Service Composition in Service Oriented Product Line

Fatemeh Mafi
Computer Department
Tehran Jonub Azad University
Tehran, Iran
Marjan_mafi_2007@yahoo.com

Shahriar Mafi
Tehran, Iran

Mehran Mohsenzadeh
Computer Department
Olum Tahghighat Azad University
Tehran, Iran

Abstract—The paradigm of Service Oriented Architecture (SOA) and Software Product Line Engineering (SPLE) discipline facilitate the development of families of software-intensive products. Software Product Line practices can be leveraged to support the development of service-oriented applications to promote the reusability of assets throughout the iterative and incremental development of software product families. Such approach enables various service oriented business processes and software products of the same family to be systematically created and integrated. [4] In this way service composition is an important task to produce new product/service with using the core assets. In this paper we try to explain the service composition in service oriented product line.

Keywords-Service Oriented Architecture; Software Product Line; Service Oriented Product Line; Service Composition

I. *INTRODUCTION*

All The term Service-Oriented Product Line is used for service oriented applications that share common parts and vary in a regular and identifiable manner. In this context, high customization and systematic planned reuse are achieved through managed variability as in SPL engineering: core assets and product development.

A service is an abstract resource that represents a capability of performing tasks that represents a coherent functionality from the point of view of provider entities and requester entities. To be used, a service must be realized by a provider agent. This provider agent is the concrete piece of software (or hardware) that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided. Service identification is to select related resources.

In service oriented applications, services are basic elements. So design and implementation of services is necessary steps in developing service oriented product line. In this way service composition is done to reuse existing services instead of implementing the new service.

Service composition can be defined as the process of combining and linking existing services (atomic or composite) to create new working services. It constitutes an essential part of service provisioning, since it leads to novel service offering thus adding value that was not existent in the individual services.

In service composition, the result of combining services is referred to as a composite service. When you use services together to achieve new functionality in a business process, the composition process itself that dictates that the order and interactions between the lower-level services is exposed as this composite service.

The rest of the paper is organized as follows. Section 2 provides background and concepts definition. Approach overview is described in section 3. Section 4 includes related works. Conclusion remarks and future works finally discussed in Section 5.

II. BACKGROUND AND CONCEPTS

A. Service Oriented Product line [11]

First, Service-Oriented Architectures (SOA) and Software Product Lines are two concepts that currently get a lot of attention in research and practice. Both promise to make possible the development of flexible, cost effective software systems and to support high levels of reuse. But at the same time they are quite different from one another.

So an approach in which SOA applications are developed as Software Product Lines (SPLs) was proposed. Thus, the term Service-Oriented Product Line is used for service oriented applications that share common parts and vary in a regular and identifiable manner. In this context, high customization and systematic planned reuse are achieved through managed variability and the use of a two life-cycle model as in SPL engineering: core assets and product development.

B. Service life cycle [8]

- The Service identification is the first major step in the life of a service. It is driven off of the business model, process definition, and semantic information model. This results in the proposal for a new service.
- In service discovery, comparing the requirements with the available services is done and the locations of the matched services are return.

- The purpose of service selection is to select optimal web service for a particular task.
- Service composition aims at providing effective and efficient means for creating, running, adapting, and maintaining services that rely on other services in some way.
    - The composition of web services could be static or dynamic. The differentiation between static and dynamic composition deals with the point of time at which a concrete Web Service is integrated into the specification of a composition. With static composition the concrete services are determined and integrated into the specification at design time. With dynamic composition on the other hand, at design time there is only a specification of the type of service given. The concrete service is then integrated at run-time.
- Service implementation
- Service monitoring

C. *Service Composition[7]*

After the service request is defined, the *service discovery and composition* phase starts. The different services that could be used in a composition are discovered according to the composition algorithm. Service discovery is performed by invoking the interface provided by the service registry, based on information contained in the published service description documents. The information published in the publication phase should be compatible with the information required in the discovery and composition phase, which can be achieved by complying to open standards even if different organizations implement their own publication and discovery mechanisms.

The composition consists of four steps:

- Service providers publish their services at a Web service registry.
- The Service Composition Engine decomposes user requirements into an abstract service and sends a SOAP request to the registry to find the proper services.
- The Web service registry returns a set of concrete services.

