Grant Agreement N° 215483

Title:	*Algorithms and techniques for splitting and merging service compositions*
Authors:	*UniHH, UPM, USTUTT*
Editor:	*Olha Danylevych (USTUTT)*
Reviewers:	*Dragan Ivanović (UPM)*
	Dimka Karastoyanova (USTUTT)
	Raman Kazhamiakin (FBK)
Identifier:	*CD-JRA-2.2.3*
Type:	*Deliverable*
Version:	*1*
Date:	*30 November 2009*
Status:	*Final*
Class:	*External*

Management Summary

This deliverable investigates techniques for split and merge of service compositions, with the emphasis on approaches that aim at optimizing costs and performance of service compositions in out- and insourcing scenarios. The deliverable provides classification criteria applicable to any split technique in the state of the art, and some novel techniques that are classified accordingly. Moreover, the deliverable presents exploratory work on merge of service compositions.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 215483 (S-Cube).

Members of the S-Cube consortium:

University of Duisburg-Essen (Coordinator)	Germany
Tilburg University	Netherlands
City University London	U.K.
Consiglio Nazionale delle Ricerche	Italy
Center for Scientific and Technological Research	Italy
The French National Institute for Research in Computer Science and Control	France
Lero - The Irish Software Engineering Research Centre	Ireland
Politecnico di Milano	Italy
MTA SZTAKI – Computer and Automation Research Institute	Hungary
Vienna University of Technology	Austria
Université Claude Bernard Lyon	France
University of Crete	Greece
Universidad Politécnica de Madrid	Spain
University of Stuttgart	Germany
University of Hamburg	Germany
Vrije Universiteit Amsterdam	Netherlands

Published S-Cube documents

All public S-Cube deliverables are available from the S-Cube Web Portal at the following URL:

http://www.s-cube-network.eu/results/deliverables/

The S-Cube Deliverable Series

Vision and Objectives of S-Cube

The Software Services and Systems Network (S-Cube) will establish a unified, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution, helping shape the software-service based Internet which is the backbone of our future interactive society.

By integrating diverse research communities, S-Cube intends to achieve world-wide scientific excellence in a field that is critical for European competitiveness. S-Cube will accomplish its aims by meeting the following objectives:

- Re-aligning, re-shaping and integrating research agendas of key European players from diverse research areas and by synthesizing and integrating diversified knowledge, thereby establishing a long-lasting foundation for steering research and for achieving innovation at the highest level.
- Inaugurating a Europe-wide common program of education and training for researchers and industry thereby creating a common culture that will have a profound impact on the future of the field.
- Establishing a pro-active mobility plan to enable cross-fertilisation and thereby fostering the integration of research communities and the establishment of a common software services research culture.
- Establishing trust relationships with industry via European Technology Platforms (specifically NESSI) to achieve a catalytic effect in shaping European research, strengthening industrial competitiveness and addressing main societal challenges.
- Defining a broader research vision and perspective that will shape the software-service based Internet of the future and will accelerate economic growth and improve the living conditions of European citizens.

S-Cube will produce an integrated research community of international reputation and acclaim that will help define the future shape of the field of software services which is of critical for European competitiveness. S-Cube will provide service engineering methodologies which facilitate the development, deployment and adjustment of sophisticated hybrid service-based systems that cannot be addressed with today's limited software engineering approaches. S-Cube will further introduce an advanced training program for researchers and practitioners. Finally, S-Cube intends to bring strategic added value to European industry by using industry best-practice models and by implementing research results into pilot business cases and prototype systems.

S-Cube materials are available from URL: http://www.s-cube-network.eu/

Contents

1 Introduction

Service compositions are one of the main components of Service Based Applications (SBAs) [15]. Service compositions allow the creation of novel, value-added services from already existing ones [25], to which applies the anti-relativistic precept that "the whole is more than the sum of its part".

Change is a constant in the practice of Service Oriented Architecture (SOA). In an enterprise world dominated by the business-IT alignment problem, SBAs must efficiently change to cope with continuously evolving requirements, environments, business goals and strategies. Changeability, i.e. the capability of facilitating the implementation of changes, is one key quality aspect of SBAs [5]. The changeability of SBAs builds directly on top of the changeability of the underpinning service compositions. In turn, the changeability of service compositions depend on which changes can be applied to them, and on how these changes simplify the alignment of the service composition and the surrounding business environment.

In the current business landscape, the in- and outsourcing of services and business processes from one organization to another are common practice. The support function of IT with respect to business demands facilities to allow the smooth, seamless realization of in- and outsourcings. In terms of SOA, SBAs must assume in- and outsourcings as first-class change scenarios, and provide efficient mechanisms to deal with them. SBAs must be able to change and adapt to changes in the availability, ownership and performance of services that are common place results of insourcing and outsourcing of services, business processes, etc.

SBAs are built on services that are "glued" together through service compositions to realize business processes [21]. The change of SBAs with respect to the in- and outsourcing can be realized through changes to the underpinning service compositions. On one hand, the organization that performs the insourcing of services and business processes needs to *merge* its own service compositions with the ones that are acquired from the other organization. On the other hand, during an outsourcing the organization that outsources must identify the services and business processes that are externalized, and modify its service compositions accordingly. This is achieved through the *split* of the existing service compositions so that the outsourcing organization keeps only the compositions of the services and business processes it retains, and the rest can be given to the insourcing organization as a reference model of what the outsourcing one requires.

The in- and outsourcing are performed for a number of motivations that range from business, e.g. to secure the production of services that are vital to the business or to reduce their cost, to technical ones, for example to optimize the performance of the IT operations. This deliverable considers in- and outsourcing performed for technical reasons. In particular, it focuses on how to optimize the performance of service compositions through their split and merge during in- and outsourcing scenarios.

1.1 Contributions of the Deliverable

The description of this deliverable treats split and merge techniques to optimize service compositions. After having examined the state of the art, the partners have found that the term "split" is rarely used, and never with a well-defined meaning. The related work adopts instead terms like "fragmentation", "decomposition" and "partitioning" – "fragmentation" being the one adopted with the highest frequency and with the most general meaning. This has had two effects on this deliverable: firstly, the term "split" has been supplanted with the more general term "fragmentation". Secondly, the partners have identified the need of a foundation for sound and comprehensive definitions of what fragmentation is and what it entails. This has ultimately resulted in a set of classification criteria that are later introduced. Moreover, the partners have developed a number of approaches for fragmentation of service compositions in different scenarios.

Specifically, the contributions of this deliverable are grouped in:

- Classification criteria for Fragmentation techniques: The classification criteria for fragmentation

have resulted into the paper "The Kaleidoscope of Fragmentation of Service Compositions" summarized in Section 2 and attached in Section A.1.

- Fragmentation techniques:
  - "Optimal Stratification of Transactions" (Section 3.1, attached in Section A.2) treats how to fragment a service compositions with transactional properties in order to optimize its costs and non-functional quality aspects
  - "Towards Identification of Fragmentation Opportunities for Service Workflows with Sharing-Based Independence" (resumed in Section 3.2, attached in Section A.3) presents a fragment identification that can be applied for different purposes such as reuse, optimization of resource utilization and optimization of the non-functional properties of a service orchestration.
  - "Towards Runtime Migration of WS-BPEL Processes" (Section 3.3, attached in Section A.4) proposes an approach for fragmentation of the process instances that enables the decentralized execution of the process instance by several parties.
  - "Executing Parallel Tasks in Distributed Mobile Processes" (Section 3.4, attached in Section A.5) presents an approach that supports the distributed parallel process execution with multiple mobile process participants.

All the papers presenting fragmentation techniques are classified in their respective summaries according to the appropriate set of classification criteria.

