Ontology-Based Abstraction of Bot Models in Robotic Process Automation

Maximilian Völker(✉) and Mathias Weske

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
{maximilian.voelker,mathias.weske}@hpi.de

Abstract. Robotic Process Automation is a technology for lightweight task automation that empowers business users to build their own software bots by combining predefined operations in a graphical user interface. However, due to the detailed nature of these operations, which are comparable to single code instructions, the graphical models become complex and hard to understand. This complicates the sharing, discussion, and maintenance of these software bots. At the same time, RPA projects typically require extensive documentation of the process and its automation aspects that must be kept in sync during the maintenance phase. This paper presents a foundation for bot model abstraction in RPA that leverages semantic information. It proposes an abstraction method to generate smaller but still expressive bot models in an automated fashion. These abstract bot models can be used for documentation purposes and to foster process understanding, as they convey key activities while hiding operational details that are not relevant to perceiving the overall automation goal.

Keywords: Robotic Process Automation · Model Abstraction · Ontology

1 Introduction

Despite recent advancements in computer technology, repetitive and tedious tasks are still the order of the day for many employees. These tasks are not only costly due to the many working hours spent, they also increase the risk of silently introducing errors by slips. Robotic Process Automation (RPA) enables companies and employees to automate such computer-based tasks by employing software robots: scripts that can operate on user interfaces but that can also make use of programming interfaces, e.g., to access databases directly [9,10,25].

As opposed to traditional process automation, often considered costly and laborious, RPA is a rather lightweight automation technique with its unintrusive approach based on user interfaces [11]. Additionally, common tools for RPA focus on business users and thus offer no-code or low-code solutions for building RPA bots [2]. Their graphical interfaces allow users to select and connect predefined operations, such as opening a program or entering text in an input field, resulting in an automation workflow displayed in process model-like notations [4,9,11].

J. P. A. Almeida et al. (Eds.): ER 2023, LNCS 14320, pp. 239–256, 2023.
https://doi.org/10.1007/978-3-031-47262-6_13

While such graphical representations enable business users to create RPA bots, they entail a major drawback: The atomic level of the individual operations quickly results in large graphical models. Whereas in business processes the level of abstraction and, to this extent, the complexity of the model can be chosen according to the model's purpose and will most likely hide complexity, the individual nodes in RPA bot models represent specific work instructions, e.g., the assignment of a value to a variable. Consequently, even smaller use cases can result in large, confusing models. While they serve the purpose of automation, they are hard to comprehend or reason about, hindering maintenance and communication tasks at later stages.

Similar to business process models, some RPA tools offer sub-process-like constructs to better structure the model. However, just as with process models, such representations are highly dependent on the modeler, who must manually specify the desired levels of abstraction that are then "cast" into the model. Furthermore, it does not help with already existing models or when using an RPA software that does not support such constructs.

This paper addresses the challenge of extensive RPA bot models by proposing an abstraction technique that automatically generates reduced and condensed models that can be used for documentation and communication purposes. Unlike traditional abstraction techniques, it does not rely on syntactic features, but rather considers the semantics of the model elements grounded on the ontology of RPA operations. With that, the goal is to create an abstract model that conveys the overall purpose and main tasks of a bot model. Additionally, a slider-based solution is devised that allows the user to control the level of abstraction.

The importance of model abstraction in RPA is motivated in Sect. 2. Subsequently, Sect. 3 presents related work on business process model abstraction and introduces RPA and the ontology of RPA operations in more detail. Section 4 elaborates on the devised abstraction technique for RPA bot models, which is extended in Sect. 5 to allow different levels of abstraction. A prototypical implementation of the approach, its application to an example, and current limitations are discussed in Sect. 6, before the paper is concluded in Sect. 7.

2 Motivating Example

During execution, RPA bots follow a sequence of work instructions that was previously modeled by a user. The available "building blocks" are predefined by the respective RPA tool provider [9] and are rather atomic operations, such as clicking a button or inserting a string into a text field [21]. Consequently, even simpler tasks may require numerous instructions. Since most RPA tools provide a graphical way to define these workflows, the created models can become accordingly large. This section illustrates this issue with an example.

In the scenario, a common task suitable for RPA should be carried out. Orders are received as PDF files attached to e-mails, and need to be populated to multiple systems. Once the relevant data has been extracted from the PDF file, it must first be entered into the company's browser-based order management

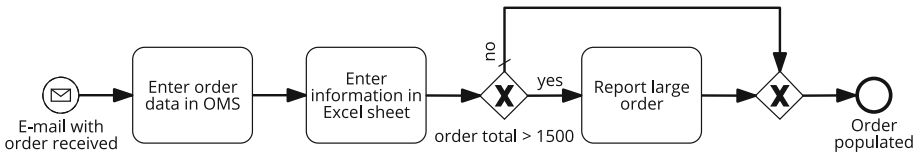

Fig. 1. High-level process model capturing the scenario

system (OMS), which returns an identifier for the order. Subsequently, some information and the identifier need to be appended to a specific spreadsheet used for reporting. Lastly, if the order surpasses a certain threshold, an e-mail needs to be generated. Figure 1 shows a business process model with the described steps.

While this scenario is not overly complex, its realization using RPA requires certain intermediate steps. For example, the text extracted from the PDF file needs to be analyzed, and the results must be cached locally, such as the address or the list of order items. The web-based OMS may require a login, and the bot occasionally needs to wait until certain UI elements are loaded. In Fig. 2, a possible sequence of RPA operations is presented, demonstrating the complexity. While all the steps shown are necessary to execute the bot properly, the model's complexity impedes communication and maintenance[1].

