Impact of Business Process Masking on Organizations' Policies

Zakaria Maamar[1][a], Amel Benna[2][b], Hirad Rezaei[3][c], Amin Beheshti[4][d] and Fethi Rabhi[3][e]

[1]*University of Doha for Science and Technology, Doha, Qatar*
[2]*Research Center for Scientific & Technical Information Algiers, Algeria*
[3]*University of New South Wales, Sydney, Australia*
[4]*Macquarie University, Sydney, Australia*

Keywords: Business Process, Masking, ODRL, Partnership, Policy.

Abstract: To preserve their competitiveness, organizations that engage in partnership have the opportunity of masking their business processes without undermining this partnership's progress. Aggregation and abstraction correspond to masking where the former groups activities together giving the impression of a limited number of activities in a business process, and the latter makes some activities invisible since they are deemed not relevant for partnership. Besides masking, organizations adopt policies to define permissions, prohibitions, and obligations on business processes at run-time. This paper examines the impact of business process masking on policies with focus on adjusting, dropping, and developing policies in Open Digital Rights Language (ODRL). A system demonstrating this impact is also presented in the paper.

1 INTRODUCTION

In the literature, it is largely reported that organizations' main assets are Business Processes (BPs). According to the activity-centric school for BP design, a BP has a process model that consists of activities and dependencies that connect these activities together. Along with BPs, organizations define policies to guide the completion of these BPs and comply with application domains' regulations. According to Weimer and Vining, policies set principles, goals, and desired outcomes for organizations (Weimer and Vining, 2017).

Besides dependencies that control the execution progress of activities as prescribed in a process model, BP designers could set additional controls on activities to define what is permitted to exercise on them (e.g., execute twice a day), what must be exercised on them (e.g., execute in return of payment), and what cannot be exercised on them (e.g., execute after 5pm) prior to instantiating their respective process models at run-time. To package additional controls on activities into policies, we resort to the Open Digital Rights Language (ODRL, (W3C, 2018)) offering a fine grained tracking in terms of who owns and approves to execute activities, for whom, for how much, and for how long, how many activities must be executed in a period of time, what prohibits from executing activities, to cite just some. Briefly, ODRL expresses that *"something is permitted, forbidden, or obliged, possibly limited by some constraints"* (W3C, 2018). Although ODRL is meant for the digital-content industry, we handle BPs' activities as assets in compliance with ODRL terminology.

Due to the sensitive nature of BPs being a know how, organizations that engage in partnership (e.g., Daiichi Sankyo and AstraZeneca in the biopharmaceutical industry in 2020) adopt 2 techniques to mask their BPs and hence, preserve their competitiveness, for example (Kallel et al., 2021): aggregation technique groups activities together giving external partners the impression of a limited number of activities in a BP, and abstraction technique makes some activities in a BP invisible to external partners since these activities are deemed not relevant for partnership and/or are intended to be hidden to mitigate privacy concern. In either technique, a BP's process model is adjusted impacting the existing policies that control either the aggregated or the abstracted activities. This adjustment means that for instance, some new policies will be defined (maybe on the fly), some existing policies will be either dropped or merged, cross-policy consistency will be checked out, etc. Indeed, organizations set policies independently of any plan of partnership in mind. In this paper, we exam-

[a] https://orcid.org/0000-0003-4462-8337
[b] https://orcid.org/0000-0002-9076-5001
[c] https://orcid.org/0000-0002-8638-0280
[d] https://orcid.org/2222-3333-4444-5555
[e] https://orcid.org/0000-0002-5988-5494

Maamar, Z., Benna, A., Rezaei, H., Beheshti, A. and Rabhi, F.
Impact of Business Process Masking on Organizations' Policies.
DOI: 10.5220/0013170700003928
In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 359-366
ISBN: 978-989-758-742-9; ISSN: 2184-4895

ine the impact of masking BPs on partnering organizations' policies. ODRL constructs like asset, policy, and rule are deemed relevant for managing masked activities, cascading masking impacts to policies, and checking the consistency of impacted policies.

In Section 2, we discuss existing works and then define ODRL and a case study. In Section 3, we present the impact of BP masking on policies and demonstrate this impact. Finally, in Section 4, we conclude the paper and present future work.

2 BACKGROUND

This section is about related work on masking and also briefly presents ODRL and a case study.

2.1 Related Work

In (Chika Eleonu, 2021), Chika Eleonu presents Aggregate Metric Model (AMM) framework to evaluate a corpus of BPs represented in different modeling approaches. These approaches adopt the principle of separation of concerns of models, make use of reference or base model for a family of BP variants, and promote the reuse of model elements. Although the author's framework is not perfectly in-line with BPs masking, selecting a modeling approach that handles aggregation and/or abstraction could help "minimize" the impact of this masking on existing policies.

In (Han et al., 2017), Han et al. advocate for task abstraction and aggregation as a means for first, obtaining customized descriptions of a BP for different users and second, deriving user interfaces of a BP related to participating users. In line with Section 1, privacy, confidentiality, and conflict of interest are some of the reasons for abstracting and aggregating tasks.

In (Kallel et al., 2021), Kallal et al. examine the impact of temporal constraint satisfaction on inter-organizational BPs where these BPs would have been subject to abstraction and/or aggregation. This impact meant generating time-constrained views, analyzing temporal consistency of these views, and addressing any inconsistency violating temporal constraints.

