The final version is published: (c) IEEE
S. Rinderle-Ma, P. Grefen: Towards Flexibility in Transactional Service Compositions,
21th IEEE International Conference on Web Services (2014)

Towards Flexibility in Transactional Service Compositions

Stefanie Rinderle-Ma
University of Vienna
Faculty of Computer Science
stefanie.rinderle-ma@univie.ac.at

Paul Grefen
Eindhoven University of Technology
School of Industrial Engineering
P.W.P.J.Grefen@tue.nl

***Abstract*—Complex services can be described by service compositions and executed by service orchestrations. Changing service compositions is a frequent requirement in practical settings. Changing the composition must not result in a violation of its functional or non-functional properties. Whereas functional aspects such as soundness have been studied quite well, non-functional properties such as transactionality have been paid little attention to in the context of change. However, in practical applications it is impossible to separate the aspects of transactionality and change. In this paper, we investigate the effects of applying changes in transactional service compositions. For this we analyze the combination of concepts from the worlds of transactional service compositions and process change. Based on the analysis results, we derive algorithms to deal with change in transactional service compositions. We discuss the algorithm design and their practical applicability.**

***Keywords*-Web services QoS; Change in Web service composition, transactional Web service compositions**

I. Introduction

The system-based support of complex services as implementation of their primary business processes constitutes a promising option for enterprises. The use of complex services requires the application of service composition at design time and service orchestration at execution time. An example from the manufacturing domain is the production of car seats consisting of several steps [1]. Another application domain is logistics [2]. However, the implementation of service compositions must not come at the price of rigidity, i.e., the limitation that the underlying process logic is hard-wired and cannot be changed without incurring high costs.

Changes may be caused by reasons external to an organization, such as changing business market conditions or new regulations. Changes may also be caused by internal reasons like business process optimizations. Flexibility and change in business processes have been well researched and several mature approaches and even commercial tools exist [3].

It is essential that changes in service compositions are applied in a controlled manner, i.e., without causing any problems in the subsequent composition execution. Existing approaches have focused on functional properties of the composition so far. More precisely, criteria for preserving structural and behavorial soundness after applying a change to an composition were introduced along with methods on how to ensure these criteria.

In turn, non-functional properties have played a minor role in the context of composition change so far, even though the importance of considering composition change together with non-functional requirements is acknowledged in literature [3]. In this paper, we focus on transactionality as a crucial non-functional requirement. It has been addressed by a body of work on transactional workflows [4], [5]. Astonishingly, a combined consideration of transactional service composition and change seems to be entirely missing.

In practical applications, however, it is impossible to separate the aspects of transactionality and change, i.e., a change can become necessary on a transactional service composition. Consider the example depicted in Fig. 1. A service composition described by schema S1 consists of three services GetData, SendOffer, and ReceiveNotification that are sequentially ordered. Transactionality of SendOffer and ReceiveNotification is handled via a compensation sphere including SendOffer and ReceiveNotification. In case of an execution failure during ReceiveNotification, e.g., a timeout for a not-received incoming message, both SendOffer and ReceiveNotification are compensated by executing a compensation service CancelOffer.

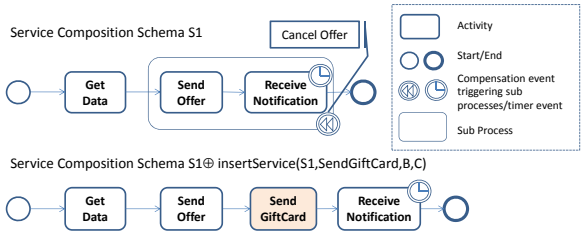

Figure 1. Service Composition and Change: Example (BPMN)

Assume now that due to a new marketing strategy, a gift card is to be sent directly after sending the offer. This is realized by inserting service SendGiftCard between services SendOffer and ReceiveNotification. The question now is how to handle the newly inserted service SendGiftCard with respect to the transactional behavior. Different options are conceivable, e.g., SendGiftCard becomes part of the compensation sphere or it stays outside the compensation sphere, i.e., marketing strategy says to not cancel gift cards. In any case, the change necessitates a reflection on the transactional behavior of the underlying service composition.

To approach the challenge of change in transactional service compositions, we address the following questions:
(1) Does a systematic classification for transactional service compositions exist that can be a basis for our work?
(2) How can we bring together the two concepts worlds of composition change and transactionality?
(3) Which formal criteria are required to control change in transactional compositions?
(4) What practical implications do arise?

Our approach allows for incremental checking of the effects of changes on transactional consistency.

To answer these research questions, this paper is structured as follows. Section II outlines the research methodology based on a mapping between the concepts worlds of transactional compositions and composition change. For the latter a study on typical change patterns exists [3]. For transactional compositions such a study is missing, hence we conduct a literature review. Its findings are analyzed (cf. Sect. III) resulting in a mapping of transaction and change concept worlds. The mapping provides the basis for algorithms which guide the process of change with respect that safeguard transactional properties of compositions when they are changed. The feasibility of the algorithms is evaluated by a discussion of the algorithm design and their applicability to practical scenarios in Sect. IV. Section V concludes this paper.