It should be noted that the focus of this deliverable is on fragmentation. This is because the state of the art of merge of service compositions is still in its infancy. Lacking a solid foundation of related work, the partners have performed exploratory research that has resulted in the following contribution:

- "From Requirements to Executable Processes – A Literature Study" (Section 3.5, attached in Section A.6) considers how the merge of use case scenarios or business process fragments can bridge the gap between Requirement Engineering (RE) and Business Process Management (BPM).

The merge of service compositions will be further investigated in the upcoming deliverables.

1.2 Relation to the Research Challenges of S-Cube

The S-Cube Vision White Paper [20] lists the following challenges related to this deliverable:

Mechanisms for fragmentation: Reasons and criteria for the fragmentation of service compositions need to be identified (e.g. outsourcing, resource workload distribution and optimization, organizational (re-distribution) and relevant mechanisms for process fragmentation need to be developed. This topic requires engineering and design principles and methodologies (SED plane) and is used for decomposition of complex service networks at the BPM layer.

Reusable process fragments: Mechanisms for creating parameterisable fragments from repeatable service compositions (and business sub-processes) are required, which are based on best practices facilitating application and systems delivery and development. Such reusable customized and/or differentiated service patterns can be offered by service providers to their customers. This topic requires engineering and design principles and methodologies (SED plane).

Coordination of fragments: There is a lack of coordination protocols to maintain the original composition logic of fragmented processes. Depending on the fragmentation reasons and criteria, as well as on the fragmentation mechanisms, the coordination protocols may be different. These protocols can be used for coordination of business transactions at the BPM layer.

Cross-layer adaptation support: Fragmentation may lead to conflicts across the layers during an adaptation of a service-based application. Thus, a deep understanding of the fragmentation techniques and strategies is needed to support devising the cross-layer adaptation strategies.

The research included in this deliverable contributes towards the fulfillment of the first, third and fourth research challenges. The fragmentation approaches proposed by the papers attached to this deliverable investigate reasons, criteria and mechanisms to perform fragmentation, hence contributing towards the **Mechanisms for fragmentation** challenge. The paper "Optimal Stratification of Transactions" contributes towards the **Coordination of fragments** challenge by illustrating a mechanism to optimize the tradeoff between granularity of fragments and Quality of Service (QoS) in service compositions. Finally, the classification criteria for fragmentation presented in the paper "The Kaleidoscope of Fragmentation of Service Compositions" are the foundation for the "deep understanding of the fragmentation techniques and strategies" described by the challenge **Cross-layer adaptation support**.

1.3 Relation to other S-Cube Work Packages and Deliverables

This deliverable is related to the following S-Cube deliverables (and by extension the respective work packages):

CD-JRA-1.2.2 Taxonomy of Adaptation Principles and Mechanisms: the fragmentation and merge techniques investigated in this deliverable can be used as Realization Mechanisms for Adaptation of SBAs, i.e. techniques to perform adaptation of an SBA.

CD-JRA-1.3.2 Quality Reference Model for SBA: the techniques for fragmentation and merge of service compositions presented investigated in the current deliverable is related to some of the quality attributes of SBAs, namely Changeability and Performance. SBAs build on top of service compositions. The capability of fragmenting and merging the service compositions allows changing the SBAs. Moreover, the optimization of QoS over the fragmentation and merging of the service compositions positively affects the QoS of SBAs built on top of them.

Moreover the results of this deliverable will be a valuable contribution to the work on Agile Service Networks (ASNs) and Business Transactionss (BTs) in the workpackage JRA 2.1. In the scope of the workpackage JRA 2.2, this deliverable is related to the previous two, namely PO-JRA-2.2.1 and CD-JRA-2.2.2. The present deliverable will be used as basis for the upcoming deliverables in the workpackage.

1.4 Structure of the Deliverable

The deliverable is structured as follows. Section 2 investigates the classification criteria for fragmentation techniques applied to service compositions. Section 3 summarizes the contributions to the techniques for fragmentation and merge of service compositions developed by the S-Cube partners in the scope of this deliverable. Section 4 concludes the deliverable presenting the final remarks.

2 Fragmentation and Merge of Service Compositions

The state of the art of SOA does not include precise, consistent definitions of what are split and merge of service compositions. Lacking definitions from within the SOA community, we have searched the related field of Modeling, in which split and merge have been precisely defined.

We have adopted as baseline for our work the framework for split and merge of generic models proposed by Brunet et al. in [1]. Let M and R be respectively the domains of models and relations defined over them. The **merge** operator is defined as:

$$\textbf{merge} : M \times M \times R \rightarrow M \tag{1}$$

The operator **merge** merges two models in the domain M into a third according to a relation in the domain R. The models are composed as specified by the relation, for example in parallel, sequentially, or any tradeoff in between.

Similarly, the **split** operator is defined as follows:

$$\textbf{split} : M \rightarrow M \times M \times R \tag{2}$$

A model is split in two other models that are related by a relation.

The merge framework of [1] is generic, and it must be instantiated for the particular modeling language, e.g. Unified Modeling Language (UML) models or Entity Relationship (ER) diagrams. Ideally, the operators **merge** and **split** are inverse, i.e.:

$$\textbf{split}(\textbf{merge}(m_1, m_2, r)) = (m_1, m_2, r) \tag{3}$$

Whether **merge** and **split** are actually inverse depends on the assumptions and limitations of the specific modeling language used to specify the models.

The formalization of merge and split presented in Equation 1 and Equation 2 are a good starting point, but in our view they have the following drawbacks:

Oversimplification: the definitions above to not reflect the nuances and complications that can be found in the state of the art. For example, it is not generally the case that split and merge are inverse operators. Moreover, the concept of relations between models is a multi-faceted one, and requires deeper investigation.

Focus on models: the **split** and **merge** operators are defined solely on models. They do not account for split and merge of *instances* (i.e. running execution of the models). The split and merge of instances of service compositions is very important because of their long-running nature. For example, changes applied to the models of the service compositions need to be migrated to some of their instances, i.e. "ad-hoc" changes.

Fixed multiplicity: the amount of models respectively in input and output to split and merge are fixed: one model is always split in two, and two models are merged into one. This in our view limits the flexibility of the definitions.

Reviewing the state of the art of BPM and SOA, we found a large body of related work about the *fragmentation* of business processes and process-based service compositions (e.g. service orchestrations and service choreographies). Fragmentation can be intuitively defined as the act of creating *fragments* from a whole (in our case, a process that realized a service composition).

However, despite the ubiquitous use of the term fragment, its definitions vary considerably. Weber et al. [24], Cossentino et al. [2] and Lu et al. [12] use the term fragment in the most general meaning, i.e. a subset of the elements (activities, control flows, etc.) that make the process.

Other authors provide more restrictive definitions. Eberle et al. [4] and Ma and Leymann [13] stress the point that fragments are not "one anymore" with the process, i.e. they are "physically" removed from

the process during the fragmentation. Tan and Fan [19] define fragments as "partitions of [a] model", using the Set Theory semantics of the term partition, i.e. subsets that do not overlap with each other. Both Khalaf [9] and Lindert and Deiters [11] define fragments as independent, stand-alone processes that, when executed in parallel, they interact with each other and realize the same semantics as the original process.

At this point, it was our conviction that the term "fragmentation", while equivalent to the term "split", should be preferred for the sake of consistency with the state of the art. Moreover, a more in-depth understanding of the characteristics of fragmentation and fragments was necessary for this deliverable, and generally beneficial to the state of the art.

In the remainder of the deliverable we will adopt the terminology and definitions presented in Section 2.1. The investigation on the characteristics of fragmentation has resulted in the classification criteria for fragmentation techniques that are presented in the first paper attached to the deliverable (included in Appendix A.1), and resumed in Section 2.2.

