„The final publication is available at Springer via
https://doi.org/10.1007/978-3-319-69462-7_22“.

Dynamic Change Propagation for Process Choreography Instances

Conrad Indiono and Stefanie Rinderle-Ma

University of Vienna, Faculty of Computer Science, Vienna, Austria
{firstname.lastname}@univie.ac.at

Abstract. Business process collaborations realize value chains between different partners and can be implemented by so called process choreographies. Change has become a major driver for costly (re-)negotiations between the participants. Static a priori prediction models exist to calculate the feasibility of a change request prior to negotiation. However, the dynamic or behavioral aspect of choreography changes at the choreography instance level has not been investigated yet, i.e., the question whether a process choreography instance is compliant with the change request and hence allows for acceptance of the change request. This work takes the dynamic perspective and analyzes the impact of a single change request from one partner on the entire (distributed) choreography based on the notion of change regions and public check points. Change strategies are elaborated to ensure choreography instance state compliance. One transaction-based approach is specified using rollback regions. It identifies probabilistically the set of activity nodes to be compensated at all levels of the business collaboration to ensure state compliance. The technical evaluation enables observing the properties of the rollback region.

流程编排谈判

静态的先验预测模型用于在协商之前计算变更请求的可行性。然而，在编排实例级别上编排更改的动态或行为方面尚未研究，即，流程编排实例是否符合更改请求并因此允许接受更改请求的问题。

该工作从动态的角度出发，基于更改区域和公共检查点的概念，分析来自一个合作伙伴的单个更改请求对整个（分布式）编排的影响。

制定变更策略以确保编排实例状态符合性。使用回滚区域指定一种基于事务的方法。它概率地标识要在业务协作的所有级别补偿的活动节点集，以确保状态遵从性。技术评估能够观察回滚区域的特性。

Keywords: collaborative business processes, dynamic change

1 Introduction

Process choreographies implement business process collaborations between different partners in order to reach a joint business goal. Examples stem from the manufacturing or logistics domain. “dynamism is the basis for agility” [10] presents an omnipresent challenge to handling change in process choreographies. Though some work on the static perspective of process choreography change exists, e.g., [12,4], approaches to deal with the dynamic perspective of choreography change are almost entirely missing [17]. This paper tackles this research gap by considering the actual execution state of each participating partner’s process instance to estimate the impact of a change request on the global business choreography. The distributed nature of process choreographies poses particular challenges as it introduces privacy elements, disallowing full insight into direct or indirect partners’ execution environments. This means that on the static model level, partners do not know what exact activities are scheduled to be executed in between the interaction activities. Similarly on the dynamic level, the concrete current execution state as well as the historic execution log of each process instance is not fully known for partners.

例子来自制造或物流领域。
“动态性是敏捷的基础”
[10]对处理流程编排中的更改提出了无所不在的挑战。

本文通过考虑每个参与伙伴的过程实例的实际执行状态来评估变更请求对全球业务编排的影响，从而解决了这一研究差距。

流程编排的分布式性质带来了特殊的挑战，因为它引入了隐私元素，使得无法全面了解直接或间接合作伙伴的执行环境。这意味着在静态模型级别，合作伙伴不知道在交互活动之间计划执行什么确切的活动。类似地，在动态级别上，合作伙伴并不完全知道每个流程实例的具体当前执行状态以及历史执行日志。

这项工作扩展了已经建立的在过程编排中的静态变化领域的工作[5]。这里，编排更改被描述为一个由几个步骤组成的过程，例如检查更改正确性（静态和动态）以及与合作伙伴协商更改请求。后者的原因在于，在完全分布式的环境中，不能对伙伴强加改变，而是必须达成一致。因此，事先估计这种谈判的成本可能是有益的[8]，特别是因为谈判可能变得昂贵。

多步骤过程[6]。在协商发生之前执行的所谓变化预测步骤中实现成本估计。初始变更请求的影响是根据允许在可用变更请求集合之间进行比较的标准化评分值来估计的。

This work extends already established work in the area of static change in process choreographies [5]. Here choreography change is described as a process consisting of several steps such as checking change correctness (static and dynamic) and negotiating the change request with the partners. The reason for the latter is that in a fully distributed setting changes cannot be imposed on partners, but have to be agreed on. Hence it can be beneficiary to estimate the costs of such a negotiation beforehand [8], particularly as negotiation might become a costly multi-step process [6]. Cost estimation is realized within the so called change prediction step that is executed before the negotiation takes place. The impact of the initial change request is estimated in terms of a normalized score value that allows comparison between the set of available change requests.

这里，编排更改被描述为一个由几个步骤组成的过程，例如检查更改正确性（静态和动态）以及与合作伙伴协商更改请求。

So far, merely the static costs of a change have been considered. The dynamic costs have only been considered in an abstracted manner, i.e., based on the choreography model level using execution probabilities [7]. In this work, the goal is to determine the *change region* of a change for different partners, i.e., the region in which a change might be critical with respect to the choreography instance state. Change regions have been proposed for business processes, but not for process choreographies. Based on the change region it can be determined whether or not a partner can apply the change right away. Further on, it can be considered whether or not the application of change compensation actions such as rollback might be meaningful in order to realize the change. In particular, the costs for such actions can be taken into consideration when estimating the change impact. This research goal is reflected in the following research questions:

在此工作中，目标是确定不同伙伴的更改的改变区域，即，其中更改对于编排实例状态可能是关键的区域。已经为业务流程提出了更改区域，但是没有为流程编排提出更改区域。

基于更改区域，可以确定合作伙伴是否能够立即应用更改。

此外，可以考虑应用诸如回滚之类的更改补偿动作是否对实现更改有意义。特别地，在估计变更影响时可以考虑这些行动的成本。本文的研究目标体现在以下几个研究问题上：

- How to identify change regions for choreography changes, i.e., those regions where changing running choreography instances could lead to an inconsistent state?
- Once the change regions are identified, how to estimate the total impact of a change request?

-如何为编排更改识别更改区域，即，那些更改正在运行的编排实例可能导致不一致状态的区域？

-一旦确定了变更区域，如何评估变更请求的总体影响？

The questions are approached following the design science method (cf. [20]). The relevance of the topic is underpinned by literature [17,8]. The created artifacts comprise definitions of fundamental concepts such as change regions as well as algorithms, i.e., algorithms that determine the rollback regions. The approach is evaluated based on a proof-of-concept implementation. Thereby the paper is structured as follows: In Sect. 2, basic terminology is introduced: Section 3 provides new concepts on choreography instance change. Section 4 provides change propagation strategies including compensation actions based on rollback. The evaluation is presented in Sect. 5, followed by related work in Sect. 6 and a conclusion in Sect. 7.

2 Motivating Example and Fundamentals

pc机 机箱制造案例

As an illustrative example, we study a pc case manufacturing use case consisting of these roles: pc case manufacturer, buyer, metal supplier and coloring supplier. The choreography model of this business collaboration is depicted in Fig. 1 and can be divided into the following areas: choosing a product type (F_1), checking

PC机箱制造商、买方、金属供应商和着色供应商。

选择产品类型（F1），检查原始资源可用性（F2）、抽象制造子过程（F4）和最终交付过程（F3）。

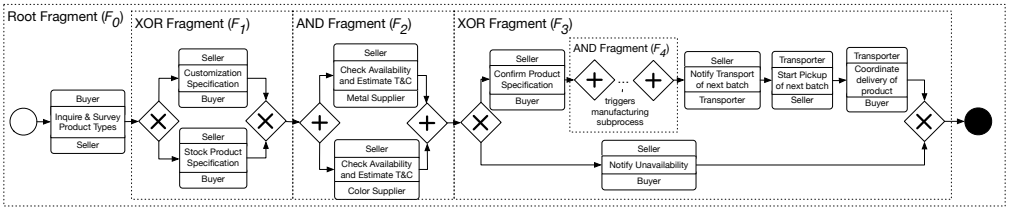


Fig. 1. Motivating Example: PC Case Manufacturing Use Case

for raw resource availability (F_2), the abstracted manufacturing subprocess (F_4) and finally the delivery process (F_3).

考虑制造商在（f1）中引入的更改，这增加了激光蚀刻自定义设计的选项。

Consider a change introduced by the manufacturer in (F_1), which adds an option for laser etching custom designs. An associated change is also implemented in Fragment F_4, which performs the activity for the laser etching. How can we estimate the impact these changes have on the other partners? From previous work [7] we are able to estimate this impact using execution probabilities and an abstract adaptation cost. In this work, we improve the change region to determine the beginning of the change and define rollback regions to mark all possible maximal activities process instances are able to progress to. A concrete transaction based adaptation is used, which informs the cost calculation.

