

---

# Development of Dynamic Composed Services Based on the Context

Francisco Javier Nieto, European Software Institute, Parque Tecnológico de Zamudio, E-48170 Zamudio-Bizkaia, Spain, [francisco.nieto@esi.es](mailto:francisco.nieto@esi.es)

Leire Bastida, European Software Institute, Parque Tecnológico de Zamudio, E-48170 Zamudio-Bizkaia, Spain, [leire.bastida@esi.es](mailto:leire.bastida@esi.es)

Marisa Escalante, European Software Institute, Parque Tecnológico de Zamudio, E-48170 Zamudio-Bizkaia, Spain, [marisa.escalante@esi.es](mailto:marisa.escalante@esi.es)

Alayn Gortazar, European Software Institute, Parque Tecnológico de Zamudio, E-48170 Zamudio-Bizkaia, Spain, [alayn.gortazar@esi.es](mailto:alayn.gortazar@esi.es)

**Keywords:** Dynamic composition, Context-aware, Services, Methodology

## 1 Introduction

One organisation interested in working in the field of services composition should answer the following question: which are the main steps in order to create a composed service, either statically or dynamically? In order to answer this question is important to have in mind the framework process [1] in which the composition development is included. This framework process has three main process areas:

- Service development/engineering process area focussing on best practices in the context of developing ‘atomic services’<sup>1</sup>. The primary purpose of this process area is to develop, test and deliver atomic services.
- Service Acquisition/provisioning process area addresses processes and activities related to the service marketplace and the role of service broker.
- Service centric system engineering area. This is the most important area for the purpose of this paper. Service centric system engineering process area includes all the functionality related to the analysis, design, development, deployment and execution of a software system based on services. The component parts involved in this process are similar to those found in

---

<sup>1</sup> Atomic Service is a single service not formed by other services.

traditional software engineering, but concentrated on the development of service-oriented architectures for the system being developed.

This paper is structured as follows. First, it provides a brief introduction about existing approaches for dynamic composition. Second, a proposal for a process to develop static composition is explained. Finally, the paper contains a detailed description of the process for developing composed services based on variability, as well as an explanation and an example of how to model the variability.

## 2 Approaches for Development of Dynamic Composed Services

In static service compositions, the activities, services and workflows are statically defined at design time. This means, after analysing the requirements, the composition with fixed elements is developed. It is not possible to change the integrated services during the execution of the composition, even when there are errors.

In a dynamic service composition, some approaches have been identified to allow certain dynamicity in the service composition, enabling the business process<sup>2</sup> to adapt to the context and manage the recovery actions when errors are detected. The objectives are to achieve high flexible service compositions, and to improve the reliability of the composition.

The different approaches identified can be used together to give the composition the maximum flexibility, because they have impact at different levels of the service compositions. It is necessary to define specific tasks and processes to fulfil the requirements of the approaches at design time as well as at runtime. These approaches are the following ones:

- **Re-Binding.** This approach enables to replace a service with other, which will be similar and will cover the same functionality. At design time, some realisation alternatives for the overall functionality of the composition are identified. If the default service has to be replaced due to an error or a QoS decrease, the re-binding process will select a new one from the alternatives list, which could be enlarged by discovering new services at runtime, and will modify the composition by binding the new selected service.
- **Variability Points.** In this approach, the parts of the composition where there will be different possible alternatives in order to execute an abstract activity<sup>3</sup>, are identified. These alternatives can be identified at design time, but it is not possible to know which one will be executed at runtime. Depending on the context when the composition is requested and the conditions are defined for each variability point, one option will be chosen and executed. The variances may be defined as different available features or as service realisation alternatives at runtime.

---

<sup>2</sup> Business process is understood as the composition flow including additional aspects such as transactions, security issues.

<sup>3</sup> Abstract activity is an activity that not necessarily has a concrete implementation.

- **Re-Planning.** The approach is used when a service participating in the composition fails, and there is no possibility to replace it with other atomic service. In this case, the composition needs to be modified in such a way that the lost functionality can be recovered. This functionality may be split (so it will be covered by various services) or may be regrouped with other part of the composition (a service that is working properly to perform part of the functionality of the composition could be used to perform also the lost functionality).