2.1 Terminology and Definitions

This section introduces the terminology that will be adopted in the remainder of the deliverable.

A **service composition** is "is a combination of a set of services for achieving a certain purpose"[1]. In the scope of this deliverable we consider **process-based service compositions**, i.e. service compositions that are structured as processes or workflows (e.g. Business Process Execution Language (BPEL) or Business Process Modeling Notation (BPMN) processes, Web Service Choreography Description Language (WS-CDL) and BPEL4Chor choreographies, etc.).

The **abstraction level** of a service composition is either **model** or **instance**. The model is the specification of the structure of the service composition, for example as a workflow (e.g. [14]). An instance is the execution of a service composition model [18, 19].

A **modeling language** specifies the syntax of service composition models and the semantics for executing their instances. Different modeling languages can be used to define service compositions of the same type, for example service orchestrations can be specified, among others, with both BPMN and BPEL, and service choreographies can be specified with Let's Dance [26], BPEL4Chor [3], etc. A modeling language provides a number of constructs that can be combined in specifying service compositions. For example, BPEL specifies several types of activities, event handlers, compensation handlers, etc. The constructs are instantiated as **elements** that compose the service composition model. In other words, a service composition model can be seen as an aggregation of elements instantiated using the constructs provided by the adopted modeling language.

A **fragment** is a subset of the elements of a service composition. Unless specified otherwise, no assumptions are taken on the elements included in a fragment (e.g. on how they relate each other), except that they are at least one. **Fragmentation** is the act of creating a set of fragments from one service composition. Conversely, **merge** is the act of creating a single service composition starting from a set of fragments. **Fragmentation-** and **merge techniques** are procedure, algorithm or methodology to perform fragmentation and merge, respectively, according to predefined criteria in order to achieve a certain goal.

2.2 Classification Criteria for Fragmentation

This section presents the classification criteria for fragmentation techniques. They are divided hierarchically in main- and subcriteria, e.g. the main criterion `What input` is further specialized in `Composition paradigm`, `Modeling language`, `Abstraction level`, `Well-formedness`, and `Self-containment`.

[1] S-Cube Knowledge Model: Service Composition – `http://www.s-cube-network.eu/knowledge-model/terms/s/service-composition`

Each main criterion treats a fundamental aspect of fragmentation techniques:

What input covers the characteristics of the input to a fragmentation, in our case process-based service compositions. It is further specialized in:

- **Composition paradigm** specifies the paradigm of service composition (e.g. service orchestration or service choreography.
- **Modeling language** refers to which modeling language is used to specify the service composition, e.g. BPEL [16], BPMN [17], WS-CDL [23] or Let's Dance [26].
- **Abstraction level** specifies whether the fragmentation technique is applicable to service composition models, instances, or both.
- **Well-formedness** covers whether the input service composition is assumed to be well-formed or not. A service composition is well-formed if it adheres to the syntax and constraints set by its modeling language.
- **Self-containment** specifies whether the input service composition is self-sufficient with respect to its intended use. For example, an executable business service composition is self-contained if and only if it contains all the data necessary for its execution.

Why treats the intended aim of a fragmentation technique, e.g. enable distributed execution of a service composition, or increase its reusability.

When considers at which point the fragmentation occurs in the lifecycle of the service composition. For example, assuming the service composition lifecycle documented in [25], the fragmentation can occur during Modeling or Execution.

Who investigates the actors that perform the fragmentation, e.g. a human service composition modeler, or an automatic tool (e.g. an execution engine for service composition).

What output analyzes the properties of the fragments that result from the fragmentation of the input service composition. `Composition paradigm`, `Modeling language`, `Abstraction level`, `Well-formedness`, and `Self-containment` are equivalent to their namesakes under `What input`, except that they apply to the output instead than to the input. The remainder criteria are:

- **Unity** specifies whether it is still "physically" comprised in the service composition. `Unity` is *preserved* if the fragment is an "high-lighted area" of the service composition. Otherwise, `Unity` is *disrupted* in fragmentation techniques that create fragments that are independent from the original service composition, e.g. if the fragments themselves are service composition models or instances.
- **Cohesion** is a measure of how much the elements in the fragment "belong together" [22].
- **Granularity** is a measure of how their sizes relate to the size of the input service composition.
- **Interdependency** defines the types of dependencies (if any) between the fragments, e.g. in terms of control dependencies during their distributed execution, e.g. control- and data dependencies and data dependencies or coordination protocols.
- **Coverage** reports the extent to which the fragments cumulatively comprise all the elements of the service composition. `Coverage` can be *full* (every element of the service composition is contained in at least one of the fragments), *partial* (the union of the fragments contains a proper subset of the elements of the service composition), *extended* (the union of the fragments contains every element in the service composition, plus others that were not part of the service composition), and *partial-extended* (the union of the fragments contains only some of the elements of the service composition, plus others that were not part of the service composition.

Overlap treats the extent to which the fragments overlap (*full*, *partial* or *nesting*).

Which specifies the order according to which the phases Fragment Identification (identifying the fragments) and Severing (modifying the service composition to "cut" the fragments out) are executed.

Where treats the *scope* of the fragmentation, i.e. the regions of the service composition that are processed during the fragmentation.

How classifies the fragmentation techniques according to aspects of the algorithms that realize them. It is further specialized in:

Fragmentation criteria captures the rationale of the fragmentation, e.g. group activities according to the role that executes them. In other words, the `Fragmentation criteria` describe in natural language the idea behind the algorithm of the fragmentation technique. Notice that `Why` and `Fragmentation criteria` do not overlap. The former covers the reasons why the fragmentation technique has been devised, while the latter describes the principles through which the fragmentation technique fulfills those reasons.

Automation refers to the extent of automation of the fragmentation technique (e.g. manual, semi-automatic or automatic).

Determinism treats whether the fragmentation technique results always in the output given the same input.

Configurability specifies whether the fragmentation technique can be configured with additional parameters that drive the fragmentation of the input service composition (e.g. specifying the desired amount of fragments, or their granularity). In a nutshell, the `Configurability` criterion captures the aspects of the fragmentation technique that, given the same service composition in input, make different fragmentation result as output.

Computational complexity is a measure of the complexity of the fragmentation according to the size of the input.

Optimality specifies whether the fragmentation technique always results in the optimal fragmentation (for some definition of optimum) or in a sub-optimal one.

3 Fragmentation and Merge Approaches

This section shortly introduces the research papers that are attached to the deliverable. The actual papers can be found in the respective sections of the Appendix (see also Section 1.1).

3.1 Optimal Stratification of Transactions

Depending on the domain in which they are used, service compositions might need to have *atomic* ("all-or-nothing") behavior. We consider service composition constructed out of a set of *basic transactions* (i.e. atomic activities), which are sequenced using a workflow. The atomicity of the overall service composition is achieved by grouping the basic transactions into *global transactions*, i.e. groups of basic transactions coordinated and synchronized through Two-Phase-Commit (2PC) protocol. The simplest way is to group all basic transactions in the service composition into one "monolithic" global transaction. However, that is impractical because of the overhead of locking of the coordination protocols.

A more refined approach is to employ the *stratification of transactions* [10]. The idea (a generalization of the concept of transaction chaining) is to divide the service composition into many "connected" global transactions called *strata*. In other words, strata are fragments of the service composition, each one coordinated by a 2PC protocol. The strata communicate with each other for coordination purposes using persistent message queues.

However, stratification of transactions is not a "push button" solution: it may also lead to performance-related drawbacks because of the overhead of messaging of the coordinations protocols. The stratification of transactions requires careful balancing between the granularity of the strata and the coordination protocols that they employed. The granularity of the strata is the ratio of the size of each one (i.e. how many basic transactions it contains), and how many strata are defined. On one hand, exceedingly fine-grained strata cause a costly overhead of message-based communication between them. On the other hand, coarse grained strata lead to overhead of resource locking and increase the abort probability of 2PC protocols and the amount of 2PC messages they require.