蚀刻

在此工作中，我们改进了变更区域以确定变更的开始，并定义了回滚区域，以标记流程实例能够进行到的所有可能的最大活动。使用基于具体事务的自适应，这通知了成本计算。

Definition 1. *[**Choreography**] Adapted from [5], we define a choreography $\mathcal{C}$ as a tuple ($\mathcal{G}$, $\mathcal{P}$, Π, $\mathcal{L}$), where*

- *$\mathcal{G}$ is the choreography model (i.e., Fig. 1).*
- *$\mathcal{P}$ is the set of all participating partners.*
- *$\Pi = \{\pi_p\}_{p \in \mathcal{P}}$ is the set of all private models.*
- *$\mathcal{L} = \{l_p\}_{p \in \mathcal{P}}$ is the set of all public models.*

细化过程结构树（RPST）[19]是结构化的表示业务流程模型的方法。它将模型划分为几个片段，每个片段都满足单个入口、单个出口属性（SESE）。片段是表示序列内的单个交互的琐碎片段（树中的叶子），或者是可以是XOR或AND片段并包含其他子片段的复杂片段。

The Refined Process Structure Tree (RPST) [19] is a structured approach for representing business process models. It divides the model into several fragments, each fulfilling the single entry, single exit property (SESE). A fragment is either a trivial one (a leaf in the tree), representing a single interaction inside a sequence, or a complex one that can be either an XOR or AND fragment and contain further sub fragments. Fig. 2 shows the pc case manufacturing use case as a collapsed RPST. It only shows the sequence under the root fragment (F_0), consisting of leaf nodes (trivial fragments, e.g. interactions) and sub trees representing either XOR or AND fragments (e.g., F_1, F_2, F_3). Note that the actual manufacturing process F_4 is not shown in the root sequence due to it being embedded under Fragment F_3.

图2显示了作为折叠RPST的PC用例制造用例。它只显示根片段(F0)下的序列，由叶节点(微小片段，例如相互作用)和表示异或或AND片段(例如，F1、F2、F3)的子树组成。

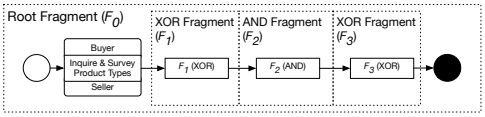


Fig. 2. Top-level RPST Tree of the PC Case Manufacturing Use Case

Definition 2. ***[Change Patterns]*** *from [5], we define the following change patterns:*

场景解释

$$\begin{aligned} ChangePattern ::= & \; REPLACE(oldFragment, newFragment) \\ & \mid DELETE(fragment) \\ & \mid INSERT(fragment, how, pred, succ) \\ how ::= & \; Parallel \mid Choice \mid Sequence \end{aligned}$$

改变模式有3种更改模式

实例：激光蚀刻的两个实施方案在图1中，“replace（F0，fnew）改变的模式将被用来，在fnew是基于下面的F0和F1片段的变化：有一个新的卖方和买方之间的相互作用对提供的customization激光蚀刻。F4冰片段的修饰，在激光蚀刻（e.g互动之特异性，与相应的供应商）是增加的。

Example: To implement the laser etching scenario in Fig. 1, the $REPLACE(F_0, F_{new})$ change pattern would be used, where F_{new} is based on F_0, with the following changes: Fragment F_1 has a new interaction between seller and buyer for offering the laser etching customization. Fragment F_4 is modified, where laser etching specific interactions (e.g, with the corresponding supplier) are added.

Definition 3. ***[Business Process Instance]***
A business process instance is a tuple (id, m, r, as), where the following holds: $m \in \Pi \wedge r \in \mathcal{P}$. as is a set of tuples each of the form (a, s), where $a \in m \wedge s \in \{inactive, activated, running, aborted, completed\}$ and id is a unique identifier that is mapped to the busines process instance.

非活动、激活、运行、中止、完成

We extend the definition of *Choreography* from Def. 1 to be: $\mathcal{C} = (\mathcal{G}, \mathcal{P}, \Pi, \mathcal{L}, \Pi^{\mathcal{I}})$, where $\Pi^{\mathcal{I}}$ is a set of business process instances (see Def. 3), which are running *instantiations* of a corresponding business process model and each activity comprising the process model having an associated state.

它运行对应业务流程模型的实例，以及包含具有关联状态的流程模型的每个活动。

它们正在运行相应的业务流程模型的实例化以及包括具有关联状态的流程模型的每个活动。

3 Dynamic Change Propagation Concepts

To determine the dynamic impacts of a change operation, two components are required: (1) identify the specific nodes that mark the beginning of the change: the *change region* and (2) identify the relative position of all business process instances ($\Pi^{\mathcal{I}}$) to that *change region*: *state compliance*. Having that relative position gives an initial indication whether there could be disproportional costs involved for the proposed change. Those process instances having their state before the *change region* are state compliant, and are thus generally non-problematic in regards to implementing the change. Those process instances whose running activities are already past the *change region* violate state compliance, and change implementation may potentially become problematic.

1）识别标记变更开始的特定节点：变更区域，（2）识别所有业务流程实例（I）与该变更区域的相对位置：状态遵从性。有了这个相对位置就初步表明了提议的改变是否涉及不成比例的成本。

速象

那些在变更区域之前具有状态的流程实例是状态兼容的，因此在实现变更方面一般没有问题。运行活动已经超过变更区域的那些流程实例违反了状态遵从性，并且变更实现可能成为问题。

3.1 Change Region

As the basis for determining the *change region*, we can start with the *smallest fragment* (see Def. 4), which returns the surrounding RPST fragment containing all supplied nodes. Example: the *smallest fragment* that contains { *Inquire & Survey Product Types* } would be the interaction itself, because a leaf node inside a RPST fragment is a fragment. The *smallest fragment* containing { *Customization Specification*, *Notify Transport of next batch* } is the root fragment F_0.

作为确定变更区域的基础，我们可以从最小的片段开始（参见def）。4），它返回包含所有提供的节点的周围RPST片段。示例：包含查询和调查产品类型的最小片段将是交互本身，因为RPST片段内的叶节点是片段。包含定制规范、通知下一批传输的最小片段是根片段f0。

Algorithm 1: Change Region Algorithm

Input:
δ - a change pattern (see Def. 2)
Begin
if $\delta.type \in \{INSERT, DELETE\}$ **then**
  return $\delta.fragment$
else
  $ins \leftarrow \delta.newFragment \setminus \delta.oldFragment$
  $del \leftarrow \delta.oldFragment \setminus \delta.newFragment$
  $F_{root} \leftarrow \alpha(ins + del)$; $F_{min} \leftarrow \emptyset$
  foreach *node in* $ins + del$ **do**
    if $F_{min} = \emptyset \vee distance(F_{root}, node) < distance(F_{root}, F_{min})$ **then**
      $F_{min} \leftarrow node$
  return F_{min}

Definition 4. ***[Smallest Fragment]*** *(from [5])*
Let σ be a public model and S be a set of nodes corresponding to σ. Then: $\alpha_\sigma(S)$ returns the smallest fragment in model σ that contains all nodes from S. Formally: $\alpha_\sigma(S) = \arg\min_{size(F)}\{F \in \sigma \mid \forall n \in S, n \in F\}$

足够的

While the *smallest fragment* is sufficient to determine the beginning of a change in the cases where the change pattern $\in \{INSERT, DELETE\}$, it does not hold for $REPLACE$. In the case of $INSERT$, we know a new RPST fragment is being inserted between *pred* and *succ* (see Def. 2). Thus taking the fragment itself to mark the beginning of the change is feasible. The same concept holds for $DELETE$ change patterns. However, in the case where we have a $REPLACE$ change pattern as in the laser etching example, the *smallest fragment* would return the root fragment F_0 as the surrounding RPST fragment. The beginning of that root fragment does not mark the real change, which is critical for determining the necessity of adaptation before implementing the change. Generally, adaptation is required once the first node marking the change enters a state past *activated* (i.e., *running*, *aborted*, *completed*). Recall that in the laser etching example we have two RPST fragments being modified inside the $REPLACE$ operation: F_1 by adding an interaction *offer laser etching* and F_4 which adds new interactions with suppliers related to the new offer. To identify the concrete change region a simple *smallest fragment* call is not sufficient. We need to further identify the fragment nearest to the start node that has been changed and set that as the beginning of the change region. To do so, we first need the fragments that have been inserted (in both F_1 and F_4) and then find the one with the shortest path from the start of the surrounding fragment: this fragment (F_1) marks the beginning of the change region. Algorithm 1 specifies such a *change region*.