### 3 Static Composition Framework

A static service composition is a composed service workflow, defined manually and a priori at design time. As consequence, after the service is executed and running, no changes are possible. In this way it is possible to compose really complex services, as interactions between used services are well known and controlled. But this static condition means a lack of adaptability to different users or environments, as it requires knowing very well the consumed services and their interactions, so if adaptability or dynamic compositions are mandatory factors, some actions should be taken during the static design to allow a dynamic service composition at run time. [2]

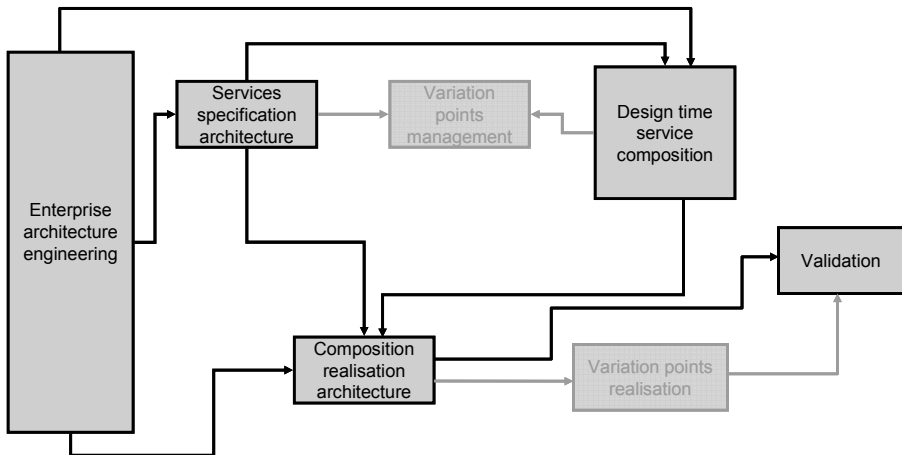


Fig. 1. Static Composition Framework

In Figure 1, the static service composition methodology is split into five processes: Enterprise Architecture Engineering, Service Specification Architecture, Design Time Service Composition, Composition Realisation Architecture, and Validation process. These processes are composed by several tasks, which are explained more in depth in [2]; including for each step a figure showing these main tasks, using the SPEM notation [6].

### 3.1 Enterprise Architecture Engineering Process

The purpose of this process is to identify, define and maintain domain or organisation wide architecture models, and assess their corresponding benefits and drawbacks to fulfil system requirements.

To achieve this objective, once the system requirements and the available candidate services are provided by a previous service discovery mechanism, it is important to gather information about previous projects to analyse similar solutions.

Different architectures that may fit the needed composition might be defined or extracted from those solutions. Once some candidate architectures are selected, the advantages and disadvantages of each one should be analysed to facilitate the decision about which architecture will be used for the composition.

The final result of this process will be a list of the available Architectural Options and Patterns, ready to be used as a skeleton for the service composition structure. This information may be enhanced with descriptions of the different alternatives, instructions on how to use them, results of a feasibility analysis and so on. To ease the future choice of an appropriate architecture, some Architectural Policies will also be obtained.

### 3.2 Service Specification Architecture Process

After obtaining the suitable architecture for the composition, the services that fulfil this architecture should be found or, at least, defined. This process defines how to specify the services that are needed to cover all the functionalities of the system, specifying abstract service specifications for the composition that will only describe their functionalities without any specific implementation.

These service specifications will be used afterwards to find concrete services that can fulfil the requirements. If some services which are necessary to fulfil the requirements can not be found, new services might be developed.

As a result of this process, a Service Specification Model will be obtained; specifying the interface of each identified abstract service, as well as its description and requirements. It could also be possible to specify some mappings with service implementations that cover the functionalities of the identified abstract services.

### 3.3 Design Time Service Composition Process

In this process, a service composition will be specified taking as inputs the Service Specification Model and the previously described Architectural Policies.

For this purpose, once the abstract services have been specified, the architecture that best fulfils the composition requirements will be selected, taking into consideration the business strategic perspectives. Then, the abstract services that fulfil the architectural requirements must be organized, obtaining an abstract service composition.

Finally, it will be necessary to describe how to fulfil other composition relevant aspects, such as error handling, security or transactions management.

At the end of this process, a document with the modelled abstract representation of the composition will be obtained. This document is called Service Composition Specification, and it also includes more information about the resulting composition, such as workflow, transactions, security issues, etc...

### **3.4 Composition Realisation Architecture Process**

After obtaining the Service Composition Specification, it is necessary to select the final services that fulfil the abstract composition. In this process, candidate services and possible alternatives will be chosen using previously described service specifications. These services can be discovered or developed, depending on the possible candidates and the system requirements, and they will be mapped with the abstract composition specification. The goal is to obtain the final business process that will fulfil the functionality expected from the system during its execution.

