key business entities are described in the use cases usually using nouns, e.g., customer and account. However, other service and component candidates or their interface operations can be obtained from the verbs contained in the use cases, e.g., the users should **authenticate** themselves and the system should **restrict** access to specific pages, can be used to identify services to authenticate users and control user access respectively [18][19].

A conceptual service model can be defined from the use cases with the purpose of facilitating the identification of services. It consists of a model of the problem domain and it is created without regard for any application or technology. Figure 4 depicted a conceptual service model [18]. Each entity in the logical model is either a stateful entity or a stateless entity. In this sense, stateful entities, such as account and customer, can be considered as components, while the managers are considered entity service candidates that manipulate these entities. The conceptual service model will also be useful during the definition of the architectural element's interfaces.

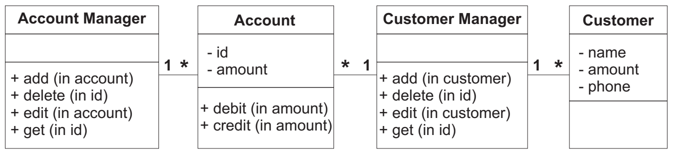

Fig. 4. Conceptual Service Model

3) Defining Services and Components from Features: The identification of service and component candidates from the feature model should be performed using the concept of service feature, which is a major functionality of a specific domain that can be added or removed of systems, and configured independently of other features [17].

In this task, each service feature identified is considered a component, service or service composition depending on its characteristics and granularity. The service features will dictate the granularity of the architectural elements identified. Thus, it should be taken into account that normally components are finer-grained than services, which are finer-grained than service compositions.

Additionally, service features considered as service candidates should share the following service characteristics, i.e., they should be stateless, autonomous, coarse-grained and interoperable [3][20]. Service features without these characteristics should be considered as components.

The architectural elements identified should be marked as mandatory, optional or alternative depending on the type of the service feature that originates them, i.e., they can be optional, mandatory or alternative. Moreover, the architectural elements may contain optional and alternative sub-features, which may impose variation on their design.

For instance, Figure 5 presents a possible way to use the feature model to identify services, compositions and components. As it can be seen, service compositions control the execution of services, while services use the functionalities that are provided by the architectural components.

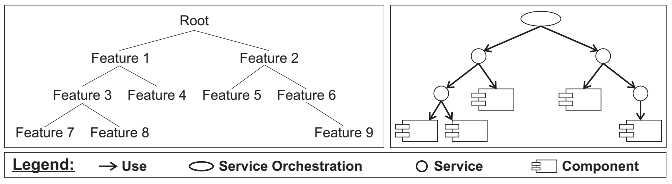

Fig. 5. Identification of Architectural Elements from Features

4) Defining Services and Components from Quality Attributes: SOPLE-DE considers a specific layer in its architectural style to deal with quality attributes. In this sense, service and component candidates can be identified using the quality attribute scenarios. The SOA and Domain architects have to analyze the quality attribute scenarios in order to identify services and components that support the accomplishment of architectural quality attributes. For instance, considering availability, techniques such as ping/echo and heartbeat can be used [21]. Thus, additional services can be identified to send messages to check the availability of other services. For security, services can be identified to authenticate users, control access to specific functionalities or limit access to some services depending on the user's profile.

In addition, the quality attribute scenarios should be used during the identification of architectural elements from business processes, use cases and feature model. For instance, if each business process activity is encapsulated into a specific service, each activity can be modified locally without impacting on other activities. On the other hand, depending on the granularity of the business activities it may lead to the identification of several fine-grained services, which may increase the number of service calls impacting performance [22]. Hence, the quality attributes scenarios and their priority should be considered during the identification of architectural elements.

It is important to note that some components and services may be identified in different techniques. In this way, at the end of the architectural elements identification activity, the service and component candidates identified using all the available techniques should be consolidated, duplicated elements should be removed and the complete list of architectural elements should be analyzed and reviewed by the business analyst and architects with the purpose of producing an accurate list of services and components. After performing the architectural elements identification, an initial set of service and component candidates including their interface operations should be listed in the architecture document.