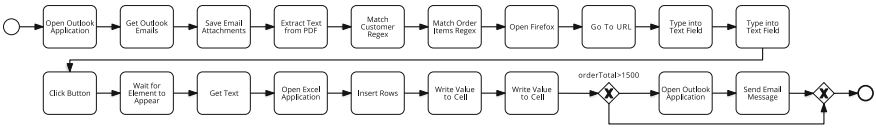

Fig. 2. RPA bot model for the described scenario (intentionally not readable)

The goal of this work is to derive from an inherently detailed bot model an abstract, yet meaningful, business process-like model, similar to the one shown in Fig. 1. That is, the abstract representation should convey the overall purpose and main functionality of the bot while hiding specific and intermediate steps. This is also in line with the main purpose of abstraction for business process models, providing a quick overview, as reported by Smirnov et al. [18]. In summary, the developed approach should address the following requirements:

R1 Reduce the size of the model.
R2 Preserve information that is vital to understand the bot's purpose.
R3 The level of abstraction should be adjustable by the user.

[1] Due to the lack of a modeling standard in RPA [22], we use BPMN for visualization. A high-resolution version can be found here: https://github.com/bptlab/onto-rpa-platform/raw/main/components/abstraction/figures/BotModelExample.svg.

R3 recognizes that there will not be a single level of abstraction that fits all needs, e.g., a coarse abstraction may be sufficient for a quick overview, while a more detailed version may be required for documentation and discussion.

3 Related Work and Preliminaries

To the best of the authors' knowledge, the topic of automated abstraction of RPA bot models has not been addressed in research so far. However, the related discipline of business process management has developed several approaches to business process model abstraction (BPMA), some of which are summarized in this section together with ontology-based approaches to model abstraction. This is followed by a brief introduction to Robotic Process Automation in general and the ontology of RPA operations in particular. In addition, two general abstraction methods used in the paper are outlined.

3.1 Business Process Model Abstraction

Business process models define a workflow consisting of activities which outline the process' steps [24], like in Fig. 1. Unlike RPA, there are no predefined activities to choose from (cf. Sect. 2) due to the complexity and diversity of the business domains. Instead, the purpose of an activity is defined by a natural language label, often without directly linking structured information [16].

Consequently, many abstraction approaches focus only on structural features of the process model, like aggregating activities of "single entry single exit" (SESE) fragments in the model [13] or applying abstraction techniques only to manually selected parts of the model to create personalized views on the process model [3]. In [19], the authors propose an abstraction approach that not only considers activities for abstraction, but that also handles aspects like roles, data objects, and messages based on a set of abstraction rules they provide.

Other approaches make use of external data not included in the model, such as execution times or costs of certain model parts [12]. Based on this information, Polyvyanyy et al. [12] propose the use of a slider that allows users to set a custom threshold for those values and to abstract model elements with a lower value.

Another abstraction approach advises the use of a vector space model to cluster activities of the process model regarding their similarity with respect to different information available in the model, such as data objects, IT systems, or roles [17]. Adding to this approach, Wang et al. [23] recently proposed a semi-supervised clustering approach that not only considers the semantic similarity, but also takes the consistency of the control flow more into account.

Theoretically, these abstraction techniques developed for business process models could be applied to RPA bot models, as they exhibit similar syntactical features and can be regarded as (part of) a business process model [7]. However, RPA with its limited set of possible operations, captured by the ontology of RPA operations presented below, offers semantic information about each task that could be exploited for abstraction, which shall be the focus of this paper.

3.2 Ontology-Based Abstraction

The lack of structured semantic information in common models hampers the development of an abstraction technique that goes beyond purely syntactical properties of the model [8]. This also applies to business process models, which have no semantic information embedded by default, but rely on text-based labels to convey their meaning, as noted above. However, since using only structural information is not always sensible, as it likely combines model parts that are semantically unrelated [16], Smirnov et al. [16] propose a semi-automated approach for BPMA that is based on ontological knowledge, more specifically, that utilizes a part-of relation defined between process activities. Groups of activities in the model that are strongly related based on this relation can be aggregated and replaced by their lowest common ancestor in the relation. As the approach requires the existence of such an ontology of the specific business environment of the process, it is not easily applicable to every business process model.

Apart from process models, there also are ontology-based abstraction techniques that aim to reduce the complexity of conceptual domain models. For example, Guizzardi et al. [8] present a rule-based abstraction approach for conceptual models that are created using the ontology-based modeling language OntoUML and which consequently can make use of semantic information. The provided rules exploit the ontological underpinning and modify the graph of the class diagram-like model to create a more abstract version of the model, for example, to abstract from classes that describe relations. Related to the idea of generating views on process models as described above, Figueiredo et al. [6] define an abstraction approach that extracts views from OntoUML models based on the ontology-induced semantics to reduce the complexity.

3.3 The Ontology of RPA Operations

Companies resort to Robotic Process Automation (RPA) for a fast and, compared to traditional approaches, cheap process automation [11]. By combining predefined automation operations, users can define automation routines that can operate on the user interface, imitating the user's behavior, and also access, for example, web services and databases [1,9].