II. Methodology

As motivated by the example in Fig. 1, change and transactionality in service compositions cannot be handled in a separated manner. To analyze how both concept worlds, i.e., change and transactions, fit together in a systematic way, the basic idea is to map these concept worlds. We build the cross-product between transaction concepts and change concepts for a theoretical analysis. In detail, we first reduce the cross-product based on selection criteria for feasible combinations. These combinations lead to algorithms to deal with change in transactional service compositions. This way, proof of completeness is implied by construction (evaluation by construction). The applicability of the algorithms is discussed by means of real-world examples.

Which change and transaction concepts are relevant for service compositions? Studying literature, we found a collection of 14 typical change patterns that are relevant in the context of service compositions [3]. For the transactional concepts we did not find such a survey. Hence in the following section, we first analyze literature and define a concept world for transactional service compositions.

A. Transactional Concepts in Service Compositions

Grefen and Vonk [4] provide a structured analysis of how workflow and transactions can be combined. However, to the best of our knowledge, a structured catalog of transactional concepts in the context of service compositions is missing. Hence, we first analyze literature by collecting transactional properties in the areas of workflows, processes, and service compositions. These properties are then aggregated into transaction concepts such that they can be analyzed together with the change patterns.

We have conducted a literature search on google scholar using keywords *transactional workflow*, *transactional process*, *transactional properties*, *transactional pattern*, and *transaction model*. In addition, we have included papers from key projects in the area: Exotica, Meteor, WIDE, WAMO, ConTracts, and SARN. Finally, we have included papers found by backward chaining through reference lists. After filtering, this literature search has resulted in a collection of 66 publications[1].

The collection of publications has been analyzed for statements about transactional concepts. A transactional concept has been included into the result if it cannot be expressed by a combination of other concepts. An example for the latter would be the concepts *Service* and *Sub-service* for the granularity property, as they can be expressed by a combination of the concepts *Intra-level* for the structure property and *Sphere* for the granularity property. The structure concepts *Intra-level* and *Inter-level* also imply structural concepts such as *flat* (Intra-level) or *hierarchical* (Inter-level).

Further on, *recovery* plays a key role in many approaches as an operational mechanism to guarantee properties such as atomicity, isolation and consistency. Recovery mechanisms are also mentioned as transactional service properties / transactional patterns in the context of Web Services [6], [7]. Hence, the result of the literature search also includes selected recovery measures. Further recovery mechanisms such as consistent completion [7] can be expressed as combination of these measures.

Figure 2 illustrates the results of the literature analysis and puts concepts and recovery measures (as a basis for recovery mechanisms) into a relation. Specifically, transactional properties atomicity and isolation occur often in a either strict or relaxed manner resulting in combined properties *relaxed atomicity*, *relaxed isolation*, *strict atomicity*, and *strict isolation*. These concepts can be seen as sub-concepts of concepts atomicity/isolation and strict/relaxed as they inherit from these concepts, but extend them based on the combination. Recovery measures *retriable* and *compensatable* can then be seen as sub concepts of strict atomicity and relaxed atomicity as well as measures *Two-phase locked* and *Openly locked* as sub concepts of strict and relaxed isolation respectively. For reasons of clarity we omitted more specialized measures such as *pivot* for relaxed atomicity. However, they can be easily classified into the transaction concept world.

Another transaction concept mentioned in several works is the concept of *spheres*. Spheres enable the definition of

[1]The list of publications is available at http://is.ieis.tue.nl/staff/pgrefen/research/publications/pdf/ICWS2014_RG_LitList.pdf

transactional properties for a set of services. Assume, for example, that a set of services is included in a sphere and the transactional measure for the sphere is set to compensatable. This means that if the execution of one of the services within the sphere fails, all services within the sphere have to be compensated. On top of the concept of sphere, *regions* of transactional properties can be used within service compositions. The concept of region is less strict than the concept of sphere. It allows for referring to parts of the composition that possess the same or similar transactional pattern.

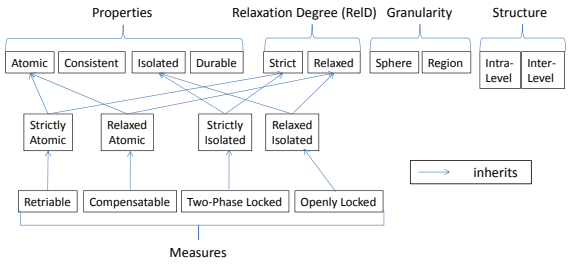

Figure 2. Transaction Concept World

Definition 1 provides a formalization of transaction concepts as basis for later considerations.

