Primary Building Blocks for Web Automation

Abstract

We present a semi-automated tool that can be used to discover basic business processes in business web applications, construct more complex business processes based on these and execute them automatically later. Our tool maps UI operations in the target business web application to conceptual operations in a database. The user of our tool is not required to have any programming knowledge in contrast to current RPA platforms. This tool comes in the form of a browser extension.

1 Introduction

Current RPA (*Robotic Process Automation*) platforms allow an increase of the work efficiency and accuracy by automatizing business processes and executing them in a more robust way by avoiding possible human errors.

In order to describe business processes in a way that can later be executed automatically by programs, RPA platforms like UiPath, Power Automate, Automation Anywhere etc. operate (i.e. create automated business processes) in the following way: RPA developers identify UI (i.e. User Interface) components of a software application like buttons, text input controls, dropdown lists and tables, and then customize activities by writing code snippets in a programming language in order to act on these UI controls (e.g. click the selected button, write data in the text input, select all data from a table or a dropdown list etc.); this code that references the selected UI controls forms the automated business process which can be executed many times later with different input parameters.

Our paper introduces a semi-automated tool that can be used to discover basic business processes and, based on these, construct more complex business processes that can be automatically executed later. This tool comes in the form of a browser extension (i.e. Chrome plugin) and can automate only business web applications, not desktop applications. To be more specific, we target CRM (i.e. Customer Relationship Management) or resource

and/or project management web applications like Microsoft Dynamics, Jira, ERP platforms, but our tool should work with any business web application that uses a relational database in the backend and exposes the data in this database through the functionality of the UI. The tool is semi-automatic, meaning that the human user must guide the tool through the various UI screens of the target web application (i.e. the human user navigates through the web application). Still, the user does not need to code for the automated business process execution, the tool handles all the understanding of the UI and the mapping of process parameters to UI controls.

The central idea of this semi-automated tool is that all the UI functionalities/ operations of a business web application that uses a relational database in the backend can be of exactly two types:

1. UI operations that only affect the UI of the application and they do not use, expose or manipulate the data in the database; these may be various functionalities for customizing the UI. These functionalities have no real value for business process automation.

2. UI operations that imply operations upon the associated database (they can be abstracted/reduced to operations on the database). Most UI operations translate into read or write operations in the database or in SQL language; they translate to 'Select', 'Insert', 'Update' or 'Delete' operations of various entities in the database - the CRUD (Create, Read, Update, Delete) operations. These are the UI functionalities that are important for business process automation and it makes sense to try to automate them.

In this sense, the proposed automation tool automatically translates human user operations on the UI onto conceptual operations in the database. So, the the central idea of the tool is to map UI operations on conceptual operations of the database. More specifically, the human user guides the automation tool (by navigating in the target business web application) such that it can discover what we call *primary blocks for process automation* - these are represented by the set of UI operations (i.e. clicks, input text fill in, etc.) that correspond to CRUD operations for the entities in the database. These *primary blocks* or *primary navigation blocks* for process automation are similar to Lego blocks that we can use in order to build more complex automated business processes out of them; these complex business processes can be later executed automatically by the tool.

BPMN 2.0 is the most used modelling notation to represent business processes. Moreover, BPMN 2.0 is not used only for graphical representations, but also for business process automation. The major aim of RPA

is to automate repetitive tasks, while Business Process Automation using BPM focuses on automation, but also considers process improvement and decision support. Another difference between BPM and RPA is that Business Process Management Systems built on BPM lifecycle need a database and a data model to store data, while RPA does not store any transactional data [21]. Some authors propose the integration of RPA into BPM lifecycle [6, 7, 12]. The mapping of BPMN patterns on Workflow Patterns defined by [19] are described in [22]. Yamasathien S. and Vatanawood W. use basic BPMN Workflow Patterns in order to formalize business processes using Promela [23]. The correctness of business processes is validated by SPIN model checker. A similar approach using Promela and SPIN is proposed by [17], where BPMN Choreography Diagrams are analysed.

In this paper, the term *concept* refers to the data stored in a database table, while *entity* refers to a row/record of a database table. A *concept* always describes a set of *entities*.

The remainder of the paper is structured as follows: Section 2 discusses related work to the tool proposed in the paper, Section 3 presents the methods used for discovering *primary blocks*, Sections 4 and 5 outlines representation and composition of *primary blocks*, and Section 6 discusses the automatic execution of complex business processes. The paper ends with conclusions and future work.

2 Related work

2.1 Robotic Process Automation

Robotic Process Automation (RPA) is defined as the application of specific methodologies and technologies that aim to automate repetitive tasks achieved usually by human users [8,10]. RPA frameworks (e.g., UiPath[1] , Automation Anywhere[2] , Blue Prism[3] , Microsoft Power Automate[4] , etc.) are designed to develop software robots that improve the business environment in various ways. RPA refers to those tools that operate on the user interface (UI) aiming to perform automation tasks using an "outside-in" approach. The information systems are kept unchanged, compared to the traditional workflow technology, that allows the improvement using an "inside-out" approach [18].

[1]https://www.uipath.com/
[2]https://www.automationanywhere.com/
[3]https://www.blueprism.com/
[4]https://powerautomate.microsoft.com/en-us/

Software robots mimic the actions achieved by humans through the use of the keyboard and the mouse in interactions with various available software. They ensure work productivity increase and costs reduction through enhanced accuracy, being available 24/7 compared to the human user. The use of software robots allows the reduction of the rule-based and repetitive work achieved by the employees and contributes to business process standardization and enhancement. In addition to reproducing human user UI interactions (mouse clicks, data entering, etc.), RPA frameworks improve software robots' potential by integrating into their actions AI capabilities and/or interactions that go beyond the user interface [4, 16]. Commercial RPA tools provide capabilities in *process mining*[5] and *process discovery*[6] . Still, human expert involvement is needed to perform analysis and decision-making according to specific process requirements. There is significant interest from academic research groups in optimizing Business Process Management in various aspects, especially in automatically extracting process steps and converting them into a software robot sequence of actions. There are several approaches that are related to the work presented in this paper.

The *first* research area is focused on studying the anatomy of tasks from the natural language descriptions of the process that details the executed routines. The automatic identification of the type of performed activities (manual, human interaction, or automated) from text documents while employing supervised machine learning techniques was investigated in [15]. In order to identify the existing relationship between various activities of a process, the authors of [9] used long short-term memory (LSTM) recurrent neural networks to learn from process description documents (PDDs). Paper [11] proposes a new grammar for complex workflows with chaining machine-executable meaning representations for semantic parsing.

The current development of RPA tools makes use of AI advances in routine identification and automation. Still, the use of human expert skills is required to analyze how the routines are executed on the application's UI. A *second* research area addresses the actual automation of routines by examining the actions performed by human users when executing their tasks using software applications.