然而，在我们有一个替换的变化模式的情况下，如激光蚀刻的例子，最小的片段将返回根片段f0作为周围的rpst片段。根片段的开始并不标志着真正的变更，这对于在实现变更之前确定适应的必要性至关重要。

一般来说，一旦第一个节点标记了更改，就需要进行自适应

进入激活后的状态（即运行、中止、完成）。回想一下，在激光蚀刻示例中，我们在替换操作中修改了两个RPST片段：F1添加了一个提供激光蚀刻的交互，而F4添加了与新报价相关的供应商的新交互。要识别具体的更改区域，一个简单的最小片段调用不是足够的。

我们需要进一步识别离已更改的开始节点最近的片段，并将其设置为更改区域的开始。为此，我们首先需要插入的片段（在f1和f4中），然后找到从周围片段开始处具有最短路径的片段：此片段（f1）标记更改区域的开始。算法1指定了这样一个变化区域。

3.2 State Compliance

The second problem with determining dynamic change impact is related to the following: while each partner is able to calculate the relative position of their

确定动态变更影响的第二个问题与以下内容有关：虽然每个合作伙伴都能够计算其相对位置流程实例来自更改区域，

由于隐私问题，合作伙伴只能估计此位置。

通过使用自己的私人执行日志，合作伙伴能够自己识别别州的合规性[13]。对于合作伙伴评估直接合作伙伴的状态遵从性，公共交互可以用作检查点。我们知道，一旦我们完成了与另一个伙伴的交互，另一个伙伴在他们自己的私有过程中同样完成了相同的交互。

process instances from the change region, partners are only able to estimate this location due to privacy issues. Using their own private execution log, partners are able to discern state compliance on their own [13]. For partners to estimate direct partners' state compliance, public interactions can be used as checkpoints. We know that once we have finished an interaction with another partner, the other partner has equally finished the same interaction in their own private process. Execution paths from that point on cannot not be accurately determined by direct partners as it is possible that the partner progresses in such a way that further interaction with the same partner never happens. A comprehensive monitoring approach at the cost of privacy would be required if indirect partner state tracking is required, as these would involve active state reports due to lack of direct interaction points. Estimating state compliance for direct partners can be achieved by taking a private execution log and abstracting by public activities or interactions where the direct partner is involved. The last activity can be seen as the *current state*, even though from the perspective of the private model, the actual progress might be more advanced. Only through passing public checkpoints can direct partners track state compliance.

从那一点开始的执行路径不能由直接合作伙伴准确确定，因为合作伙伴可能以这样一种方式进行，即永远不会发生与同一合作伙伴的进一步交互。

如果需要间接的合作伙伴状态跟踪，则需要以隐私为代价的全面监控方法，因为由于缺少直接交互点，这些方法将涉及活动状态报告。

评估直接合作伙伴的状态遵从性可以通过获取私有执行日志和通过公共活动或直接合作伙伴参与的交互进行抽象来实现。
最后一个活动可以看作是当前状态，即使从私有模型的角度来看，实际进度可能会更高级。只有通过公共检查点，合作伙伴才能跟踪状态遵从性。

Definition 5. ***[Choreography Instance]***
We extend the definition of *Choreography* from Def. 3 to include *choreography instances*: $\mathcal{C} = (\mathcal{G}, \mathcal{P}, \Pi, \mathcal{L}, \Pi^{\mathcal{I}}, \kappa, \kappa', \mathcal{G}^{\mathcal{I}})$, where
- $\kappa : corr_{id} \rightarrow \{(id, m, r, as)\}$ is a total function that maps a unique correlation identifier to a set of business process instances working on the same business case (see Def. 3), each assigned a unique private case id, with $corr_{id} \in \mathcal{G}^{\mathcal{I}}$.
- $\kappa' : (id, m, r, as) \rightarrow corr_{id}$, a non-injective surjective function that maps a unique business process instance to a correlation identifier.
- $\mathcal{G}^{\mathcal{I}}$ is the set of all active unique correlation identifiers, each representing a single choreography instance.

In our example, a business case is started by the buyer who intends to have a product manufactured. That buyer process starts with its own case id to track the product. The seller creates a unique case id once a product buying confirmation comes in and uses the same id when contacting suppliers. The two suppliers create their individual private case ids. Finally, the transporter is contacted by the seller, who may create a separate case id to handle the shipment. Even though many different case ids exists, the same product is being handled and referenced. The correlation id groups these unique case ids to the same business case. Thus a choreography instance groups together all partners and their private process instances working on the same business case. The known public markings are used to set the states of each public interaction activity. However, for privacy reasons, the partner who has direct connections with the most partners has the most accurate view. The choreography instance from the perspective of the buyer is more limited compared to the one of the buyer.

在我们的示例中，一个业务案例是由打算生产产品的买方启动的。买方流程从其自己的案例ID开始跟踪产品。一旦收到产品购买确认书，卖家就会创建一个唯一的案例ID，并在联系供应商时使用相同的ID。

跟踪

最后，卖方与运输方联系，后者可以创建单独的案例ID来处理装运。尽管存在许多不同的案例ID，但正在处理和引用同一个产品。

相关ID将这些唯一的案例ID分组到同一个业务案例。因此，编排实例将所有合作伙伴及其处理同一业务案例的私有流程实例分组在一起。

已知的公共标记用于设置每个公共交互活动的状态。但是，出于隐私考虑，与多数合作伙伴有直接联系的合作伙伴拥有最准确的视图。从买方的角度来看，编排实例比买方的更为有限。

聚合编排实例

检索

An *aggregated choreography instance* can be created (illustrated in Fig. 3) by taking all choreography instances (in $\mathcal{G}^{\mathcal{I}}$), applying κ on each to retrieve all the relevant business process instances and abstracting on public activities of the directly involved partners. The last known public activity can be registered as

通过获取所有编排实例（在Gi中），在每个实例上应用检索所有相关业务流程实例，并抽象直接涉及的合作伙伴的公共活动，可以创建聚合编排实例（如图3所示）。

最后一个已知的公共活动可以注册为当前活动节点。这样的节点可以计数，并且频率标记在编排模型（G）上。每个计数表示一个编排实例。

现在可以确定哪些编排实例违反了状态遵从性：在图3中，这是变更区域内的三个实例（在内部），以及变更区域后的七个实例（在之后）。

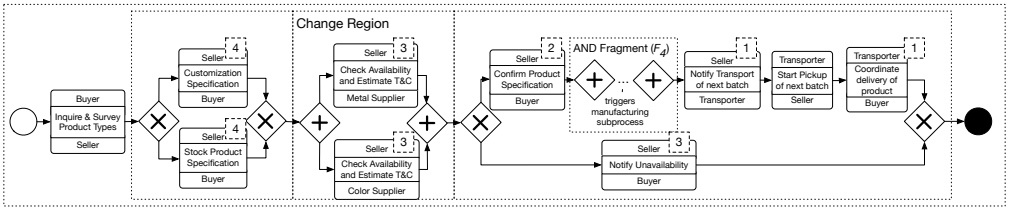


Fig. 3. Example of an Aggregated Choreography Instance

the *current active node*. Such nodes can be counted and the frequency marked on the choreography model ($\mathcal{G}$). Each count represents a single choreography instance. It is now possible to determine which choreography instances violate state compliance: in Fig. 3 these are the three instances inside the change region (WITHIN), and the seven instances after the change region (AFTER).

4 Dynamic Change Propagation Strategies

Having defined the necessary concepts, we will focus our attention on the mechanics of propagating changes to partners, focusing on the question of consistency from the dynamic point of view. At the core, every process instance needs to be stopped to avoid having process instances in the BEFORE state (compliant) entering the WITHIN or AFTER state, which makes these process instances non-compliant in terms of the change requested. This is the pessimistic approach because we assume that regardless of the future path actually taken, it might lead to structural conflicts due to the changes to be committed. Activities that are still before the *change region* are free to be executed according to the previous model. The *change region* represents a critical section, and partners still need to decide on a common understanding for this region (see Change Negotiation [6]). Thus any partners proceeding the execution past the *change region* have at that point already made a choice with which change alternative to proceed. Note that choosing the old version is an alternative as well. The outcome of the change negotiation needs to match this implicitly chosen change alternative for those process instances to remain structurally compliant. Anyone diverging from the common understanding means that any work resulting from that becomes unusable and has to be discarded. Partners are tied to each other with their decisions on which change alternative to pick.