In (Kammerer et al., 2022), Kammerer et al. present Process Query Language (PQL) in the context of Process-Aware Information Systems (PAISs). The authors note that PAISs have become critical to many organizations wishing to increase maintainability and reduce costs of changes by separating process logic from application code. PQL permits to create, query, and update process models in a large size process repositories enabling personalized views over these models thanks to abstraction and aggregation techniques. Kammerer et al. mention managers who prefer an abstracted BP while process participants prefer a detailed view of the process parts.

In (Reijers et al., 2009), Reijers et al. address the high-number concern of process models that organizations end-up managing over time along with how to disclose them to a mixed audience of stakeholders such as end-users and auditors. A potential approach is process-model aggregation that takes advantage of these models' common constructs such as tasks and decision controls. An aggregate model will, on top of unique constructs, combine both models into one, where the common constructs are considered only once. As a result of applying aggregate models, fewer process will be maintained and updates to common constructs will only need to be performed once. Running queries over aggregated process models to extract common and/or unique constructs is also part of Reijers et al.'s approach.

Contrasting the originality of our work to those above, it is clear that none examines the impact of BPs masking on policies controlling BPs in terms of who owns and approves initiating them, for whom, for how much, and for how long, how many must be initiated in a period of time, what prohibits from initiating them, etc. The works above acknowledge masking' benefits to make complex process models easy to maintain, but little is given about revisiting existing policies and even developing new ones.

2.2 ODRL

ODRL provides a flexible and interoperable information model, vocabulary, and encoding mechanisms to specify how to control using assets. An asset is an identifiable resource or a collection of resources such as data/information, content/media, applications, and services. ODRL policies refer to what is permitted, prohibited, and obliged actions that stakeholders can, cannot, and should exercise over assets, respectively.

2.3 Case Study

Our case study builds upon the work of Kallal et al. who examined temporal consistency of inter-organizational BPs (Kallel et al., 2021). When a customer orders online, the seller Amazon, the shipper FedEx, and the financial institution BankX have their respective BPs intertwined/synchronized exchanging messages according to items ordered, delivery dates chosen, payment types selected, etc. Fig. 1 captures this intertwine with emphasis on activities that partners carry out and may not be willing to let others know about them. For instance,

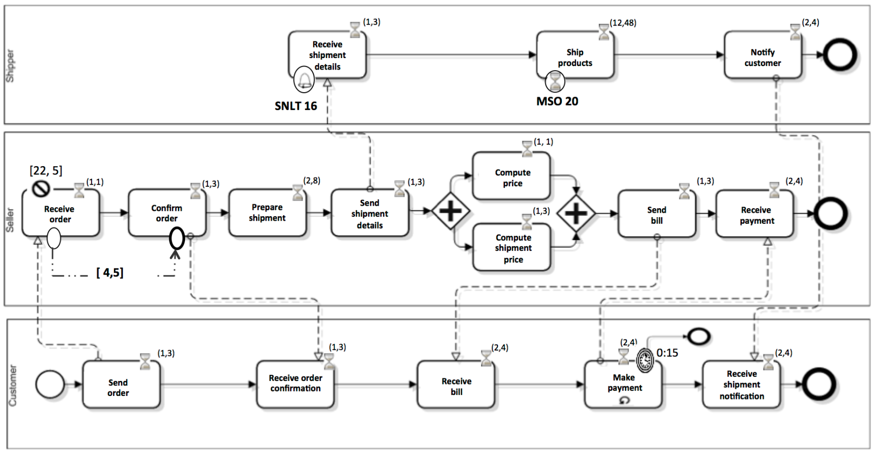


Figure 1: BPMN-based Amazon on-line shopping (Kallel et al., 2021).

compute price remains private offering Amazon a competitive advantage despite the partnership with FedEx and BankX.

In addition to BPs' process models, Amazon, FedEx, and BankX develop policies to control some activities for various reasons such as budgetary, privacy, and ownership. For instance, *ship products* must be executed by a level-3 staff, *send bill* is permitted once every 2h during high-seasons, and *make payment* is prohibited from any delay upon item delivery. Before engaging in any partnership, Amazon, FedEx, and BankX have the opportunity of masking particular activities using aggregation and/or abstraction. This could result in defining new policies, dropping some existing ones, combining existing ones as well, etc. For instance, *receive bill* and *make payment* are aggregated into a new activity calling for developing a new policy for this new activity and setting new dependences for this new activity with existing pre- and post-conditions. And, *send shipment details* is abstracted calling for revisiting its pre- and post-dependent activities including probably their respective policies.

3 MASKING AND POLICIES

This section consists of 4 parts. The first part defines masking. The second and third parts apply masking to BPs. Finally, the fourth part implements the masking.

3.1 Foundations

Fig. 2 is about the respective outcomes of applying abstraction and aggregation masking techniques to a BP. A BP consists of activities, $\mathsf{A}_{1,\cdots,i,j}$, and dependencies between activities, $\{\mathsf{D}_{i,j}(\mathsf{A}_i,\mathsf{A}_j)\}$. We associate each activity with an activity policy, $\mathcal{AP}_{1,\cdots,i,j}$, and each dependency with a dependency policy, $\mathcal{DP}_{i,j}(\mathcal{AP}_i,\mathcal{AP}_j)$, that captures case study-driven interactions between $\mathsf{A}_i/\mathcal{AP}_i$ and $\mathsf{A}_j/\mathcal{AP}_j$.