Robidium [13] is a tool that discovers automatable routine tasks from the user interface (UI) logs and generates RPA scripts to automate these routines. This is a Software as a Service (SaaS) tool that implements the robotic process mining pipeline proposed in [14]. *Robidium* uses UI log files that consist of data and events that are not related to a specific task identified

[5]https://www.uipath.com/product/process-mining
[6]https://www.uipath.com/product/task-capture

beforehand. Its architecture emphasizes a preprocessing step on UI logs, that allows the routine extraction and discovery of automatable routines that are compiled into a UiPath script.

SmartRPA [1] is a cross-platform tool that attempts to tackle the discovery and the automation of routine tasks, that current practice proves to be time-consuming and error-prone. The tool uses its own action logger to record UI actions on the actions system, Microsoft Office applications, or web browser (e.g., Google Chrome, Mozilla Firefox) into a log file, used as input for routine identification. The tool allows the generation of a high-level flowchart diagram that can be studied by expert users for potential diagnosis operations and to generate executable RPA scripts based on the most frequent routine variant. Some input fields of the selected routine variant can be personalized before executing the related RPA scripts, supporting those steps that require manual user inputs.

There are also tools that automate users' actions. Ringer is a web replayer developed as a Chrome extension. Based on the trace of DOM (i.e. Document Object Model) events performed by users, it provides a script replaying users' actions [2]. This tool is only used for recording users' actions. Rousillon uses Ringer to further develop complex web automation scripts using Helena web scripting language [5].

In contrast with *Robidium* [13] and *SmartRPA* [1], our tool does not use an external logging tool to record user actions. Instead, it uses the human user actions at UI level in order to compute a navigation flow in the target web application and it uses the web DOM itself and the data model of the target web application in order to identify the elements that are available on UI and can be included in future interactions. As a result, the tool allows the generation of the workflow blocks together with the inventory (or a map) of the UI elements the user may interact with (inputs, buttons, etc.). The identification of the relevant interaction elements is accomplished by considering a data model that corresponds to the concepts processed within the examined software application. Robidium, SmartRPA and Ringer are all record-and-replay tools. Both *Robidium* and *SmartRPA* emphasize a particular type of routine, i.e., filling out a web form (Google Forms or other web application) with data taken from a desktop application (Microsoft Office Excel file), that is equivalent to inserting a new record into a database. These are very simple examples. Our tool is capable of discovering processes (i.e. *primary blocks*) on complex business web applications (e.g. CRM and ERP apps.) which imply complex browsing flows in the target web application. Also, our tool can automatically execute complex business processes that the human user didn't even recorded/executed (because complex automatizable business processes can be composed from individual *primary blocks* discov-

ered by our tool and these complex business process can be automatically executed by our tool); i.e. our tool is not a simple record-and-replay tool. Also, our browser plugin is capable of *understanding the UI* using the data model of the target web application - so the human user does not need to indicate to the plugin relevant input fields or property controls on the UI.

3 Primary Blocks construction/discovery

Existing RPA platforms have powerful features like OCR, Computer Vision, or DOM-based for *recognizing* UI controls, but they do not have a *deep understanding* of the UI of an application (e.g. they don't detect structural elements and structural relations like all the controls belonging to a panel/tab, detecting complete menus made of menu items, detecting entities operated on by the UI of the application, etc.). By abstracting the whole UI of the target web application to data operations (CRUD) on the database, our tool brings a semantic, more interconnected understanding of the UI and the whole application. This makes process automation possible on a higher, more abstract layer that just simple recording and replaying UI events.

The set of *primary blocks* discovered by the browser plugin functions similarly to a basis in a vector space. The *primary blocks* may be combined with special operators (that will be introduced later in the paper) like the sequence operator (i.e. logical AND) in order to form complex automatable business processes.

The browser plugin determines automatically the *primary blocks* corresponding to CRUD operations on all the concepts in the database of the applications. The human user still has to guide the plugin (i.e. the human user has to navigate) through the target web application, but there is no other required input from the human user except this navigation through the screens of the web application and the initial settings of the plugin for a target web application - which consists of the Data Model of the database used by the target web application. The human user mainly has to trigger click events in the targeted web application's UI in a logical order (so that to register the order of clicks on the UI, by navigating through the web application from the first web page to the desired UI form, which allows the user to perform a specific CRUD operation in the database) and the browser plugin understands all the web content that is loaded in the browser like: identifying HTML elements/tag that the user clicked on, identifying entities or concepts from the database shown in the HTML document loaded in the browser window, associating HTML text input fields with the data fields of an entity from the backend database etc.

The browser plugin can not determine automatically *partial CRUD operations* like a functionality of the application that links two entities together from the database using foreign keys (this would be implemented using an SQL Update operation that updates only one or two fields of those entities - the foreign key value) or *cross-concepts operations* like the functionality of showing complex reports implemented with multiple SQL Select queries on different tables joined together by the JOIN operator. But these *partial CRUD or cross-concepts operations* can still be recorded/saved by the plugin and the human user can tag them (i.e. annotate them) as such.

The browser plugin functionality is depicted in Fig. 1. The user starts the process of recording a new *primary block* by clicking on the "Start recording primary block" button. Up to the point when the user clicks on the "End recording primary block" button [7], all actions happening on the UI of the target web application will be recorded by the plugin and at the end, they will be serialized in a JSON format and persisted to a database using an Rest API server or saved to the browser's local storage. But the plugin is not a record and replay tool for web UI actions, it is more than that - it abstracts UI actions to semantic/conceptual operations on the database.

During the time when the *primary block* is recorder/discovered, the plugin has two states: *idle* and *processing*. When the plugin is *idle* (depicted in Fig. 1 by the green progress bar), it awaits for a UI operation (i.e. a click event) from the human user. After the user clicks on a button or other clickable element in the web page, the plugin saves to local storage the signature of the element that was clicked and then moves to the state of *processing*. In this new state, it waits until the result of the click event on the UI is finished (i.e. this would usually imply, for single-page application, that one or more XHR requests are sent by the browser to the HTTP server and a part of the loaded document (i.e. DOM) is updated). After the handling of the click event is complete, the plugin determines the *diff DOM* (i.e. the part of the DOM that was updated) and tries to identify the concept which is rendered in this *diff DOM*, if possible, and the operation (i.e. Select, Update, Insert, Delete or SelectAll) on this concept that is facilitated by this *diff DOM*. For example, if the user clicks a "New" button that constructs a form which allows the user to insert a new entity of type "Account" to the database, then this *diff DOM* (i.e. the web form) is semantically defined by the concept "Account" and the operation "Insert".

