# On the Nature of Data in RPA Bots

Maximilian Völker(✉) and Mathias Weske

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

**Abstract.** Robotic process automation (RPA) automates tasks traditionally performed by employees, reducing repetitive and error-prone work. While RPA bot models are based on graphical notations, data, a key component of RPA, is often not explicitly represented, making it difficult to understand how data contributes to the automation. This paper explores the role of data in RPA, extends the ontology of RPA operations by data aspects, and proposes a visualization of data in RPA bot models, promoting the design of more comprehensible RPA bot services and enabling different bot analysis techniques.

**Keywords:** Robotic Process Automation · Ontology · Modeling · Data

## 1 Introduction

Modeling processes to enable their automation, including the orchestration of services, has been a strong motivation for business process management (BPM) for decades. With robotic process automation (RPA), a new technology emerged that does not require changes to the existing IT systems, as it operates primarily on the user interface level [1] to automate legacy systems, for example. Furthermore, RPA features various capabilities beyond simulating mouse clicks and key presses, like accessing databases or connecting to modern cloud services by using APIs [4], thus bridging the gap between legacy systems and modern services. Tasks automated with RPA are usually of a structured nature and centered around digital data [9], and common use cases include extracting or transferring data between applications [5].

Targeting business users, RPA workflows can typically be defined in a graphical manner by composing predefined building blocks [4,11], such as for clicking a button. However, the configuration of inputs, outputs, and parameters of building blocks is mainly done in a form-like manner. Consequently, data is typically not represented graphically, which is especially problematic as the role of data in the RPA bot and its data-flow cannot be conceived easily. To determine which parts of the bot are dependent from a data-perspective, the configuration of each building block needs to be reviewed individually.

In this paper, we conceptualize the role of data in RPA by refining the ontology of RPA operations [11] (Sect. 3), and suggest an approach for visualizing data and its flow in RPA bot models (Sect. 4). Furthermore, we highlight practical implications in Sect. 5, such as data-flow analysis to prevent run-time errors.


F. Monti et al. (Eds.): ICSOC 2023, LNCS 14420, pp. 29–37, 2023.
https://doi.org/10.1007/978-3-031-48424-7_3

## 2 Preliminaries and Motivating Example

The lack of RPA standards leads to varied terminology and modeling interfaces across vendors [2,11]. To address this, the ontology of RPA operations (ORPAO) was introduced in [11], featuring the various types of RPA operations (the building blocks), software applications and services that can be automated, and relevant data file types. Figure 1 shows some of the main concepts of the ORPAO, including the taxonomy of RPA operations with its three main classes, where `AutomationOperations`, for example, represent operations that actually interfere with the system and applications outside the bot.

As shown in Fig. 1, the ontology already includes a rudimentary conceptualization of data based on `CSO:Data`, originating from the upper *Core Software Ontology (CSO)* by Oberle et al. [6]. For the ORPAO, `CSO:Data` was extended by `File`, comprising a taxonomy of different file types [11]. Furthermore, the `CSO:accesses` relation was refined to hold between `RPAOperation` and `CSO:Data` and specialized by the subtypes `reads`, `writes`, and `transforms` [11].

The CSO further includes a notion of inputs and outputs, which was reused in [11] to define that the `reads` relation defines the input and `writes` the output of an operation: According to the CSO, `CSO:Input` and `CSO:Output` are roles played by certain `CSO:Data` [6]. The relations `CSO:inputFor` and `CSO:outputFor` connect `CSO:Inputs` and `CSO:Outputs` to operations, and `reads`, `writes`, and `transforms` were introduced as "shortcut" for this construct [11].

In [12], the ontology was extended by information regarding the control flow by mapping operations to the meta-meta-model for process model languages by Heidari et al. [3]. This mapping enables the vendor-independent modeling of RPA bots based on the concepts in the ORPAO in any common process modeling language, such as BPMN. At the same time, these conceptual models can be translated into RPA bot models of specific vendors and vice versa [12].