The main result of this process will be an Executable Script written in a concrete implementation language for the composition, i.e. WS-BPEL [5], which will include all the issues described in the Service Composition Specification. Furthermore, a Service Deployment Architecture document will also be produced. This document will explain how services will interact with each other and which Quality of Service (QoS) levels will be covered.

### **3.5 Validation Process**

As a final step, the composed service will be validated to assure that selected services provide the appropriate functionalities with the optimal QoS. For this purpose, and to guarantee that no violations or deadlocks are possible, the workflow should be iterated. To assure that the obtained results are always correct, and that the system maintains the required stability, the system should be overloaded during this process.

Finally, the updated versions of the documents produced on the previous process will be obtained; i.e. a Verified Executable Script and a Verified Service Deployment Architecture.

## **4 Dynamic Composition Framework based on Variability**

Software variability has become an important topic in software architectures and product family approaches [3] to derive different product configurations through the use of variation points to describe those parts of a software system that may vary. This approach has been applied to the development of service compositions since different configurations could be possible in a service composition depending on different factors. The main idea of variability for the business process workflow is to introduce one more abstraction layer in the business process. In this way, developers include tasks at design time to define explicitly the variability that can be concreted in later phases and implemented in very different ways.

The application of variability needs to be supported during the design of a service composition, because introducing variability once the business process has

been developed and implemented is very difficult. Maintaining the coherence of the composition and the fulfilment of the requirements is not a trivial matter. This happens because adding variability to the composition might affect many tasks as well as global policies like those for security, transactions and QoS.

To add this abstraction level to the business process, we have proposed to carry out two extra processes during the development of service composition (Variation points management and Variation points realisation), which will help developers to select the composition parts in which to introduce dynamicity using variability, as well as how to implement and define possible alternatives for each variation point.

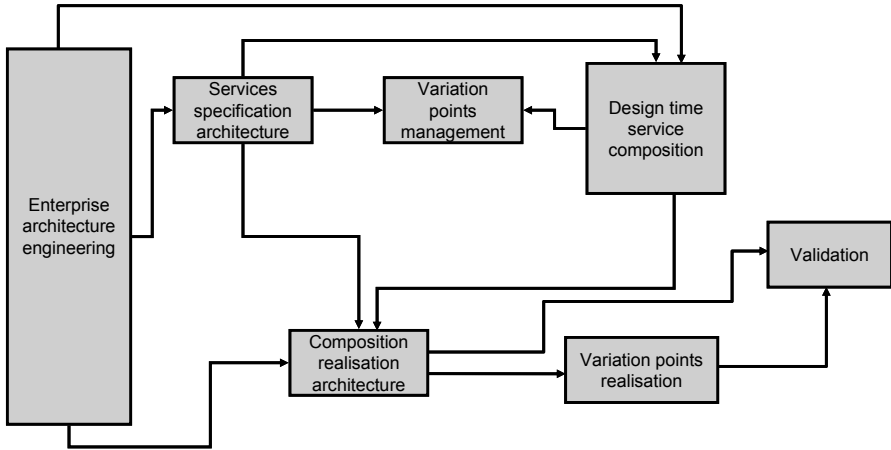


Fig. 2. Dynamic Composition Framework based on Variability

These processes are expected to be used in any development process, improving any methodology in such a way that can guide developers to introduce some dynamic parts in their compositions at design time. In the figure above, we show how these processes can be fitted in the static composition framework.

#### 4.1 Variation Points Management Process

As explained before, some parts of the composition might change at run-time. The purpose of this process is to describe at design time which parts of the composition might change at runtime and define a set of requirements to perform this variability and maintain the coherence with the business process goal.

At this point, a Service Composition Specification has been modelled in the previous step, so the basic abstract business process is defined. This specification has to be analysed in order to get which parts of the composition might have a different behaviour at runtime depending on context information. The selected architecture, the composition requirements and the abstract services descriptions identified are some of the composition aspects that will determine where the variable parts might be.

Those variable parts will be defined as variation points and the next step will consist on describing each one. This is, defining the requirements and the criteria

(functional, non functional, dependencies...) to be fulfilled in order to guarantee that the new composition will be coherent and compatible with the initial requirements and goal. It is important to put special attention in aspects like transactions, security and QoS policies which are sensitive to changes.