On the one hand, abstraction "hides" for instance, $\mathsf{A}_i/\mathcal{AP}_i$ in the BP raising questions about the impact of this hiding on between $\{\mathcal{AP}_{i-1}\}$ and $\mathcal{AP}_i$ and between $\mathcal{AP}_i$ and $\{\mathcal{AP}_{i+1}\}$ based on dependencies initially associated with $\mathcal{DP}_{\{i-1\},i}(\mathcal{AP}_{\{i-1\}},\mathcal{AP}_i)$ and $\mathcal{DP}_{i,\{i+1\}}(\mathcal{AP}_i,\mathcal{AP}_{\{i+1\}})$, respectively.

On the other hand, aggregation combines for instance, $\mathsf{A}_i/\mathcal{AP}_i$ and $\mathsf{A}_j/\mathcal{AP}_j$ into $\mathsf{A}_{ij}/\mathcal{AP}_{ij}$ in the BP raising questions about this combination's impact on between $\{\mathcal{AP}_{i-1}\}$ and $\mathcal{AP}_{ij}$ and between $\mathcal{AP}_{ij}$ and $\{\mathcal{AP}_{j+1}\}$ based on dependencies initially associated with $\mathcal{DP}_{\{i-1\},i}(\mathcal{AP}_{\{i-1\}},\mathcal{AP}_i)$ and $\mathcal{DP}_{j,\{j+1\}}(\mathcal{AP}_j,\mathcal{AP}_{\{j+1\}})$, respectively. During aggregation, it is deemed necessary to work out the details of A_{ij}, $\mathcal{AP}_{ij}$, $\mathcal{DP}_{\{i-1\},ij}(\mathcal{AP}_{\{i-1\}},\mathcal{AP}_{ij})$, and $\mathcal{DP}_{ij,\{j+1\}}(\mathcal{AP}_{ij},\mathcal{AP}_{\{j+1\}})$.

Let us define $\mathcal{AP}$s and $\mathcal{DP}$s. First, $\mathcal{AP}$ definition in terms of necessary ODRL constructs depends on the case study's requirements with focus on (*i*) the construct asset that corresponds to an activity's name and (*ii*) the construct constraint that both corresponds to a case study's requirements and drives the implementation of the masking techniques. In ODRL, constraint (though optional) could be applied to 2 constructs namely, rule and action. When a constraint on a triggered rule in an $\mathcal{AP}$ is satisfied, we declare the successful completion of the rule and the $\mathcal{AP}$.

Second, $\mathcal{DP}$ definition in terms of necessary ODRL constructs depends on the types of dependencies between activities in a BP's process model. In the literature, there exist several dependency types such as those used to compose services namely, *sequence*, *choice*, *parallelism*, and *synchronization* (Daoud

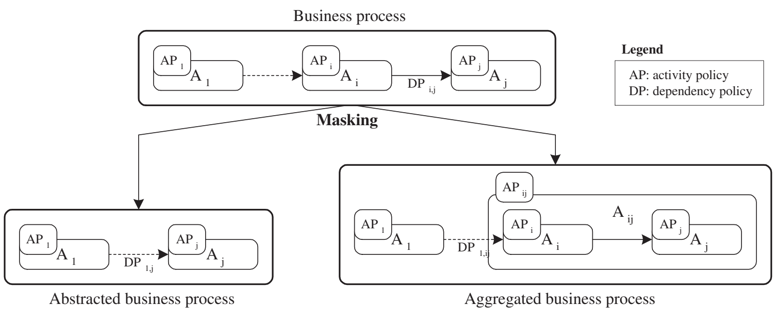


Figure 2: Masking applied to a business process.

et al., 2021). Based on our previous work (Maamar et al., 2022), we adopt service composition-based dependency types and represent their respective $\mathcal{DP}s$ in 2 stages. In the first stage, we define a $\mathcal{DP}$ rule-composition $\mathcal{DP}_{i,j}(\mathcal{AP}_i, \mathcal{AP}_j)$ to identify its assets, $\mathcal{AP}_i$ and $\mathcal{AP}_j$ (Listing 1), and then, define additional $\mathcal{DP}s$ (Listings 2 and 5), one per dependency type, that inherit from the $\mathcal{DP}$ rule-composition.

Listing 1: Dependency policy rule-composition.

```
1 {"@context": "http://www.w3.org/ns/odrl.jsonld",
2 "@type": "Set",
3 "uid": "http:/.../DPi_j:01",
4 "permission": [{
5    "uid": "http://.../DPi_jrule-comp",
6    "target":["http://.../DP/APi",
7             "http://.../DP/APj"],
8    "action": "use"}]}
```

To illustrate, we present *sequence*- and *choice*-based dependency policies, $\mathcal{DP}^{seq}_{i,j}(\mathcal{AP}_i, \mathcal{AP}_j)$ and $\mathcal{DP}^{cho}_{i,j}(\mathcal{AP}_i, \mathcal{AP}_j)$, respectively. The former dependency type, $\mathcal{DP}^{seq}_{i,j}(\mathcal{AP}_i, \mathcal{AP}_j)$, means that $rule_i$ in $\mathcal{AP}_i$ and $rule_j$ in $\mathcal{AP}_j$ should be triggered sequentially, i.e., trigger$(rule_i \prec rule_j)$. Based on some ODRL constructs, we achieve rule sequencing as follows (Listing 2):