Definition 1 (Transaction Concept): A transaction concept builds upon the following dimensions:

- *Properties* :={Atomicity, Isolation, Consistency, Durability }
- *RelD* := {Strict, Relaxed}
- *Granularity* := {Sphere, Region}
- *Structure* := {Intra-Level, Inter-Level}

A transaction concept TC is defined as

$$TC \in \text{Properties} \times \text{RelD} \times \text{Granularity} \times \text{Structure}$$

Examples for transaction concepts are (Atomic, Relaxed, Sphere, Intra) and (Isolation, Strict, Region, Intra-level).

The concepts discussed above have their origins in the database domain and the domain of transactional workflows [8]. They have been mapped to the service-oriented domain, most specifically the WS Transaction standard [8]. This standard defines abstract transaction models that inherit from the concepts in Fig. 2: the Atomic Transaction model inherits from *Strictly Atomic* and *Strictly Isolated*, the Business Activity model inherits from *Relaxed Atomic* and *Relaxed Isolated*; both inherit from the *Sphere* granularity concept to denote transactional scope. The notion of service composition is reflected in our *Intra-level* structure concept, the notion of service nesting in our *Inter-level* structure concept.

B. Change Concepts in Service Compositions

Change and flexibility in service composition has been studied for more than a decade now. In [3], 14 well-defined change patterns for business processes are described. We transfer the concepts *activity* and *process fragment* as used in [3] to the corresponding concepts of *Service* and *ComposedService*. We understand a composed service as a process-oriented composition of services in a, for example, sequential, parallel, or alternative way. This corresponds to the concept of process fragment. Note that the considered change patterns operate on control flow only, i.e., data flow changes are not considered. Definition 2 provides a formalization of change concepts in service compositions.

Definition 2 (Change Concept): Change concepts are classified along the dimensions Change Type and Change Granularity with the following characteristics [3]:

- *ChangeType* := {Insert, Delete, Order-changing, Copy, Hierarchy-changing}
- *Change Granularity* :={Service, ComposedService}

A change concept CC is defined as combination of characteristics of dimensions ChangeType and ChangeGranularity. Formally:

$$CC \in \text{ChangeType} \times \text{ChangeGranularity}$$

An *Insert* operation adds a new service (*Service*) or composed service (*ComposedService*) into a composition model. Typical options here include serial insert between two subsequent services or parallel insert to an existing service or composed services. A *Delete* operation removes an service or composed service from the composition model. *Order-changing* operations refer to moving services or composed services from their old to a new position. Furthermore, serially ordered services or composed services within a composition model can be parallelized or vice versa. Finally, two services or composed services might be swapped.

Hierarchy-changing operations inline or extract composed services within a composition model, i.e., either a composed service is lifted up one hierarchy level in the composition model or a composed service is lifted down one hierarchy level. The *Copy* operation duplicates an existing service or composed service by inserting it at another position within the composition model. Examples for change concepts are (Insert, Service) and (Copy, ComposedService).

C. Analysis of Cross-Product between Transactional and Change Concepts

After summarizing the transactional and change concept worlds, the next step of the methodology is to map the concept worlds onto each other. More specifically, a full mapping between all concepts builds the basis for later considerations. For this, we determine the number of concepts for each of the worlds. Both worlds are constructed by different dimensions. Basically, the number of concepts unfolds by multiplication of characteristics of the dimensions. In detail:

- *Change concepts:* we encounter 4 characteristics of dimension change type (i.e., Insert, Delete, Order-changing, Copy) that are to be multiplied by the 2 characteristics of dimension change granularity (i.e., Service and ComposedService). Change operation Hierarchy-changing only operates on composed ser-

vices such that the number of change concepts turns out as $8 + 1 = 9$.

- *Transactional concepts:* along the 4 dimensions Properties, Granularity, Structure with 4, 2, 2, and 2 characteristics, respectively, the number of change concepts turns out as $4 * 2 * 2 * 2 = 32$

In order to reduce the full mapping of 288 combinations, we apply the following selection criteria:

1) If a concept can be expressed by a combination of other concepts, we remove it from the mapping. This results in a removal of change types *Order-changing* and *Copy* since they can be expressed by concepts *Insert* and *Delete*.
2) Regarding change type *hierarchy-changing* the only meaningful matching partner at the transactional concept side is *Inter-level*. We discuss this assumption further in Sect. III-C.
3) We omit consideration of concepts *durable* and *consistent* since we assume that all spheres and regions (Intra-level/Inter-level) shall deliver consistent and durable output.
4) We assume that relaxation usually holds likewise for atomicity and isolation as the case for Sagas [9].

The reduction results in 20 mappings. In Section III, the reduced mapping is elaborated into algorithms that prevent violations of transactionality of service compositions when applying change operations.

III. Strategies for Preventing Transactionality Violations when Applying Changes

In this section, we investigate the 20 combinations of transactional and change concepts for distilling strategies to safeguard transactional properties when changing service compositions. In order to formulate the algorithms for change in service compositions properly, we first introduce necessary notions in Sect. III-A and provide the algorithms in Sect. III-B.