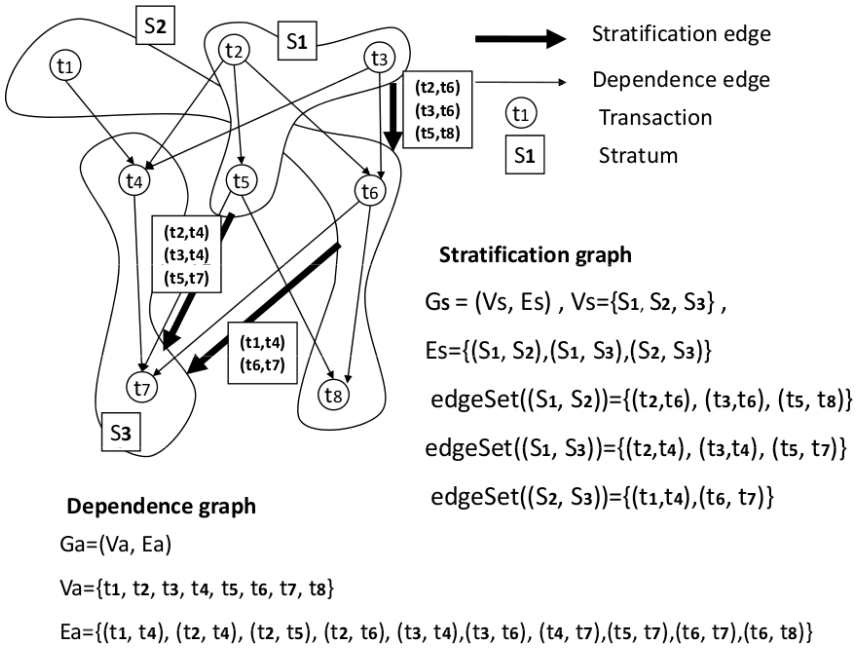


Figure 1: An example of stratification graph

Since each stratum can be seen as a fragment of the original service composition, the strafication of transactions is a way to fragment the service composition while maintaining its atomicity and improving

the non-functional properties. This paper presents a method to identify the optimal fragmentation by taking into account properties of the basic transaction like execution time, recovery cost, invocation cost, timeliness cost, abort probability, start and due time. The fragmentation optimizes the service composition in terms of its overall execution time, concurrency, message costs (for both 2PC and persistent message queues), invocation/recovery cost, timeliness cost, etc. The outcome of the stratification are the strata (i.e. fragments of the service composition), and the stratification graph that connects them (an example of stratification graph is shown in Figure 1).

The approach builds on the following assumptions. The service composition is constructed out of a set of activities (i.e. the basic transactions). The basic transactions are described through properties like execution time, recovery cost, invocation cost, timeliness cost, abort probability, start and due time, etc. The predefined execution order between the basic transactions is described by the *dependency graph*. The dependency graph abstracts from the concrete data- and control- flow dependencies among the basic transactions. It is acyclic and contains only simple control-flow constructs such as `AND-JOIN` and `SPLIT`. The basic transactions can be executed only if all of its predecessors basic transactions have been successfully executed (`AND-JOIN`) and if the constraints on its start time are met.

Each stratum has two message queues, one for incoming messages and one for outgoing. The dependencies between basic transactions are translated in dependencies in the stratification graph between the strata that contain them. In order to avoid deadlocks during the execution of the *stratified global transaction* (i.e. the global transaction restructured around the strata), the stratification graph must be acyclic.

The stratification of a global transaction is not necessarily unique, i.e. a service composition can be divided into strata in multiple ways. The stratification alternatives differ in their costs/quality characteristics. To pick the best alternative, the technique employs the hill-climbing and simulated annealing search optimization approaches, which are combined in a hybrid algorithm to overcome the particular shortcomings of each.

The classification of this approach according to the criteria for fragmentation techniques is presented in Table 1.

Criterion	Value
What input is given to the fragmentation	
Composition paradigm	Service coordination
Modeling language	Dependency graph
Abstraction level	Model
Well-formedness	Yes
Self-containment	No
Why is the service composition fragmented	Optimization of performance and costs
When the fragmentation is performed in the service composition lifecycle	Design-time or deployment-time
Who performs the fragmentation	Modelling tool or infrastructure
What output	
Composition paradigm	Service coordination
Modeling language	Stratification graph
Abstraction level	Model
Well-formedness	Yes
Self-containment	No
Multiplicity	1+
Unity	Yes
Cohesion	Unspecified
Granularity	Variable
Interdependency	Yes: Stratification graph
Coverage	Full-extended
Overlap	Disjoint
Where is the fragmentation performed in the service composition	Global
Which phases does the fragmentation perform	Fragment Identification & Fragment Severing
How is the fragmentation performed	
Fragmentation criteria	Cost reduction and QoS optimization
Automation	Yes
Determinism	No
Configurability	Yes: which aspects to optimize
Computational complexity	Unspecified
Optimality	Not guaranteed

Table 1: Classification of the fragmentation technique presented in "Optimal Stratification of Transactions"

3.2 Towards Identification of Fragmentation Opportunities for Service Workflows with Sharing-Based Independence

Composition notations and languages, such as BPMN [17], and WS-BPEL [16], allow process modelers and designers to view a composition from the point of business logic and processing requirements related to parallelism and data flow. The now fashionable service mash-ups are also tools for building (usually simplified) customized workflows from known service components in a user-centric way. Finally, compositions can be programmed in any common programming language, such as Java, with infrastructure that provides the necessary constructs and libraries for establishing client connections to, and exposing Web services. This calls for a neutral, language independent notion of fragmentation and fragmentation possibilities.

Our approach is based on the very general notion of independence between parts of a composition. The underlying idea is that workflows have a certain degree of freedom in (re-)arranging their activities, without violating the overall inter-process business protocol, and while preserving their essential properties, such as correctness and transactional integrity. These properties are, on one hand, dictated by the control structures (usually already presented at the level of workflow design) but also at the level of data flow, which is usually not present in many workflow designs, but which may induce dependencies between fragments which may, for some aims, disrecommend treating them as fragments.

Sharing-based independence analysis is a general technique that can be applied to both upper and lower layers of software architecture, and consequently, to various parts of the service stack. On the service composition layer, we model different entities used within a workflow as data structures subject to sharing analysis. These can include:

- Variables (e.g. in BPEL) that hold the state of the executing processes. This is a classical case of data sharing found in many programming languages. The analysis needs to take into account that these variables usually complex XML-like structures.
- Services used in the composition (partners in BPEL parlance). Accesses to partners need to respect their business protocol, which can be as simple as a synchronous *request-reply* cycle, or as complex as an asynchronous exchange of messages with multiple stateful callbacks and conversations going on in the course of such exchange. We model dependencies on partner services by mapping services to new variables that are used at invocation.
- External resources, such as databases, which have to be accessed following a protocol, and which may hold persistent information. These may be mapped on either local activities directly implemented by the composition execution engine, or to external services that implement them. To model dependencies on these resources, we map database identifiers to logical variables that are used in database operations.

The overall idea of using the sharing-based approach to detection of fragmentation opportunities is shown on Figure 2. A workflow description, in an adequate abstract or executable notation, is first translated into an intermediate notation, in the form of a logic program that is amenable to sharing analysis the existing tools for analysis of logical programs (e.g., those developed in Ciao / CiaoPP [7, 6]). The intermediate notation tends to capture relevant information that allows sufficiently accurate sharing and subsequent analysis steps.