5) Define Flows: In this task, the communication among services will be defined. The role responsible to perform the flow identification is the SOA architect, who is responsible to analyze the services identified and define their communication protocols, e.g., SOAP or REST, and their communication types, e.g., synchronous or asynchronous, that will be used by the services to communicate. The quality attributes should be considered in this task as well, since the protocol and type of communication impact some quality attributes, e.g., asynchronous communication can increase performance.

Moreover, the integration mechanism that will be used in the SOA should be defined. For example, service consumers and providers can communicate directly without any broker such as the peer-to-peer communication pattern, or they can be mediated by a middleware, which in the context of SOA, it is known as the Enterprise Service Bus (ESB) [20].

At this point, it is also important to decide if a service registry will be used. Service-oriented product lines that require dynamic discoverability and binding should use it. In this way, services will not communicate directly and will use the registry to register their services and find service providers [20].

It is important to note that in the context of SOPLE, the communication protocol and type can be treated as variation points. In other words, the same service can be accessed using several protocols in a synchronous or asynchronous way depending on the system of the product line [23]. At this point, the information about service communications should be added to the list of architectural elements produced previously.

B. Variability Analysis Activity

At this point, the components, services, service compositions and their communication flows were defined. During the variability analysis activity, it will be defined how the variability contained in the feature model, business processes, use cases and quality attributes will be implemented. This task is a refinement of the architectural elements identified previously. After the variability analysis activity, the components and services are no longer candidates anymore. The following steps should be performed in this activity.

1) Analyzing Architectural Elements: During this step, the granularity and cohesion of the architectural elements should be analyzed. Additionally, cross-cutting variability concerns that cannot be isolated into a specific component or service should be identified.

In this sense, the operations contained in the architectural elements interfaces should be analyzed in order to identify related operations, e.g., the ones related to the same business entities, that can be joined in a unique architectural element with the purpose of increasing cohesion and reducing the number of candidates. Interface variability, e.g., parameterization, can be introduced here to group similar operations into a single parameterized operation.

The granularity analysis will be useful to identify fine-grained and coarse-grained variation points. Fine-grained variation points are the ones that can be implemented by changing a class attribute or method. Thus, the selection of variability mechanisms will be easier, since some variability implementation mechanisms can be more appropriated to implement fine-grained variability, such as aspects, object-oriented and SOA design patterns, conditional compilation and configuration files, while others fit better in coarse-grained variability, e.g., component-based development and service-oriented development, in which components or services are substituted/changed to implement the variability [24], [25].

Finally, crosscutting variability concerns should be analyzed. In some cases, a logging component for instance, has its code spread in several components. Thus, the domain architect should analyze these cases in order to decide how this variability is going to be implemented. Some mechanisms are well-suited for this kind of variability, e.g., aspects [26].

2) Defining Variability Implementation Technique: In this step, the variability implementation technique will be defined and documented. It is important to note that the binding time and the variability type of a variation point should be considered during this step because different variability mechanisms support specific types of variability and binding times [27]. For instance, conditional compilation cannot be used to implement dynamic (runtime) binding times, since using conditional compilation the selection of variants is realized at compile-time.

C. Architecture Specification Activity

In the architecture specification activity, the high-level design of components, services, service compositions and their flows will be specified. In this activity, architectural views are produced, and they may contain variability as the artifacts of the core assets development cycle. Thus, architecture specification requires notations with support for variability representation, such as [28][29].

The architecture specification activity receives the list of architectural elements (components, services and service compositions) and their flows identified in the previous activities as inputs, and produces the architectural views that will be documented in the architecture document. The domain and SOA architects should perform this activity.

A SO-PLA, as any other software architecture, is a complex entity that cannot be represented in a simple one-dimensional fashion. Since there are different stakeholders involved in a product line project with particular concerns about the systems, it is important to use multiple views to represent the SO-PLA [21]. Moreover, the use of multiple architectural views are essential in order to handle separately the functional and non-functional requirements [30]. The views used in SOPLE-DE are depicted in Figure 6 and described next [31].

Layer View: the objective of this viewpoint is to represent the layers of the SOA solution. Thus, in this perspective, the architectural elements identified in the previous activities are represented in their respective layer.