Compared to business processes with their plenitude of application domains and user-defined activities, workflows in robotic process automation are restricted to the predefined set of RPA operations provided by the respective RPA tool vendor [9,21]. Völker and Weske [21] introduced the *ontology of RPA operations (ORPAO)*, that provides a vendor-agnostic view on *RPA operations* and relates them to the *software* and the *data* that can be automated. It defines a taxonomy (type-of relationship) of operations that classifies the available operations regarding their purpose, differentiating three main types of operations: `AutomationOperations` that actually operate on the computer, such as accessing data, performing mouse clicks, or starting software; `InternalOperations` that are used for local data storage and checks; and `ControlFlowOperations`, such as decisions that affect the execution [21]. Instances of the operations-taxonomy,

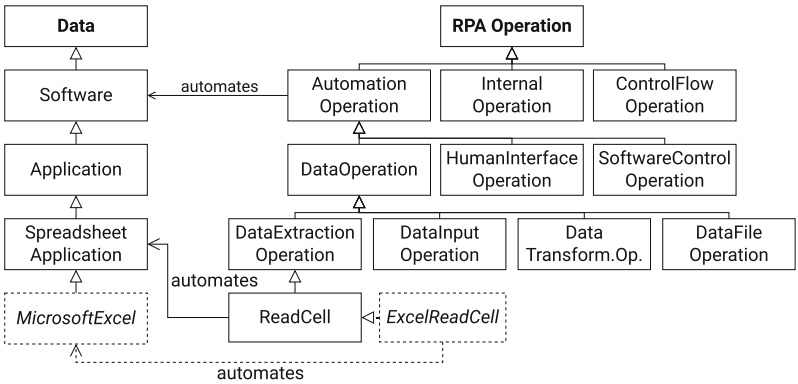

Fig. 3. Excerpt of the operations-taxonomy in the ontology of RPA operations, showing its main concepts and the branch for the *ExcelReadCell* operation with its connection to the concept of `Software` in the ORPAO. Adapted from [21].

such as *ExcelReadCell*, represent a specific operation that is available for building an RPA bot [21]. Traversing the taxonomy, partly depicted in Fig. 3, reveals that *ExcelReadCell* belongs to the concept (class) `ReadCell`, which in turn is a `DataExtractionOperation`, and so on. Figure 3 also depicts the `automates` relation, that connects `AutomationOperations` and `Software`.

In this paper, we refer to the operations-taxonomy as a rooted tree T_O with `RPAOperation` as its root r_O. Let o be an element (node) in T_O, then we define $type : T_O \rightarrow T_O \cup \{\perp\}$ as the function that returns for each element in the taxonomy its parent element and undefined ($\perp$) for $type(r_O)$. For example, $type$(ExcelReadCell) = ReadCell. Furthermore, we denote the set of all specific operations (instances in the taxonomy) as $\mathcal{O}$, which are the leafs in T_O. Likewise, we introduce $\mathcal{S}$ as the set of all specific software programs in the ORPAO, including, for example, *MicrosoftExcel*.

3.4 Abstraction Methods

While the different abstraction approaches address different goals and consider different information for abstraction, they usually rely on the same two methods on *how* to abstract elements in the model, i.e., how the model itself is pruned: *elimination* and *aggregation* [3,5,14,15,18,19]. Likewise, the abstraction approach presented in this paper makes use of these two methods, which is why they and their effects on process models are introduced in more detail.

Dimensions of Abstraction. Smirnov et al. [18] characterize the effects of elimination and aggregation operations on the model and highlight two dimensions to consider: *granularity* and *coverage*. The degree of granularity describes how detailed the model's activities are. At a high level of granularity, the model consists of many very detailed activities, whereas at lower levels, it consists of only a few activities that outline the process. Coverage in turn considers the

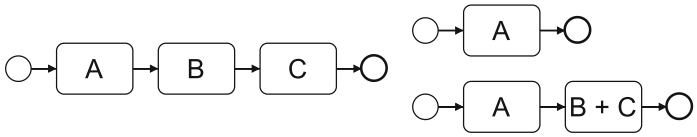

Fig. 4. Application of the abstraction methods *elimination* (top right) and *aggregation* (bottom right) to `B` and `C`.

amount of information included in the model. A high degree of coverage means that most of the available information is represented, while a lower level results in a loss of information.

Elimination. Elimination removes elements from the model and thus decreases its complexity. The level of coverage is reduced because the information carried by the eliminated elements is no longer included and cannot be recovered from the abstract model. Figure 4 illustrates the effect with a simple example. Assuming activities B and C are supposed to be eliminated from the original model at the left, the resulting abstracted model is depicted at the top right. Any information about the eliminated activities is lost. The granularity remains unchanged, as the level of detail (of the remaining activities) is unaffected.

Aggregation. The goal of aggregation is to combine a number of connected elements of the original model and represent them as a new, single element in the abstract model. The new element describes the subsumed parts in more general, resulting in a lower degree of granularity. However, by reducing the granularity, some detailed information may also be lost, slightly decreasing the coverage of the model. Aggregating activities B and C in the example in Fig. 4 results in the abstract model shown at the bottom right. Instead of two activities, a new activity representing both is incorporated.

.

4 Generating Abstract RPA Bot Models

With the goal of creating an abstract bot model that conveys the bot's key functionality as motivated in Sect. 2, this section discusses the application of the two abstraction methods, elimination and aggregation, in the context of RPA. By leveraging ontological knowledge, the presented abstraction approach focuses on the content of the model instead of syntactic features, with the aim of providing a more meaningful abstraction.

Similar to business processes, RPA bots follow a well-defined sequence of operations to achieve their automation goal [21]. This order is, as with business processes, typically defined graphically using a notation based on workflow graphs.