The ORPAO recognizes the importance of data in RPA by a separate class of operations, the `DataOperations` with its subclasses [11]. `DataExtractionOperations` access external data resources to retrieve data and cache it internally, for example, `ReadCell` operations that extract a value of spreadsheets for future use. `DataInputOperations` can write content to external data resources based

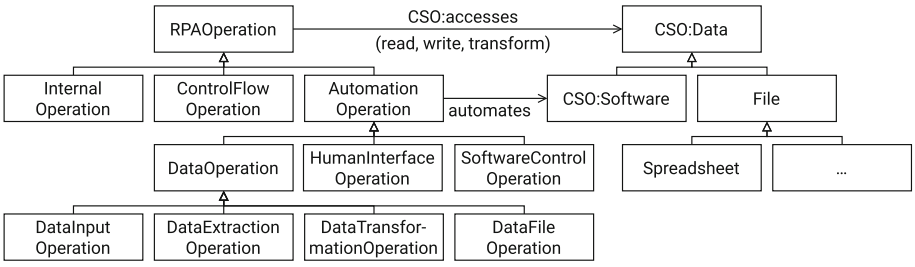


**Fig. 1.** Excerpt of important elements in the ontology of RPA operations, based on [11]


**Fig. 2.** Exemplary RPA bot model

on internally available data, like `WriteCell` operations write a value to a spreadsheet. `DataTransformationOperations` read the contents of an external data resource, apply a specific transformation to the data, and immediately write the result back to the data resource, like `SortTable`. `DataFileOperations` modify the properties of the data "container", such as creating, moving, or deleting a file.

Consequently, we need to differentiate two types of data in RPA: *External data resources*, such as files, exist independently of the RPA bot itself and are not bound to RPA in any way. *Internal data* is only available within the scope of the RPA bot and thus lost when it terminates. It can be a dynamic value determined at run-time of the bot, i.e., a variable, or a hard-coded value specified in advance.

Although data plays a critical role in RPA, this aspect is typically not expressed visually, but hidden in the configurations of the operations. Figure 2 shows a sample model of an RPA bot. This small example illustrates the need to visualize data and its flow through the process: What URL is the bot visiting? Which Excel file is being manipulated? Is there a relationship between the value extracted from the web page and the value inserted into the spreadsheet cell?

This problem multiplies as bots become more complex and handle multiple data sources. Data dependencies between operations are not visible, but can only be uncovered incrementally by examining the configuration of each operation.

The example exhibits another peculiarity of data access in RPA: The "Get Text From Element" and "Set Cell Value" operations are each preceded by operations that provide the appropriate data context for the operations to work in. These preparatory operations determine the data on which the operations will be performed, i.e., their input is not configured but determined by the context.

## 3 Conceptualizing Data in RPA Bots

To capture the dualism of data in RPA, the ontology of RPA operations and its data relationships are refined and extended in the following.

Foremost, `DataResource` is introduced as an intermediate class between `CSO:Data` and `File`, and `TransientData` as its sibling, representing internal data. The more abstract `DataResource` class is intended to reflect that there are more data sources than `Files`, like `Databases` or `WebResources`. `DataResources` exist independently of RPA bots and can also be accessed by other services or users. `TransientData`, on the other hand, only exists within the scope of the bot instance and will be discarded when the bot instance terminates. It can be further divided into `SimpleTransientData`, which represents data of primitive types like strings, and `ComplexTransientData`, such as `TabularData`.

The ORPAO already introduced the relations `reads`, `writes`, and `transforms` as specializations of the generic `CSO:accesses`, connecting `RPAOperations` and `CSO:Data` (cf. Sect. 2). With the newly introduced classes of `DataResources` and `TransientData`, we can refine them to now hold between `AutomationOperations` and `DataResources`. This complies with the definition of `AutomationOperations` being the class of operations that can access data outside the bot [11]. At the same time, focusing solely on read and write access does not encompass the full spectrum of data access that services can typically perform, such as *create*, *read*, *update*, and *delete* (CRUD), and that can also be implemented with RPA. Therefore, we first introduce two new relations, `creates` and `deletes`, to be able to express that an operation creates or destroys a `DataResource`.