Integration View: the purpose of this view is to depict the integration mechanism that will be used in the SOA, e.g., peer-to-peer (direct service communication) or hub-and-spoke, which is mediated by an ESB [20].

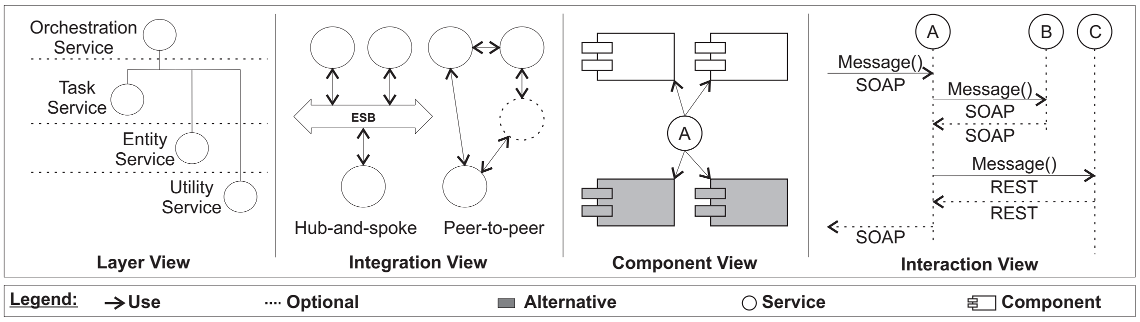

Fig. 6. SOPLE-DE Architectural Views

Interaction View: the main goal of this viewpoint is to show the communication protocols and the messages exchanged among service consumers and providers. It is also used to represent concurrency issues and depicts the dynamic behavior of service compositions.

Component View: the purpose of this perspective is to represent the logical structure of the components that will be used by the services. It is a best practice to create high-level and coarse-grained service interfaces that implement a complete business process or set of business activities. Thus, services should expose the functionality of a couple of components [18].

The domain and SOA architects have to decide which architectural views will be used to represent the SO-PLA depending on the stakeholders involved in the project, and the size and domain of the product line being constructed. The architects also decide the level of details of each view.

D. Architectural Elements Specification Activity

In this activity, the low-level design and the detailed description of components and services will be defined and documented. The purpose of this activity is to design and document the internal behavior, pre-conditions, invariants, post-conditions and contracts (interfaces) of the services and components of the architecture. This activity receives the list of architectural elements identified in the previous activities, their flows and the architectural views produced as inputs, and it produces UML diagrams, and service and component description documents.

At the end of the architectural elements specification activity, the component and service descriptions should be documented in the architecture document. These descriptions should contain the following fields:

- Description: describes the general purpose of the service or component, i.e., functional requirements;
- Interfaces: describes the component and service operations, and its input and output parameters;
- Pre-conditions: lists the conditions that must be satisfied before using the component or service functionalities;
- Invariants: describes the conditions that must be satisfied during the whole execution of the service operations, otherwise, the functionality being executed should stop as soon as the condition fails;
- Post-conditions: lists the conditions that must be satisfied after the execution of the component or service operations;
- Quality Attributes: describes the non-functional requirements (e.g., service level agreements) that are satisfied by the service, e.g., considering a performance attribute, the response time can be defined between 0.75 and 1.5 seconds.

E. Design Decisions Documentation Activity

Design decisions are very important parts of the design discipline [32]. As it can be seen in Figure 2, these decisions can be made during the whole SOPLE-DE, since design decisions are made for problems that the domain and SOA architects face during the project. In this context, problems mean specific situations that need a special attention during the design and must be discussed and documented. Such decisions can be the selection of technologies that are going to be used and the variability technique that will be used to implement specific variation points. This activity receives as input the design decisions that appear during the SOPLE-DE, and its output is a set of solutions for each specific problem found with the rationale that motivates the selection of each solution.

V. EXPERIMENTAL STUDY

An initial experimental study on the *Travel Reservation* domain was performed with the purpose of evaluating and refining the SOPLE-DE. In this experiment, the process of Wohlin [33] was used to define, plan and execute the experimental study. In addition, the Goal Question Metric (GQM) framework was also used to define the experiment [34] as described in the next sections.