In previous work, we introduced the notion of a conceptual RPA bot as "a vendor-independent representation of an RPA bot that is based on the concepts of the ontology of RPA operations" [22]. Along this line, we introduce the concept

of a *conceptual RPA bot model* in Definition 1 as a workflow graph whose nodes are connected to instances in the ontology of RPA operations and thus yields semantic information for each of its nodes.

Definition 1 (Conceptual RPA Bot Model). A conceptual RPA bot model B is a tuple $(N, F, concept)$, where

- N is a finite and nonempty set of nodes in the model;
- $F \subseteq N \times N$ is the flow relation between operations, so that (N, F) is a weakly connected graph; and
- $concept : N \to \mathcal{O}$ maps each node in the model to the corresponding individual in the operations-taxonomy T_O, part of the ORPAO.

Existing, vendor-specific RPA bot models can be linked to the ontology using the knowledge base of RPA operations which connects implementations of operations by the various RPA vendors to their conceptual counterparts in the ontology, thus enabling an automated transformation to conceptual RPA bot models that can also be applied to text-based RPA bot models [21,22].

The central idea of the abstraction approach detailed in this section is built on two pillars described in Sect. 4.1. First, it requires a mapping that assigns (classes of) RPA operations in the ontology an abstraction method to apply. Second, the bot model is analyzed to determine the execution context of each node in the model, which is used to constrain and scope aggregation operations during the abstraction. Both, the general mapping and the model-specific context analysis, are prerequisites for performing the abstraction as described in Sect. 4.2.

4.1 Prerequisites and Preparation of the Abstraction

To be able to perform the abstraction, we introduce two new concepts: the abstraction mapping and the operation context analysis. While the abstraction mapping is independent of bot models, the automatic context analysis must be performed on the model prior to abstraction.

Abstraction Mapping. The abstraction mapping determines how specific operations should be treated during abstraction based on their importance regarding the abstraction goal of creating a smaller model for communication purposes (cf. Sect. 2). As defined in Definition 2, (classes of) operations in the operations-taxonomy can be assigned one of the abstraction methods, *elimination* or *aggregation*.

Definition 2 (Abstraction Method Mapping). Let T_O be the taxonomy of operations in the ontology of RPA operations as introduced in Sect. 3.3. α is a partial mapping that assigns elements in the taxonomy an abstraction method:

$$\alpha : T_O \rightharpoonup \{elimination, aggregation\}$$

Based on the partial mapping, the abstraction method that should be applied to any element $o \in T_O$ can be determined as follows:

$$abs(o) = \begin{cases} \alpha(o) & \text{if } \alpha(o) \text{ is defined} \\ abs(type(o)) & \text{elseif } type(o) \neq \bot \\ \bot & \text{else} \end{cases}$$

As each node n in a conceptual bot model is linked to the ontology via the model's *concept* function, the appropriate abstraction method to apply can be retrieved using $abs(concept(n))$. According to the definition of $abs(o)$, the function either returns the abstraction method directly assigned to o (case 1) or the method of the nearest ancestor of o in T_O that has a method assigned by applying the function recursively to the parent in T_O (case 2). Otherwise, $abs(o)$ will return $\perp$ to indicate that o and none of its ancestors are assigned an abstraction method and thus o is not to be abstracted (case 3). Inheriting the abstraction method (case 2) allows assigning a method to classes of operations conveniently instead of having to specify a method for each operation in $\mathcal{O}$. At the same time, it is still possible to overwrite the inherited method by assigning it directly (case 1).

For the mapping, we differentiate operations that (a) are essential for understanding the bot's purpose and hence should not be affected by the abstraction, (b) convey important information and whose semantics should therefore be preserved as well as possible (both relate to **R2**), and (c) operations that do not contribute to understanding and thus can be concealed in the abstract model (**R1**). Operations of type (a) are not assigned a method and consequently will not be modified by the abstraction. This applies, for example, to `ControlFlowOperations`, which are essential to understand the execution logic.

Operations of type (b) are, in their sum, important for the understanding and are therefore assigned *aggregation*, i.e., they can be merged and represented at a higher level of abstraction, but should not be removed (**R2**). This applies primarily to `DataOperations`, since data processing is the essence of RPA. In addition, certain interactions with the UI may be relevant to understanding the purpose of the bot. In the example, the operations performed in the OMS are not that important individually, but taken together they convey that data is being entered into this system.

Operations of type (c) are assigned *elimination*, reducing the size of the model (**R1**) and focussing the attention to more important parts. Regarding the example, this applies, among others, to the operations concerning the internal data structure of the bot, such as matching a regular expression. While it is essential for the bot to be functional, it is not necessary for grasping its purpose, and is therefore also not included in the high-level process model presented in the motivation (Fig. 1). Other examples include operations that resize windows, wait for UI elements to appear, or navigating in the UI to a specific point.

In this paper, we establish the theoretical foundation for the abstraction approach, but do not elaborate on a specific mapping. The development of such a mapping is beyond the scope of this paper and should be done in collaboration with domain experts (cf. Sect. 6.3).

Operation Context Analysis. Like in the motivating example, an RPA bot will most likely automate different software and access various data during its execution as defined in the bot model. Thus, each of its nodes is performed in a specific *context*, defined in Definition 3. It comprises the specific application on which the operation will be executed, and, if applicable, the data on which the operation is being performed. For example, after opening the browser once

(application context), the bot navigates to and operates on different websites (data context) within this browser instance.