1. Make $\mathcal{DP}^{seq}_{i,j}(\mathcal{AP}_i, \mathcal{AP}_j)$ inherit from $\mathcal{DP}_{i,j}(\mathcal{AP}_i, \mathcal{AP}_j)$ using inheritFrom (line 4).
2. Define a rule $\mathcal{R}$ (line 5) having $RuleijDP_{i,j}$ as a uid (line 6), *permission-with-duty* as a type (lines 5 and 9), and $rule_j$ as an asset (line 7).
3. Include the action trigger in $\mathcal{R}$ to exercise over $rule_j$ (line 8).
4. Define the property duty in $\mathcal{R}$, so the successful completion of $rule_i$ can be declared. To this end, include the action *nextPolicy*, the uid of $\mathcal{AP}_i$, and the constraint *event* in this property (lines 13-15). The constraint *event* specifies the triggering order of rules, so that triggering $rule_j$ happens after successfully completing $rule_i$.

Listing 2: *Sequence*-based dependency policy.

```
1 {"@context": "http://www.w3.org/ns/odrl.jsonld",
2 "uid": "http://.../SeqDPij:02",
3 "type": "Agreement",
4 "inheritFrom": "http://.../DPi_j:01",
5 "permission": [{
6     "uid": "http://.../rules/DPij",
7     "target": "http://.../rules/APj/rulej",
8     "action": "trigger",
9     "duty": [{
10        "action": "nextPolicy",
11        "uid": "http://.../policies/APi",
12        "constraint": [{
13            "leftOperand": "event",
14            "operator": "lt",
15            "rightOperand": {"@id": "odrl:
                 policyUsage"}}]}]}]}
```

To instantiate Listing 2, let us consider Fig. 1's *receive shipment details* and *ship products*. We define first, a $\mathcal{DP}$ rule-composition to treat these 2 activities' $\mathcal{AP}s$ as assets (Listing 3) and then, $\mathcal{DP}^{seq}_{receiveShipmentDetails,shipProducts}(\ldots)$ to implement the sequencing of the rules in their respective $\mathcal{AP}s$ (Listing 4). In this $\mathcal{DP}^{seq}_{\ldots}$'s permission rule (line 5), the action trigger (line 8) is exercised over the asset *shipProductsRule* (line 7). The permission rule also has a duty property (line 9) to declare the successful completion of *receiveShipmentDetailsRule* included in *receive shipment details*'s $\mathcal{AP}$. Thanks to the constraint in line 11, the sequencing of triggering *receiveShipmentDetailsRule* then *shipProductsRule* is achieved.

Listing 3: Instantiation of Listing 1.

```
1 {"@context": "http://www.w3.org/ns/odrl.jsonld",
2 "@type": "Set",
3 "uid":"http://../DPShipDetails_ShipProducts:01",
4 "permission": [{
5   "uid": "http://.../
         DPShipDetails_ShipProductsRulecomp",
6   "target":["http://.../DP/
         APreceiveShipmentDetails",
7             "http://.../DP/APshipProducts"],
8   "action": "use"}]}
```

Listing 4: Instantiation of Listing 2.

```
1  {"@context": "http://www.w3.org/ns/odrl.jsonld",
2  "uid": ".../SeqDPShipDetails_ShipProducts:02",
3  "type": "Agreement",
4  "inheritFrom":"/DPShipDetails_ShipProducts:01",
5  "permission": [{
6     "uid":"http://.../DPShipDetailsShipProducts",
7     "target":"http://.../shipProductsRule",
8     "action": "trigger",
9     "duty": [{
10       "action": "nextPolicy",
11       "uid":".../receiveShipmentDetailsRule",
12       "constraint": [{
13           "leftOperand": "event",
14           "operator": "lt",
15           "rightOperand": {"@id": "odrl:
                 policyUsage"}}]}]}]}
```

Regarding $\mathcal{DP}^{cho}_{i,j}$(A$_i$,A$_j$), it means that $rule_i$ in $\mathcal{AP}_i$ and $rule_j$ in $\mathcal{AP}_j$ should be triggered inclusively; i.e., trigger($rule_i \vee rule_j$). Based on some ODRL constructs, we achieve rule choice as follows (Listing 5):

1. Make $\mathcal{DP}^{cho}_{i,j}(\mathcal{AP}_i, \mathcal{AP}_j)$ inherit from $\mathcal{DP}_{i,j}(\mathcal{AP}_i, \mathcal{AP}_j)$ using inheritFrom (line 4).
2. Define 2 rules $\mathcal{R}$ and $\mathcal{R}'$ having $\mathcal{R}uleDP_{ij}$ and $\mathcal{R}uleDP_{ji}$ as uids, *obligation-with-consequence* as a type, and $rule_i$ and $rule_j$ as assets, respectively (lines 5-11 and lines 12-17).
3. Define the property consequence in both $\mathcal{R}$ and $\mathcal{R}'$ to handle, respectively, the exclusive non-fulfillment of the obligation in a way that if $rule_i$ is not triggered, then $rule_j$ will be triggered (lines 10-11) and *vice-versa* (lines 16-17). Should both obligations be fulfilled, then $rule_i$ and $rule_j$ will be triggered.
4. Include the action trigger in $\mathcal{R}$ to exercise over either $rule_i$ or $rule_j$ (lines 8 and 11).
5. Include the action trigger in $\mathcal{R}'$ to exercise over either $rule_j$ or $rule_i$ (lines 14 and 17).