我们将把注意力集中在向合作伙伴传播变化的机制上，从动态的角度关注共存问题。

在核心，需要停止每个流程实例，以避免流程实例处于“前”状态（符合）进入“内”或“后”状态，这使得这些流程实例在请求的更改方面不符合要求。

这是一种悲观的方法，因为我们假设，不管实际采取的未来路径如何，它都可能由于要进行的更改而导致结构性冲突。

仍在更改区域之前的活动可以根据以前的模型自由执行。变更区域代表一个关键部分，合作伙伴仍然需要决定对此区域的共同理解（见变更谈判[6]）。

因此，任何在变更区域之后进行执行的合作伙伴都已经做出了选择，以进行变更替代。请注意，选择旧版本也是一种选择。

变更
协商的结果需要匹配这些流程实例在结构上保持兼容的隐式选择的变更备选方案。任何偏离共同理解的人都意味着由此产生的任何工作变得不可用并且必须被丢弃。合作伙伴之间是相互联系的，他们的决定是选择哪种改变。

谈判

隐式选择

There is a significant disadvantage with this approach: all choreography instances being affected by the change are stopped and no work can be executed that falls beyond the first node in the *change region*, which means there is an opportunity cost: no productive work can be performed due to the waiting time until a change alternative has been agreed on for the contested *change region*. This waiting time ensures the collaboration stays consistent. An alternative approach is the *optimistic change strategy*, which assumes that paying the opportunity cost (of waiting) is higher than just proceeding with execution and pay the cost only if actual state violations occur, meaning a different change alternative has been agreed on than the one partners have implicitly chosen to proceed execution with. In order to accomplish this, we need an approach to estimate this actual *repair* cost, which is the cost of transforming non-compliant

这种方法有一个显著的缺点：所有受更改影响的编排都将停止，并且无法执行超出更改区域中第一个节点的工作，这意味着存在机会成本：由于等待时间的原因，在更改替代方案被激活之前，无法执行生产性工作。为有争议的变革区域而努力。这个等待时间确保协作保持一致。

另一种方法是乐观的变更策略，该策略假定
支付机会成本（等待成本）高于仅仅继续执行，并且仅在实际状态发生冲突时支付成本，这意味着已经商定了不同于一个合作伙伴隐式选择继续执行的变更替代方案。.为了实现这一点，我们需要一种方法来估计实际的维修成本，这是转换不符合项的成本。

流程实例将再次符合。将这两种不同的成本进行权衡（悲观成本与乐观成本），可以在向合作伙伴提出变更传播之前对工作进行公平的估计，并在变更谈判开始后进行谈判。在这项工作中，重点放在确定修复成本的技术上，这是为使不符合的流程实例符合而发生的。

process instances to become compliant again. Weighing these two different costs together (pessimistic cost vs optimistic cost), allows a fair estimation of effort before change propagation is proposed to partners, and bargaining leverage once change negotiations start. In this work, the focus is placed on the technique for determining the *repair* cost, which occurs for making non-compliant process instances compliant.

在本工作中，我们将着重于基于事务支持的乐观变更策略，并且基于执行回滚来计算修复成本，从对事务模型的假设开始。

4.1 Transaction-based Optimistic Change Strategy

事务模型假设存在几篇探讨工作流事务模型应该具有的必要属性的论文（[3]）。这里，我们总结了这些属性，并假设它们是支持的，独立于所使用的具体事务模型。本文的工作建立在这些假设的基础上。

In this work we will focus on an optimistic change strategy that is based on transaction support and calculates the repair cost based on performing rollback, starting with the assumptions on the transaction model.

Transaction Model Assumptions Several papers exist that explore the necessary properties a workflow transaction model should entail ([3]). Here we summarize these properties and assume they are supported, independent of the concrete transaction model being used. The work in this paper builds on these assumptions.

工作流事务是（i）单个活动或（ii）一系列活动，并将其转换为另一个一致状态的一致状态作为输入。此外，工作流事务通过子进程和子活动的存在分层地嵌套。关于原子性，工作流事务放松了传统事务全有或全无概念的无属性[3]。

A workflow transaction is either (i) a single activity or (ii) a sequence of activities and takes as input a consistent state transforming it into another consistent state. Furthermore, workflow transactions are hierarchically nested with the presence of subprocesses and thus sub activities. Regarding atomicity, workflow transactions relax the *nothing* property of the all-or-nothing concept of traditional transactions [3]. Whereas traditional transactions expect an opaque transformation step from one consistent state to another, workflow transactions are made transparent, where each intermediate consistent state is *opened up*. This transparency allows a more fine-grained ability to perform rollbacks to past consistent states. A rollback path can be defined, which marks the backward sequence of past activities from the current activity up to the desired past consistent state. This paper takes these eligable states as rollback target in the context of inter-organizational business processes. We assume that the past consistent states, which have been traversed so far are represented and available in the form of private and public execution logs. The visibility of which is dependent on the partner accessing it. These execution logs are one of the required inputs for performing selective rollback. Note that for all private activities, the owner of the process instance is able to decide, without coordination with the associated partners, to which past state to rollback to. But once a public activity occurs inside the rollback path, the directly associated partner is bound to the same rollback operation. A transitive effect can be observed as this partner directing its partners to rollback to the relevant public activity, due to their interaction activities in the same rollback path. A decision to rollback then, by the nature of public interactions and the message dependency, affects other partners directly as well as indirectly. Another assumption is the ability to assign compensation tasks to activities, which are executed in the case an activity needs to be rolled back. If a compensation task cannot be semantically mapped, then the activity is marked as a *critical activity*. The effects of *critical activities* are further discussed in section 4.2.

尽管传统事务期望从一个一致状态到另一个一致状态的不透明转换步骤，但是工作流事务是透明的，其中打开了每个中间一致状态。

这种透明性允许更细粒度的能力，以执行回滚到过去的一致状态。可以定义回滚路径，它标记从当前活动到期望的过去一致状态的过去活动的向后序列。

可消除的

我们假设到目前为止已经遍历的过去的一致状态是以私有和公共执行日志的形式表示和可用的。它的可见性取决于访问它的合作伙伴。这些执行日志是执行选择性回滚所需的输入之一。

但是一旦在回滚路径中发生了公共活动，直接关联的伙伴就被绑定到相同的回滚操作。当该伙伴指示其伙伴回滚到相关的公共活动时，可以观察到传递效应，这是由于它们在相同的回滚路径中的交互活动。

然后，由于公共交互和消息依赖的性质，回滚的决定会直接或间接地影响其他合作伙伴。

另一个假设是将补偿任务分配给活动的能力，这些任务在需要回滚活动的情况下执行。如果补偿任务不能在语义上映射，则该活动被标记为关键活动。关键活动的影响在第4.2节中进一步讨论。

关于一致性，我们假设在执行单个活动之后提交的每个事务都处于一致的状态。
因此，每个回滚操作以及因此补偿任务的完成也导致一致的状态。

Regarding consistency, we assume that each committed transaction after the execution of a single activity results in a consistent state. Accordingly, each rollback operation and thus the completion of a compensation task results in a consistent state as well.

关于隔离，回滚活动不应影响同时运行的工作流事务（假设它们不是同一编排实例的一部分）。对于子活动，我们在父活动和被调用的子活动之间具有父子依赖性。由于这种父子关系，每当决定回滚父活动时，所有执行的子活动都需要回滚。相反，子活动上的回滚并不一定需要父活动上的回滚。

Regarding isolation, rolling back activities should not affect concurrently running workflow transactions (assuming they are not part of the same choreography instance). In the case of sub activities we have a parent-child dependency between the parent activities and the called sub activities. Due to this parent-child relationship, whenever a decision is made to rollback the parent activity, all of the executed sub activities need to be rolled back as well. Inversely, a rollback on a sub activity does not necessarily require a rollback on the parent activity.

关于持久性，我们一致地假设所有提交的事务以及执行的任何补偿任务都是持久性的。

Regarding durability, we assume together with consistency that all committed transactions, as well as any compensation tasks executed, are made persistent.