Definition 3 (Operation Context). Let $B = (N, F, concept)$ be a conceptual bot model. The contexts of B, C_B, is a tuple $(c_{software}, c_{data})$, where:

- $c_{software} : N \rightharpoonup \mathcal{S}$ where $\mathcal{S}$ are the specific software programs (instances) in the ORPAO, and
- $c_{data} : N \rightharpoonup D$ where D is the set of strings describing all possible data contexts, such as file names or URLs.

As some nodes and their respective operations are not executed in a specific context, such as `InternalOperations` that have no effect outside the bot, the context is defined as a partial mapping.

To be able to assign the correct context to nodes in the model, it needs to be analyzed for *context switches*, i.e., points where the active software or data changes during the bot's execution. As there should be no external influence on the context (only the bot operates the computer), context switches can be determined by analyzing the operations linked in the bot model.

For this, we differentiate between two types of operations, application-specific and generic operations. For application-specific operations, i.e., operations that are tailored to a certain application, the software context can be directly derived from their *automates* relation to a software in the ontology (cf. Sect. 3.3). In contrast, generic operations can be applied to various applications, such as *GetText*. Therefore, we also explicitly analyze the model for preceding operations that change the active application to determine their software context, such as `SoftwareControlOperations`. The software context can either be derived by their *automates* relation as well, or, in the case of a generic `SoftwareControlOperation`, by analyzing the operation's configuration. Generic operations are then assigned to the context previously started by such an operation.

For the data context, we similarly analyze for operations that change the data context, such as operations that open a file. As the data context is very specific to each RPA bot, e.g., a URL or file path, the data context needs to be derived from the specific configurations of the operations, i.e., the configured inputs and outputs. If this is not possible, a unique string can be used to identify a new data context and thus a context switch.

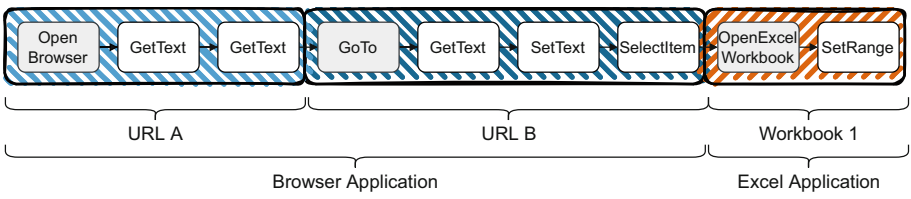

Fig. 5. Exemplary bot model sequence with annotated context

In Fig. 5, an excerpt of a bot model is shown, highlighting the different contexts in which operations are executed. The *OpenBrowser* operation causes a context switch to the browser application and a specific configured URL, on which the subsequent two operations are performed. *GoTo* is a browser-specific operation and navigates to a new URL, i.e., the data context is changed. Finally, *OpenExcelWorkbook* causes another context switch.

4.2 Performing the Abstraction

With the abstraction mapping and the results of the context analysis at hand, the given (conceptual) bot model B can be abstracted by applying first elimination operations and then aggregation operations as described below.

In the **elimination** step, all nodes $n_e \in N$ in the model B that reference an operation that should be eliminated according to the abstraction method mapping, i.e., $abs(concept(n_e)) = elimination$, are removed from the set of nodes N. At the same time, the flow relation F needs to be adjusted so that the predecessor and successor of n_e are now directly connected, thus omitting n_e. As `ControlFlowOperations` are neither considered for elimination nor aggregation, n_e cannot have multiple incoming or outgoing control flow arcs.

Compared to elimination, which only removes individual nodes, **aggregation** requires determining suitable groups of nodes in the model.

Techniques for BPMA typically aggregate regions or fragments (cf. Sect. 3.1), since the models' complexity often stems from the involvement of multiple roles, nested choices, and parallel execution, in addition to the overall intricacy of the processes themselves. RPA processes, in turn, rarely exhibit parallel behavior, do not have the concept of roles, and also do not feature too many choices, as the behavior to automate would become too complex to handle and maintain. Consequently, we focus only on the abstraction of sequences of nodes in this work (cf. Sect. 6.3). This focus also ensures that ordering constraints are preserved, as parallel or exclusive operations will not be aggregated.

An aggregation group $A \subseteq N$ consists of the nodes that form a (maximal) sequence of nodes in the model B according to its flow relation F, where for all nodes $n_{a1}, n_{a2} \in A$ the following constraints hold.

Constraint 1 $abs(concept(n_{a1})) = abs(concept(n_{a2})) = aggregation$
Constraint 2 $c_{software}(n_{a1}) = c_{software}(n_{a2})$
Constraint 3 $c_{data}(n_{a1}) = c_{data}(n_{a2})$

Accordingly, each sequence of maximal length of nodes in the model that reference an operation intended for aggregation and that are performed in the same software context and the same data context forms an aggregation group. Thus, since only nodes in the same context are considered together, the aggregation is context-preserving.

The nodes of an aggregation group A will then be replaced by a common abstract node n_A in the model. To maintain the connection to the ontology and thus retain semantic information, n_A is linked to the lowest common ancestor of

Fig. 6. Example sequence after applying elimination and aggregation

the operations referenced by the abstracted nodes in the operations-taxonomy T_O. By this, n_A refers to the class of operations that describes the aggregated operations as specific as possible.

Considering the example in Fig. 5 again, after performing the elimination on the nodes marked gray and the aggregation step for each context present, three abstract activities remain, as shown in Fig. 6.

4.3 Semantic Label Generation

Structural abstraction approaches often have problems providing meaningful labels for aggregated activities [18]. Here, the semantic underpinning of the abstract bot model can facilitate the automatic generation of labels.