Listing 5: *Choice*-based dependency policy.

```
1  {"@context": "http://www.w3.org/ns/odrl.jsonld",
2  "uid": "http://.../ChoRuleDPij:001",
3  "type": "agreement",
4  "inheritFrom": "http://.../DPi_j:01",
5  "obligation": [
6      {"uid": "http://.../rule1/ChoRuleDPij",
7      "target": "http://.../policies/APi/rulei",
8      "action": "trigger",
9      "consequence": [{
10        "target": "http://.../policies/APj/rulej",
11        "action": "trigger"}]},
12     {"uid": "http://.../rule2/ChoRuleDPji",
13     "target": "http://.../policies/APj/rulej",
14     "action": "trigger",
15     "consequence": [{
16        "target": "http://.../policies/APi/rulei",
17        "action": "trigger"}]}]}
```

3.2 Masking Techniques

Post-Abstraction Analysis. Abstraction makes some activities, e.g., A$_i$, in a BP invisible. These activities either are deemed not relevant for exposure to external partners or are hidden to address privacy concern. Despite being invisible, abstracted activities are invoked when partnership is enacted at run-time. By "dropping" A$_i$ from the BP, we create an "artificial" (a) $\mathcal{DP}^{a,seq}_{fi-1,i+1}(\mathcal{AP}_{i-1}, \mathcal{AP}_{i+1})$ that (*i*) inherits from a new artificial $\mathcal{DP}^a$ rule-composition similar to Listing 1 (with $\mathcal{AP}_{i-1}$ and $\mathcal{AP}_{i+1}$ as assets) and (*ii*) adopts a *sequence* dependency between A$_{i-1}$ and A$_{i+1}$ by default as per the recommendation of Kallel et al. who refer to this dependency as *finish-to-start* (Kallel et al., 2021). In line with Kallal et al. who examined BPs masking/abstraction from a time perspective when they used a delay period to hold on the abstracted activity before executing afterwards the non-abstracted activities, we restrict our ODRL constraints on actions in $\mathcal{AP}s$ to time, only, and integrate the delay period into the "artificial" dependency policy. Listing 6, that corresponds to $\mathcal{DP}^{a,seq}_{i-1,i+1}(\mathcal{AP}_{i-1}, \mathcal{AP}_{i+1})$, inherits from $\mathcal{DP}^{a}_{i-1,i+1}(\mathcal{AP}_{i-1}, \mathcal{AP}_{i+1})$ rule-composition (line 4). However, contrarily to $\mathcal{DP}^{seq}_{...}(\cdots, \cdots)$ in Listing 2, the action trigger (line 9) to exercise on $\mathcal{AP}_{i+1}$'s rule (line 7) is now constrained to *delayPeriod* (lines 11-13) ensuring that A$_i$'s time remains satisfied at run-time (line 13).

Listing 6: Artificial sequence-based dependency policy for abstraction.

```
1  {"@context": "http://www.w3.org/ns/odrl.jsonld",
2  "uid": "http://.../SeqDPi-1,i+1",
3  "type": "Agreement",
4  "inheritFrom": "http://.../DPi-1,i+1",
5  "permission": [{
6     "uid": "http://.../rules/RuleDPi-1,i+1",
7     "target": "http://.../rules/APi+1/rulei+1",
8     "action": [{
9         "rdf:value": {"@id": "odrl:trigger"},
10        "refinement": [{
11          "leftOperand": "delayPeriod",
12          "operator": "eq",
13          "rightOperand": {"@value": "AiDur", "
                 @type": "xsd:duration"}}]}],
14    "duty": [{
15        "action": "nextPolicy",
16        "uid": "http://.../policies/APi-1",
17        "constraint": [{
18            "leftOperand": "event",
19            "operator": "lt",
20          "rightOperand": {"@id": "odrl:
                policyUsage"}}]}]}]}
```

To instantiate Listing 6, we abstract *ship products*; its execution duration could take 48hours, for example. Listing 7 corresponds to $\mathcal{DP}^{a,seq}_{receiveShipmentDetails,notifyCustomer}(\mathcal{AP}_{rec\ldots},\mathcal{AP}_{not\ldots})$ that inherits from $\mathcal{DP}^{a}_{receiveShipmentDetails,notifyCustomer}(\mathcal{AP}_{rec\ldots},\mathcal{AP}_{not\ldots})$ rule-composition (line 4). Relying on Kallel et al. (Kallel et al., 2021), *delayPeriod* is set to 48h (line 13) so that *ship products* would have enough time to be executed and *notify customer* is not executed just after completing *receive shipment details*.