The intermediate notation is an abstract model of a workflow, which is on one hand more concrete than e.g. Petri-net based workflow models, but less concrete than an executable implementation, e.g. in BPEL, since we do not necessarily need to mimic the actual operational semantics. Depending on the focus and objectives of a particular analysis (identification of fragments, security concerns, resource management) different abstractions of the original process can be generated.

The analysis step takes the translation in the intermediate notation and produces results in forms of code and type annotations, which are combined with the input to the analysis to produce an annotated

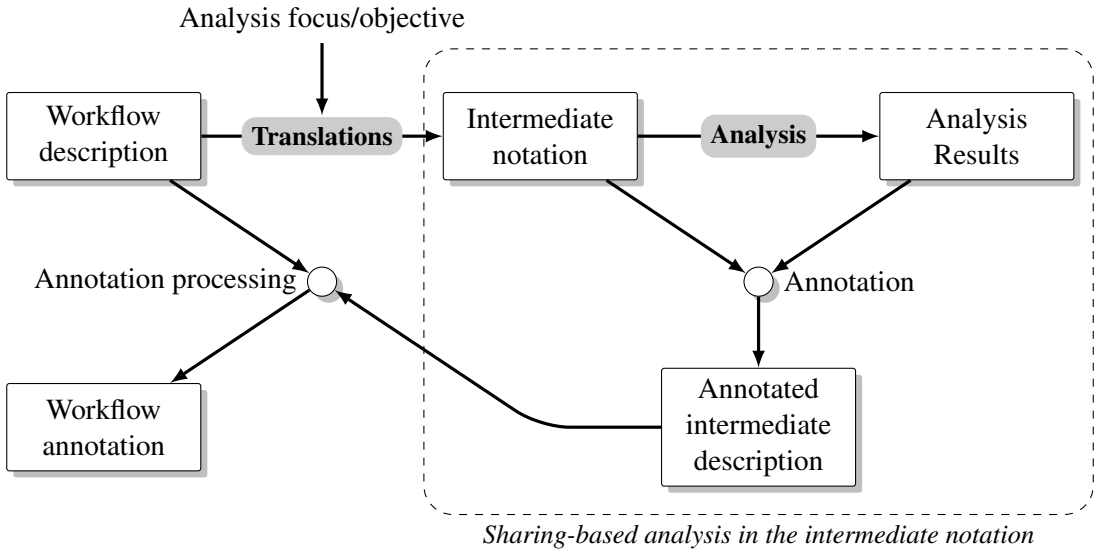


Figure 2: Overview of the sharing-based fragmentation analysis

description of the workflow in the intermediate notation. Finally, the annotated intermediate description, combined with the original workflow description, can be used used to produce an workflow annotation containing results of the analysis. Alternatively, the intermediate notation or the analysis results can be presented in a suitable form to human workflow designers/implementors, or to tools that perform static or dynamic fragmentation and merging.

This paper is concerned with the part of the sharing-based fragmentation analysis on Figure 2 that deals with workflows represented in the intermediate notation.

It is worth noting that sharing analysis is non-trivial and that doing it manually is usually infeasible, and anyway inefficient, error prone, and, which is especially important, incompatible with the requirements of automated service adaptation and evolution. Automatic sharing analysis, on the other hand, although hard and computationally intensive, can be employed automatically on demand, and guarantees safety of its results: although it may sometimes miss an opportunity for split, by applying conservative approach it guarantees that the results of sharing analysis do not contain false negatives. In our future work, we plan to examine requirements and constraints for the runtime application of this analysis method.

This work contributes to the deliverable and the activity aims by proposing a syntactic-based approach to automatic fragment identification which is nonetheless semantics-preserving, assuming correct initial information about the process is given. It does not aim, at this stage, at replacing other fragmentation methods but rather at complementing them by, e.g., giving some initial fragmentation conditions to which splitting algorithms can be applied [8].

The classification of this approach according to the criteria for fragmentation techniques is presented in Table 2.

Criterion	Value
What input is given to the fragmentation	
Composition paradigm	Service orchestration
Modeling language	Logic-based notation, in perspective to be automatically deduced from BPEL-style, control flow dependency based orchestrations.
Abstraction level	Model and instance
Well-formedness	Yes
Self-containment	Yes
Why is the service composition fragmented	Analysis, Abstraction, Optimize non-functional properties
When the fragmentation is performed in the service composition lifecycle	Design-time, Runtime
Who performs the fragmentation	Analyst, Automated tools
What output	
Composition paradigm	Service orchestration
Modeling language	Logic-based notation, in perspective convertible to other representations.
Abstraction level	Model
Well-formedness	Yes
Self-containment	No
Multiplicity	1+
Unity	Yes
Cohesion	Variable
Granularity	Variable
Interdependency	Yes
Coverage	Full
Overlap	No
Where is the fragmentation performed in the service composition	Global, Partial
Which phases does the fragmentation perform	Fragment Identification
How is the fragmentation performed	
Fragmentation criteria	Sharing-Based Independence
Automation	Yes
Determinism	Yes
Configurability	Yes: different objectives
Computational complexity	Variable, depending on the algorithm applied and precision trade-offs
Optimality	Proportional to precision of analysis

Table 2: Classification of the fragmentation technique presented in "Towards Identification of Fragmentation Opportunities for Service Workflows with Sharing-Based Independence"

3.3 Decentralized execution of service compositions by runtime process migration

The decentralized execution of business process instances is a promising approach for enabling flexible reactions to contextual changes at runtime. To realize this, this contribution focuses on the *runtime fragmentation of process instances*, allowing several (potentially pre-selected) parties to execute a given process instance in a decentralized way. The main goal here is to enable a flexible adaptation of the responsibilities for the execution of the process (in whole or in part) to dynamically changing situations at runtime.

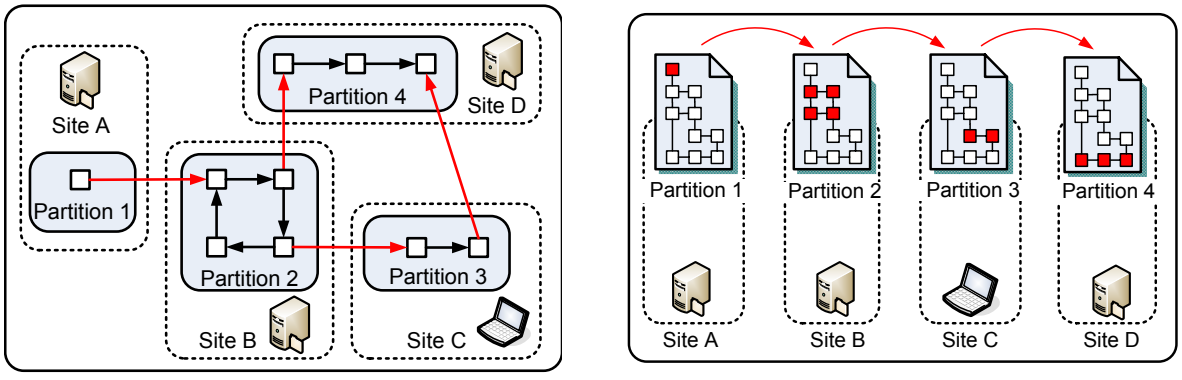
In contrast to most current approaches which address process distribution by *physical fragmentation of processes* and by dynamic assignment of resulting static process parts to different business partners (see Figure 3(a)), this approach proposes to enable a more dynamic segmentation of responsibilities at runtime by using *runtime process migration* as a means of *logical process fragmentation* (see Figure 3(b)). As this variant of process fragmentation offers a high degree of flexibility, it is particularly suitable for environments where potential process participants (i.e. process execution engines) can enter and leave the system dynamically or where quality-of-service-parameters change very often (see A.4).