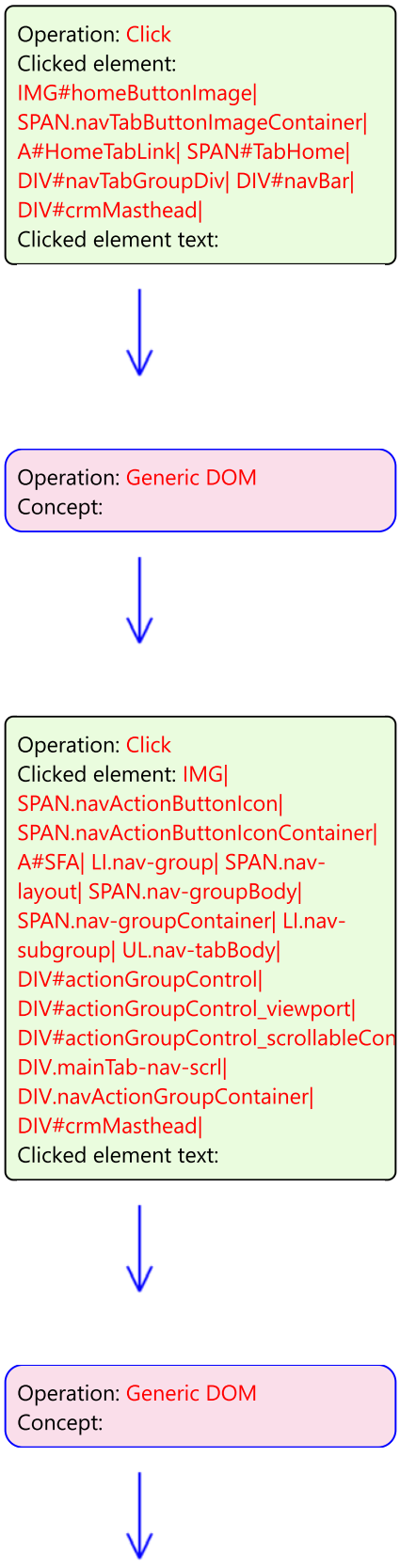
So, actions happening on the UI of the target web application are recorded

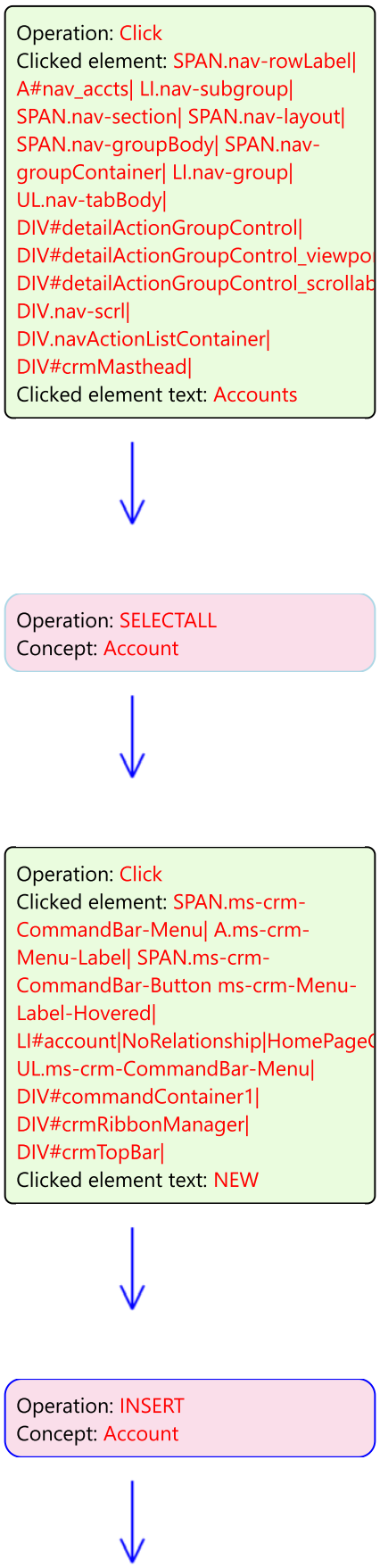
[7] please note that the "End recording primary block" button is not visible in Fig. 1 as this button replaces the "Start recording primary block" button, after the user clicks on it to trigger the recording of a new *primary block*

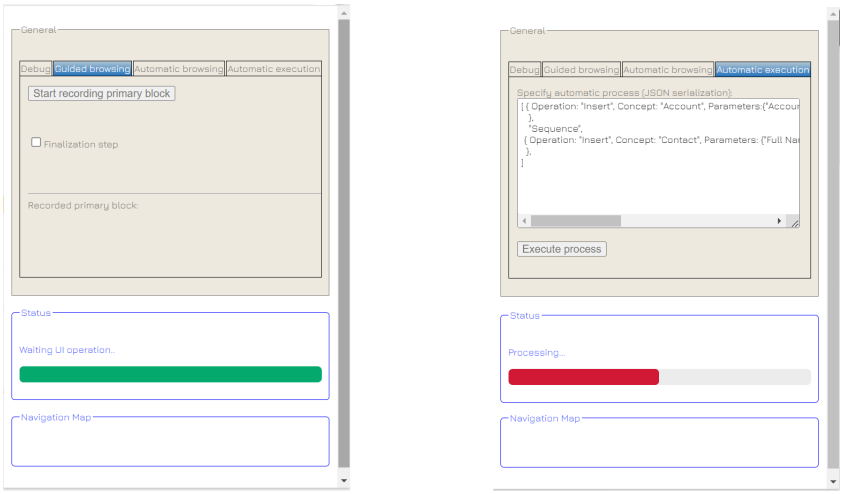
in pairs: a click event is first recorded (by saving the signature of the DOM element that was clicked - the signature is just its XPATH up to the document's <body>tag) and the resulted *diff DOM* is then saved in a semantically abstract form defined by: *concept* : *operation*. The *concept* is just a concept from the data model of the target web application (i.e. from the database) and *operation* is one of the following: *Insert*, *Update*, *Delete*, *Select* and *SelectAll*, although the human user can also annotate custom operations. These operation are the basic CRUD operations on a concept in the database; we distinguish *SelectAll* which basically means selecting several entities of a concept (not necessarily all entities in the database) from *Select* because the entities resulted from a *SelectAll* are usually displayed as a list or a table on the UI of a web application (which is different than the way a single entity resulted from a *Select* operation is rendered on the web UI). If the plugin does not identify any concept or entity from the database in the *diff DOM*, then this *diff DOM* will be represented as *Concept: "Generic DOM", Operation: ""*.

When the user finally clicks the "End recording primary block" button, the recording of a *primary block* is complete and the user can tag this chain of UI actions / events with a custom string; by default, the tag/name of the *primary block* chain is the *concept* : *operation* pair discovered by the plugin at the end of the recording of the primary block (e.g. "Account" : "Update" pair). A full example of a *primary block* recorded in the CRM web application Microsoft Dynamics 2016 CRM, i.e. the *primary block* of the CRUD operation: *Account : Insert* is visible below (i.e. all the steps from the *primary block*, except the finalization step are visible).