To generate labels, the approach makes use of the ontology connection as well as the context. It is composed of the brief natural language description of the class in the ORPAO, the name of the software program of the context ($c_{software}$), and the string describing the data context (c_{data}).

In the example (Fig. 5), the two operations found in the first context are the same (*GetText*). Thus, this operation name can be used to describe the abstracted operations as well. The second context comprises three different operations, their lowest common ancestor in the taxonomy being `DataOperation`, as we observe both, reading and writing operations. Instead of the class name, its description in the ontology, "Handle Data", is used for the label. The third context consists of only one operation that can be used directly again for the description. The labels are then enriched with the respective context information, i.e., in which application and on which data they are performed, as shown in Fig. 6.

In sum, this allows the reader of the abstract bot model to comprehend which applications are being automated and what types of operations are being performed on them.

5 Slider-Driven Abstraction

The ontology-based abstraction technique introduced in the previous section already satisfies **R1** and **R2** by removing unimportant operations and grouping detailed work instructions to more high-level tasks. But it only provides a single level of abstraction, which is potentially too fine or too coarse for certain use cases. Therefore, we introduce an extension to this technique which allows the user to adjust the degree of abstraction using a "slider approach" and that addresses **R3**.

To match the two different employed abstraction methods, elimination and aggregation, this dynamic approach utilizes two sliders. One for determining the level of coverage, and a second for adjusting the granularity.

5.1 Coverage Slider

The *coverage slider* gives the user control over how much information should be removed from the model, i.e., how extensive operations should be eliminated, based on weights. To enable an incremental elimination, an element o in the operations-taxonomy T_O can be assigned not only the abstraction method but, if $abs(o) = elimination$, also a relevance weight w. Similar to abs, the weight of an element $o \in T_O$, $w_{elim}(o)$, is either directly annotated at the element or inherited from the closest ancestor in the operations-taxonomy that features such a value.

Using the slider, the user can now set a minimum weight w_{min}. Nodes referencing operations o with $abs(o) = elimination$ and $w_{elim}(o) < w_{min}$ are eliminated in the abstract model.

This enables a step-wise reduction of the model and allows differentiating the various concepts that are marked for elimination, but are not necessarily of the same importance. For example, internal operations that are used to set up and check the bot's internal data structure could now be removed first, as they might be considered less relevant than other types of operations.

5.2 Granularity Slider

The aggregation technique described in the previous section aggregates nodes that occur in the same context. In Fig. 5, for example, the combination of input and extraction operations results in a very abstract element, which might be too coarse in some cases.

The *granularity slider* controls the extent of the aggregation by setting a maximum depth d in the operations-taxonomy to which nodes will be aggregated. Let $concept_d(n)$ be the function that returns for a node n the ancestor of $concept(n)$ in T_O at the depth d. With the granularity slider, an additional constraint for determining the aggregation groups at a slider value d is added to the constraints presented for aggregation in Sect. 4.2:

Constraint 4 $concept_d(n_{a1}) = concept_d(n_{a2})$, i.e., nodes refer to the same operation (type) at currently set taxonomy depth d

That is, for each operation referenced in the model, its ancestor at the currently set depth in the taxonomy is determined. Sequential nodes intended for aggregation that are performed in the same context and that have the same ancestor at that depth are aggregated and replaced with that ancestor. The aggregation groups grow larger with each step in the slider as the considered depth in T_O decreases, elements become more abstract and cover more descendants.

At the very first, the operations-taxonomy is considered to full extent, and the aggregation is performed as prescribed by the constraints, i.e., only for the nodes referring to the very same operations intended for abstraction. Applied to the model in Fig. 5, this results in model 1 depicted in Fig. 7 where only the two *GetText* operations in the first context are aggregated (elimination is applied as well, concealing the operations marked in gray).

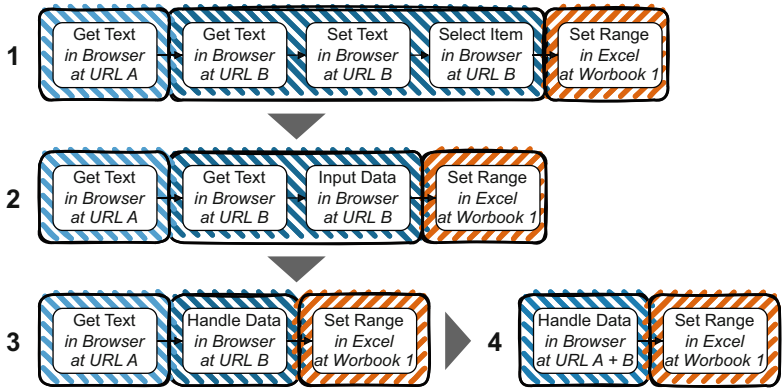

Fig. 7. Different levels of aggregation applied to the example in Fig. 5

For each slider step hereinafter, the depth of the operations-taxonomy to which the operations are abstracted is reduced stepwise and the aggregation groups determined using the given constraints. To ensure that each slider step entails a change, the slider range is set to the depth of the operations-taxonomy pruned to the branches with operations that are actually referenced in the model.

In Fig. 7, this causes *SetText* and *SelectItem* to be combined next, as both are of the type `DataInputOperation` (model 2). At the next level in the taxonomy, all three operations in the original model are `DataOperations` and, consequently, they are aggregated with the next slider step. The resulting model 3 corresponds to the abstracted model of the static approach in Fig. 6.