基于前面部分提到的事务模型的假设，我们介绍了回滚区域。回滚区域是一种可以归类为基于事务的乐观更改策略的技术。它是乐观的，因为它允许合作伙伴在变更请求尚未提交时继续执行。无论更改请求是否被提交，乐观方法都假定并非每个工作结果都需要被丢弃。它是基于事务的，因为使用了回滚（特别是补偿任务）来使不兼容的流程实例恢复到遵从性。

4.2 Rollback Region

回滚区域确定相应的伙伴可以执行到哪个上限，并基于该上限计算在出现遵从性转换的情况下所需的成本估计。

Based on the assumptions on the transaction model mentioned in the previous section, we introduce *rollback regions*. *Rollback region* is a technique that can be classified as a transaction-based optimistic change strategy. It is optimistic because it allows partners to proceed the execution even while the change request has not been committed yet. Whether or not the change request will be committed, the optimistic approach assumes that not every work result needs to be discarded. It is transaction-based due to the use of rollback, specifically compensation tasks, to bring non-compliant process instances back to compliance. *Rollback regions* determine the upper bound up to which a corresponding partner may proceed execution, and based on that calculates a cost estimation that would be required in case compliance transformation occurs. Stopping the execution of a process instance at an interaction gives us certain information about the corresponding partner we are interacting with. When we are waiting for a message then we know that the corresponding partner has not yet reached its corresponding send activity. Conversely, when there is an upcoming receive message activity with a partner and we haven't yet sent the required message for that partner to proceed, we are sure that at the worst case, that partner is waiting for us. These *checkpoints* form the basis for *rollback regions* to estimate the farthest activity direct partners are able to proceed. All possible paths that can be drawn between the current activity node up to the *checkpoints* represent the rollback paths to be traversed (i.e., by running the sequence of compensation activities). The cost of a rollback path is defined in Def. 6. The act of committing a change request is itself conducted within the context of a transaction: either all choreography instances are transformed into a consistent state, or the change commit fails and a rollback occurs which results in the consistent state before the change request was proposed. The *rollback region* captures the worst case cost in the event of a rollback. It is important to note that not all rollback paths need to be traversed, as only the actually traversed path represented by the private and public execution log needs to be compensated. In the following

在交互中停止流程实例的执行给我们提供了关于我们正在与之交互的相应伙伴的某些信息。

当我们在等待消息时，我们知道相应的伙伴尚未到达其相应的发送活动。

相反，当与合作伙伴有一个即将到来的接收消息活动，并且我们还没有发送该合作伙伴继续进行所需的消息时，我们确信在最坏的情况下，该合作伙伴正在等待我们。

这些检查点构成了回滚区域的基础，以估计直接合作伙伴能够进行的最远活动。

可以在当前活动节点之间绘制直到检查点的所有可能路径表示要遍历的回滚路径（即，通过运行补偿活动序列）。

提交更改请求的行为本身在事务的上下文中进行：要么将所有编排实例转换为一致状态，要么更改提交失败，并且发生回滚，从而导致在提出更改请求之前处于一致状态。

回滚区域捕获发生回滚时的最坏情况成本。需要注意的是，并非所有回滚路径都需要遍历，因为只需要补偿由私有和公共执行日志表示的实际遍历路径。在下面

我们将讨论可能确定构建回滚区域的终端节点的最远检查点的影响因素。

sections we will discuss the influencing factors that may determine the farthest *checkpoint* that builds the terminal node of a *rollback region*.

确定最远活性的第一个影响因素是临界活性的存在。

Critical Activities The first influencing factor in determining the farthest activity is the presence of a *critical activity*. Recall that *critical activities* are those activities not associated with compensation tasks (c.f. Section 4.1), and thus cannot be compensated in the event of a failed transaction (e.g. rollback). Since the *rollback region* requires the presence of compensation tasks to work, *critical activities* constrain the possible execution paths for process instances in the context of a transaction-based optimistic change strategy. Concretely, from the perspective of a single process instance, if the next activity to be executed is a *critical activity*, then the optimistic change strategy becomes impossible due to not having the prerequisite compensation task to perform a rollback. This results in a hybrid change strategy where the change strategy can be optimistic up to the point of the first *critical activity*, and switching to a pessimistic change strategy from that point on. Of course, some process instances might emit a different execution log which does not entail such *critical activities*. In that case, the optimistic change strategy persists. A *critical activity* thus affects how far a process instance, under an optimistic change strategy regime, may proceed and in the same way limits the choices for the farthest activity.

回想一下，关键活动是那些与补偿任务无关的活动（参见4.1节），因此在事务失败（例如，回滚）的情况下不能进行补偿。由于回滚区域需要补偿任务的存在才能工作，所以关键活动在基于事务的乐观更改策略上下文中限制流程实例的可能执行路径。

具体地说，从单个流程实例的角度来看，如果下一个要执行的活动是关键活动，那么由于没有执行回滚的先决补偿任务，乐观的变更策略变得不可能。

这导致混合的变更策略，其中变更策略可以乐观到第一个关键活动的点，并且从该点开始切换到悲观的变更策略。

当然，
一些流程实例可能发出一个不同的执行日志，该日志不包含这样的关键活动。在这种情况下，乐观的改变策略仍然存在。因此，关键活动影响在乐观的变更策略制度下流程实例可以继续多远，并且以相同的方式限制对最远活动的选择。

Sync vs Async Message Passing in Interactions The W3C WS Choreography Model defines two distinctive types of interaction activities[1]: The (a) *one-way interaction*, for sending a single message and (b) the *request-response interaction*. In the latter case, the sender expects a response from the receiver of the initial message. In this work we define interactions to be either *async* or *sync*, corresponding to (a) and (b) respectively. The main difference between the two

[1] https://www.w3.org/TR/ws-chor-model/

W3C WS Choreograhy Model定义了两种独特的交互活动类型1：(a)单向交互，用于发送单个消息和(b)请求-响应交互。在后一种情况下，发送方期望从初始消息的接收方得到响应。在本文中，我们将交互定义为异步或同步，分别对应(a)和(b)。这两者的主要区别

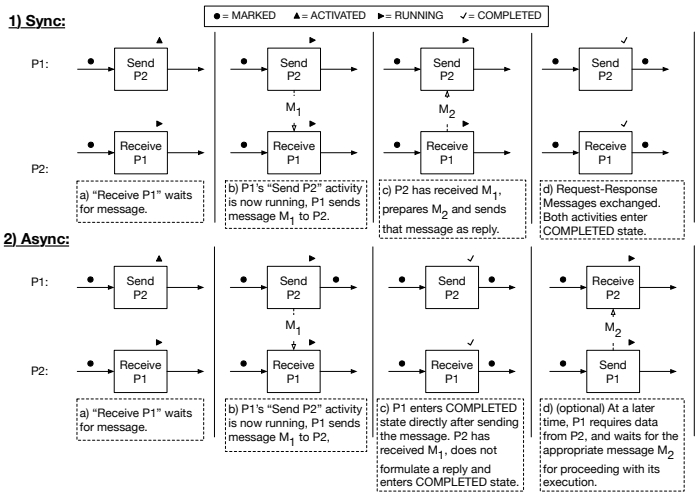


Fig. 4. Execution semantics of sync vs async interactions

交互类型如下：在异步消息传递样式的情况下，合作伙伴不必在向其合作伙伴发送消息之后等待响应（参见图4）。
这种发送语义影响回滚区域的边界（即最远的活动）。在同步交互中，由于必要的消息协调，两个伙伴彼此处于锁定状态。

以请求-响应方式。从同步交互中的发送伙伴p1的角度来看，只要交互尚未完成，对应伙伴p2的最远活动是清楚的：当前交互节点。

初始发送方在同步交互中有两个状态要执行：初始消息的第一次发送，以及用于接收对初始消息的回复的隐式接收状态。

后者防止了初始发送方进行流程实例的执行，从而也防止了回滚区域与对应伙伴的分歧。

interaction types is the following: in the case of the asynchronous message passing style, partners are not obliged to wait for a response after sending a message to their partner (cf. Figure 4). This sending semantic affects the boundary of the *rollback region* (i.e. the farthest activity). In a *sync* interaction, the two partners are in lock-step with each other due to the necessary message coordination in a request-response fashion. From the perspective of the sending partner p_1 in a *sync* interaction, the farthest activity for the corresponding partner p_2 is clear as long as the interaction has not completed: the current interaction node. The initial sender has two states to execute inside a *sync* interaction: the first sending of the initial message, and an implicit receive state for receiving the reply to the initial message. The latter prevents the initial sender to progress the execution of the process instance, and thus also prevent the *rollback region* from diverging with the corresponding partner.

在异步交互中，由于缺少等待应答的阻塞接收活动，因此它不那么透明。

在异步交互完成之后，只有在初始异步发送活动之后发送方具有显式的接收活动时，发送方才会进入等待状态。

In an *async* interaction it is not as transparent, due to the lack of a blocking receive activity that would wait for a reply. After an *async* interaction completes, the sender will come to a waiting state only if the sender has an explicit receive activity after the initial *async* send activity. Note that after a *sync* interaction completes (i.e., the receiving partner has received the reply and proceeds execution), the same non-transparency issue exists for *sync* interactions. *Rollback region* are then non-deterministic, as well as dynamic. It is non-deterministic due to not knowing which actual execution path will be taken and which blocking node reached. It is also dynamic, because the *rollback region* changes as soon as partners pass checkpoints. A *rollback region* becomes a cost estimate based on a temporary snapshot of the current state of the choreography instance, to estimate the effort and cost required to propagate a change request.