Show navigation




Operation: Click
Clicked element: SPAN.nav-rowLabel| A#nav_accts| LI.nav-subgroup| SPAN.nav-section| SPAN.nav-layout| SPAN.nav-groupBody| SPAN.nav-groupContainer| LI.nav-group| UL.nav-tabBody| DIV#detailActionGroupControl| DIV#detailActionGroupControl_viewpo DIV#detailActionGroupControl_scrollab DIV.nav-scrl| DIV.navActionListContainer| DIV#crmMasthead|
Clicked element text: Accounts
Operation: SELECTALL
Concept: Account
Operation: Click
Clicked element: SPAN.ms-crm-CommandBar-Menu| A.ms-crm-Menu-Label| SPAN.ms-crm-CommandBar-Button ms-crm-Menu-Label-Hovered| LI#account|NoRelationship|HomePageG UL.ms-crm-CommandBar-Menu| DIV#commandContainer1| DIV#crmRibbonManager| DIV#crmTopBar|
Clicked element text: NEW
Operation: INSERT
Concept: Account



(a) The "Primary block Discovery" UI (b) The "Automatic execution" UI

Figure 1: The user interfaces of the browser extension

Although a *primary block* can contain a sequence of pairs of the form: "UI Click event", "Conceptual operation", we can group these pairs in some templates/models, that are valid for all CRUD operations for all concepts (and for all web applications). Most CRM applications process use specific sets of business concepts among which there exist a wide variety of relationships. CRUD operations that are performed over the business domain concepts allow data processing that is finally stored in a database. Fig. 2 shows the primary building block that corresponds to the *concept detection* as a first action achieved for almost all other primary blocks, i.e., insert, update, delete, select, select all. The action that defines the *Detect Concept* primary block consists of clicking on the UI element that corresponds to the abstract concept included in the data model, e.g., Contact, Account, etc. As a subsequent result, the content of the DOM object is updated. Every primary building block makes use of some parameters ensuring the reusability of that particular operation.

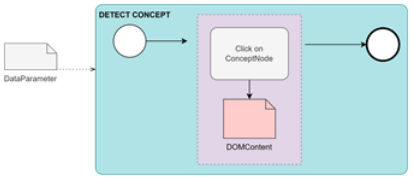


Figure 2: *Detect concept* operation representation

Detect Entity emphasized by Fig. 3 is the primary building block that is connected after the *Detect Concept* block for several CRUD operations, as it is required to indicate the actual data to act on for operations like update or delete. The identification of the particular entity to operate on suggests the need to have to perform several click operations in order to identify the desired entity indicated by the input parameters.

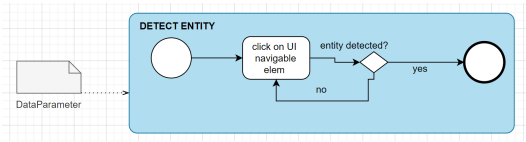


Figure 3: *Detect entity* operation representation

The actions accomplished during CRUD operations may be organized into three sections, according to the particularity of every operation: *pre-operations* (i.e. operations required before we can execute the main conceptual operation), *specific operations* (i.e. the main conceptual operation), and *post-operations* (i.e. usually these are final confirmations from the user of the main operation and are usually achieved by click events on 'Confirm' or 'Save' or 'Ok' buttons). For example the *Insert* conceptual operation depicted in Fig. 4 requires achieving first the concept identification only, without any entity to operate on. Subsequently, the actions refer to choosing the UI element that allows entering new data (usually a `New` button), followed by the data entering step (filling in the input texts). The *Insert* operation finishes with the confirmation of adding new data into the database (e.g. clicking on a 'Save'-like button).

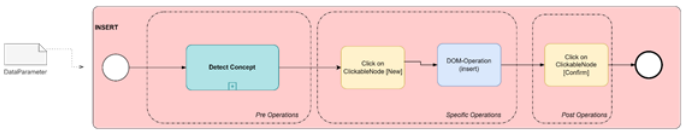


Figure 4: *Insert* operation representation

Fig. 5 and Fig. 6 emphasize similar templates for the *Update* and *Delete* operations on concepts, acting on some existing entity that should be updated or deleted. Both *Detect Concept* and *Detect Entity* operations should be executed as pre-operations. *Select* operation and *Select All* operation are different from other CRUD operations in terms of the category of the performed actions. They allow data extraction from the database and its presentation to the user, altering the content of the DOM object. See Fig. 7 and 8.

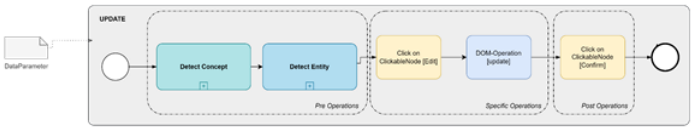


Figure 5: *Update* operation representation

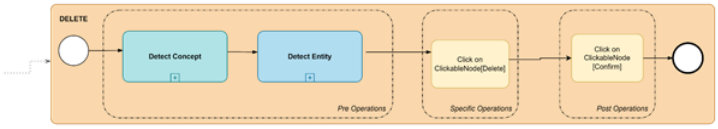


Figure 6: *Delete* operation representation

Depending on the conceptual operation (i.e. Update, Delete, Insert, Select, SelectAll) and on the targeted web application, as we have seen in the above templates, usually a *primary block* ends with UI click events that are meant to confirm the operation by the user; for example, an Insert conceptual operation will usually end with a click on a 'Save' or 'Confirm' button; an Delete conceptual operation will usually end with a click on 'Are you sure ?' or 'Confirm' buttons. These click events form the *post-operations* and these steps have to be signalized as such to the plugin by checking the 'Finalization steps' checkbox of the plugin (so that when the plugin switches to the automatic execution of the *primary block*, it knows that the DOM that allows the main conceptual operation to be perform lies in the chain before these finalization steps).

As we said before, the recorded *primary block* is automatically tagged by the plugin with the text *concept* : *operation* where *concept* and *operation* represent the conceptual operation discovered by the plugin at the end of the recording of the primary block (e.g. "Account" : "Update"). The plugin also allows the user to change this tag/name of the recorded *primary block* if he/she wants. The user can also change the attributes of the pre-recorded steps that form the *primary block*.