As soon as no further aggregation is possible, as in model 3, the next slider step will lift constraint 3, i.e., only the software context is considered and operations can be aggregated across different data contexts (see model 4). While relaxing constraint 3 enables an even more compact abstract model, aggregating data contexts may result in the loss of the corresponding information.

The label for the nodes is created similarly to the previous approach. As each links to a concept, its brief description is used for label generation, along with the respective context information, as described in Sect. 4.3.

Building on the abstraction approach presented in the previous section, the slider extension provides a considerably more flexible solution that can generate both less and more abstract models, depending on the current needs of the user and thus satisfying **R3**.

6 Evaluation

To demonstrate the applicability and usefulness of the abstraction approach, we provide a prototypical implementation of the slider approach presented in Sect. 5 and report on the application of the abstraction to the motivating example. Additionally, we briefly discuss current limitations of the presented approach.

Fig. 8. Application of abstraction method to motivating example

6.1 Prototype

The described abstraction technique has been implemented as an open-source prototype[2], extending the conceptual RPA bot modeler described in [22]. The prototype features two interactive sliders, one for the elimination threshold and one for the aggregation level, as introduced in Sect. 5, and displays the corresponding abstracted bot model to the user. Internally, the conceptual bot model is translated into a process tree [20] first, which then undergoes the context analysis (cf. Sect. 4.1). The result is an enriched version of the process tree, whose leafs, i.e., the operations, are annotated with the respective application and data context, if applicable. After the elimination candidates are determined, the process tree is pruned and all sequences of operations are analyzed to determine the aggregation groups. Based on the list of candidates for elimination and aggregation, specific model transformations are derived and applied to the bot model, more specifically which operations to remove and which to rename, since aggregation is performed by renaming the first operation of the aggregation group using the abstract label and removing the other operations from the group.

6.2 Application to Motivating Example

Figure 8 shows the bot model of the motivating example introduced in Sect. 2, abstracted using the slider approach. Here, the coverage slider was set high to remove many not-so-relevant operations, and the granularity slider was configured to match the level of abstraction of step 2 in Fig. 7, e.g., `DataOperations` may be aggregated up to the main data classes that differentiate between input and extraction of data, and the data context is preserved. Overall, it provides an overview of what is happening in each application, while reducing the number of elements from 20 to 6 (also counting *if*, as it is a control-flow operation [22]).

6.3 Current Limitations

This work provides a first step towards a more complex framework for abstraction of RPA bot models. In the following, some current assumptions for simplification made by the presented approach and points for improvement that should be addressed in the future are discussed.

First, this work focuses on linear RPA workflows and does not consider parallel and exclusive behavior. While not too common, especially parallel behavior,

[2] https://github.com/bptlab/onto-rpa-platform/tree/main/components/abstraction.

it poses interesting challenges and opportunities to further improve the abstraction. For one, abstracted models could contain empty branches in case every node on them is eliminated, which should be removed from the model in a post-processing step. In addition, the relation between the branches could be analyzed. For example, adjacent branches could involve different types of operations in the same context (e.g., different ways to achieve the same based on certain conditions) or the same operations in slightly different contexts (e.g., crawling different documents simultaneously), opening up new opportunities for aggregation.

Another aspect is the detection of patterns that can be observed in RPA bot models, such as alternating operations or contexts. For example, alternating read and write operations in two contexts implementing a data transfer would currently not be abstracted at all. They could be replaced by a node representing the data transfer between the two contexts, or by a loop construct.

Regarding the context analysis, implicit context switches are currently not considered. For example, the software context could also be changed by starting a software using a sequence of UI commands, or the data context could be affected by clicking a hyperlink or a button in a browser. Detecting such sequences could be incorporated in the future to improve the context analysis.

Also, the paper does not provide a specific abstraction method mapping. While it is intended as a flexible solution that can be adapted to specific needs, expert interviews and user studies could yield a reasonable basis to start with.

7 Conclusion

Generating abstract views on process models is a crucial step to improve understanding and to get a quick overview of the process goal without having to work through all the details. In this paper, we motivated and established the foundations for the abstraction of RPA bot models that leverage contextual and semantic information provided by the ontology of RPA operations. In addition, we contributed a prototype that realizes the described abstraction approach.

Some RPA tools offer constructs similar to sub-processes in BPMN to structure the bot process. Our approach, however, is vendor-independent, can be applied to existing bot models, and the level of abstraction can be dynamically adjusted to the current needs thanks to the slider extension.

In the future, it is conceivable to extend the abstraction approach by more facets, such as highlighting important areas or, making more use of the ontology, focusing on specific applications or the flow of data. Other ideas for future work include considering control-flow constructs besides sequences, such as decisions or loops. In addition, certain RPA-specific patterns of operations could be explored and addressed in the abstraction, such as interleaving read and write operations. The generation of labels that are more model-specific than the solution in this paper should also be further investigated, as they convey the semantic information to the reader. Finally, the abstraction approach and different mappings should be evaluated in a user study to analyze the perceived usefulness and quality of the abstraction.