Also, the different types of access to `DataResources` need to be extended and detailed further. First, there are operations that can directly modify `DataResources` in a persistent way, like `AppendToFile`. However, many operations, such as those related to office or the browser, do not directly operate on the data but on a working copy of it. For example, `ExcelWriteCell` requires that an Excel workbook has been opened before to which it can write content, as pointed out in Sect. 2. At the same time, only after an explicit save operation (`SaveWorkbook`) has been performed, the change is persisted. Consequently, we can observe additional subtypes of access related to these working copies:

`provisions(New)` describes that an operation provides a working copy of the related (newly created) `DataResource` for the subsequent operations, e.g., `OpenNewExcelWorkbook provisionsNew ExcelWorkbook`. `persists` describes that an operation saves the changes made to the provisioned `DataResource`, like `SaveWorkbook persists ExcelWorkbook`. Finally, `closes` describes that an operation destroys a working copy, such as `CloseWorkbook closes ExcelWorkbook`.

Furthermore, we can specialize the relations `reads` and `writes`: Operations that can access the resource directly without any preceding data provisioning are related to `DataResources` by `directlyReadsFrom` or `directlyWritesTo`. For example, `AppendToFile directlyWritesTo TextFile`. In contrast, the relations `implicitlyReadsFrom` and `implicitlyWritesTo` indicate that data provisioning is required and thus that the data access is not performed directly on the original resource. For example, `ExcelWriteCell implicitlyWritesTo ExcelWorkbook`. Such implicit changes will be lost if not followed by a `persists` operation.

As described in Sect. 2, the access relations in the ORPAO were already associated with the notion of inputs and outputs using roles. We retain this definition for the newly introduced specialized relations that reflect direct and implicit access. Thus, given an operation and its relation to a `DataResource`, we can deduce which data plays the role of an input for the operation and which data is considered an output of the operation.

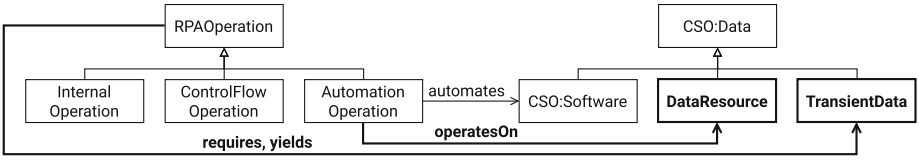


**Fig. 3.** Updated abstract of important elements in the ORPAO (cf. Figure 1), including the new main data classes and relations (depicted in black)

Next, the relations to `TransientData` are investigated. Similar to the read and write relations, we introduce `requires` and `yields` to express that certain `TransientData` plays the input/output role for a given `RPAOperation`.[1]

While `*ReadsFrom`/`*WritesTo` and `requires`/`yields` all represent the same idea of inputs and outputs, there are important differences. The former denote access of `AutomationOperations` to external data, i.e., they affect the state of the computer outside the RPA bot. The latter represent the use of bot-internal data, i.e., `TransientData`, by any `RPAOperation`. For example, `InternalOperations` like `MatchRegularExpression` operate on internal data and `ControlFlowOperations` may use them for decision-making.

The updated overview of the main concepts is shown in Fig. 3. The different relations between `AutomationOperations` and `DataResources` are subsumed under a generic relation `operatesOn`, which replaces the previously used `CSO:accesses`, since it now encompasses more concepts than just read and write.

Overall, using the new relations, we can express at a conceptual level what type of access an `AutomationOperation` performs on which `DataResources`, and if and how an `RPAOperation` works with `TransientData`, i.e., bot-internal data. For example, `ExcelReadCell implicitlyReadsFrom ExcelWorkbook` and `yields StringTransientData`. Consequently, it requires data in form of an Excel workbook as input, and produces data in form of a string to be used internally. As it performs an implicit access, it can be inferred that the operation reads the data from a working copy of the workbook which needs to be provisioned before.

## 4 Visualizing Data in RPA Bot Models

To be able to model RPA bots based on the ontology of RPA operations, it was extended in [12] by a mapping of its concepts to the meta-meta-model for business process model languages created by Heidari et al. [3]. In order to model and visualize the discussed data aspects, the newly introduced concepts must be mapped to the meta-meta-model as well.