A. The Definition

The goal of this experiment was to *analyze the SOPLE-DE* for the purpose of *evaluation* with respect to its *efficacy and applicability* from the point of view of *researcher* in the context of *service-oriented product line projects*. The questions used to define the experiment are described next:

- Does the SOPLE-DE aid architects to identify services with low coupling?
- Does the SOPLE-DE aid architects to design services with low instability?

- Does the SOPLE-DE aid architects to define service operations with high cohesion?
- Do the subjects have difficulties to apply the SOPLE-DE in practice?

The following metrics were analyzed in the experimental study with the intention of validating the SOPLE-DE quantitatively.

Service Coupling (SC): Coupling is a measure of the extent to which interdependencies exist between software modules [35]. Low coupling indicates a well-partitioned system and avoids problems of service redundancy and duplication [36]. In this sense, the following metric was evaluated [37]: $SC\,(s) =$ number of service providers used by a service consumer (s), where (s) is a service of a given system. This coupling metric has range [0, n], where n is the number of service providers different from (s) of a given system. SC = 0 indicates a totally loosely coupled service, and SC = n indicates a maximally coupled service [37].

Service Instability (SI): The reason for a design to be rigid, fragile and difficult to reuse is the interdependency among its modules. A design is rigid if it cannot be changed easily, and a single change in a specific service causes a cascade of changes in several independent modules. In this sense, the following metric was evaluated [38]: $SI\,(s) = P/(P+C)$, where C is the number of service consumers that call service (s), and P is the number of service providers that service (s) uses. The service instability metric has range [0, 1], where SI = 0 indicates a maximally stable service and SI = 1 indicates a totally unstable service [38].

Lack of Service Cohesion (LSC): Cohesion is the degree of the strength of functional relatedness of operations within a service [36]. Highly cohesive service operations indicate good functionalities subdivision, and imply high reusability. Lack of cohesion increases complexity, thereby increasing the likelihood of errors during the development process [39]. In this sense, the following metric was evaluated: LSC (s) = Number of business entities accessed by the operations of a service (s). This lack of service cohesion metric has range [1, n], where n is the number of business entities of a specific domain. LSC = 1 indicates totally cohesion among service operations, and LSC = n indicates maximally low cohesion. This metric assumes that the service operations of a specific service should access at least one business entity of the domain.

Applicability Problems (AP): This issue will be used to identify possible applicability problems during the execution of the SOPLE-DE. The applicability problems found will be mapped to the respective activity of the approach according to the information provided by the subjects using a questionnaire. In this sense, the following metric was evaluated: AP = % of subjects that had difficulties to apply the SOPLE-DE.

B. The Planning

The experiment definition determines the foundations for the experiment, i.e., why the experimental study will be conducted, while the experiment planning prepares for how the study will be conducted [33]. As any other type of engineering activity, the experiment must be planned and the plans must be followed in order to control the experiment. Its results can be disturbed, or even destroyed if not planned properly.

Context: The objective of this experiment is to evaluate the efficacy, understanding and applicability of the SOPLE-DE in the context of service-oriented product line projects. The experiment was conducted in a university laboratory with postgraduate students using a project on the Travel Reservation domain. The experimental study was conducted as a Replicated Project, which is characterized as being a study which examines object(s) across a set of teams, and a single project [40]. The subjects of the study will be requested to act as the roles defined in the SOPLE-DE, i.e., domain architect and SOA architect. However, a subject can play more than one role during different activities and tasks of the SOPLE-DE. All the subjects were trained to use the approach.

The Null Hypotheses: In the context of experimental studies, there are two types of hypotheses: null and alternative hypotheses. The *null hypotheses* are the ones that the experimenter wants to reject with as high as significance as possible, while the the *alternative hypotheses* are the ones in favor of which the null hypotheses are rejected [33].