The service composition engine sends a SOAP request to the concrete services and binds to them.

D. *Static or dynamic service composition*

Composition of software components/services during system design time (a.k.a., static software composition) is not flexible and agile enough in cases when there are frequent runtime changes of requirements and/or operational circumstances that cannot be anticipated. Static composition is sufficient for constructing applications with well-defined specific requirements that are not likely to change frequently. If a software system has a loosely defined set of operations to carry out or it has to adapt to relatively frequent changes in the environment that might not even be predicted during design time, static composition is too limited. Redesigning the system to accommodate the changes often requires considerable human involvement, which significantly slows down the overall reaction to change. Further, modifying or updating statically composed software usually requires disrupting its operation, which is not suitable for high-availability, mission-critical, and hard real-time systems. As will be discussed later in this paper, many business systems would benefit from greater runtime flexibility, agility, and availability of software systems.[12]

Dynamic composition of software services/components is an important step forward in achieving these goals. It enhances flexibility of software systems since it enables the runtime construction of new services, if they do not already exist, to address a specific problem. A large number of useful services can be created from a set of basic services. Some of these services may not have been designed or even conceived of ahead of time. The services can be assembled based on the demands of the system or its users. The involvement of humans in the composition process is minimized. The users do not need to be interrupted during upgrades or the addition of new functionality into the system. To conclude, dynamic service composition provides an ability to rapidly and autonomously (i.e., with minimal human involvement) adapt even to changes that were not envisioned during design time, while keeping the running software system constantly available to users. Dynamic service composition is a very challenging undertaking and there are a number of issues to take into consideration. It has some elements in common with static service composition but it also has some unique features. One of these features is the crucial nature of time limits. The dynamic service composition process often must complete within some specified, relatively short, time limits or it becomes impractical. Generally, it is an automated process with limited human involvement.[12]

There have been several benefits to dynamic service composition [13]:

- *Greater flexibility* – the customization of software, based on the individual needs of a user, can be made dynamic through the use of dynamic composition without affecting other users on the system.
- *New services can be created at runtime* – the application is no longer restricted to the original set of operations that were specified and envisioned at the design or compile times. The capabilities of the application can be extended at runtime.
- *Users are not interrupted during upgrades of applications* – instead of being brought offline and having all services suspended before upgrading, through the dynamic composition infrastructure, users can continue to interact with

the old services while the composition of new services are taking place. This will provide continuous and seamless upgrading service capabilities to existing applications.

- *Unlimited set of services* – unlike static composition, where the number of services provided to end users is limited and the services are specified at design time, dynamic composition can serve applications or users on an on-demand basis. With dynamic composition, theoretically an unlimited number of new services can be created from a limited set of service components.

E. Software product line process

In traditional software product line, domain requirements engineering defines the required component features which are considered during the component selection. They define the required functionality and quality that a component should offer and a component must match the variability desired for the software product line.

As a result of a component selection process, adaptations of requirements can be required. One reason for such an adaptation is the identification of functionality or quality offered by a component that was not considered by the product line, but which will improve the product line and is thus added as a new feature. Another reason for an adaptation is the fact that it is quite unlikely for a component to match all the desired requirements artifacts and/or to comply fully with the desired variability. Also in this case an adaptation of the requirements or the variability is required.

The output of component selection includes the identified candidate components. Typically, rankings of the components with regard to several criteria are provided. A detailed valuation is conducted only for components that perform well in a preliminary screening activity.

As the selected component has to become an integral part of the reference architecture, domain design imposes architecture constraints to be considered during component selection, such as the architectural styles and patterns that the component must conform to, compatibility constraints, and constraints caused by the process structure of the reference architecture.

Domain design develops the reference architecture, which is the basis for the application architecture. The reference architecture determines common components and interfaces. The application architect binds the architectural variability according to the bindings defined in the application variability model.

The goals of the domain realization are to provide the detailed design and the implementation of reusable software assets, based on the reference architecture. It means that domain realization delivers components and interfaces for reuse by application realization. The reusable software assets are mainly reusable components and interfaces. In addition, domain realization incorporates configuration mechanisms that enable application realization to select variants and build an application with the reusable components and interfaces.