Listing 7: Instantiated sequence-based artificial dependency policy for abstraction.

```
1  {"@context": "http://www.w3.org/ns/odrl.jsonld",
2  "uid":"http://.../SeqDPreceiveShipmentDetails,
       notifyCustomer",
3  "type": "Agreement",
4  "inheritFrom":".../DPreceiveShipmentDetails,
       notifyCustomer",
5  "permission": [{
6      "uid": ".../RuleDPreceiveShipmentDetails,
           notifyCustomer",
7      "target": "http://.../ruleNotifyCustomer",
8      "action": [{
9          "rdf:value": {"@id": "odrl:trigger"},
10         "refinement": [{
11             "leftOperand": "delayPeriod",
12             "operator": "eq",
13             "rightOperand": {"@value": "48", "
                   @type": "xsd:duration"}}]}]},
14     "duty": [{
15         "action": "nextPolicy",
16         "uid": ".../APreceiveShipmentDetails",
17         "constraint": [{
18             "leftOperand": "event",
19             "operator": "lt",
20             "rightOperand": {"@id": "odrl:
                   policyUsage"}}]}]}]}
```

Post-Aggregation Analysis. Aggregation groups activities, e.g., A$_i$ and A$_j$, together giving partnership partners the impression of a limited number of activities in a BP. To discuss BPs masking/aggregation's impact on policies, we consider additional activities, A$_{i-1}$ and A$_{j+1}$, as well as other necessary policies namely, $\mathcal{AP}_{i-1,i,j,j+1}$, $\mathcal{DP}_{i-1,i}(\mathcal{AP}_{i-1},\mathcal{AP}_i)$, $\mathcal{DP}_{i,j}(\mathcal{AP}_i,\mathcal{AP}_j)$, $\mathcal{DP}_{j,j+1}(\mathcal{AP}_j,\mathcal{AP}_{j+1})$, $\mathcal{DP}^{seq|cho|\ldots}_{i-1,i}(\mathcal{AP}_{i-1},\mathcal{AP}_i)$, $\mathcal{DP}^{seq|cho|\ldots}_{i,j}(\mathcal{AP}_i,\mathcal{AP}_j)$, and $\mathcal{DP}^{seq|cho|\ldots}_{j,j+1}(\mathcal{AP}_j,\mathcal{AP}_{j+1})$. By aggregating A$_i$ and A$_j$ into A$_{ij}$, we create several "artificial" activity and dependency policies namely, $\mathcal{AP}^a_{ij}$, $\mathcal{DP}^{a,\cdots}_{i-1,ij}(\mathcal{AP}_{i-1},\mathcal{AP}_{ij})$, $\mathcal{DP}^{a,\cdots}_{ij,j+1}(\mathcal{AP}_{ij},\mathcal{AP}_{j+1})$, $\mathcal{DP}^{a}_{i-1,ij}(\mathcal{AP}_{i-1},\mathcal{AP}_{ij})$, and $\mathcal{DP}^{a}_{ij,j+1}(\mathcal{AP}_{ij},\mathcal{AP}_{j+1})$. By analogy with BPs masking/abstraction, we adopt first, ODRL temporal constraints on actions in both A$_i$'s $\mathcal{AP}_i$ and A$_j$'s $\mathcal{AP}_j$ and second, a *sequence*-based dependency between A$_i$ and A$_{ij}$ and between A$_{ij}$ and A$_{j+1}$. For the sake of illustration, we also adopt a *sequence*-based dependency between A$_i$ and A$_j$ to calculate A$_{ij}$'s execution duration that is the sum of A$_i$'s and A$_j$'s respective execution durations (Dumas et al., 2013). Durations for other dependency types like *choice* are presented in (Dumas et al., 2013) as well. To define necessary artificial policies, we proceed as follows:

- $\mathcal{AP}^a_{ij}$: we define $\mathcal{AP}^a_{ij}$ by setting a $\mathcal{DP}$ rule-composition $\mathcal{DP}_{i,j}(\mathcal{AP}_i,\mathcal{AP}_j)$ as per Listing 1, a *sequence*-based dependency policy $\mathcal{DP}^{seq}_{i,j}(\mathcal{AP}_i,\mathcal{AP}_j)$ as per Listing 2, and the policy itself as per Listing 8. In this listing, line 7 refers to the asset that is the aggregating activity's name, line 4 establishes the inheritance with $\mathcal{DP}^{seq}_{i,j}(\mathcal{AP}_i,\mathcal{AP}_j)$, and lines 11-13 are related to the time constraint on the action trigger.

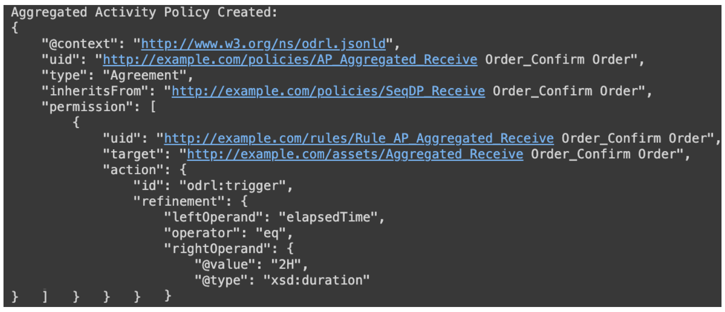
Listing 8: Artificial policy for an aggregating activity.

```
1  {"@context": "http://www.w3.org/ns/odrl.jsonld",
2  "uid": "http://.../AijPolicy:001",
3  "type": "Agreement",
4  "inheritsfrom": "http://.../SeqDPij:02",
5  "permission": [{
6      "uid": "http://.../rules/AijRule",
7      "target": "http://.../assets/Aij",
8      "action": [{
9          "rdf:value": {"@id": "odrl:trigger"},
10         "refinement": [{
11            "leftOperand": "elapsedTime",
12            "operator": "eq",
13            "rightOperand": {"@value": "AijDur",
                  "@type": "xsd:duration"}}]}]}]}
```

We now aggregate *receive shipment details* and *ship products* into *complete product shipment* considering the *sequence*-based dependency between these 2 activities as well as their 48h and 4h respective execution durations, for example. Listing 9 corresponds to $\mathcal{AP}^a_{completeProductShipment}$ that instantiates Listing 8 where the execution duration is set to 52h (line 13).