The initial approach for a decentralized execution of service compositions by runtime process migration contains two parts:

- First, a concept for the general execution of process parts and for the (potential) delegation of the (remaining) process including all required state information, so that exactly one participant at a time is responsible for the execution of the process instance.
- Second, the provision of adequate privacy and security mechanisms in order to protect private information carried within the process (e.g. private data, activities or identities).

The approach proposed here is based on a concept which has been established in the context of highly dynamical mobile environments (see [27]). Therein, the runtime assignment of single process tasks or larger sub processes is realized by migrating process instances from one partner to another, so that each partner is able to perform a particular subset of the process. The assignment can either happen dynamically, e.g. in dependence of a partner's special capabilities or his current workload - or the process can contain a predefined description of the desired partner to fulfill a particular task (or a set of tasks), based on functional and/or on non-functional requirements.

Compared to physical fragmentation, process migration provides more flexibility by allowing the distribution of running process instances while at the same time respecting the guidelines of the process modeler. It allows for fragmenting the responsibility to execute a process at runtime - depending on the availability of business partners or other contextual incidences. Furthermore, the granularity of fragmentation and the range of distribution can be selected on the fly by each executing participant. At least in the



(a) Process fragmentation

(b) Process migration

Figure 3: Process decentralization variants

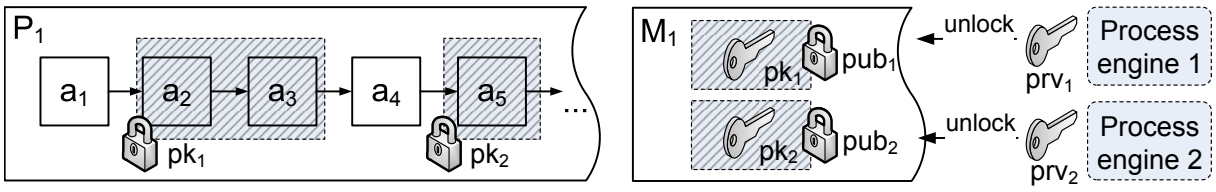


Figure 4: Process encryption and key distribution

case of sequential execution, coordination and merging of multiple process fragments is not necessary. Global variables, scopes, errors and transactions are easier to handle, because all aspects of the process (i.e. data and control flow) are available to all executing parties. Thus, there is less communication and coordination overhead.

However, during decentralized execution of a migrated process, its entire information is generally public to subjects which potentially belong to foreign organizations. To prevent such privacy and security threats, the access to process data can be restricted to specified subjects or roles. Therefore, the existence of a basic cryptographic key infrastructure such as *PKI (Public Key Infrastructure)* is assumed. Critical parts of a process description are *masked*, in order to ensure that only authorized participants can execute sensitive activities and access corresponding data. Masked process parts are encrypted with a session key which can be obtained from the migration metadata. The session key itself is encrypted with the public key of the authorized party (see Figure 4). If more than one participant needs access to a shared object (e.g. to a global process variable), the session key can be provided for each of them. In addition, the integrity of process description and migration metadata can be ensured by standard procedures such as e.g. *MAC (Message Authentication Code)* and *digital signatures*.

The classification of this approach according to the criteria for fragmentation techniques is presented in Table 3.

Criterion	Value
What input is given to the fragmentation	
Composition paradigm	Service orchestration
Modeling language	Executable PDL, e.g. WS-BPEL
Abstraction level	Instance
Well-formedness	Yes
Self-containment	Yes
Why is the service composition fragmented	Distributed execution
When the fragmentation is performed in the service composition lifecycle	Runtime
Who performs the fragmentation	Execution engine (potentially based on the guidelines of process modeler)
What output	
Composition paradigm	Service orchestration
Modeling language	Executable PDl, e.g. WS-BPEL with task assignment
Abstraction level	Instance
Well-formedness	Yes
Self-containment	Yes
Multiplicity	1+
Unity	Yes
Cohesion	Unspecified
Granularity	Variable
Interdependency	Yes: control flow and data dependencies
Coverage	Full-extended
Overlap	No
Where is the fragmentation performed in the service composition	Global
Which phases does the fragmentation perform	Identification and Fragment Severing
How is the fragmentation performed	
Fragmentation criteria	Depending on configuration
Automation	Yes
Determinism	Depending on external context
Configurability	Yes
Computational complexity	Linear
Optimality	Depending on configuration

Table 3: Classification of the fragmentation technique presented in "Decentralized Execution of Service Compositions by Runtime Process Migration"

3.4 Executing Parallel Tasks in Distributed Mobile Processes

In case of a sequential execution of process fragments, the efforts of coordination can often be reduced to a (relatively simple) delegation resp. migration protocol (see A.4). However, advanced synchronization and coordination mechanisms are required, if *parallel process fragments* have been distributed to several different parties. If furthermore shared data objects are used in more than one of these parallel fragments, a separate execution could lead to undesired or even wrong results. (see A.5). This contribution considers the concurrent execution of several parallel paths of the process instance by replication of the process description and respective execution of the parallel section of the process by different participants, including synchronization of control flow and data variables.

As long as the process is transferred to exactly one participant, the execution of a parallel section of the process is uncritical as the states of each included activity are well defined and data dependencies can be handled locally (e.g. by synchronization). However, since the execution of parallel paths on a single machine cannot be considered as "real parallelism", a copy of the (entire) process can be distributed to different participants, which are each responsible for the execution of one of the parallel paths. In order to synchronize the parallel paths a meeting point has to be defined. In consequence, the distributed parallel execution requires advanced coordination mechanisms.

The presented concept is based on the data replication and respective methods for detection of data dependency conflicts, decentralized coordinated execution and synchronization of parallel process fragments. If the effort for synchronizing variables on parallel paths is considered to be critical (e.g. in mobile environments where communication networks have a low bandwidth) the process designer can reduce synchronization overhead already at design time (see Figure 5). Therefore, the process model can be checked for process variables, which are read and/or written in more than one parallel path. In case a data conflict is detected (i.e. coordination and/or synchronization of process variables may be necessary in order to ensure a correct execution), the process modeler can solve unintended data dependencies or assign so-called *dataclasses*, which specify under which circumstances the variables have to be synchronized.

Considering runtime, the process is executed the usual way until the process execution reaches a branch, which results in parallel execution (i.e. an AND-SPLIT). In this case, the process instance description including all migration metadata is replicated and distributed to suitable process engines, which are now each responsible for the execution of one of the paths (see Figure 6). Together with the information extracted from the dataclasses, an algorithm for optimistic runtime conflict resolution is applied in order to reduce coordination overhead among replicates. Final synchronization and merging of replicates can be induced by specifying a fixed participant for the execution of the respective AND-JOIN activity or by a distributed incremental procedure, which allows for determining the synchronization point at runtime. However, this increased flexibility for synchronization also implies a slightly increased