The final result of this process will be an enhanced and improved Service Composition Specification, which will now also include variation point specifications to be instantiated at runtime.

## 4.2 Variation Points Realisation Process

Until this process, there is not enough information to instantiate variable points. This process guides developers through the process of gathering that information indicating which services are candidates for each variation point and how those services fulfil the requirements of the variation point, as well as defining some rules to indicate when to use one service or another.

Before starting this process, some candidate services should be discovered, and the composition specification should be completed as a service deployment architecture. With this information, composition developers will define the candidate services for each variation point, mapping the requirements of the variation points and the abstract service descriptions. A ranking may be created to facilitate the selection of services and their alternatives.

At the same time, the variation points policies have to be described. To perform this activity, developers will define a set of rules or requirements to guide the execution engine in the services selection task in such a way that can fit better the executed composition with the user requirements and needs. Defining which context information is needed is critical, so the execution engine will know which data must be gathered.

From this process, an enhanced Service Deployment Architecture and an enhanced Execution Script will be obtained. The Execution Script will contain specific language elements to indicate variabilities and policies to manage them, so the execution engine can interpret them correctly. In this way, the execution engine will be ready to instantiate those variation points at runtime before starting the execution of the composition.

## 5 Modelling Dynamic Composition based on Variability

To model the variability identified in a dynamic composition, it is necessary to define the information required to describe variation points in composite services and to configure them at different points in the lifecycle (e.g. design time or runtime). Then, this information model should be attached to a composition language, such as WS-BPEL [5]. However, the details about how to attach these variability concepts to a composition language are still under our research.

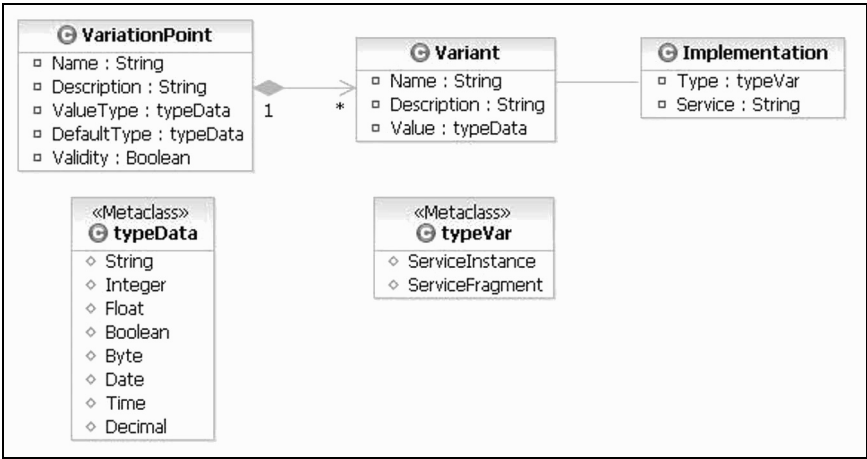
This section provides a definition of the variability concepts used in composition language and also a conceptual model that relates all these concepts. The figure 3 gathers all these variability concepts and their relationships.

A **variation point** (VP) identifies a point in a composition context which varies according to the value of a decision taken using context information. Each variation is defined with a set of information such as:

- A *Name* identifying the variation point with a unique identifier within the composition.
- A *Description* describing the variation point, giving the necessary plain text information to support the process of decision making.
- A *ValueType* specifying the data type of the variation value.
- A *DefaultValue* a pre-set value, indicating the value automatically applied to the variation point if the decision is not taken explicitly.
- The *Validity*, indicating if the variation point has to be taken during composition execution. This attribute is true by default.

A **variant** is one of the different alternatives in a variation point. Only one variant is active in the system at any given moment. Each variant is defined with a set of information such as:

- A *Name* identifying the variant with a unique identifier within its owner.
- A *Description* describing the variant, giving the necessary text plain information about the alternatives gathered in that variant.
- A *Value* indicating the value associated to that variant. This means, the variant will be the alternative chosen in case that value coincides with the decision taken.
- An *Implementation* indicating the candidate service attached to the variant. This candidate service could be a single concrete service to an abstract service composition or several abstract/concrete services to replace a part of the composition.