A. Basic Notions: Transactionality, Sphere, Region

We start with definitions for service composition models:

Definition 3 (Service Compositions): A service composition model is defined as P:=(N, E) with N denotes the set of services and composed services and E denotes the set of control edges setting our the execution order between the services.

How can we transform the mapping between change and transactional concepts into algorithms? At first, a notion of transactionality is required. For this, we define transactionality as a projection of a selection of the enumeration of transactional concepts as in Def. 1. Definition 4 puts this into a formal frame.

Definition 4 (Atomicity and Isolation Level): Atomicity level AL and isolation level IL can be defined as cross-product between the associated transactional concept dimensions Properties and RelD:

- AL:= {Not Atomic, Relaxed Atomic, Strictly Atomic}
- IL:= {Not Isolated, Relaxed Isolated, Strictly Isolated}

Atomicity and isolation levels can each be ordered:

- Not Atomic $\prec$ Relaxed Atomic $\prec$ Strictly Atomic
- Not Isolated $\prec$ Relaxed Isolated $\prec$ Strictly Isolated

where $a \prec b$ means that a measure a is ensuring a less strict transactional property than b.

In the following, we apply the atomicity and isolation levels in order to define a transactionality notion for services, spheres, and regions. This covers the change concept dimension change granularity and transaction granularity.

Definition 5 (Transactionality of Services): Let P:=(N, E) be a service composition model. The transactionality of a service n $\in$ N is defined as: $trans : N \mapsto AL \times IL$

- trans(n1) < trans(n2) if n1.AL $\prec$ n2.AL $\wedge$ n1.IL $\prec$ n2.IL
- trans(n1) > trans(n2) if n2.AL $\prec$ n1.AL $\wedge$ n2.IL $\prec$ n1.IL
- trans(n1) = trans(n2) if n1.AL = n2.AL $\wedge$ n1.IL = n2.IL

Due to the assumption that both levels are either relaxed, strict, or not transactional, this is the full set of relations between the transactionality of two services.

As shown in our motivating example (cf. Fig. 1), services can be combined into spheres to express a transactional measure that refers to a set of services.

Definition 6 (Sphere): Let P:=(N, E) be a service composition model. Then a sphere S on P is defined as follows: $S := (N_S, trans_S)$ with $N_S \subseteq N$ and $trans_S \in AL \times IL$ with $\forall\ n \in N_S : trans(n) \geq trans_S$.

We do allow single-service spheres. We assume that spheres are exclusive, i.e., they do not overlap. In particular, spheres define transactional patterns.

As an additional concept to spheres, we introduce the notion of *region* within a service composition. A region groups a set of services within a composition with respect to their transactionality. With respect to the composition, regions can define transactional patterns. For contracting processes, for example, a two-phase transactional behavior is typical. Over the composition, the following transactional patterns (spheres) occur in sequence: 1) non-transactional sphere (information phase), 2) relaxed-transactional sphere (negotiation phase), 3) single-service strict-sphere (contracting phase), 4) relaxed-transactional-sphere (service phase). In a purchase order composition, for example, the transactionality of the involved services at the beginning of the composition is rather low, e.g., when searching for adequate offers no specific measure is necessary. Then the transactionality increases when customer and provider start to get into some contractual status by exchanging offers, notifications, invoices, and even payments. For services reflecting such kind of business logic, a higher transactionality becomes necessary, e.g., by compensating payments in case

of failures. At the end of the composition execution, the transactionality tends to go down again.

A region R forms a partitioning of a service composition P and is assigned a transactionality $trans_R \in AL \times IL$. Spheres form a partitioning of each region. We do allow single-sphere regions. For all definitions we assume that each element is at least as transactional as the element it is contained in.

B. Transactionality of Service Compositions under Change

In the previous section, definitions have been introduced that cover the concept mapping between change and transactionality. In the following, algorithms are presented for systematic perusal of the effects on transactional properties within a service composition after the application of a change. The algorithms are organized along the change type and transaction granularity. In the tables we use line numbering to explain and illustrate the individual steps.

Table I starts with checking insertion of a service n into a composition P at position pos. A first distinction is whether n is inserted into a sphere S or a region R. If neither is the case, no specific action regarding transactionality is necessary. In case n is inserted into a sphere, it is checked whether the transactionality of n is greater or equal the transactionality level of the sphere (1). If yes, n can become part of the sphere. If n is, for example, retriable and sphere S is compensatable, n can still be inserted and retried in case of a failure. If n becomes part of S and transactionality of S is relaxed, compensation specification of n has to be updated. If transactionality of n is lower than transactionality of S (2), n cannot immediately become part of S. The following checks follow the principle that we want n to become part of S and that we want to maintain the transactionality of S. Hence, we check whether transactionality of n could be lifted up to trans_S (3) or n could be replaced by another service n' with same functional properties and a minimum transactionality of trans_S (4). In both cases, n or n' respectively can become part of S accompanied with updates of compensation specification of S (if existing).