In general, the introduced concept of `DataResources` matches the concept of *DataStores* in the meta-meta-model, since data can be read from or written to

[1] The naming is inspired by the relations `CSO:methodRequires` and `CSO:methodYields` that relate `CSO:Methods` with `CSO:Data` in the Core Software Ontology [6].

**Table 1.** Mapping of ORPAO relations between `AutomationOperations` and `DataResources` to process model patterns.

| Relation | Model Pattern |
|---|---|
| `creates` | Task with *DataStore* as output |
| `directly[ReadsFrom\|WritesTo]` | Task with *DataStore* as [input\|output] |
| `deletes` | *No model concept for destructing model elements* |
| `provisions` | Task with *DataStore* as input and *ProvisionedDataStore* as output |
| `implicitly[ReadsFrom\|WritesTo]` | Task with *ProvisionedDataStore* as [input\|output] |
| `persists` | Task with *ProvisionedDataStore* as input and *DataStore* as output |
| `closes` | *No model concept for destructing model elements* |

it, and it stores data permanently beyond the scope of the bot. But the concepts in the meta-meta-model do not allow expressing different types of data access beyond inputs and outputs. Thus, the finer-grained access types discussed before cannot be expressed directly, such as the implicit access via provisioned data. To address this issue, we differentiate between *DataStores* that represent the actual data and *ProvisionedDataStores*, the provisioned version of it. They represent a working copy of the data, e.g., created by opening a file in a software program.

Table 1 details the mapping of relations in the ORPAO between `DataResources` and `AutomationOperations` to model patterns based on the meta-meta-model. Due to the lack of data associations besides read and write in the meta-meta-model, the relations `creates` and `directlyWritesTo` share the same pattern. Still, both relations should be modeled to ensure the data-flow

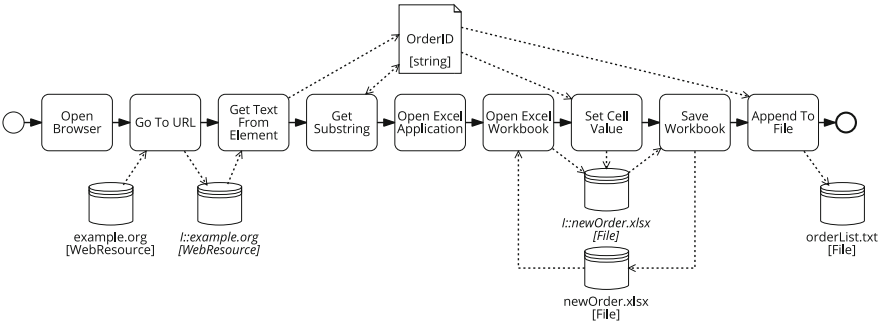


**Fig. 4.** Exemplary RPA bot model with data annotations

can be captured as detailed as possible and textual annotations could be used for clarification.

`TransientData` is mapped to *DataObjects*, more specifically, to a *DataObjectInput* if the `TransientData` is `required` by the operations and mapped to *DataObjectOutput* in case the operation `yields` the data. Data objects in general, also in the case of BPMN [7], represent data that only exists in the scope of the (bot) process instance and is lost at instance termination. This corresponds to the definition of `TransientData` as bot-internal, instance-specific data.

This definition also clarifies why the provisioned version of data is of type *DataStore* and not *DataObject*. Even though they are not persisted, they still exist outside the bot, such as an opened file in Excel that could remain open after the bot terminated, or could be accessed externally as well.

The extended mapping is applied to the example bot model given in Fig. 2. Figure 4 features the same process, now annotated with data information. As there is no concept for the *ProvisionedDataStores*, the difference is marked by italicizing the label. The model now shows which steps of the bot access external data, and whether it is a direct access, like for `AppendToFile`, or an indirect access, like `SetCellValue`. Moreover, it also visualizes the flow of data in the model: The data extracted from the website "example.org" is first internally manipulated by `GetSubstring`, further used to modify the Excel file "newOrder.xlsx", and later appended to the content of "orderList.txt".

To reduce the complexity of the model, the concept of `ContextContainers` introduced in [12] could be adapted, which allows subsuming operations that handle the context for indirect operations and could thus help reduce the overall model size and improve its clarity.[2]

## 5 Improving the Modeling of Bots by Considering Data

In the following, we outline possible applications of the introduced conceptualization and visualization.