注意，在同步交互完成之后（即，接收伙伴已经接收到应答并继续执行），对于同步交互也存在相同的非透明性问题。

回滚区域既是非确定性的，也是动态的。由于不知道将采用哪个实际执行路径以及到达哪个阻塞节点，所以它是不确定的。

它还是动态的，因为回滚区域在伙伴通过检查点时立即改变。

回滚区域变成基于编排实例的当前状态的临时快照的成本估计，以估计传播更改请求所需的努力和成本。

Definition 6. ***[Cost of a rollback path]***

Let RP be the set of private/public activities constituting a rollback path, which does not contain a critical activity. Assume further a function $cost_{comp}(a)$ that returns the cost of performing the compensation task associated with an activity a. Then the cost of a rollback path is the sum of the cost of compensation tasks to be executed: $cost_{rbp}(RP) = \sum_{a \in RP} cost_{comp}(a)$.

Definition 7. ***[Blocking Node]***

如果节点是同步交互（包括隐式接收活动）、显式接收活动、批评活动（没有映射补偿任务的活动）或标记流程实例结束的结束节点，那么它就是阻塞节点。

A node is a *blocking node* if it is either a *sync* interaction (which includes an implicit receive activity), an explicit receive activity, a criticial activity (an activity without a mapped compensation task), or an end node marking the end of the process instance.

$$BlockingNode ::= SyncInteractionActivity \mid ReceiveActivity \mid CriticalActivity \mid EndNode$$

Rollback Region Algorithm Depending on the perspective of the partner applying the *rollback region*, it can have different purposes. On the one hand, from the perspective of the change initiator, the *rollback region* is used to calculate the effort required for executing compensation tasks in order to make process instances compliant according to the old process model. After initiating

回滚区域算法根据合作伙伴应用回滚区域的角度，可以有不同的目的。

一方面，从变更发起者的角度出发，使用回滚区域来计算执行补偿任务所需的工作量，以便根据旧流程模型使流程实例兼容。启动后

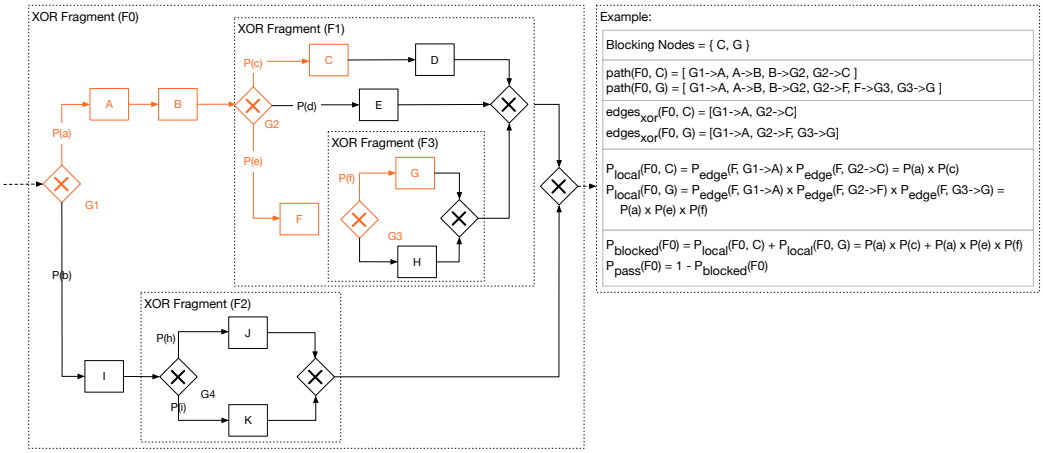


Fig. 5. Fragment-Local Blocking Node Probabilities

碎片局部阻塞节点概率

the change request, in an optimistic change strategy, the change initiator assumes the change request becomes accepted and proceeds execution, while the decision to commit to the change is not yet made. On the other hand, from the perspective of the change acceptor, the *rollback region* is used to calculate the effort required in order to make process instances compliant according to the new process model. This is due to the process execution maintaining the old version for running the current process instances. The effort then is to make the process instances compliant to the new process models. The *rollback region* is used as the core mechanism to determine the effort required to make process instances compliant, whether for the old process model or the new one. In both of the above cases, the *rollback region* algorithm requires the following inputs: (1) the *change region* marking the beginning of the change, (2) the private process model of the partner for which the dynamic impact is calculated, and (3) the private execution log of the same partner. Since the purpose of the *rollback region* is to calculate the cost of transforming non-compliant process instances to become compliant, we define the start node to be the first node within the *change region* (i.e., input (1) *change region*). The end node to be determined is the farthest activity as discussed earlier. In this section we will focus on the steps required to determine a *rollback region* from the perspective of a single partner. The full specification can be found in Algorithm 2.

变更请求，在一个乐观的变更策略中，变更发起人假定变更请求被接受并继续执行，而提交变更的决定尚未作出。另一方面，从变更接受者的角度，使用回滚区域来计算根据新流程模型使流程实例兼容所需的工作量。这是由于流程执行维护了用于运行当前流程实例的旧版本。

然后，努力使流程实例符合新的流程模型。回滚区域用作核心机制，以确定使流程实例兼容所需的工作，无论是对于旧流程模型还是新流程模型。在上述两种情况下，回滚区域算法都需要以下输入：（1）标记更改开始的更改区域；（2）计算动态影响的伙伴的私有进程模型；（3）同一伙伴的私有执行日志。

由于回滚区域的目的是计算将不符合的流程实例转换为符合的成本，因此我们将起始节点定义为更改区域（即输入（1）更改区域）中的第一个节点。要确定的结束节点是前面讨论过的最远的活动。在本节中，我们将重点介绍从单个合作伙伴的角度确定回滚区域所需的步骤。完整的规范可以在算法2中找到。

Step 1: Determine Fragment-Local Blocking Node Probabilities The first step of the *rollback region* algorithm is to determine for each occuring *blocking node* (cf. Def. 7) its local probability of becoming activated. The *rollback region* is not deterministic, as the actual path to be executed from the current activity is still not yet known. In the simplest case we have a single blocking node on the level of the top-most RPST fragment (inside a sequence). In this case we can determine that in all cases, landing on this node causes the execution to stop. In the case of inside an RPST fragment (XOR gateways), we require branching probabilities (e.g. based on actual historical execution data) to determine the probability of the execution engine reaching that blocking node. Thus we add

步骤1：确定片段局部阻塞节点的概率回滚区域算法的第一步是为每个片段确定发生阻塞节点（参见定义7）其被激活的局部概率。

回滚区域不具有确定性，因为从当前活动执行的实际路径仍然未知。在最简单的情况下，我们在最上面的rpst片段（在序列中）的级别上有一个阻塞节点。在这种情况下，我们可以确定在所有情况下，登录此节点都会导致执行停止。在RPST片段（XOR网关）内的情况下，我们需要分支概率（例如基于实际的历史执行数据）来确定执行引擎到达该阻塞节点的概率。因此我们添加

另一个假设：我们可以访问一个分支概率表，该表是我们想要估计其影响的合作伙伴的私有模型。

深嵌套子碎片内的阻塞节点通过乘以到达该子碎片的分支概率来确定。

由于我们只关心通过分支概率阻塞节点的可到达性，因此同一子碎片内的多个阻塞节点被认为是相等的，并且只考虑第一个。通过总结到达每个阻塞节点的这些局部概率，我们可以确定这个$_{rpst}$片段成为最终节点的概率：$_{pblocking}$（$_{f}$）（参见$_{def}$。9）。在不遇到阻塞节点的情况下，通过相同片段的执行的逆概率为$_{ppass}$（$_{f}$）$_{=1-pblocking}$（$_{f}$）。

another assumption: we have access to a table of branch probabilities for the private model of the partner whose impact we want to estimate. Blocking nodes inside deeply nested subfragments are determined by multiplying the branch probabilities of reaching that subfragment. Since we are only interested in the reachability of blocking nodes through branching probabilities, multiple blocking nodes inside the same subfragments are considered equal and only the first is considered. By summing up these local probabilities of reaching each blocking node, we can determine the probability of this RPST fragment becoming a terminal node: $P_{blocking}(F)$ (cf. Def. 9). The inverse probability of the execution passing through the same fragment without encountering a *blocking node* would then be $P_{pass}(F) = 1 - P_{blocking}(F)$.

path（f，node），一个函数，返回从fragment f开始到node的最短边序列。

edges xor（f，node），返回xor节点从指向节点的fragment f开始处的路径中经过的边，定义为xor（path（f，node））经过的边。

pedge（边），返回被遍历边的局部（xor）分支概率的函数。

Definition 8. ***[Reachability of Local Blocking Node]*** Given
- $path(F, node)$, a function that returns the shortest sequence of edges starting from the beginning of Fragment F leading to $node$.
- $edges_{xor}(F, node)$, a function that returns the edges preceeded by an XOR node from the path from the beginning of Fragment F leading to $node$, defined as $\alpha_{preceeded\_by\_xor}(path(F, node))$.
- $P_{edge}(edge)$, a function that returns the local (XOR) branching probability of that edge being traversed.