Product management defines the major application features for all applications of the product line. The development of the applications is supported by the commonality and variability of the platform.

Application requirements engineering reuses the common parts and chooses the variant parts that are suitable to match the features defined by product management for the application. Certain features are application specific, i.e. they only apply for a single application. The main output of application requirements engineering is the application requirements specification which is a complete specification of the application. It includes the application variability model.

The main goal of the application design is to produce the application architecture. The application architecture is a specialization of the reference architecture developed in domain design. Application architects bind the variability of the reference architecture and introduce application-specific changes according to the application requirements specification.

Application realization builds the application based on the application architecture. The application architecture determines the structure of the application to be built as well as the rules how to build it, which are contained in the texture. The application architecture also determines the configuration of reused domain components and interfaces that are part of the application as well as their interrelation with application-specific components and interfaces.

III. APPROACH OVERVIEW

[1] is the most related work to our work. In this paper we modeled the development process of service oriented applications with product line approach. In this model because of two separate processes in software product line paradigm (Domain engineering and Application engineering), each phase has two sections .First section is for domain (all products in product line) and second is for the application that is requested. For this reason, for developing a family of SOA application, in application section, we have service identification, discovery, selection, composition and monitoring.

Since that in each service oriented application, there are two kinds of service compositions (static and dynamic), in this model both static and dynamic composition are considered in separate steps. Static composition implies that the compositions is performed at design or compile time. Dynamic service composition, on the other hand, composes an application autonomously when a user queries for an application at runtime. Therefore, dynamic composition involves adapting running applications by changing their functionalities and/or behavior via the addition or removal of service components at run time.

A. Feature Analysis [9]

Feature modeling is the activity of identifying externally visible characteristics of products in a product line and organizing them into a model called feature model. The primary goal of feature modeling is to identify commonalities and differences of products in a product line and represent them in an exploitable form, i.e., a feature model. Once we have a feature model, it is further analyzed through feature analysis. Feature analysis starts with identification of service features. A service feature represents a major functionality of a system and may be added or removed as a service unit. The output of this process is a set of features that need to be bound together into a product to provide a service correctly, a product can be considered as a composition of features.

B. Identification

In this phase, the required features which are considered during the selection and the required functionality and quality that a service should offer are defined.

C. Discovery

In discovery phase, comparing the features with the available services is done and the locations of the matched services are return. If there are not any matched services, we have three alternatives for selection phase:

- Implement the required service(s) ourselves
- Do static service selection and composition at design time
- Do dynamic service selection and composition at run time

So after discovery, if there is a list of possible service candidates, we should select them from reusable service repository according to our architecture and selection criteria. Otherwise, we have three alternatives. If we select the static composition, the output of design section will be composite service and if we want to do dynamic composition, we should plan the composition structure for the run time. The service composition planner should select qualified atomic services and make an appropriate composition plan. Once an optimal service composition plan is determined, it is passed on to the construction phase, where the preparation for composite service execution is performed.

D. Design time with static compositionme

If there is a list of possible service candidates, we should select them from reusable service repository. The purpose of service selection is to select optimal web service for a particular task. As the selected service has to become an integral part of the reference architecture, domain design imposes architecture constraints to be considered during this selection, such as the architectural styles and patterns that the service must conform to, compatibility constraints, and constraints caused by the process structure of the reference architecture.

At this time, if there are not any matched services and we select the static composition, we should decompose the main feature into sub features, chose the best available services that are matched with these sub features, linked together these atomic services and finally compiled and deployed the new (composite) service.

Two main approaches are currently investigated for static service composition. The first approach, referred to as web service *orchestration*, combines available services by adding a central coordinator (the orchestrator) that is responsible for invoking and combining the single sub-activities. The second approach, referred to as web service *choreography*, does not assume the exploitation of a central coordinator but rather defines complex tasks via the definition of the conversation that should be undertaken by each participant.

Static composition is purely manual i.e. firstly, the user problem must be defined and then a manual selection of services according to desired outputs is performed. There are many potential problems, exceptions, and errors that may occur during this process. The challenge lies in dealing with these unexpected issues in the limited time frame that is permitted for a particular composition. Also, it is not possible to precisely predict or test at design time what the exact environmental circumstances of operation will be at composition time and whether the process will be successful. While steps are taken to decrease the chance of a failed composition, it cannot always be avoided.