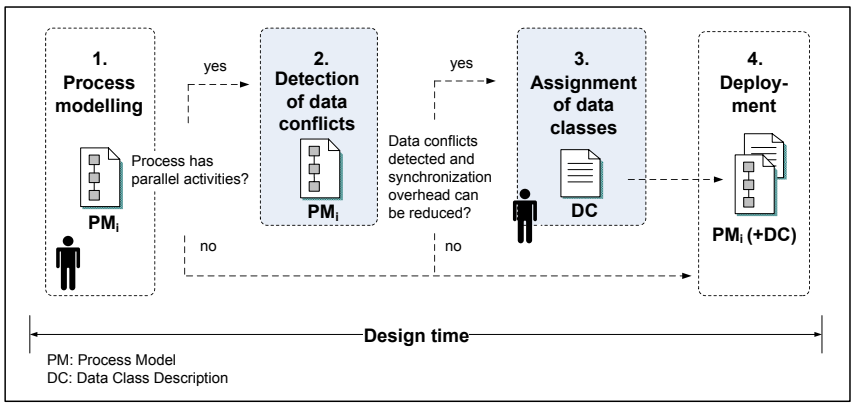


Figure 5: Preparation for reduction of synchronization overhead

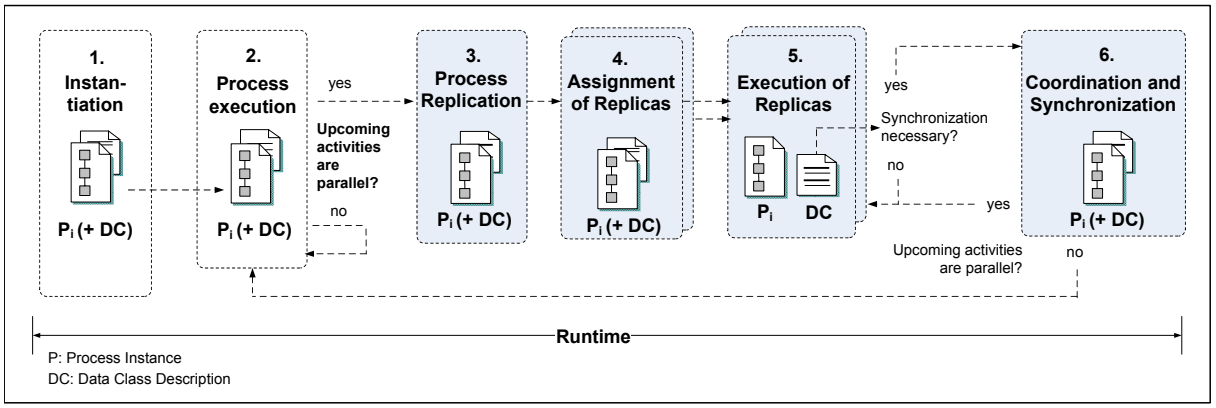


Figure 6: Execution and synchronization of replicated parallel process paths

coordination overhead.

The classification of this approach according to the criteria for fragmentation techniques is presented in Table 4.

Criterion	Value
What input is given to the fragmentation	
Composition paradigm	Service orchestration
Modeling language	Executable PDL, e.g. WS-BPEL
Abstraction level	Instance
Well-formedness	Yes
Self-containment	Yes
Why is the service composition fragmented	Distributed execution
When the fragmentation is performed in the service composition lifecycle	Runtime
Who performs the fragmentation	Execution engine (potentially based on the guidelines of process modeler)
What output	
Composition paradigm	Service orchestration
Modeling language	Executable PDl, e.g. WS-BPEL with task assignment
Abstraction level	Instance
Well-formedness	Yes
Self-containment	Yes
Multiplicity	1+
Unity	Yes
Cohesion	Unspecified
Granularity	Variable
Interdependency	Yes: control flow and data dependencies
Coverage	Full-extended
Overlap	No
Where is the fragmentation performed in the service composition	Global
Which phases does the fragmentation perform	Identification and Fragment Severing
How is the fragmentation performed	
Fragmentation criteria	Depending on configuration
Automation	Yes
Determinism	Depending on external context
Configurability	Yes
Computational complexity	Linear
Optimality	Depending on configuration

Table 4: Classification of the fragmentation technique presented in "Decentralized Execution of Service Compositions by Runtime Process Migration"

3.5 Bridging Requirements and Executable Processes using Merge Techniques

This paper is of exploratory nature. It aims at connecting the area of RE and BPM, which up to now have evolved mostly in separation. The paper contributes an investigation of how merge techniques can be applied to use case scenarios and service composition models.

The paper is motivated by the observation that use case scenarios and models of process-based service compositions are of similar nature. On one hand, a use case contains set of scenarios focused on achieving the same (business) goal. Each scenario describes the steps performed by the system (usually including its interactions with the users) in order to achieve the scenario's goal. On the other hand, a service composition model defines the ordering of the activities (e.g. service invocations) that have to be performed by a system (e.g. a process execution engine) to achieve the goal of the service composition, i.e. provide the composite service. Scenarios and service composition models differ in their degree of abstraction and in the modeling languages they adopt.

Both scenarios and service composition models correspond to stages in the SBA proposed in [15]: scenarios are created during the EARLY REQUIREMENTS phase, while process models are the artifacts created during the REQUIREMENT ENGINEERING AND DESIGN phase that are later deployed and executed. The development of SBAs would benefit from the semi-automation of the transition between the EARLY REQUIREMENTS and REQUIREMENT ENGINEERING AND DESIGN phase, which can be realized through methods and tools to (semi-)automatically create service compositions from the use case scenarios.

The paper envisions two alternative approaches to bridge the use case scenarios and service composition models:

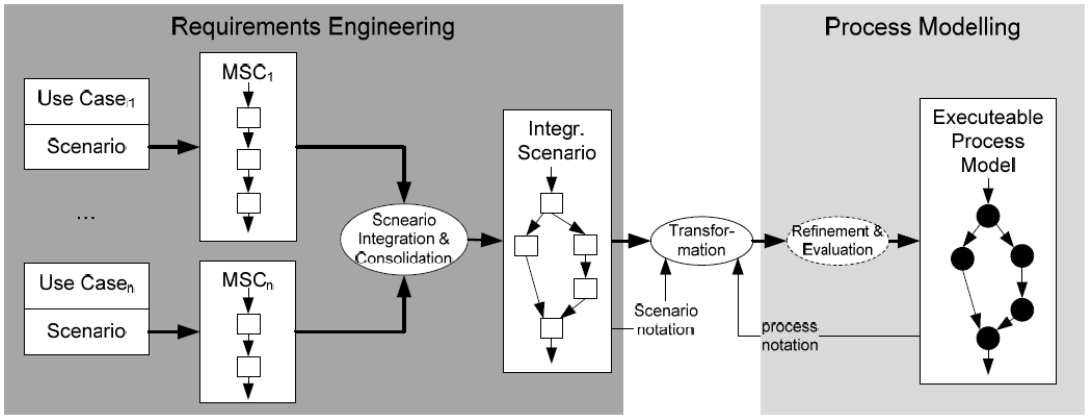


Figure 7: Deriving Process Models from Use Cases - Alternative 1

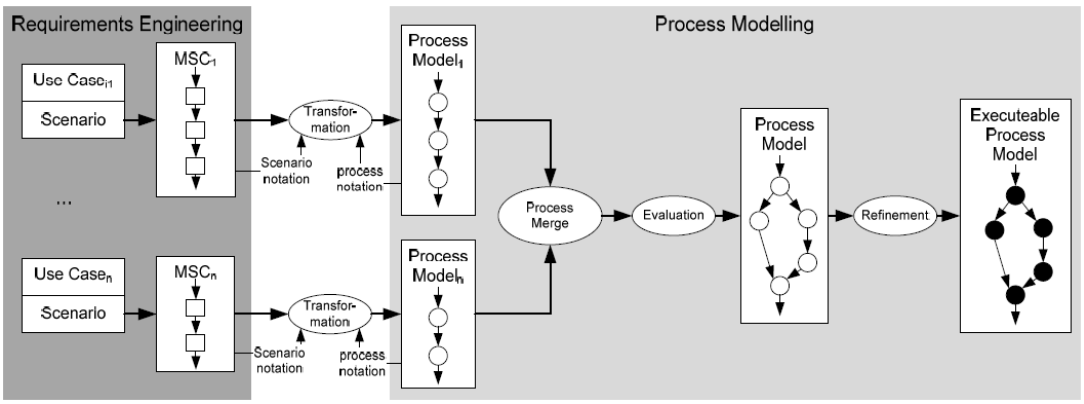


Figure 8: Deriving Process Models from Use Cases - Alternative 2

Merge first, Transform later: the scenarios are first merged, and then transformed into a service composition model (Figure 7);

Transform first, Merge later: each scenario is transformed into a distinct service composition; the service compositions so produced are later merged into one (Figure 8).