The *reachability of a local blocking node* is defined as

$$P_{local}(F, node) = \begin{cases} \prod_{x \in edges_{xor}(F,node)} P_{edge}(F, x) & \text{if is\_xor\_fragment(F)} \\ 1.0 & \text{otherwise} \end{cases}$$

Figure 5 shows an example which calculates the probabilities of reaching blocking nodes $\{C, D\}$ inside a XOR RPST Fragment $F0$.

给定到达$_{rpst}$片段$_{f.plocal}$（$_{f}$，$_{node}$）内阻塞节点的局部概率，我们可以定义阻止执行的整个$_{rpst}$片段的概率：

Definition 9. ***[Probability of a blocking RPST Fragment]***

Given the local probability of reaching a *blocking node* inside a RPST fragment F: $P_{local}(F, node)$, we can define the probability of the whole RPST fragment blocking the execution:

$$P_{blocking}(F) = \sum_{\forall x \in nodes(F): is\_blocking\_node(x)} P_{local}(F, x)$$

Step 2: Determine Terminal Node A terminal node is the farthest activity a process instance may reach. In the simplest case, the terminal node is the *end* node of the process model. This case happens when there is no more blocking nodes to suspend the execution for a process instance, except the last node (i.e., the *end* node). Another case is a blocking node in the main sequence of the top-most RPST fragment. This becomes a terminal node due to the certainty of this node becoming activated in future executions. It could be either due to partners having to synchronize messaging (receive activity) or reaching a critical activity. The last case is an RPST fragment one level below the top-most RPST fragment, which has $P_{blocking}(F) = 1.0$ (c.f., Def. 9), meaning an absolute certainty of activating a *blocking node* once entered. In contrast, any $P_{blocking}(F) < 1.0$ will have the possibility of executions avoiding *blocking nodes*

步骤2：确定终端的终端节点 节点最远的流程实例的活动是可以达到的。在箱的案例，是在终端节点的节点端的流程模型。

这个案例发生时，有没有更多的是起到阻塞节点的过程实例的执行，除了最后的节点（即，端节点）。

另一个案例是一个阻塞的主节点在最顶$_{rpst}$序列片段。这成为终端节点，由于节点的数量在未来成为激活的指示执行。

这可能是由于到的任何信息（$_{Synchronize}$）接收具有深远的活动）或关键的活动。

最后一个案例是一个$_{rpst}$片段水平低于上最$_{rpst}$片段，其中有$_{pblocking}$（$_{f}$）
$_{=}$（一，$_{1}$）。一个绝对的确定性，即$_{9}$）阻塞节点激活一次进入大学。在对比度，任何$_{pblocking}$（$_{f}$）＜$_{1.0}$将有阻塞的可能性，避免节点的执行

？？？

Algorithm 2: Local Rollback Region (LRR) Algorithm

```
   Input:
1        Q - change region
2        π_i - Private Process Model of partner i
3        L_i - Private Execution Log of partner i
4  Begin
5  Δ ← Q(π_i); S ← head(Δ); cur ← last(L_i)
6  if S ∉ L_i ∧ S.state ∉ {running, stopped, completed} then
7      return 0
8  else
9      // Step 1: Determine terminal node
10     foreach f in nodespath_rpst(cur, π_i) do
11         if P_blocking(f) = 1.0 then
12             node_terminal ← f
13             break;
14     // Step 2: Adjust local probabilities
15     residual ← 1.0; probs ← ∅; subfrags ← ∅
16     foreach f in nodespath_rpst(cur, π_i) until f = node_terminal do
17         foreach n in nodes(f) do
18             if is_blocking_node(n) ∧ n ∉ subfrags then
19                 probs ← probs + {(f, P_local(f, n) * residual)}
20                 subfrags ← subfrags + {f}
21         residual ← (1 − P_blocking(f)) * residual
22     assert( ∑_{(bn,p)∈probs} p = 1.0 )
23     // Step 3: Calculate expected cost of rollback region
24     cost_expected ← 0.0
25     foreach (bn, p) in probs do
26         cost_expected ← cost_expected + cost_rbp(nodespath(cur, bn)) * p
27     return cost_rbp(nodespath(S, cur)) + cost_expected
```

为了一般地确定终端节点，我们采用最顶层的RPST片段，并按照该序列计算每个节点成为阻塞节点的概率。第一个$P_{blocking}(f) = 1.0$标记为终端节点。在图6中可以看到一个例子，其中端节点被标记为终端节点，因为F_2是更改区域，而F_3具有$P_{blocking}(t_3) = 0.6$。

inside this RPST fragment. To generically determine the terminal node we take the top-most RPST fragment and in that sequence take each node's probability of it becoming a *blocking node*. The first $P_{blocking}(F) = 1.0$ is marked as the terminal node. An example can be seen in Figure 6, where the end node is marked as the *terminal node*, due to F_2 being the *change region* and F_3 having a $P_{blocking}(F_3) = 0.6$. ???

Step 3: Determine Absolute Probabilities of Rollback Paths inside the Rollback Region Having determined the *terminal node*, we can now calculate the absolute probability of reaching each *rollback path* inside the *rollback region*. The only *blocking nodes* we consider are those starting from the current activity up to the terminal node (c.f., example in Figure 6). The *rollback paths* are built by pairing each candidate *blocking node* as the end activity with the current active node as the starting activity. For the adjustment of the probabilities we again take the sequence of the top-most RPST fragment. For each node, we accumulate the residual, which is the probability of not being blocked by the current node (starting with 1.0) and adjust the local probabilities by multiplying it with the

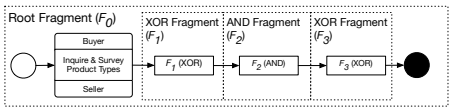
确定回滚区域内回滚路径的绝对概率

在确定了终端节点之后，我们现在可以计算到达回滚区域内每个回滚路径的绝对概率。我们考虑的唯一阻塞节点是从当前活动到终端节点的那些节点（c.f.，图6中的示例）。

回滚路径是通过将每个候选阻塞节点作为结束活动与当前活动节点作为开始活动配对来构建的。为了调整概率，我们再次采用最上面的rpst片段序列。

对于每个节点，我们累积残差，即当前节点未阻塞的概率（从1.0开始），并通过将其与残差相乘来调整局部概率。此步骤在终端节点处停止。要维持的不变量是所有绝对概率的和=1.0。

???



			Change Region	Current Active	
# of Blocking Nodes:	1	2	2	5	1
$P_{blocked}$:	1.0	1.0	1.0	0.6	1.0
$P_{adjusted}$:	-	-	1.0	0.6	0.4
$Cost_{rbp}$:	-	-	0.5	4.0	0.0
$Cost_{expected}$:	-	-	1.0 x 0.5 = 0.5	0.6 x 4.0 = 2.4	0.0
Total $Cost_{expected}$:	0.5 + 2.4 + 0.0 = 2.9				

Fig. 6. Example of a *rollback region* calculation.

residual. This step stops at the terminal node. The invariant to be upheld is the sum of all absolute probabilities = 1.0.

Step 4: Calculate Expected Cost of a Rollback Region Since *rollback regions* are probabilistic and having calculated the absolute probabilities of each *rollback path*, we can calculate a final value that represents the expected cost of that *rollback region*. Using Def. 6 we can determine the cost of a single *rollback path*. This *rollback path* cost is multiplied by the absolute probability of that *rollback path* actually occuring (from the previous step). The sum of all these individual *rollback path* costs represents the expected cost of the *rollback region*. A fixed cost part exists, which is the *rollback path* cost that is accrued due to the already traversed path starting from the change region up to the current active node.