Currently, the configuration of an operation in a bot is often hidden, e.g., in a sidebar. By using the semantic information provided by the ontology, the input and output configuration for an operation could be automatically derived from the visual bot model. For example, if an operation is connected to a data store using a read association, it can be concluded that the modeled `DataResource` is an input for the modeled operation. Consequently, the operation can be configured accordingly, given that the actual data resource is referenced in a defined way, such as using the label as in Fig. 4. However, it is important to note that associations to *ProvisionedDataStores* do not result in a configuration, as these operations depend on data provisioned before. Similarly, the model could be automatically updated as soon as the input/output configuration of an operation is modified.

[2] A model of the running example using these context containers and including data can be found here: https://github.com/bptlab/onto-rpa-platform/raw/main/components/data/figures/ContextContainerExample.svg.

By leveraging ontological knowledge about the inputs and outputs of operations, the bot model repository can be searched for a specific data usage, such as accessing a specific website or file, which can facilitate the maintenance of bots in case a web service or file structure changes.

It also enables the application of well-established techniques for data-flow analysis and error-detection in process models to RPA bot models, such as the data validation problems described in [8,10]. In particular, problems such as *redundant data*, where `TransientData` is written but never read; *missing data*; or *lost data*, where data is overwritten without being read in between, are relevant to RPA. Relevant in the context of RPA are also the problems of *mismatched data*, i.e., the (in)compatibility of data structures produced as output and later used as input, and *inconsistent data* due to concurrent data access.

In addition, RPA-specific data issues can be analyzed in the bot model. For example, related to the concept of "working copies", models can be checked for *missing context* or *lost changes*, i.e., an implicit data access that is not preceded by an appropriate provisioning step or that is not eventually persisted.

## 6 Conclusion

In this paper, we discussed the role of data in RPA services and their bot models, and presented a corresponding extension to the ontology of RPA operations along with a possible graphical representation for data in bot models. In addition to the improved representation, it enables vendor-agnostic data-flow analysis that can provide valuable insights and prevent errors. There are several aspects that can be further developed in the future. For example, differentiating between implicit and persisting accesses may help in error handling to determine which changes have already been persisted and thus need to be reverted or compensated for. Besides, the understandability and perceived usefulness of the approach by users should be investigated, especially since the additional elements increase the complexity of models. At the same time, different possible notations in addition to the one outlined in the paper could be assessed.

## 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. Correia, C., Rodrigues da Silva, A.: Platform-independent specifications for robotic process automation applications. In: MODELSWARD 2022, pp. 379–386, SciTePress (2022). https://doi.org/10.5220/0010991200003119
3. Heidari, F., Loucopoulos, P., Brazier, F., Barjis, J.: A meta-meta-model for seven business process modeling languages. In: 2013 IEEE 15th Conference on Business Informatics, pp. 216–221, IEEE (2013), https://doi.org/10.1109/CBI.2013.38
4. Hofmann, P., Samp, C., Urbach, N.: Robotic process automation. Electron. Mark. **30**(1), 99–106 (2020). https://doi.org/10.1007/s12525-019-00365-8
5. Lacity, M., Willcocks, L., Craig, A.: Robotic process automation at telefonica O2. MIS Quart. Executive **15**(1) (2015)

6. Oberle, D., Grimm, S., Staab, S.: An Ontology for Software. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 383–402. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-540-92673-3_17
7. Object Management Group: Business Process Model and Notation (BPMN) (2014). https://www.omg.org/spec/BPMN/
8. Sadiq, S., Orlowska, M., Sadiq, W., Foulger, C.: Data flow and validation in workflow modelling. In: Proceedings of the 15th Australasian Database Conference, pp. 207–214, Australian Computer Society (2004)
9. Syed, R., et al.: Robotic process automation: contemporary themes and challenges. Comput. Indust. **115**, 103162 (2020). https://doi.org/10.1016/j.compind.2019.103162
10. Trčka, N., van der Aalst, W.M.P., Sidorova, N.: Data-flow anti-patterns: discovering data-flow errors in workflows. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 425–439. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02144-2_34
11. 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
12. 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