References

1. van der Aalst, W.M.P., Bichler, M., Heinzl, A.: Robotic process automation. Bus. Inf. Syst. Eng. **60**(4), 269–272 (2018). https://doi.org/10.1007/s12599-018-0542-4
2. Aguirre, S., Rodriguez, A.: Automation of a business process using robotic process automation (RPA): a case study. In: Figueroa-García, J.C., López-Santana, E.R., Villa-Ramírez, J.L., Ferro-Escobar, R. (eds.) WEA 2017. CCIS, vol. 742, pp. 65–71. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66963-2_7
3. Bobrik, R., Reichert, M., Bauer, T.: View-based process visualization. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 88–95. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_7
4. Enriquez, J.G., Jimenez-Ramirez, A., Dominguez-Mayo, F.J., Garcia-Garcia, J.A.: Robotic process automation: a scientific and industrial systematic mapping study. IEEE Access **8**, 39113–39129 (2020). https://doi.org/10.1109/ACCESS.2020.2974934
5. Eshuis, R., Grefen, P.: Constructing customized process views. Data Knowl. Eng. **64**(2), 419–438 (2008). https://doi.org/10.1016/j.datak.2007.07.003
6. Figueiredo, G., Duchardt, A., Hedblom, M.M., Guizzardi, G.: Breaking into pieces: an ontological approach to conceptual model complexity management. In: 2018 12th International Conference on Research Challenges in Information Science (RCIS), pp. 1–10. IEEE (2018). https://doi.org/10.1109/RCIS.2018.8406642
7. Flechsig, C., Völker, M., Egger, C., Weske, M.: Towards an integrated platform for business process management systems and robotic process automation. In: Marrella, A., et al. (eds.) BPM 2022. LNBIP, vol. 459, pp. 138–153. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-16168-1_9
8. Guizzardi, G., Figueiredo, G., Hedblom, M.M., Poels, G.: Ontology-based model abstraction. In: Kolp, M., Vanderdonckt, J., Snoeck, M., Wautelet, Y. (eds.) 2019 13th International Conference on Research Challenges in Information Science (RCIS), pp. 1–13. IEEE (2019). https://doi.org/10.1109/RCIS.2019.8876971
9. Hofmann, P., Samp, C., Urbach, N.: Robotic process automation. Electron. Mark. **30**(1), 99–106 (2019). https://doi.org/10.1007/s12525-019-00365-8
10. Lacity, M.C., Willcocks, L.P.: A new approach to automating services. MIT Sloan Manag. Rev. **58**(1), 41–49 (2016)
11. Penttinen, E., Kasslin, H., Asatiani, A.: How to choose between robotic process automation and back-end system automation? In: Bednar, P.M., Frank, U., Kautz, K. (eds.) ECIS 2018 Proceedings. AIS (2018)
12. Polyvyanyy, A., Smirnov, S., Weske, M.: Process model abstraction: a slider approach. In: 12th International IEEE Enterprise Distributed Object Computing Conference, pp. 325–331. IEEE (2008). https://doi.org/10.1109/EDOC.2008.17
13. Polyvyanyy, A., Smirnov, S., Weske, M.: On application of structural decomposition for process model abstraction. In: Abramowicz, W., Maciaszek, L., Kowalczyk, R., Speck, A. (eds.) Business Process, Services Computing and Intelligent Service Management. LNI, pp. 110–122. Gesellschaft für Informatik e.V. (2009)
14. Polyvyanyy, A., Smirnov, S., Weske, M.: Business process model abstraction. In: vom Brocke, J., Rosemann, M. (eds.) Handbook on Business Process Management 1. IHIS, pp. 147–165. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-642-45100-3_7
15. Smirnov, S.: Structural aspects of business process diagram abstraction. In: Wilde, O. (ed.) 2009 IEEE Conference on Commerce and Enterprise Computing, pp. 375–382. Wiley (2009). https://doi.org/10.1109/CEC.2009.18

16. Smirnov, S., Dijkman, R., Mendling, J., Weske, M.: Meronymy-based aggregation of activities in business process models. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 1–14. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16373-9_1
17. Smirnov, S., Reijers, H.A., Weske, M.: From fine-grained to abstract process models: a semantic approach. Inf. Syst. **37**(8), 784–797 (2012). https://doi.org/10.1016/j.is.2012.05.007
18. Smirnov, S., Reijers, H.A., Weske, M., Nugteren, T.: Business process model abstraction: a definition, catalog, and survey. Distrib. Parallel Databases **30**(1), 63–99 (2012). https://doi.org/10.1007/s10619-011-7088-5
19. Tsagkani, C., Tsalgatidou, A.: Process model abstraction for rapid comprehension of complex business processes. Inf. Syst. **103**(C) (2022). https://doi.org/10.1016/j.is.2021.101818
20. van Zelst, S.J., Leemans, S.J.J.: Translating workflow nets to process trees: an algorithmic approach. Algorithms **13**(11), 279 (2020). https://doi.org/10.3390/a13110279
21. Völker, M., Weske, M.: Conceptualizing bots in robotic process automation. In: Ghose, A., Horkoff, J., Silva Souza, V.E., Parsons, J., Evermann, J. (eds.) ER 2021. LNCS, vol. 13011, pp. 3–13. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-89022-3_1
22. Völker, M., Weske, M.: Ontology-supported modeling of bots in robotic process automation. In: Ralyté, J., Chakravarthy, S., Mohania, M., Jeusfeld, M.A., Karlapalem, K. (eds.) ER 2022. LNCS, vol. 13607, pp. 239–254. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17995-2_17
23. Wang, N., Sun, S., OuYang, D.: Business process modeling abstraction based on semi-supervised clustering analysis. Bus. Inf. Syst. Eng. **60**(6), 525–542 (2016). https://doi.org/10.1007/s12599-016-0457-x
24. Weske, M.: Business Process Management: Concepts, Languages, Architectures, 3 edn. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-662-59432-2
25. Willcocks, L.P., Lacity, M., Craig, A.: The IT function and robotic process automation: the outsourcing unit working research paper series (15/05) (2015). http://eprints.lse.ac.uk/64519/