步骤4：计算回滚区域的预期成本，因为回滚区域是概率的，并且计算了每个回滚路径的绝对概率，所以我们可以计算一个表示该回滚区域的预期成本的最终值。使用Def.。我们可以确定单个回滚路径的成本。此回滚路径成本乘以实际发生回滚路径的绝对概率（从上一步开始）。所有这些单独回滚路径成本的总和表示回滚区域的预期成本。存在一个固定成本部分，这是由于已遍历的路径从更改区域开始到当前活动节点而累积的回滚路径成本。

Using Rollback Region for Change Impact Analysis Recall that the goal of this work is to have the ability to evaluate several change alternatives in terms of change impact on the whole choreography. We have tackled that challenge through the lens of the dynamic state of the individual process instances that together realize the collaboration, in relation to the position of the proposed change alternatives individually. By using *rollback regions*, it is possible to answer these questions. While the specified *rollback region* algorithm LRR works from the perspective of a single partner, and in that context for a single process instance, we can now aggregate the individual expected costs of each *rollback region* for a single choreography instance:

$$ARR(corr_{id}, q) = \sum_{i \in \kappa(corr_{id})} LRR(q, i.m, i.as)$$

变更替代方案相对于完全协作的预期成本然后会变成：

The expected cost of a change alternative over the complete collaboration would then become:

$$CRR(q) = \sum_{corr_{id} \in C^T} ARR(corr_{id}, q)$$

With $CRR(q)$, it is now possible to compare change alternatives in terms of their impacts by estimating compensation task costs, where the individual

用回滚区域进行变更影响分析。回想一下，本工作的目标是能够根据变更对整个编排的影响评估几个变更备选案。

们已经通过单个过程实例的动态状态来应对这一挑战，这些过程实例共同实现了与提议的变更备选方案的位置相关协作。

过使用回滚区域，可以回答这些问题。

然指定的回滚区域算法lrr从单个合作伙伴的角度工作，并且在该上下文中，对于单个流程实例，我们现在可以为单编排实例聚合每个回滚区域的单个预期成本：

rollback paths dynamically evolve as execution progresses. Note however, that the LRR expects the private model π_i of the partner for whom to calculate the impact for. The change initiator does not have access to the private models of the other partners in the collaboration. This means the aggregation conducted in $ARR(k, q)$ is a fan-out process, where the change initiator asks each partner for the results of applying the local *rollback region* algorithm (LRR) and in the end aggregates the returned individual expected costs. One approach exists to avoid the communication overhead. The change initiator could estimate the change impact of a change alternative by substituting the private model of the intended partner with the corresponding accessible public model. The required execution log can be either (i) derived through abstraction on the public nodes of the intended partner or (ii) directly requested. The result of appliying the LRR on this adjusted input cannot be accurate, as no private activities are accessible and thus the complete compensation cost of those *rollback paths* unknown. What is known are the distances of these *rollback paths* through the number of *checkpoints* past the *change region*. The expected cost of change alternatives based on these inputs are only approximations. One way to increase the accuracy would be by all partners making public several metrics: branching probabilities, average number of private nodes inside RPST Fragments and average compensation cost of private activities.

5 Evaluation

As a technical evaluation we have implemented the concepts introduced in this work, mainly *rollback regions* and the dependent concepts, as a proof-of-concept. Furthermore we evaluate the output of the *rollback region* variations and ensure the critical invariants are upheld[2]. The evaluation setup follows this methodology: (1) Load a pre-defined choreography specified with BPMN 2.0 XML Format. (2) Specify a change region. (3) Randomly scale private activities for each role in the choreography with the following set as the number of private activities per fragment: $\{2, 5, 10, 30, 50\}$. (4) Randomly generate business process instances (by assigning activity states) and creating the associated choreography instances, grouping these business process instances together. (5) Calculate the expected cost using the different *rollback region* variations: (a) default LRR, (b) public *rollback region*, (c) public *rollback region* with added information (public branching probabilities, number of private activities inside each fragment, average compensation task cost of private activities inside each fragment). (6) Determine the error rate between (a) and (b) as well as (a) and (c), defined as the difference between the final costs of each respective algorithms.

The evaluation shows that the error rate is positively correlated with the number of private activities: as the number of private activities inside a fragment is scaled up, so does the error rate. Error rate reduction through the *rollback region* variant (c) can be observed. Thus it is possible to choose between variants of the *rollback region* algorithm depending on the readiness of communication

[2] The implementation can be retrieved under https://github.com/indygemma/rollback-regions

overhead for determining dynamic change impact and sensitivity to the error rate.

6 Related Work

The survey presented in [17] sets out the related areas for this work. One dimension is static versus dynamic change and the other dimension process orchestrations versus process choreographies where this work sits at the intersection of dynamic change and process choreography. A plethora of approaches exists for static and dynamic change in process orchestrations [12]. Propagation strategies for process evolution have been at first defined in [2] including abort, flush, and migrate. For an efficient decision on migrating process instances change regions have been proposed in [1]. Depending on the instance state relatively to the change region the possibility to migrate this instance can be quickly made. The most interesting case are instances that are within the change regions - this holds also true for this work. Static change in process choreographies has been investigated by different approaches. The survey in [17] only names DYCHOR [14] and C^{3}Pro [5] to deal with change propagation, i.e., other approaches focus on structural correctness of the choreography and consistency between public and private processes. So far only [18] has provided a first approach addressing dynamic change in process choreographies according to [17]. [18] addresses the problem of migrating instances after a choreography change by proposing strategies for handling the migration in an ordered way, i.e., by avoiding concurrent changes and by a protocol for the instances to accept or decline the migration. As opposed to [18], this work focuses on the instance states and the costs of the migration. Both approaches seem to be complementary.

Several transactional models for processes have been proposed (for an overview see [15]). Particular focus was put on how to deal with long-running transactions (i.e., instances) such as SAGAS [9] and Spheres [11]. Rolling back instances in order to reach a compliant state again was proposed for process orchestrations by [16]. This approach exploits selected ideas from transactional process management and transfers them to the context of dynamic choreography change.

7 Summary

In this work we have defined *rollback regions*, an algorithm to determine the expected cost of a change request in the context of process choreography instances. The algorithm is based on a transactional model that supports compensation tasks, the messaging semantics of interaction activities (sync vs async), as well as the actual states each single business process instance are situated in relation to the *change region*. There are several variations to the algorithm, and the evaluation shows that it is possible to choose the most appropriate one depending on sensitivity to communication overhead as well as error rate. In future work we would like to tackle the data perspective of applying change propagation requests, both in how it affects dynamic change impact analysis and state compliance, as well in the context of the change propagation algorithms themselves. *Rollback regions* can be further extended to support loops as well as studying alternative adaptations (e.g., versioning) to ensure state compliance.

Acknowledgment This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-072.

References

1. van der Aalst, W.: Exterminating the dynamic change bug: A concrete approach to support workflow change. Information Systems Frontiers 3(3), 297–317 (2001)
2. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data Knowl. Eng. 24(3), 211–238 (1998)
3. Eder, J., Liebhart, W.: Workflow transactions. In: Workflow Handbook 1997, pp. 195–202 (January 1997), handbook of the Workflow Management Coalition (WfMC)
4. Eshuis, R., Norta, A., Roulaux, R.: Evolving process views. Information and Software Technology 80, 20 – 35 (2016)
5. Fdhila, W., Indiono, C., Rinderle-Ma, S., Reichert, M.: Dealing with change in process choreographies: Design and implementation of propagation algorithms. Inf. Syst. 49, 1–24 (2015)
6. Fdhila, W., Indiono, C., Rinderle-Ma, S., Vetschera, R.: Finding collective decisions: Change negotiation in collaborative business processes. In: On the Move to Meaningful Internet Systems. pp. 90–108 (2015)
7. Fdhila, W., Rinderle-Ma, S.: Predicting change propagation impacts in collaborative business processes. In: Symposium on Applied Comp. pp. 1378–1385 (2014)
8. Fdhila, W., Rinderle-Ma, S., Indiono, C.: Change propagation analysis and prediction in process choreographies. Int. J. Cooperative Inf. Syst. 24(3) (2015)
9. Garcia-Molina, H., Salem, K.: Sagas. In: Special Interest Group on Management of Data. pp. 249–259 (1987)
10. Grefen, P., Rinderle-Ma, S., Dustdar, S., Fdhila, W., Mendling, J., Schulte, S.: Charting process-based collaboration support in agile business networks. IEEE Internet Computing (2017)
11. Guabtni, A., Charoy, F., Godart, C.: Spheres of isolation: Adaptation of isolation levels to transactional workflow. In: Business Process Management. pp. 458–463 (2005)
12. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer, Heidelberg New York Dordrecht London (2012)
13. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems: A survey. Data Knowl. Eng. 50(1), 9–34 (Jul 2004)
14. Rinderle, S., Wombacher, A., Reichert, M.: Evolution of process choreographies in DYCHOR. In: On the Move to Meaningful Internet Systems. pp. 273–290 (2006)
15. Rinderle-Ma, S., Grefen, P.W.P.J.: Towards flexibility in transactional service compositions. In: International Conference on Web Services. pp. 479–486 (2014)
16. Sadiq, S.W.: Handling dynamic schema change in process models. In: Australasian Database Conference. pp. 120–126 (2000)
17. Song, W., Jacobsen, H.A.: Static and dynamic process change. IEEE Transactions on Services Computing (2015)
18. Song, W., Zhang, G., Zou, Y., Yang, Q., Ma, X.: Towards dynamic evolution of service choreographies. In: Asia-Pacific Services Computing Conference. pp. 225–232 (2012)
19. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Business Process Management. pp. 100–115 (2008)
20. Wieringa, R.: Design Science Methodology for Information Systems and Software Engineering. Springer (2014)