3.1 Detecting concepts and conceptual operations in the DOM

The detection of an operation on a concept in the current DOM (i.e. the difference DOM, *diffDOM*) happens according to the algorithm depicted in listing 1. The algorithm begins with the function $TableOfEntitiesDetected(diffDOM)$ which tries to detect any tables in the *diff DOM*. Tables are important because the may indicate a SelectAll operation on a concept. We can not

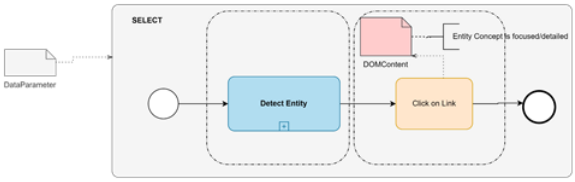


Figure 7: *Select* operation representation

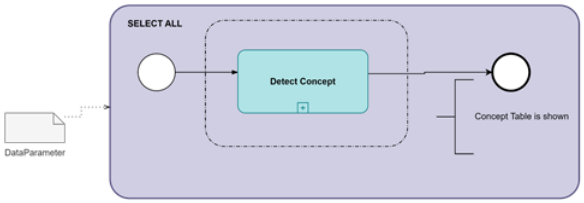


Figure 8: *Select All* operation representation

present here the body of this function due to space constraints, but in essence, the algorithm for detecting a tables tries to horizontally align leaf html tags containing only text into horizontal row clusters and then tries to vertically align these horizontal rows into a table. After detecting a table, the algorithm tries to detect if the entities of a concept are rendered in that table through function $DetectConceptInTable(diffDOM)$. Again, we omit the details here, but simply said, the concept is determined by searching its attributes in the table cells of the table header or, if these are not found (e.g. the table may not have a header), the concept is determined by inspecting (i.e. triggering click events) the entities from the first two rows of the table; detecting an entity is done similarly to the way function $FindOneConcept(diffDOM)$ determines that an entity (of that concept) is rendered in the $diffDOM$: several text attributes (labels) of a concept from the Data Model are found in the $diffDOM$ - more than 75% of the labels of a concept are found.

Next, if there was no *SelectAll* or *Delete* operation on a concept/entity detected, the algorithm tries to find text input elements/tags (i.e. "<input type=text >,<select >,<textarea >") associated to the text labels (i.e. the text attributes of the found entity). We do this in the $FindInputFieldsForTextLabels()$ function. The associated text input tag is searched in the neighborhood of the label, more specifically, the associated input tag is searched in the South-East quarter of a circle with the center in the middle of the label - the closest such text input tag from this South-East quarter is associated to the label. If no such input html tags were detected, that the operation is *Select* on an entity of the detected concept. If, on the other hand, there was text input html elements found in the *diffDOM*, but they are empty (i.e. don't contain

values), the conceptual operation would be *Insert*. Otherwise, the conceptual operation is *Update* on an entity of the detected concept.

Algorithm 1 The conceptual operation detection in DOM algorithm

The DetectConceptualOperation algorithm is:

if $TableOfEntitiesDetected(diffDOM) == TRUE$ **then**
  $DetectedConcept = DetectConceptInTable(diffDOM)$
  return $(Concept : DetectedConcept, Operation : SELECTAll)$
else
  if $(DetectedConcept = FindOneConcept(diffDOM))! = NULL)$ **then**
    {Several text attributes (labels) of a Concept from the Data Model are found in the DOM}
    if (The previous click event was triggered on a tag with the caption "DELETE") **then**
      return $(Concept : DetectedConcept, Operation : DELETE)$
    end if

    $InputFieldsFound = FindInputFieldsForTextLabels()$
    if $InputFieldsFound == FALSE$ **then**
      return $(Concept : DetectedConcept, Operation : SELECT)$
    else
      if (Found Text Input elements are empty (i.e. don't contain values)) **then**
        return $(Concept : DetectedConcept, Operation : INSERT)$
      else
        return $(Concept : DetectedConcept, Operation : UPDATE)$
      end if
    end if
  end if
end if

3.2 Configuration settings for the plugin

The initial settings of the plugin for a target web application are the following:

- URL of the target web application together with access credentials
- the Data Model of the database used by the target web application

The Data Model configuration does not need to match exactly the structure of the database used by the application, but it should match the text labels used for each concept of the database on the UI of the application. This is why, primary access to the database used by the target web application is not required in order to use the plugin.

A small snippet from a data model example used for the Microsoft Dynamics 2016 CRM application is given below in JSON format:

```
DataModel = {
    "Account" : ["Account Name", "Phone", "Fax", "Website", "Parent Account", "Ticker Symbol",
    "Address", "Primary Contact", "Description", "Industry", "SIC Code", "Ownership"],

    "Contact" : ["Full Name", "Job Title", "Account Name", "Email", "Business Phone", "Mobile Phone",
    "Fax", "Preferred Method of Contact", "Address", "Gender", "Marital Status", "Birthday",
```

```
    "Spouse/Partner Name", "Anniversary", "Personal Notes", "Company", "Originating Lead",
    "Last Campaign Date", "Marketing Materials", "Contact Method", "Email", "Bulk Email", "Phone",
    "Fax", "Mail"],

    "PrimaryKeys" : [{Concept : "Account", PrimaryKey : "Account Name"}]

    "ForeignKeys" : [{ForeignKey : "Company", ForeignConcept : "Contact", PrimaryKey : "Account Name",
    PrimaryConcept : "Account"}]
        ...
};
```

The data model describes the concept *Account* with its associated attributes and the concept *Contact* with its associated attributes. It also describes that the *Company* attribute from *Contact* is a foreign key and refers to the primary key *Account Name* from the *Account* concept. Again, we emphasize that these attributes don't have to match exactly the attributes of the database tables, but instead they have to match the text labels for the respective entities on the UI of the target web application. The foreign key relation is used when for example the plugin tries to add a new (i.e. an Insert operation) entity of type *Contact* and the UI interface of the Microsoft Dynamics CRM application only allows the input of values for the main attributes of *Contact* entity and assumes that the *Account* to which this new *Contact* entity is linked already exists in the database, so the UI only allows the user/plugin to specify the *Account Name* attribute of the *Account* entity, without specifying other attributes for the *Account*.

4 Representation of primary building blocks

Internally, a *primary block* is represented as a JSON object. The representation stores the *pre-operations*, *specific operations* and *post-operations* of the *primary block* as a sequence of steps. But except the *post-operations* which are usually represented as a UI click event on a 'Save'-like button, the steps from *pre-operations* and *specific operations* come as pairs: a UI click event and a conceptual operation in the *diff DOM*. An example of a JSON representation of a $\{concept : "Account", operation : "Insert"\}$ *primary block* for the Microsoft Dynamics CRM 2016 web application is depicted in the following listing.

```
PrimaryBlock=[
     {operation: "Click", target: ".../IMG#homeButtonImage", targetText : ""},
     {concept: "", operation : "Generic DOM"},
     {operation: "Click", target : ".../IMG", targetText : ""},
     {concept: "", operation : "Generic DOM"},
     {operation: "Click", target: ".../SPAN.nav-rowLabel", targetText: "Accounts"},
     {concept : "Account", operation: "SELECTALL"},
     {operation: "Click", target: ".../SPAN.ms-crm-CommandBar-Menu", targetText: "NEW"},
     {concept: "Account", operation: "INSERT"},
```

```
    {operation: "Click", target: ".../BUTTON#saveBtn", targetText: "SAVE", finalizationStep: "True"}
]
```

In this listing, the UI click operations are identified by the attributes: operation="Click", the target of the click event (i.e. the html tag that was clicked) identified by its XPATH (only the last step of the XPATH is present in the listing for brevity reasons) and the textual caption of the target tag (e.g. the text shown by the button that was clicked); sometimes there is no such target text as the button may just contain an icon and no text (for example a delete button can contain a recycle bin icon). A conceptual operation identified in the *diff DOM* is specified in the JSON using the attributes: *concept* and *operation* (i.e. Select, SelectAll, Insert, Update, Delete); sometimes, when the plugin identifies no operation on a concept, then only a $\{concept : "", operation : "GenericDOM"\}$ generic step is saved. The last step from the listing identifies the *post-operations* step (specified by the $finalizationStep$ attribute).