E. Run time with dynamic service composition

Dynamic service composition is the process of creating new services at runtime from a set of service components. This process includes activities that must take place before the actual composition such as locating and selecting service components that will take part in the composition, and activities that must take place after the composition such as registering the new service with a service registry.

A very important aspect of dynamic service composition is that the new composite service need not be envisioned at design time. This feature, known as unanticipated dynamic composition, provides considerable flexibility for modifying and extending the operation of software systems during runtime. However, it also introduces a number of complications and problems for designing and operating software systems that support dynamic service composition. In this paper, we will describe our experiences with dynamic service composition and discuss how it can be used to improve the agility, flexibility, and availability of business software systems, particularly for e- and m commerce systems.

In dynamic composition, automated tools are used to analyze a user problem, select and assemble web service interfaces so that their composition will solve the user problem. Furthermore, even if the dynamic composition process seems successful, there is the potential for unexpected feature interactions that cannot be easily and rapidly discovered and recovered from. A feature interaction is the way a service component (i.e., a feature) modifies or affects at

runtime the behavior of other service components in a particular composition. The problem is similar to a program that compiles without errors but still fails to execute properly.

Compilation is only one part of the successful execution of a program just as the composition process will not guarantee the composite service will function correctly. When unexpected feature interactions arise despite all measures taken to avoid them, it might be almost impossible for the composition infrastructure to correct the situation. Human (i.e., user) input is needed to determine if the side effects are neutral or service affecting. If the feature interactions cause the composite service to function incorrectly or behave erratically, the composite service can be terminated and never reassembled. However, in many situations it may be appropriate to simply ignore those feature interactions that do not seriously affect the operation of the composite service.

There is also a lack of support for dynamic composition techniques in programming languages and other development tools. The fundamental challenge in composing services at runtime is the design and implementation of an infrastructure that will support the process. Locating components at runtime requires a component library or code repository that is integrated with the software infrastructure that is actually per forming the composition. The infrastructure should also support mechanisms to recover (e.g., rollback) from an unsuccessful composition and to discover and, if possible, recover from unexpected feature interactions. All these and other issues make the dynamic composition process inherently complex. Consequently, cost-benefit analysis must be taken into consideration before applying dynamic service composition techniques to a particular circumstance.

F. Evaluation

The final step contains the evaluation of the proposed service compositions. The evaluation is carried out partly by means of performance tester (as overall speed, reliability or costs of a composition) and partly by means of voting where each participant can input non-rational elements to express his preference for a certain composition. The outputs of this step are ranked list of compositions including the data of the votes and possibly include important comments/remarks of the voting session.

IV. RELATED WORK

After In [1] we propose an approach to modeling service oriented software product line. In this approach we first model service oriented architecture and software product line architecture separately and then we put them together. We present common phases in modeling these two paradigms for developing a service or product. We have described how a family of business process can be modeled and variability can be captured in different development stages to approach the flexible and cost-effective development and deployment of a family software products.

The [2] work presents a contribution to the combination of SOA and SPL concepts. In particular, how these concepts can be used together to achieve desired benefits such as improved reuse, decreased development costs and time to market, and production of flexible applications customized to specific customers or market segment needs. In order to achieve these goals, they presented an approach for service-oriented product line architectures that introduces the concepts of managed variability into service oriented world and uses a two life-cycle model as in SPL engineering, however, only core assets development is considered in this work. These concepts were introduced in order to provide support for high customization and systematic planned reuse during service-oriented development. In this context, services are developed to be reused in specific contexts and service-oriented applications can be developed rapidly and customized according to specific customer requirements. They also present a case study on the conference management domain clarifying and explaining the activities of the approach.

In [3] they have two main conclusions of this work. First, the modification of the classical product line lifecycle where they include a specific process for composing web services. Second, the definition of specific variation points in the architecture for specifying different alternatives for composing such services. This becomes a key aspect to facilitate the evolution of these systems and for customizing web services during the design and implementation phases. Another key point related to the evolution of systems is the ability of the variation points to support dynamic changes depending of the context in which the system runs. In addition to this, more variation points can be defined (e.g.: user preferences, type of network protocol, etc.) to improve the product derivation process and to provide other alternatives that can be selected at different binding times. Finally, they argue that agile models such as lightweight product lines constitute a good approach for those systems that have strong time to market requirements. This definitely holds for Web systems. In this way and compared to other approaches, they improve the evolution of service oriented systems through the use of specific variability information in the service description.