**Fig. 3.** Variability Conceptual Model



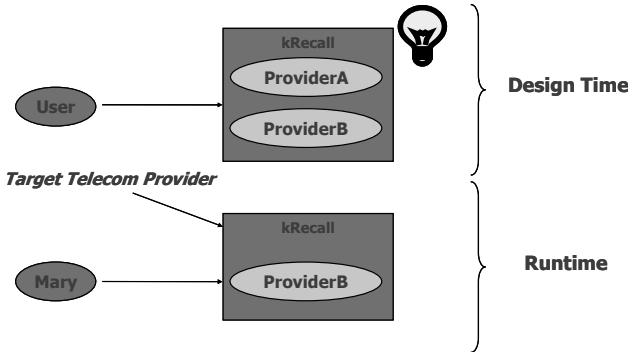
## Example of Usage

This section provides an example of the usage of variability concepts explained in the previous section. The example shows how a specific configuration impacts on the final composition. Keep in mind that this example only manages simple variability and disregard dependencies that could appear between variation points.

Suppose that we want to use an IP voice service called *kRecall* that, after checking the user identity through an external service denominated *kLogin*, provides different alternative ways to phone, using different telecommunication providers. The *kRecall* service provides two alternative telecommunication providers, ProviderA and ProviderB. If a mobile phone user wants to phone to a ProviderB user, the best choice would be to use the same telecom provider as the target mobile user, this means, Provider B service is preferred.

To model this in the composition, we have to add a variation point for *kRecall* depending on the telecom provider of the target mobile phone and define the associated variants. The best variant is chosen on the basis of the target mobile phone. This decision is taken based on the value given to the variation point associated at runtime.

For example, suppose that Mary uses the *kRecall* service to get her through John and ProviderB is the telecommunication provider that supplies phone services to John. Then, the composition flow of *kRecall* service, after configuring the variation points identified, would be denoted by the following figure:



**Fig. 4.** Example of Variability Choices based on Telecommunication Providers

If the target phone provider is 'ProviderA', the first alternative or variant will be chosen. If the target phone provider is 'ProviderB', then the second variant will be chosen.

Now, the *kRecall* service is configured using the user's context. Depending on a specific configuration (in this case, the type of telecom provider for target phone), the variation point is instantiated and final the composition flow is generated and specifically adapted to the context.

## 6 Conclusions and Future Work

Existing services composition approaches focus mainly on static composition based on service description and orchestration standards. Adding variability to a business process is a good solution to get some level of dynamicity.

The static framework presented in the paper can be used as a guide to develop any service composition. Following all the steps, a robust and complete composition could be obtained with a good quality level.

But there are many domains where dynamic compositions offer many advantages, so the paper shows how to apply a product family engineering approach as a feasible solution for developing composed services, as they share some commonalities regarding their variable behaviour in some circumstances. The advantage of this approach is that provides the appropriate functionality with the optimal quality depending on the context in the execution time.

This approach can be considered as a dynamic re-composition since there is already a service composition defined. To perform a real dynamic composition, we need to create a service composition from scratch and in real time. This kind of approach is known by on-the-fly composition, which needs only a few input parameters provided by the user, like a composition goal, to develop low-complex compositions. This approach will be researched in the context of future European projects.

The variability conceptual model included in section 5 describes a first approach. This model should be also improved to express dependencies between variation points, in order to get a more complete approach using variances. Besides, the issue about how to attach these variability concepts to a composition language is still under development.

As well as some tasks can be variable, the number of alternatives may not be static, in such a way that new alternatives might be found at runtime. This issue has not been taken in account, but it could be tackled in later researches allowing adding alternatives automatically as they are detected when new services are discovered by the system.

## 7 References

- [1] SeCSE Project: SeCSE Methodology. (2005), <http://secse.eng.it>
- [2] SeCSE Project: Report on methodological approach to designing service compositions. (2005), <http://secse.eng.it>
- [3] Pohl, K., Böckle G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer-Verlag, Berlin Heidelberg (2005)
- [4] Lawrence, W.: Composite Applications. Best practice report in CBDI Journal, (September 2004)
- [5] OASIS Technical Committee: OASIS Web Services Business Process Execution Language. (May 2003). [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel)
- [6] OASIS OMG: Software Process Engineering Metamodel, version 1.1, (January 2005). <http://www.omg.org/technology/documents/formal/spem.htm>

Enterprise Interoperability

New Challenges and Approaches

Doumeingts, G.; Müller, J.; Morel, G.; Vallespir, B. (Eds.)

2007, XVI, 587 p., Hardcover

ISBN: 978-1-84628-713-8