A graphical representation of a small part from the *primary block* represented as JSON in listing 4 is visible in fig. 9. The graphical representation of the full *primary block* is presented above. Another representation of a *primary block* that can be interpreted by a business analyst is obtained by transforming the JSON serialization into XPDL serialization, which can be visualized graphically using a BPMN tool (e.g. Together Workflow) as in Fig. 10. Similarly, diagrammatic visualizations can be generated for complex processes, by composing the XPDL serialization of the *primary blocks*.

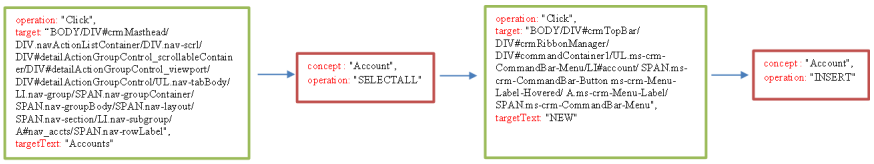


Figure 9: Snippet of an $\{concept : "Account", operation : "Insert"\}$ *primary block* for the Microsoft Dynamics CRM 2016 web application

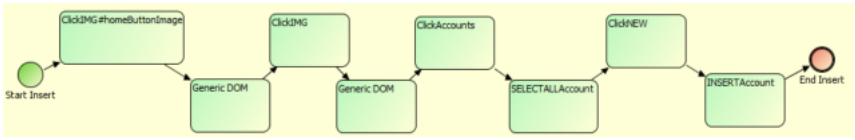


Figure 10: BPMN representation of XPDL format for (Account : Insert)

5 Composition of complex business processes using primary navigation/building blocks

In 2003, van der Aalst et al. defined a series of 20 workflow patterns [19]. Their initial goal was to use these patterns as qualitative criteria in workflow systems' assessment or standards' assessment. A comparison based on the workflow patterns, where BPMN models and UML Activity Diagrams are analysed, is depicted in [20]. Another similar approach that analyzes BPMN models, workflow patterns and YAWL models is presented in [3]. Wohed et al. studied the application of these patterns on BPMN models [22]. They do not focus only on control-flow perspective, but also resource and data perspectives are analyzed. Control-flow patterns are described in 6 categories: Basic control-flow patterns, Advanced branching and synchronisation patterns, Structural patterns, Multiple instances patterns, State-based patterns, and Cancellation patterns. A similar approach where business process models are built based on BPMN Workflow Patterns is treated in [23].

The *Sequence* pattern involves at least 2 tasks that should precede and succeed each other. This pattern requires tasks and sequence flows concepts. Modeling parallel tasks is performed using *Parallel Split* and *Synchronization* patterns, while the choice of one branch out of multiple branches is described by *Exclusive Choice* and *Merge* patterns. These patterns use tasks and gateways (exclusive of parallel gateways).

Complex processes could be obtained by combining *primary blocks* using Basic control-flow patterns [22]. To achieve this we may consider *primary blocks* as sub-processes, and assimilate them to tasks. Sequencing the CRUD operations is a natural composition, but we may also allow discriminating composition based on some conditions (i.e. Insert or Update depending on the existence or not of the considered entities) or even parallel composition (e.g. several Insert operations executed in parallel). Parallel composition is achieved by operating on two or more browser tabs in which the target web application is loaded. The composition correctness is simple to prove for sequencing and/or discriminating compositions; for parallel composition, some extra conditions should be added for data race exclusion.

6 Automatic execution of complex business processes. Experiments

A complex, automizable business process is a sequence of *primary block*s linked by the operators presented in the previous section. An example of

such an automizable business process is presented in the following listing.

```
ComplexProcess=[
    {Operation: "Insert", Concept: "Account", Parameters:{"Account Name": "UBB",
    "Phone" : "0000000000", "Fax" : "000000000", "Website" : "www.company.com", ...} }
    Operator,
    {Operation: "Insert", Concept: "Contact", Parameters:{"Full Name": "John Doe",
    "Job  Title": "Professor", "Account Name" : "UBB", "Email" : "John.Doe@company.com",
    ...} },
    Operator,
    ...
]
```

The complex process is given in the listing in a JSON representation. The Operator can be one of *"Sequence"*, *"Choice"* and *"Parallel"*. The *"Choice"* operator is implemented using *Exclusive choice* and *Merge* patterns, and the *"Parallel"* operator is implemented using the *Parallel Split* and *Synchronization* patterns. Each *primary block* in the listing is identified by the *Operation* and *Concept* pair and also contains the *Parameters* of the *primary block* (e.g. the $\{Operation : "Insert", Concept : "Account"\}$ *primary block* specifies value parameters for each property of the Account entity that is to be inserted in the database). Of course, there can be more than two *primary blocks* linked by different operators in a complex automizable process.

The plugin's UI allows the user to specify a complex process for automatic execution in the above JSON format (see Fig. 1 (b)).

The full code of our browser plugin is available at:
https://github.com/KiralyCraft/WAPlugin

We tested the *primary block* discovery and the automatic execution of complex processes functionalities of our plugin on 3 business applications: 1) Microsoft Dynamics 2016 CRM [8], 2) Microsoft Dynamics 365 Business Central [9] and 3) Atlassian Jira [10]. We used similar complex processes as the one in the above JSON listing, having 2-3 operators. The examples of complex processes we automatically executed on Microsoft Dynamics 2016 CRM application are presented on the following listing. We used similar processes for the other two applications for experimentation. Our browser plugin was able to successfully discover the *primary blocks* and execute the complex processes in the 3 aforementioned business applications. Some screenshots and additional details of these experiments are presented in the figures below.

```
ComplexProcess1=[
```

[8] https://learn.microsoft.com/en-us/lifecycle/products/dynamics-crm-2016-dynamics-365