Listing 9: Instantiated artificial *sequence*-based dependency policy for aggregation.

```
1  {"@context": "http://www.w3.org/ns/odrl.jsonld",
2  "uid":"http://.../DPcompletProductShipment:001",
3  "type": "Agreement",
4  "inheritsfrom":"http://.../
       SeqDPreceiveShipmentDetailsShipProduct:02",
5  "permission": [{
6     "uid":".../completeProductShipmentRule",
7     "target":".../completeProductShipment",
```



```
Aggregated Activity Policy Created:
{
    "@context": "http://www.w3.org/ns/odrl.jsonld",
    "uid": "http://example.com/policies/AP_Aggregated_Receive Order_Confirm Order",
    "type": "Agreement",
    "inheritsFrom": "http://example.com/policies/SeqDP_Receive Order_Confirm Order",
    "permission": [
        {
            "uid": "http://example.com/rules/Rule_AP_Aggregated_Receive Order_Confirm Order",
            "target": "http://example.com/assets/Aggregated_Receive Order_Confirm Order",
            "action": {
                "id": "odrl:trigger",
                "refinement": {
                    "leftOperand": "elapsedTime",
                    "operator": "eq",
                    "rightOperand": {
                        "@value": "2H",
                        "@type": "xsd:duration"
}   ]   }   }   }   }
```

Figure 3: Aggregated *receive order* and *confirm order* $\mathcal{AP}$s.

```
 "action":[{
    "rdf:value": {"@id": "odrl:trigger"},
    "refinement": [{
      "leftOperand": "elapsedTime",
      "operator": "eq",
      "rightOperand": {"@value": "52", "@type"
          : "xsd:duration"}}]}]}]}
```

- $\mathcal{DP}^{a}_{i-1,ij}(\mathcal{AP}_{i-1},\mathcal{AP}_{ij})$ and $\mathcal{DP}^{a,seq}_{i-1,ij}(\mathcal{AP}_{i-1},\mathcal{AP}_{ij})$: are defined in compliance with Listing 1 and Listing 2, respectively.
- $\mathcal{DP}^{a}_{ij,j+1}(\mathcal{AP}_{ij},\mathcal{AP}_{j+1})$ and $\mathcal{DP}^{a,seq}_{ij,j+1}(\mathcal{AP}_{ij},\mathcal{AP}_{j+1})$: are defined in compliance with Listing 1 and Listing 2, respectively.

3.3 Experiments

We present the experiments conducted to assess the impact of masking techniques on BPs' policies using Section 2.3's case study. The experiments, available at Google Colab, assessed performance- and process-based metrics in terms of process flow, policy consistency, and computational resources. They were performed on a Python-based ODRL policy engine combined with a BPMN parser[1], executed within Google Colab. This one provides a cloud-based platform with Python 3 support, backed by Google Compute Engine, which allows for code execution in virtualized environments. The system ran on a free Colab session with 12.7 GB of system RAM, 107.7 GB total disk space (with 32.5 GB used), and a Python 3 Google Compute Engine backend for the CPU.

In the aggregation scenario, we selected a sequence-based dependency between *receive order* and *confirm order* (Fig 1). The system generated the necessary policies to reflect this aggregation. The generated policies include:

- Data dependency-based rule composition composes the rules of the individual activities being aggregated, i.e., Listing 1.
- Sequence-based dependency policy defines the sequence dependencies between the aggregated activities, i.e., Listing 2.
- Aggregated activity policy in Fig. 3 represents the aggregated activity as a single unit, i.e., Listing 8.
- Sequential dependency policies with the aggregated activity in Fig. 4 define the links between the aggregated activity and other activities in the process, i.e., Listing 2.
- Message synchronization policies represent the message that the aggregated activities receive and transmit.

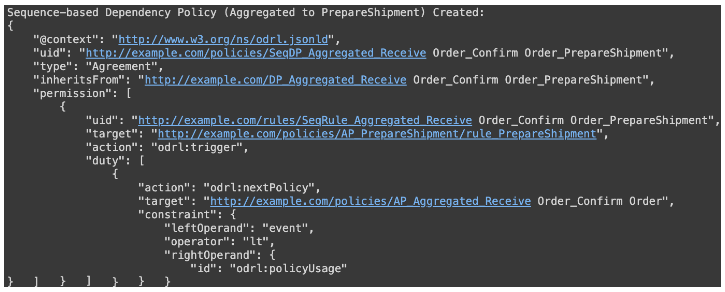
In the abstraction experiments, abstraction was applied to activities such as *send bill*. We made the following observations:

- Completion time: The time to generate abstracted policies was minimal, with an average completion time of 0.006 seconds. This demonstrates negligible delay in generating abstraction policies.
- Resource utilization: CPU usage during abstraction was minimal, with negligible changes observed indicating that abstraction does not significantly strain computational resources.
- Policy complexity: Abstraction required a fixed number of policies and rules, regardless of the number of activities abstracted together. Specifically, 3 policies and 3 rules per abstraction. This demonstrates that abstraction maintains a low level of policy complexity while effectively hiding activities.