In [4] they presented a novel methodology for development of business process family by exploiting SPLE and SOA. In their methodology they introduce variability modeling derived from different levels of SOA development to support a high level of reuse and to facilitate the development of variant rich business process model. They have described how a family of business process can be modeled and variability can be captured in different development stages to approach the flexible and cost-effective development and deployment of a family software products. Furthermore, with the comparison of current approaches for model-driven development of semantically rich business processes and supporting SOAs they described how they improve the state of the art in model-driven development of families of SOAs. Furthermore, they have also made the initial steps toward realization of supporting tools for our vision.

In [5] they defined a web service-based web application (WSBWA) as a collection of web services or reusable proven software parts that can be discovered and invoked using

standard Internet protocols. In particular, they used the lightweight product line model proposed in Capilla and Yasemin Topaloglu (2005) and extended it to support a domain-specific visual language and environment at the application engineering level thus allowing them to perform agile calibration and customization of WSBWAs. At the domain engineering level, their approach includes the identification of commonalities and variability of WSs in WSBWAs domain as well as the construction of a Model View Workflow (MVWf)-based framework that is instantiated to identify a specific product or WSBWA. At the application level, their approach supports agile methods, in particular by relying on a domain-specific visual language and environment with innovative extraction capabilities of WSs directly from web sites that are ‘‘imported” into our visual environment. This speeds up the development process by facilitating the composition and customization (or calibration) of a product or WSBWA for a specific customer.

V. CONCLUSION AND FUTURE WORK

In this paper with considering our proposed model in [1], we introduce an approach for explain the service composition process in one of the development phases of service oriented software product line (design or implementation). In this way we separated static with dynamic service selection and composition for developing a family of SOA application.

As future work, we will try to use a model driven approach for modeling the dynamic service composition in families of SOA applications.

REFERENCES

[1] Fatemeh Mafi, Shahriar Mafi, Ma Seyyedi, An approach to modeling service oriented product line. Tehran Jonub University.

[2] Fl´avio Mota Medeiros, Eduardo Santana de Almeida,Silvio Romero de Lemos Meira, Towards an Approach for Service-Oriented Product Line Architectures.

[3] Rafael Capilla1, N. Yasemin Topaloglu .Product Lines for Supporting the Composition and Evolution of Service Oriented Applications

[4] Mohsen Asadi, Bardia Mohabbati , Nima Kaviani, Dragan Gašević, Marko Bošković, Marek Hatala . Model-Driven Development of Families of Service-Oriented Architectures.

[5] Marcel Karam, Sergiu Dascalu, Haidar Safa, Rami Santina, Zeina Koteich .A product-line architecture for web service-based visual composition of web applications. The Journal of Systems and Software (2008),855–867.

[6] Klaus Pohl, Günter Bckle, Frank van der Linden. Software Product Line Engineering, Foundations, Principles, and Techniques. Springer-Verlag Berlin Heidelberg 2005.

[7] Mike Rosen, Boris Lublinsky, Kevin T. Smith, Marc J. Balcer. Applied SOA Service-Oriented Architecture and Design Strategies. Wiley, 2008

[8] Thomas Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall , 2005

[9] Ali Gondal, Michael Poppleton, Colin Snook. Feature Composition Towards product lines of Event-B models

[10] ANR PERSO. State of the art: Models and Algorithms for Service Composition. 2009.

[11] Andreas Helferich, Georg Herzwurm, and Stefan Jesse. Software Product Lines and Service-oriented Architecture: A Systematic Comparison of Two Concepts, Universidad Stuttgart, Stuttgart, Germany, sei,2007.

[12] Antonio Bucchiarone , Stefania Gnesi . A Survey on Services Composition Languages and Models.

[13] Mohamad Eid, Atif Alamri , Abdulmotaleb El Saddik . A reference model for dynamic web service composition systems. 2008Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.