[9] https://dynamics.microsoft.com/en-us/business-central/overview/

[10] https://www.atlassian.com/software/jira

```
    {Operation: "Insert", Concept: "Account", Parameters:{"Account Name": "UBB", "Phone": "00000000000",
    "Fax": "00000000000", "Website": "www.ubbcluj.ro", "Parent Account": "", "Ticker Symbol": "a",
"Address": "Str. M. Kogalniceanu, Cluj", "Primary Contact" : "Virginia Niculescu", "Description": "University",
"Industry": "Academic", "SIC Code": "00000", "Ownership": ""} }
    Operator,
    {Operation: "Insert", Concept: "Contact", Parameters:{"Full Name": "Adrian Sterca",
    "Job  Title": "Professor", "Account Name" : "UBB", "Email" : "adrian.sterca@ubbcluj.ro",
    "Business Phone": "00000000000", "Mobile Phone": "00000000000","Fax": "000000000",
    "Preferred Method of Contact": "telephone", "Address": "Str. M. Kogalniceanu, Cluj", "Gender": "M",
    "Marital Status": "married", "Spouse/Partner Name": "", "Birthday" : "20/01/2020", "Anniversary": "",
    "Personal Notes": "", "Company": "UBB", "Originating Lead": "", "Last Campaign Date": "",
    "Marketing Materials": "", "Contact Method": "", "Email": "other@gmail.com", "Bulk Email": "", "Phone": "",
    "Fax": "", "Mail": "" } },
 ]

ComplexProcess2=[
    {Operation: "Delete", Concept: "Account", Parameters:{"Account Name": "UBB"} }
    Operator,
    {Operation: "Update", Concept: "Contact", Parameters:{"Full Name": "Camelia Chisalita-Cretu",
    "Job  Title": "Professor", "Account Name" : "UBB", "Email" : "maria.cretu@ubbcluj.ro",
    "Business Phone": "00000000000", "Mobile Phone": "00000000000","Fax": "000000000",
    "Preferred Method of Contact": "telephone", "Address": "Str. M. Kogalniceanu, Cluj", "Gender": "M",
    "Marital Status": "married", "Spouse/Partner Name": "", "Birthday" : "20/01/2020", "Anniversary": "",
    "Personal Notes": "", "Company": "UBB", "Originating Lead": "", "Last Campaign Date": "",
    "Marketing Materials": "", "Contact Method": "", "Email": "other@gmail.com", "Bulk Email": "", "Phone": "",
    "Fax": "", "Mail": "" } },
 ]

ComplexProcess3=[
    {Operation: "SelectAll", Concept: "Account", Parameters:{"Account Name": "UBB"} }
    Operator,
    {Operation: "Select", Concept: "Contact", Parameters:{"Full Name": "Camelia Chisalita-Cretu"} },
 ]
```

6.1 Limitations

In this subsection, we want to list some of the limitations of our browser plugin. Firstly, our tool is able to detect only entity CRUD operations in the UI, other UI operations which do not translate to CRUD operations in the database (e.g. operations that modify a single property of an entity), must be tagged manually by the user (also using our browser plugin). Secondly, we have tested our tool on 3 commercial, business web applications and although, we based our plugin implementation on common web design principles (like the fact that an input field is always placed in the web UI either on the right or below or in the south-east part of its corresponding text label), it may not work correctly on other business web applications. As shown in Figs. 4 - 8, we used general templates for CRUD operations which are not specific to a particular web application, so our web automation plugin should, in principle, work with any typical business web applications that uses a relational database. Another limitation is the fact that, currently, the complex process which is executed automatically must be specified by the

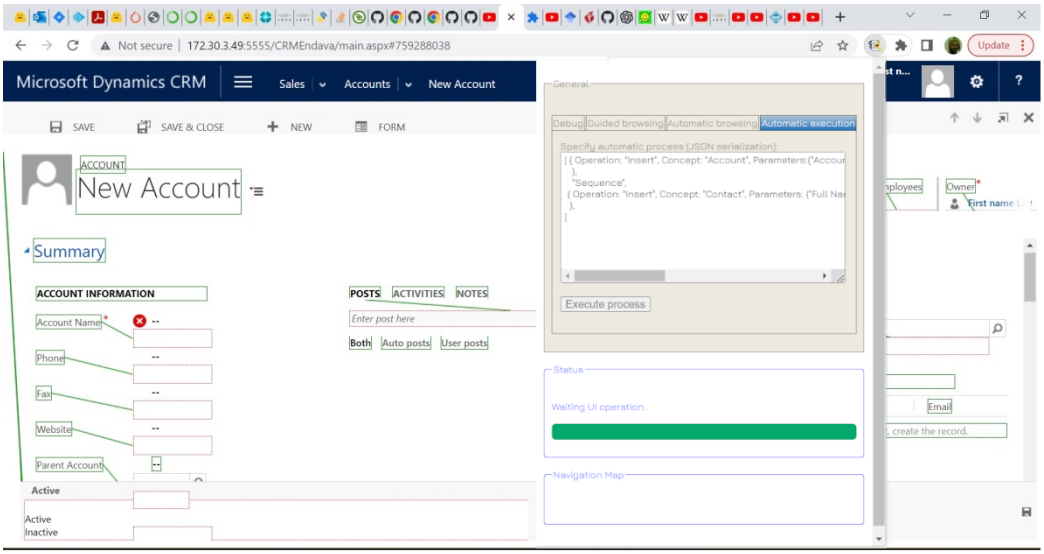


Figure 11: A screenshot performing an (Account : Insert) in Microsoft Dynamics CRM 2016 (the link between labels and text inputs of the Account entity are outlined with green lines by the plugin)

human user in the JSON format which is not necessary business friendly. Other limitation is that currently, once saved, we can not edit a recorded *primary block*.

7 Conclusions and Future Work

We presented in this paper a browser tool that can be used in order to semi-automatically detect basic business processes in business web applications, construct more complex business processes based on these and execute them automatically later. Our tool maps UI operations in the target business web application to conceptual operations in a database. The user of our tool is not required to have any programming knowledge, all he/she needs to do is to provide a small startup configuration (like the URL of the target web application, credentials for accessing it and the concepts from the database with their properties) and to "guide" the plugin in the discovery of basic business processes (i.e. *primary blocks*) by browsing through the target web application. This is different than the current RPA platforms which function on a record-and-replay principle (i.e. they can only execute automatically what they have previously recorded in the UI) and also require the user to write code in order to program the automated process. We have shown that our idea is viable by experimenting on 3 commercial, business web applications.