In this experimental study, the null hypotheses determine that the use of the SOPLE-DE in service-oriented product line projects does not produce benefits that justify its use and that the subjects will have difficulties to understand and apply the approach in practice. Thus, according to the selected criteria, the following null hypotheses were defined:

H1. μSC of services without SOPLE-DE $<$ μSC of services with SOPLE-DE

H2. μSI of services without SOPLE-DE $<$ μSI of services with SOPLE-DE

H3. μLSC of service operations without SOPLE-DE $<$ μLSC of service operations with SOPLE-DE

H4. μMore than 50% of the subjects will have difficulties to understand the SOPLE-DE

H5. μMore than 50% of the subjects will have difficulties to apply the SOPLE-DE in practice

The Alternative Hypotheses: In this experimental study, the alternative hypotheses determine that the use of the SOPLE-DE in service-oriented product line projects produces benefits that justify its use and that most of the subjects will not have difficulties to understand and apply the approach in practice. Thus, the following alternative hypotheses were defined:

H1. μSC of services without using SOPLE-DE $>=$ μSC of services with SOPLE-DE

H2. μSI of services without SOPLE-DE $>=$ μSI of services with SOPLE-DE

H3. μLSC of service operations without SOPLE-DE $>=$ μLSC of service operations with SOPLE-DE

H4. μMore than, or 50% of the subjects will not have difficulties to understand the SOPLE-DE

H5. μMore than, or 50% of the subjects will not have difficulties to apply the SOPLE-DE in practice

The Project used in the Experimental Study: The business process [16] and some variation points [10] of the project used in the experimental study is presented in Figure 7. The subjects received the inputs required by the SOPLE-DE and produced an architecture document describing the design performed. At the end of the experimental study, the artifacts produced by the subjects were analyzed both quantitatively and qualitatively as described in the results of the experiment.

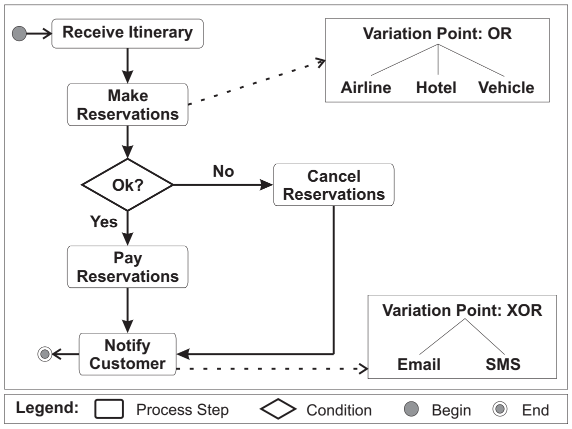

Fig. 7. The Project used in the Experimental Study

C. The Operation

This section presents the details about the execution of the experimental study performed with the purpose of evaluating and refining the SOPLE-DE.

The Environment: The experimental study was conducted during 8 hours at the Federal University of Pernambuco (UFPE). The experimental study was composed of 8 subjects that performed the experiment in parallel. In the experiment, three service-oriented systems were designed as a service-oriented product line.

Training: The subjects were trained before the experimental study began. The training took 22 hours, divided into 10 lectures with two hours each, during a postgraduate course at the university, and 2 hours independently of the university course presented by the experimenter. In addition, the subjects who used the proposed approach were trained 2 hours more to use the SOPLE-DE.

As previously described, the study was performed in two steps: initially, the subjects were trained in several aspects of software reuse, SPL, SOA, reuse processes and SOPLE-DE, and after, they performed the design of the service-oriented product line project in 8 hours.

Subjects: The subjects were four M.Sc. and four Ph.D. students from the Federal University of Pernambuco. All the subjects considered had industrial experience in software development, more than one year at least. Two subjects had participated in industrial projects involving some kind of reuse activity, for instance, component-based development, framework, or web services development. In addition, all the subjects had participated in SPL academic projects, and two subjects have taken part of an academic SOA project.

D. The Analysis and Results

In this section, the results obtained with the experimental study are presented. This section is divided into quantitative and qualitative analyses.

1) Quantitative Analysis: The quantitative analysis was divided in four analyses: coupling and instability of the service-oriented product line architecture, service operations cohesion and difficulties found during the use of the SOPLE-DE in practice.

After collecting the information about the service coupling, instability and cohesion, the data collected was analyzed. Figure 8 shows the metric results for the services identified by the subjects. In the graphics, the axis (X) shows the ID of the subjects, while the axis (Y) represents the service coupling mean, service instability mean and the average cohesion of the service operations.