In the aggregation experiments, aggregation was applied to activities like *compute price*, *compute shipment price*, and *prepare shipment*. We made the following observations:

- Completion time: Aggregation required more time than abstraction, with the average completion time increasing proportionally to the number of activities aggregated. For example, aggregating 3 activities took on average 0.014 seconds. This increase is due to the additional time required

[1]https://pypi.org/project/bpmn-python.



```
Sequence-based Dependency Policy (Aggregated to PrepareShipment) Created:
{
    "@context": "http://www.w3.org/ns/odrl.jsonld",
    "uid": "http://example.com/policies/SeqDP_Aggregated_Receive Order_Confirm Order_PrepareShipment",
    "type": "Agreement",
    "inheritsFrom": "http://example.com/DP_Aggregated_Receive Order_Confirm Order_PrepareShipment",
    "permission": [
        {
            "uid": "http://example.com/rules/SeqRule_Aggregated_Receive Order_Confirm Order_PrepareShipment",
            "target": "http://example.com/policies/AP_PrepareShipment/rule_PrepareShipment",
            "action": "odrl:trigger",
            "duty": [
                {
                    "action": "odrl:nextPolicy",
                    "target": "http://example.com/policies/AP_Aggregated_Receive Order_Confirm Order",
                    "constraint": {
                        "leftOperand": "event",
                        "operator": "lt",
                        "rightOperand": {
                            "id": "odrl:policyUsage"
}   ]   }   ]   }   }   }
```

Figure 4: Sequence-based $\mathcal{DP}s$ of the aggregated activities and *prepare shipment* $\mathcal{AP}s$.

to create multiple policies and manage dependencies.

- Resource utilization: Aggregation consumed more CPU resources compared to abstraction due to the increased number of policies generated. Despite this, CPU and memory usage remained within acceptable limits, indicating the system's efficiency in handling aggregated processes.
- Policy complexity: Aggregation led to an increase in policy complexity, with the number of policies and rules significantly higher than in abstraction. Aggregating activities led to creating multiple policies, including rule composition policies and dependency policies linking existing activities to the aggregated activities. This reflects the additional complexity introduced by aggregation.

4 CONCLUSION

This paper examined the impact of BP masking techniques on organizations' policies that engage in partnership. We relied on ODRL to define policies that control BPs' activities and capture application domains' regulations. 2 techniques are adopted for masking namely, abstraction and aggregation, that impact the way policies are defined. Experiments were conducted to demonstrate the technical doability of adjusting, dropping, and/or developing policies because of abstraction and aggregation. Unlike abstraction that introduced minimal delay and maintains a low level of policy complexity, aggregation required more time and generated a higher number of policies. In term of future work, we would like first, to re-engineer masked BPs so that the steps of applying abstraction and/or aggregation to unmasked BPs are traced back, and, second, to analyze cross-policy consistency when masking is applied to separate BPs.

REFERENCES

Chika Eleonu, H. (2021). Aggregate Metric Model for Evaluating Business Processes. *Business Process Management Journal*, 27(2).

Daoud, M., El Mezouari, A., Faci, N., Benslimane, D., Maamar, Z., and El Fazziki, A. (2021). A Multi-Model based Microservices Identification Approach. *Journal of Systems Architecture*, 118:102200.

Dumas, M., La Rosa, M., Mendling, J., and Reijers, H. (2013). *Fundamentals of Business Process Management*. Springer.

Han, L., Zhao, W., and Yang, J. (2017). An Approach towards Task Abstraction and Aggregation in Business Processes. In *Proceedings of ICWS'2017*, Honolulu, HI, USA.

Kallel, S., Cheikhrouhou, S., Maamar, Z., Guermouche, N., and Jmaiel, M. (2021). From Generating Process Views over Inter-Organizational Business Processes to Achieving their Temporal Consistency. *Computing*, 103(7).

Kammerer, K., Rüdiger Pryss, R., and Reichert, M. (2022). Retrieving, Abstracting, and Changing Business Process Models with PQL. In Polyvyanyy, A., editor, *Process Querying Methods*. Springer.

Maamar, Z., Faci, N., and El Haddad, J. (2022). Microservices Deployment on a Multi-platform Ecosystem: A Contract-Based Approach. In *Revised Selected Papers - ICSOFT'2022*, Lisbon, Portugal.

Reijers, H., Mans, R., and van der Toorn, R. (2009). Improved Model Management with Aggregated Business Process Models. *Data Knowledge Engineering*, 68(2).

W3C (2018). ODRL Information Model 2.2. https://www.w3.org/TR/2018/REC-odrl-model-20180215/. (Visited in January 2024).

Weimer, D. and Vining, A. (2017). *Policy Analysis, Concepts and Practice*. Taylor & Francis Group, New York.