If lifting up $trans(n)$ to $trans_S$ or replacing n is not possible, it has to be checked whether S can be split by n into two spheres S_1 and S_2 (5). For the example in Fig. 1, we could argue that `Send Gift Card` splits sphere S=({`Send Offer, Receive Notification`}, (relaxed atomic, relaxed isolated)) into two spheres.

Note that for a sphere S its transactionality is defined by the lower bound of the transactionalities of all services contained with the sphere. Hence, if the transactionality of S can be decreased to the transactionality of newly inserted service n, then n can become part of S (by updating S and possible compensation specifications accordingly) (6). Otherwise, n is inserted and S and n have to marked as critical, i.e., in case of a failure during runtime, the information on S and n can be taken into consideration for recovery. We follow the premise that transactionality "problems" do not prevent changes from being executed.

When inserting a service n into a region R, again it is checked whether transactionality of n is greater or equal the transactionality of R (8). Then, n can become part of R. If trans(n) is lower than trans_R, the business semantics has to be checked. Assume, for example, that in a region with strict transactionality a service with relaxed or none transactionality is inserted. Strict transactionality is required for business-critical service. It can be desired that within such business-critical services, e.g., a service is inserted that collects, customers' feedback and hence has to be neither compensated nor handled in a strictly isolated manner.

When deleting a service from a sphere S, it can be checked whether n is the only service in S with minimal transactionality. As the transactionality of a sphere is constituted by the lower bound of the transactionalities of all services in S, deleting n leads to an increase of the transactionality of S to the minimum transactionality over the services remaining in S (9). Assume, for example, sphere S = ({`receive request`, `offer car`, `book hotel`}, (relaxed atomic, relaxed isolated)) with $trans$(`remove car offer`) = (relaxed atomic, relaxed isolated) and $trans$(`receive request`) = $trans$(`book hotel`) = (strictly atomic, strictly isolated). If now service `offer car` is deleted as is not cost-effective anymore, the service with minimal transactionality is deleted from S and the transactionality of S can be increased to (strictly atomic, strictly isolated). A region consists of spheres. Hence, deleting a service from a region also means to delete a service from a sphere. In more circumstances the transactional effects of the deletion of a service is handled at a sphere level and not at the region level.

Table II summarizes the algorithms for insertion into spheres and regions at granularity composed service. Note that changes at granularity service are specializations of changes at granularity composed service since we can see a single service as a composed service consisting of a single service. Hence, we only provide the algorithm for insertion of composed service here and omit the algorithm for deletion. How the latter works can be seen from the deletion of single services.

When inserting composed service CS, the same considerations as for inserting services hold, but for each of the services contained within CS. Hence, we distinguish between the following cases: (10) for all services of CS, their transactionality is greater or equal the transactionality of the sphere. Then CS can become part of sphere S. (10) summarizes those cases, for which a subset of services within CS exists for which their transactionality is lower than the transactionality of sphere S. In this case, it depends whether the services of CS can be either lifted up to transactionality of sphere S with respect to their transactionality

InsertService(P, n, pos) **inserts service** n **into composition** $P = (N, E)$ **at position** $pos = (pre, post)$
resulting in composition schema $P' := (N', E')$ **with** $N' = N \cup \{n\}$, $E' = (E \cup \{(pre, n), (n, post)\}) \setminus \{(pre, post)\}$

Sphere $S = (N_S, trans_S)$ *with* $N_S \subseteq N$	switch (1) trans(n) $\geq$ trans$_S$ $\Longrightarrow$ n becomes part of S, i.e., update S to S':=($N_S \cup$\{n\}, trans$_S$) $\wedge$ if $trans_S$ = (relaxed atomic, relaxed isolated) $\Longrightarrow$ adapt compensation specification of S (2) trans(n) $<$ trans$_S$: switch (3) trans(n) can be lifted up to trans$_S$ $\Longrightarrow$ n becomes part of S, i.e., update S to S':=($N_S \cup$\{n\}, trans$_S$); if trans$_S$ = (relaxed atomic, relaxed isolated) $\Longrightarrow$ adapt compensation specification of S (4) n can be replaced by service n' with trans(n') $\geq$ trans$_S$ $\Longrightarrow$ replace n by n'; n becomes part of S, i.e., update S to S':=($N_S \cup$\{n\}, trans$_S$); if trans$_S$ = (relaxed atomic, relaxed isolated) $\Longrightarrow$ adapt compensation specification of S (5) S separable $\Longrightarrow$ n splits S into two spheres S_1 and S_2 with $S_1 := (N_{S_1} = N_S \cap pred^*(P', n), trans_S)$ and $S_2 := (N_{S_2} = N_S \cap succ^*(P', n), trans_S)$ where $pred^*(P, n)$ / $succ^*(P, n)$ denotes all direct and indirect predecessor/successor services in P. (6) transactionality of S can be decreased to $trans(n) \Longrightarrow n$ becomes part of S i.e., update S to S':=($N_S \cup$\{n\}, $trans(n)$) $\wedge$ if $trans_{S'}$ = (relaxed atomic, relaxed isolated) $\Longrightarrow$ define compensation specification of S' (7) otherwise $\Longrightarrow$ insert n, mark n and S as critical
Region R	(8) trans(n) $\geq$ trans(R) $\Longrightarrow$ check business semantics