The subjects with Id = 1, 2, 3 and 4 used the SOPLE-DE during the experiment, while subjects with Id = 5, 6, 7 and 8 designed the project without following a structured method. This is a threat of the experiment, since we cannot determine if the results are consequence of using the SOPLE-DE, or just because a structured method was used [41]. The experiment was performed in this way due the lack of service-oriented product line processes in the literature. In addition, we could not found baselines for the metrics used in the experiment. However, the metric results of this experiment can be used as baselines for new experiments.

As it can be seen in Figure 8, the coupling, instability and cohesion of the services generated using the SOPLE-DE (subjects with Id = 1, 2, 3 and 4) are lower when compared with the services identified by the subjects without use the structured method (subjects with Id = 5, 6, 7 and 8). Analyzing the answers of the subjects using the feedback questionnaire with the difficulties found to apply the SOPLE-DE in practice, it was identified that two subjects (Id = 3 and 4) had difficulties to apply the approach in practice.

The subject (Id = 3) had difficulties during the architectural elements identification activity. It was highlighted that sometimes it was difficult to determine if a specific module identified was a component or a service. Thus, the documentation was modified, and a set of characteristics of services was added to ease the identification of architectural elements. The other subject (Id = 4) mentioned difficulties during the production of the architectural views. In order to clarify this point, some examples with architectural views were introduced in the documentation. In this sense, two subjects had problems to apply the SOPLE-DE activities in practice. It represents 50% of the total number of subjects that used the approach. In this sense, the null hypotheses could be rejected.

A correlation could be identified between the experience of the subjects and the difficulties found to apply the SOPLE-DE in practice. The subjects that had difficulties did not have much experience in software projects, i.e., they have participated of two industrial software development projects as developers, and have never worked with SPL or SOA, different from the other two subjects that used the SOPLE-DE.

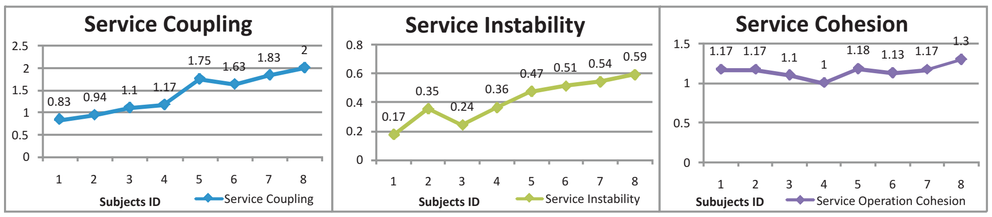

Fig. 8. Metric Results

2) Qualitative Analysis: After concluding the quantitative analysis of the experiment, the qualitative analysis was performed. This analysis was based on the answers of the subjects using in a feedback questionnaire.

Training Analysis: The training was applied to all the subjects who participated of the experimental study and was composed of a set of slides involving topics related to software reuse, software product lines and service-oriented architectures. The training was performed in 24 hours. Two subjects considered the training very good (Id = 6 and 7), four subjects classified it as good (Id = 1, 2, 5 and 8) and two subjects classified the training as regular (Id = 3 and 4). The scale defined was: very good, good, regular, and unsatisfactory.

Usefulness of the SOPLE-DE: The four subjects that used the SOPLE-DE reported that the approach was useful to perform the service-oriented domain design. However, one subject (Id = 3) indicated some improvements in the architectural elements identification activity with the purpose of facilitating the identification of modules and their classification as components or services. All the issues raised during the experiment were considered, and the SOPLE-DE was refined.

Quality of the Documentation and Instruments: One subject (Id = 2) complained about the lack of examples in the documentation of the SOPLE-DE to clarify the different activities of the approach. In this sense, examples were put in the approach documentation to ease understanding. Regarding the instruments of the experiment, two subjects (Id = 2 and 6) requested more information about the requirements of the service-oriented product line on the Travel Reservation domain. Thus, the requirements were carefully detailed considering the questions of the subjects.