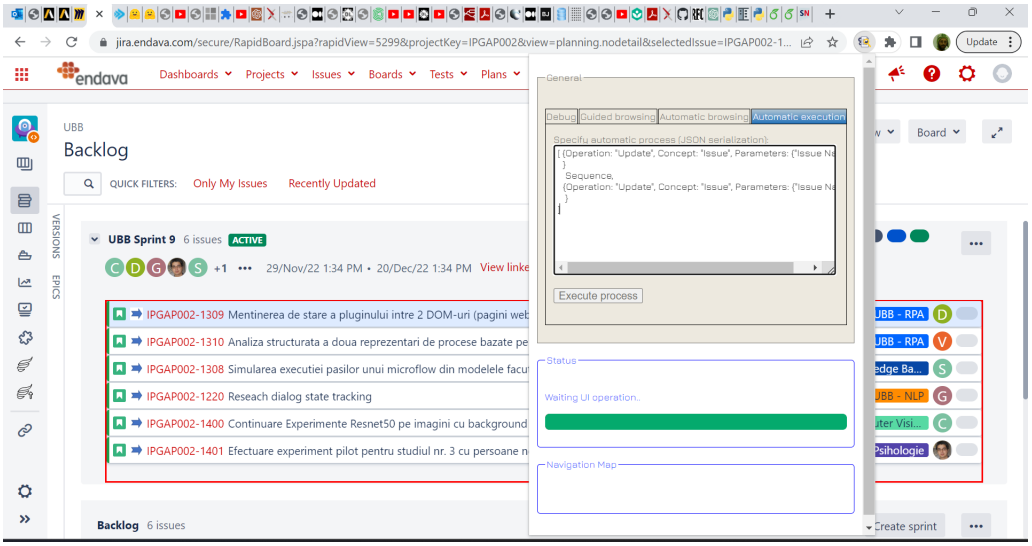


Figure 12: A screenshot performing an (Issue : SelectAll) in Atlassian Jira (the table with the Issue entities is highlighed in red by the plugin)

As future work, we intend to test our tool on other business web applications, to improve its execution interface with the user through NLP (Natural Language Processing), to automatically discover the entities and concepts used by the target web application, to automatically detect operations different than CRUD operations and to allow the user a greater flexibility in editing the recorded *primary blocks*.

References

[1] Agostinelli, S., Lupia, M., Marrella, A., and Mecella, M. *Automated Generation of Executable RPA Scripts from User Interface Logs*. 2020.

[2] Barman, S., Chasins, S., Bodik, R., and Gulwani, S. Ringer: web automation by demonstration. In *Proceedings of the 2016 ACM SIGPLAN international conference on object-oriented programming, systems, languages, and applications* (2016), pp. 748–764.

[3] Börger, E. Approaches to modeling business processes: a critical analysis of bpmn, workflow patterns and yawl. *Software & Systems Modeling 11* (2012), 305–318.

[4] Chakraborti, T., Isahagian, V., Khalaf, R., Khazaeni, Y., Muthusamy, V., Rizk, Y., and Unuvar, M. From robotic process automation to intelligent process automation: Emerging trends. *CoRR abs/2007.13257* (2020).

[5] Chasins, S. E., Mueller, M., and Bodik, R. Rousillon: Scraping distributed hierarchical web data. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (2018), pp. 963–975.

[6] Flechsig, C., Lohmer, J., and Lasch, R. Realizing the full potential of robotic process automation through a combination with bpm. In *Logistics Management: Strategies and Instruments for digitalizing and decarbonizing supply chains-Proceedings of the German Academic Association for Business Research, Halle, 2019* (2019), Springer, pp. 104–119.

[7] Flechsig, C., Völker, M., Egger, C., and Weske, M. Towards an integrated platform for business process management systems and robotic process automation. In *Business Process Management: Blockchain, Robotic Process Automation, and Central and Eastern Europe Forum: BPM 2022 Blockchain, RPA, and CEE Forum, Münster, Germany, September 11–16, 2022, Proceedings* (2022), Springer, pp. 138–153.

[8] for Robotic Process Automation, I. Introduction to robotic process automation. a primer, June 2015.

[9] Han, X., Hu, L., Dang, Y., Agarwal, S., Mei, L., Li, S., and Zhou, X. Automatic business process structure discovery using ordered neurons LSTM: A preliminary study. *CoRR abs/2001.01243* (2020).

[10] Hofmann, P., Samp, C., and Urbach, N. Robotic process automation. *Electronic Markets 30*, 1 (March 2020), 99–106.

[11] Ito, N., Suzuki, Y., and Aizawa, A. From natural language instructions to complex processes: Issues in chaining trigger action rules. *CoRR abs/2001.02462* (2020).

[12] König, M., Bein, L., Nikaj, A., and Weske, M. Integrating robotic process automation into business process management. In *Business Process Management: Blockchain and Robotic Process Automation Forum: BPM 2020 Blockchain and RPA Forum, Seville, Spain, September 13–18, 2020, Proceedings 18* (2020), Springer, pp. 132–146.

[13] Leno, V., Deviatykh, S., Polyvyanyy, A., Rosa, M. L., Dumas, M., and Maggi, F. M. Robidium: Automated synthesis of robotic process automation scripts from UI logs. In *Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track at BPM 2020 co-located with the 18th International Conference on Business Process Management (BPM 2020), Sevilla, Spain, Sept. 13-18, 2020* (2020), vol. 2673, CEUR-WS.org, pp. 102–106.

[14] Leno, V., Polyvyanyy, A., Dumas, M., Rosa, M. L., and Maggi, F. M. Robotic process mining: Vision and challenges. *Business Information Systems Engineering: The International Journal of WIRTSCHAFTSINFORMATIK 63*, 3 (2021), 301–314.

[15] Leopold, H., van der Aa, H., and Reijers, H. *Identifying Candidate Tasks for Robotic Process Automation in Textual Process Descriptions*. 01 2018, pp. 67–81.

[16] Rajawat, A. S., Rawat, R., Barhanpurkar, K., Shaw, R. N., and Ghosh, A. Chapter one - robotic process automation with increasing productivity and improving product quality using artificial intelligence and machine learning. In *Artificial Intelligence for Future Generation Robotics*, R. N. Shaw, A. Ghosh, V. E. Balas, and M. Bianchini, Eds. Elsevier, 2021, pp. 1–13.

[17] Solaiman, E., Sun, W., and Molina-Jimenez, C. A tool for the automatic verification of bpmn choreographies. In *2015 IEEE international conference on services computing* (2015), IEEE, pp. 728–735.

[18] Van-der Aalst, W. M. P., Bichler, M., and Heinzl, A. Robotic process automation. *Business and Information Systems Engineering 60* (08 2018), 269–272.

[19] van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. Workflow patterns. *Distributed and Parallel Databases 14*, 1 (2003), 5–51.

[20] White, S. A., et al. Process modeling notations and workflow patterns. *Workflow handbook 2004*, 265-294 (2004), 12.

[21] Willcocks, L. P., and Lacity, M. *Service automation robots and the future of work*. SB Publishing, 2016.

[22] Wohed, P., van der Aalst, W. M., Dumas, M., ter Hofstede, A. H., and Russell, N. Pattern-based analysis of bpmn.

[23] Yamasathien, S., and Vatanawood, W. An approach to construct formal model of business process model from bpmn workflow patterns. In *2014 Fourth International Conference on Digital Information and Communication Technology and its Applications (DICTAP)* (2014), IEEE, pp. 211–215.