DeleteService(P, n) **deletes service** n **from composition** $P = (N, E)$ **resulting in composition schema** $P' := (N', E')$
with $N' := N \setminus \{n\}$ **and** $E' := (E \setminus \{(pred(P, n), n), (n, succ(P, n))\}) \cup \{(pred(P, n), succ(P, n))\}$
where $pred(P, n)$ **/** $succ(P, n)$ **denotes the direct predecessor / successor service of** n **in** P**.**

Sphere $S = (N_S, trans_S)$ *with* $N_S \subseteq N$	switch (9) $\forall\, n' \in N \setminus \{n\} : trans(n') > trans(n)$ delete n and increase transactionality of S to $trans_{min} := min(\{trans(n') \mid n' \in N \setminus \{n\}\})$ i.e., S is updated to $S' = (N \setminus \{n\}, trans_{min}))$ otherwise delete n

Table I
ALGORITHMS FOR CHANGE GRANULARITY *Service*

InsertComposedService(P, CS, pos) **inserts composed service** $CS := (N_{CS}, E_{CS})$ **into composition** $P = (N, E)$ **at position** $pos = (pre, post)$
resulting in composition schema $P' := (N', E')$ **with** $N' = N \cup N_{CS}$, $E' = (E \cup E_{CS} \cup \{(pre, n_{start}), (n_{end}, post)\}) \setminus \{(pre, post)\}$
where n_{start} **denotes the initial service of** CS **and** n_{end} **the final service**†

Sphere $S = (N_S, trans_S)$ *with* $N_S \subseteq N$	switch (10) $\forall\, n \in N_{CS}$: trans(n) $\geq$ trans$_S$ $\Longrightarrow$ CS becomes part of S, i.e., update S to S':=($N_S \cup$ N$_{CS}$, trans$_S$) $\wedge$ if $trans_S$ = (relaxed atomic, relaxed isolated) $\Longrightarrow$ adapt compensation specification of S (11) $\exists\, \tilde{N_{CS}} \subseteq N_{CS}$ with $\forall\, n \in \tilde{N_{CS}}$: trans(n) $<$ trans$_S$ switch (12) $\forall\, n \in \tilde{N_{CS}}$: (trans(n) can be lifted up to trans$_S$ $\vee$ n can be replaced by service n' with trans(n')$\geq$ trans$_S$: $\Longrightarrow$ lift up or replace n; CS becomes part of S, i.e., update S to S'; adapt compensation specification of S (13) $\tilde{N_{CS}} = N_{CS} \wedge$ S separable $\Longrightarrow$ CS splits S into two spheres S_1 and S_2 with $S_1 := (N_{S_1} = N_S \cap pred^*(P', n_{start}), trans_S)$ and $S_2 = (N_{S_2} := N_S \cap succ^*(P', n_{end}), trans_S)$ (14) transactionality of S can be decreased to $min(\{trans(m) \mid m \in N_{CS}\}) \Longrightarrow CS$ becomes part of S i.e., update S to S':=($N_S \cup$ N$_{CS}$, $min(\{trans(m) \mid m \in N_{CS}\})$) $\wedge$ if $trans_{S'}$ = (relaxed atomic, relaxed isolated) $\Longrightarrow$ define compensation specification of S' (15) otherwise $\Longrightarrow$ Insert CS, mark S and CS as critical
Region R	(16) $\exists\, \tilde{N_{CS}} \subseteq N_{CS}$ with $\forall\, n \in N'$: trans(n) $<$ trans$_R$ $\Longrightarrow$ check business semantics

†We assume exactly one initial (on incoming edges) and one final service (no outgoing edges) within a composition.

Table II
ALGORITHMS FOR CHANGE GRANULARITY *ComposedService*

or replaced by services with minimum transactionality of S.

Consider the scenario displayed in Fig. 3 where composed service CS is inserted into sphere S with $trans_S$ = (relaxed atomic, relaxed isolated). If `Receive Notification` fails after a timeout `Send Offer` is compensated by executing compensation service `Cancel Offer`. Both services in CS have a lower transactionality than $trans_S$, hence cannot become part of S instantly. In Case (12), we assume that transactionality of `Send Quiz` and `Receive Quiz` can be lifted up to $trans_S$. Then CS can become part of S and compensation of S is updated to executing a sequence of compensation services `Cancel Offer`, `Cancel Quiz`, `Send Substitute Gift`. In Case

(13), transactionality of services in CS cannot be lifted up and they cannot be replaced either, but S is separable. Then S is split into two spheres $S1$ and $S2$ and the compensation specification for $S1$ and $S2$ are updated.