Quality of the Architecture Document Produced by the Subjects: We used the following scale to measure the quality of the architecture documents produced by the subjects: very good, good, regular, and unsatisfactory. It was noticed that subjects (Id =1, 2 and 3) produced a well-structured document, since they followed the SOPLE-DE template strictly. In this sense, we classified the architecture documents produced by these subjects as good. Considering the architecture documents produced by the subjects without use the SOPLE-DE, we could detect that subject (Id = 5 and 6) produced good architecture documents as well, however, with different sections, since they do not followed a template. Regarding subjects (Id = 4, 7 and 8), the architecture documents were classified as regular, since they were not organized and well-structured as the documents of the other subjects.

Even with the analysis not being conclusive, the experimental study indicates that the SOPLE-DE allows the architects to design service-oriented product line architectures with a good coupling and stability, and services with cohesive operations. Additionally, the aspects related to the applicability of the SOPLE-DE also returned satisfactory results for subjects with a certain experience with SPL and SOA. Moreover, with the results identified in this experiment, the metric values can be calibrated in a more accurate way for new experiments. However, two aspects should be considered: the repetition of the study in different contexts and new studies based on observation in order to identify more problems and new points for improvements. The full description of the experimental study discussing its definition, planning, operation, analysis and interpretation can be found in [42].

VI. Conclusions and Future Work

This work proposed an approach to design service-oriented product line architectures, which combines SPL and SOA concepts focusing on increasing reuse and flexibility, and supporting customization during the development of service-oriented systems.

The SOPLE-DE approach was based on an extensive review of the available service-oriented processes, their weak and strong points and gaps in the area [42]. It can be seen as a systematic way to design service-oriented product line architectures through a well-defined sequence of activities with clearly defined inputs and outputs.

Additionally, the approach was evaluated in an initial experimental study, which analyzed it both quantitatively and qualitatively. This experimental study presented findings that the SOPLE-DE can be viable to aid software architects to design service-oriented product line architectures with good coupling and instability, and identify services with cohesive operations.

Even it being a relevant contribution for the field, new routes need to be investigated in order to define a more complete process that consider all the software development disciplines, such as requirements, design and implementation, for product lines based on services. Moreover, the SOPLE-DE can be extended to consider a bottom-up strategy that uses legacy systems to identify and design services [3].

In addition, new experiments in different domains are necessary to gather more evidences about the efficacy of the proposed approach. The complete documentation about this work can be found in the following M.Sc. thesis [42].

VII. Acknowledgement

This work was partially supported by the National Institute of Science and Technology for Software Engineering[1], funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

References

[1] C. W. Krueger, "Software reuse," *ACM Computing Surveys*, vol. 24, no. 2, 1992.

[2] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[3] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall, 2005.

[4] P. Istoan, "Software product lines for creating service-oriented applications," Master's thesis, Irisa Rennes Research Institute, 2009.

[5] F. M. Medeiros, E. S. de Almeida, and S. R. L. Meira, "Towards an approach for service-oriented product line architectures," in *SOAPL'09: 3rd Workshop on Service-Oriented Architectures and Software Product Lines*, 2009.

[6] J. Lee, D. Muthig, and M. Naab, "An approach for developing service-oriented product lines," in *SPLC'08: 12th International Software Product Line Conference*. IEEE Computer Society, 2008, pp. 275–284.

[7] G. Booch, *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley, 1995.

[8] N. Boffoli, D. Caivano, D. Castelluccia, F. M. Maggi, and G. Visaggio, "Business process lines to develop service-oriented architectures through the software product lines paradigm," in *SOAPL'08: 2nd Workshop on Service-Oriented Architectures and Software Product Lines*, 2008, pp. 143–147.

[9] S. Günther and T. Berger, "Service-oriented product lines: Towards a development process and feature management model for web services," in *SOAPL'08: 2nd Workshop on Service-Oriented Architectures and Software Product Lines*, 2008, pp. 131–136.

[10] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[11] M. Abu-Matar, "Toward a service-oriented analysis and design methodology for software product lines," *IBM Developer Works*, 2007.

[12] A. Arsanjani, "Service-oriented modeling and architecture," Service-Oriented Architecture and Web services Center of Excellence, IBM, Tech. Rep., 2004.

[13] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, "S3: A service-oriented reference architecture," *IT Professional*, vol. 9, no. 3, pp. 10–17, 2007.