The paper compares the two alternatives in light of the state of the art of merging use case scenarios and service compositions. The outcome of the paper's investigation is that, while there is a consistent body of work that can be used to bridge use case scenarios and service composition models in the scope of SBA, the state of the art suffers from some deficiencies that require further research efforts.

4 Conclusions

This deliverable investigates fragmentation and merge approaches for service compositions. The contributions of this deliverable are of three different types. First of all, it provides classification criteria for fragmentation techniques for service compositions. Secondly, the deliverable introduces four novel approaches for fragmentation criteria of service compositions applied in different domain that range from transactional service compositions to migration of instances of service orchestrations. Finally, it investigates the role of merge in bridging the gap between Requirement Engineering and Business Process Management.

The content of this deliverable will be used as basis for the upcoming CD-JRA 2.2.4: Models, Mechanisms and Protocols for Coordinated Service Compositions. CD-JRA 2.2.4 will include the specification of mechanisms and coordination protocols for controlling the consistent execution of fragmented processes. The fragmentation and merge techniques presented in this deliverable are a first step towards the specification of those mechanisms. Moreover, the classification criteria for fragmentation techniques provided in the scope of this deliverable will act as a baseline for defining the properties of the fragmented processes, as well as a catalogue of important aspects of current and future techniques for the fragmentation of processes.

References

[1] Greg Brunet, Marsha Chechik, Steve Easterbrook, Shiva Nejati, Nan Niu, and Mehrdad Sabetzadeh. A manifesto for model merging. In *GaMMa '06: Proceedings of the 2006 international workshop on Global integrated model management*, pages 5–12, New York, NY, USA, 2006. ACM.

[2] Massimo Cossentino, Salvatore Gaglio, Stéphane Galland, Nicolas Gaud, Vincent Hilaire, Abderrafiaa Koukam, and Valeria Seidita. A MAS metamodel-driven approach to process fragments selection. In Michael Luck and Jorge J. Gómez-Sanz, editors, *AOSE*, volume 5386 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2008.

[3] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. BPEL4Chor: Extending BPEL for modeling choreographies. In *ICWS*, pages 296–303. IEEE Computer Society, 2007.

[4] Hanna Eberle, Tobias Unger, and Frank Leymann. Process fragments. In *OTM*, volume 5870 of *Lecture Notes in Computer Science*, pages 398–405. Springer, November 2009.

[5] Andreas Gehlert and Andreas Metzger. Quality reference model for SBA. Contractual Deliverable CD-JRA-1.3.2, S-Cube Consortium, March 2008. http://www.s-cube-network.eu/results/deliverables/wp-jra-1.3.

[6] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor). *Science of Computer Programming*, 58(1–2):115–140, October 2005.

[7] M. V. Hermenegildo, F. Bueno, M. Carro, P. López, J.F. Morales, and G. Puebla. An Overview of The Ciao Multiparadigm Language and Program Development Environment and its Design Philosophy. In Jose Meseguer Pierpaolo Degano, Rocco De Nicola, editor, *Festschrift for Ugo Montanari*, number 5065 in LNCS, pages 209–237. Springer-Verlag, June 2008.

[8] R. Khalaf and F. Leymann. E Role-based Decomposition of Business Processes using BPEL. In *IEEE International Conference on Web Services (ICWS'06)*, 2006.

[9] Rania Khalaf. *Supporting business process fragmentation while maintaining operational semantics: a BPEL perspective*. Doctoral thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, March 2008.

[10] Frank Leymann and Dieter Roller. *Production Workflow - Concepts and Techniques*. PTR Prentice Hall, Januar 2000.

[11] Frank Lindert and Wolfgang Deiters. Modelling inter-organizational processes with process model fragments. In *Enterprise-wide and Cross-enterprise Workflow Management*, volume 24 of *CEUR Workshop Proceedings*, pages 33–41. CEUR-WS.org, 1999.

[12] Ruopeng Lu, Shazia Wasim Sadiq, and Guido Governatori. On managing business processes variants. *Data Knowl. Eng.*, 68(7):642–664, 2009.

[13] Zhilei Ma and Frank Leymann. BPEL fragments for modularized reuse in modeling BPEL processes. *Networking and Services, International conference on*, 0:63–68, 2009.

[14] Mangala Gowri Nanda, Satish Chandra, and Vivek Sarkar. Decentralizing execution of composite web services. In *OOPSLA*, pages 170–187. ACM, 2004.

[15] Elisabetta Di Nitto. State of the art report on software engineering design knowledge and survey of hci and contextual knowledge. Deliverable PO-JRA-1.1.1, S-Cube Consortium, July 2008. http://www.s-cube-network.eu/results/deliverables/wp-jra-1.1.

[16] OASIS. Web Services Business Process Execution Language Version 2.0. OASIS Standard, OASIS, April 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[17] OMG. Business Process Modeling Notation Version 1.2. OMG Recommendation, OMG, February 2008. http://www.omg.org/spec/BPMN/1.2/.

[18] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. Disjoint and overlapping process changes: Challenges, solutions, applications. In *CoopIS/DOA/ODBASE (1)*, volume 3290 of *Lecture Notes in Computer Science*, pages 101–120. Springer, 2004.

[19] Wei Tan and Yushun Fan. Dynamic workflow model fragmentation for distributed execution. *Computers in Industry*, 58(5):381–391, 2007.

[20] The S-Cube Team. S-Cube's global research vision, version 1.0. Vision document, S-Cube Consortium, April 2009.

[21] Willem-Jan van den Heuvel. Survey on business process management. Deliverable PO-JRA-2.1.1, S-Cube Consortium, July 2008. http://www.s-cube-network.eu/results/deliverables/wp-jra-2.1.

[22] Irene T. P. Vanderfeesten, Hajo A. Reijers, and Wil M. P. van der Aalst. Evaluating workflow process designs using cohesion and coupling metrics. *Computers in Industry*, 59(5):420–437, 2008.

[23] W3C. Web Services Choreography Description Language Version 1.0. Candidate Recommendation, W3C, November 2005.

[24] Barbara Weber, Stefanie Rinderle, and Manfred Reichert. Change patterns and change support features in process-aware information systems. In *CAiSE*, volume 4495 of *Lecture Notes in Computer Science*, pages 574–588. Springer, 2007.

[25] Branimir Wetzstein. Overview of the state of the art in composition and coordination of services. Deliverable PO-JRA-2.2.1, S-Cube Consortium, July 2008. http://www.s-cube-network.eu/results/deliverables/wp-jra-2.2.

[26] Johannes Maria Zaha, Alistair P. Barros, Marlon Dumas, and Arthur H. M. ter Hofstede. Let's Dance: A language for service behavior modeling. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 4275 of *Lecture Notes in Computer Science*, pages 145–162. Springer, 2006.

[27] Sonja Zaplata, Christian P. Kunze, and Winfried Lamersdorf. Context-based Cooperation in Mobile Business Environments: Managing the Distributed Execution of Mobile Processes. *Business and Information Systems Engineering (BISE)*, 2009(4), 10 2009.

A Attached Papers

A.1 The Kaleidoscope of Fragmentation of Service Compositions

Authors:

Tilburg: Michele Mancioppi

USTUTT: Olha Danylevych

USTUTT: Dimka Karastoyanova

USTUTT: Frank Leymann

Tilburg: Mike P. Papazoglou