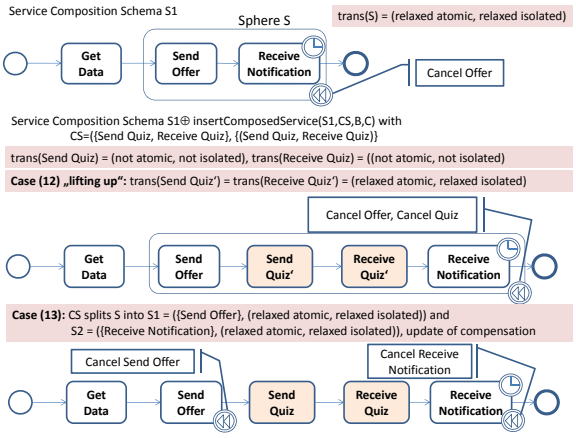

Figure 3. Inserting a Composed Service: Example (in BPMN Notation)

If S is not separable (13), CS is inserted into S and both are marked as critical. For inserting composed service CS into region R, if there are services contained in CS with lower transactionality than trans_R, it should be checked whether this contradicts the intended business semantics.

C. Transaction Concept and Hierarchy-changing operations

The algorithms that we have discussed above are focused on changes that take place at a single service level, i.e., relate to the Intra-level concept of Fig. 2. The class not yet discussed is that of changes that take place at two levels of a nested service, i.e., hierarchy-changing operations related to the Inter-level concept of Fig. 2. Conceptually, hierarchy-changing operations can be modeled by a complex combination of three insert and delete operations. For example, an Inline operation requires deleting a placeholder for a sub-service, inserting the composed service that is the sub-service at the position of the placeholder, and deleting the sub-service at the lower level. The complexity is caused by the fact that these three operations take place within a multi-level transaction construct. This multi-level transaction construct can be homogeneous or heterogeneous across the levels. If it is homogeneous, all levels obey the same transaction constructs. We find this for example in the closed and open nested transaction models from the database world [8], which cater for strict respectively relaxed transactionality (cf. Def. 1 and Fig. 2). If it is heterogeneous, each level has its own transactional constructs (like for example in the WIDE model, which has two different levels [8]). Obviously, heterogeneity increases complexity here.

IV. Evaluation

This paper performs an analysis of flexibility requirements in transactional service composition and provides algorithms to fulfill these requirements. To evaluate the contribution of this paper, we look at both the requirements analysis and the algorithm design.

For the requirements analysis, we consider its completeness the most important quality aspect: to provide a usable analysis, we ensure that all relevant cases are indeed covered. We have guaranteed this quality aspect by construction of the analysis. As a first step, we have started with full and independent enumerations of both transaction characteristics and change patterns as identified in well-recognized literature. As the next step, we have taken the cross-product of these two enumerations to find all relevant change primitives to transactional service composition constructs.

For the design of the provided algorithms, we consider correctness as the most important quality aspect: the algorithms should indeed guarantee correct transactional behavior of service composition when changes are applied. As this paper makes the design of the algorithms at an abstract level, we cannot *field test* them in application environments (this is for future work). Providing formal proof of the correctness of the algorithms would require complex mathematical formalisms (such as dynamic logic), where the chance of error in the proof would be higher than chance in the algorithm under scrutiny. Thus, we have so far resorted to *operational desk* examination" of the algorithms.

Finally, the application of the approach in real-world scenarios is a crucial point to discuss. Transactionality of complex services is generally considered important to guarantee robustness of the execution of these services. The ability to easily change complex services is also generally considered essential to achieve the business agility required by current volatile markets. The lack of approaches combining these two aspects, however, currently hampers broad application in practice. We observe research efforts, however, that contribute towards the development of this application.

The CrossFlow project [4] has developed an approach that combines agility and transactionality, but heavily restricts the kind of changes allowed to obtain agility (only dynamic substitution of sub-services for outsourcing is supported). This project developed applications in the telecom/logistics and insurance industry domains. Projects like CrossWork [10], ADVENTURE [1], and GET Service [2] do pay explicit attention to more advanced kinds of changes (design-time and/or run-time). They cover manufacturing and logistics as their application domains. The approaches developed in CrossWork and ADVENTURE do not cover transactionality (even though transactionality is considered important in their domain). The approach currently under development in GET Service does include transactional constructs, most notably measures that are indicated as Compensatable in Fig. 2.