[14] T. Erl, *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.

[15] L. G. Azevedo, F. Santoro, F. Baião, J. Souza, K. Revoredo, V. Pereira, and I. Herlain, "A method for service identification from business process models in a SOA approach," in *10th International Workshops on Business Process Modeling, Development and Support (BPMDS)*, ser. Lecture Notes in Business Information Processing, vol. 29. Springer Berlin Heidelberg, April 2009, pp. 99–112.

[16] M. Havey, *Essential Business Process Modeling*. O'Reilly, 2005.

[17] J. Lee and K. C. Kang, "Feature binding analysis for product line component development," in *PFE'03: 5th International Workshop on Software Product-Family Engineering*, 2003, pp. 250–260.

[18] J. McGovern, S. Tyagi, M. Stevens, and S. Mathew, *Java Web Services Architecture*. Morgan Kaufmann, 2003.

[19] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall, 2004.

[20] N. M. Josuttis, *SOA in Practice*. O'Reilly, 2007.

[21] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practices*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[22] A. Erradi, S. Anand, and N. Kulkarni, "SOAF: An architectural framework for service definition and realization," in *SCC'06: International Conference on Services Computing*. IEEE Computer Society, 2006, pp. 151–158.

[23] S. Segura, D. Benavides, A. Ruiz-Cortés, and P. Trinidad, "A taxonomy of variability in web service flows," in *SOAPL'07: 1st Workshop on Service-Oriented Architectures and Software Product Lines*, 2007.

[24] C. Kästner, S. Apel, and M. Kuhlemann, "Granularity in software product lines," in *30th International Conference on Software Engineering (ICSE)*, 2008, pp. 311–320.

[25] T. Erl, *SOA Design Patterns*. Prentice Hall PTR, 2009.

[26] R. Laddad, *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning Publications, 2003.

[27] C. Gacek and M. Anastasopoules, "Implementing product line variabilities," *SSR'01: Symposium on Software Reusability*, vol. 26, no. 3, pp. 109–117, 2001.

[28] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley, 2004.

[29] M. Razavian and R. Khosravi, "Modeling variability in business process models using uml," in *ITNG'08: 5th International Conference on Information Technology - New Generations*, 2008, pp. 82–87.

[30] P. Kruchten, "Architectural blueprints - the 4+1 view model of software architecture," *IEEE Software*, 1995.

[31] J. J. L. Dias Jr., "A software architecture process for soa-based enterprise applications," Master's thesis, Federal University of Pernambuco, Brazil, 2008.

[32] A. P. J. Jarczyk, P. Löffler, and F. M. Shipman, "Design rationale for software engineering: A survey," in *HICSS'92: 25th Hawaii International Conference on System Sciences*, 1992.

[33] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering: An Introduction*. Springer, 2000.

[34] V. Basili, G. Caldiera, and D. H. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*. Wiley, 1994.

[35] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-oriented designs," in *ASWEC'07: Australian Software Engineering Conference*, 2007.

[36] M. P. Papazoglou and W.-J. V. D. Heuvel, "Service-oriented design and development methodology," *International Journal of Web Engineering and Technology (IJWET)*, vol. 2, no. 4, pp. 412–442, 2006.

[37] H. Hofmeister and G. Wirtz, "Supporting service-oriented design with metrics," in *EDOC'08: 12th Enterprise Distributed Object Computing Conference*, 2008.

[38] P. T. Quynh and H. Q. Thang, "Dynamic coupling metrics for service–oriented software," *IJCSE'09: International Journal of Computer Science and Engineering*, 2009.

[39] L. H. Rosenberg and L. Hyatt, "Applying and interpreting object oriented metrics," in *Proceedings of the Software Technology Conference*, 1998.

[40] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *IEEE Transactions on Software Engineering*, 1986.

[41] B. Kitchenham, L. Pickard, and S. L. Pfleeger, "Case studies for method and tools evaluation," *IEEE Software*, 1995.

[42] F. M. Medeiros, "An approach to design service-oriented product line architectures," Master's thesis, Federal University of Pernambuco, 2010.

[1]INES - http://www.ines.org.br