Figure 4 sketches different implementation options for supporting flexibility in transactional service compositions comparable to the taxonomy of options for transactional

workflow support in [4]. If the process engine does not support in-built transactionality or flexibility, but enables the implementation of plug-ins or additional services, it can be extended by a transactional engine or change extension respectively. The WIDE project has demonstrated the feasability of extending an existing process engine with a transaction engine: the FORO process engine has been extended with a two-level transaction engine [11]. The Cloud Process Execution Engine CPEE (cpee.org) is a lightweight process engine, but with change extensions in terms of, e.g., a repair service [12]. Systems such as AristaFlow (www.aristaflow.com) feature built-in change extensions and can be extended by transaction engines via their API. In principle, limited forms of transactionality (specifically measures shown as Retriable and Compensatable in Fig. 2) can be also implemented through exception handling functionality offered by several commercial systems such as jBPM, WebSphere, or Oracle BPM and academic tools such as CPEE. Most commercial systems offer only restricted support for flexibility in terms of supporting the change patterns set out in [3]. The CPEE process engine provides a suitable platform for the concepts presented in this paper by implementing a combination of flexibility and enhanced exception handling towards full transactional support.

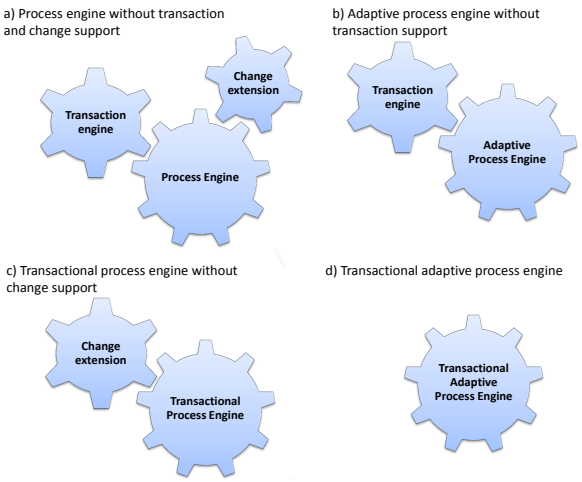

Figure 4. Different architectural options for flexibility and transactionality

V. Summary and Outlook

In this paper, we have developed the first version of an approach to combine transactionality and change for composed services. This combination is essential to guarantee both robustness and agility in the execution of modern-day, service-oriented business processes. The conceptual core of the approach is formed by a mapping of concepts from the transaction world and the process change world, applied to the service domain. The operational part of the approach is laid down in algorithms that guarantee transactional correctness of services under change. Extensions to this paper can be found in several directions. Firstly, the algorithms need to be further completed for the hierarchy-related change operations. Further refinement of described algorithms is possible to allow for optimizations. Secondly, the algorithms can be integrated in practical change approaches or even service/process evolution tools. The detailed level at which the algorithms are described supports this task. Thirdly, future work will incorporate transactional concepts *durable* and *consistent* which were not considered in this paper for simplicity reasons as well as more specific transactional models, e.g., multi-level transactions. Finally, we will pay attention to correctness considerations of our approach.

References

[1] S. Schulte, D. Schuller, R. Steinmetz, and S. Abels, "Plug-and-play virtual factories," *IEEE Internet Comp.*, vol. 16, no. 5, pp. 78–82, 2012.

[2] P. Grefen and R. Dijkman, "Hybrid control of supply chains: a structured exploration from a systems perspective," *Int'l J. of Prod. Mgmt. and Eng.*, vol. 1, no. 13, pp. 39–54, 2013.

[3] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems." *Data and Knowledge Engineering*, vol. 66, no. 3, pp. 438–466, 2008.

[4] P. Grefen and J. Vonk, "A taxonomy of transactional workflow support," *Int'l Journal of Coop. Information Systems*, vol. 15, no. 01, pp. 87–118, 2006.

[5] A. Sheth and M. Rusinkiewicz, "On transactional workflows," *Data Engineering Bulletin*, pp. 20–25, 1993.

[6] S. Bhiri, K. Gaaloul, O. Perrin, and C. Godart, "Overview of transactional patterns: Combining workflow flexibility and transactional reliability for composite web services," in *Business Process Management*, 2005, pp. 440–445.

[7] K. Hahn and H. Schweppe, "Analysis of non-functional service properties for transactional workflow management," in *Workshop Non Functional Prop. and Service Level Agreements in Service Oriented Comp.*, 2008.

[8] T. Wang, B. Kratz, J. Vonk, and P. Grefen, "A survey on the history of transaction management," *Distributed and Parallel Databases*, vol. 23, no. 3, pp. 235–270, 2008.

[9] H. Garcia-Molina and K. Salem, "Sagas," in *ACM SIGMOD Int'l Conf. on Management of data*, 1987, pp. 249–259.

[10] P. G. et al., "Internet-based support for process-oriented instant virtual enterprises," *IEEE Internet Computing*, vol. 13, no. 6, pp. 65–73, 2009.

[11] P. Grefen, J. Vonk, E. Boertjes, and P. Apers, "Two-layer transaction management for workflow management applications," in *Database and Exp. Syst. Appls.*, 1997, pp. 430–439.

[12] G. Stürmer, J. Mangler, and E. Schikuta, "Building a modular service oriented workflow engine," in *Service-Oriented Comp. and Appls., 2009 IEEE Int'l Conf.* IEEE, 2009, pp. 1–4.