



**UNIVERSITATEA DIN
BUCUREȘTI**



**FACULTATEA
DE
MATEMATICĂ ȘI INFORMATICĂ**

SPECIALIZAREA INFORMATICĂ

Disertație

RPA2Graph – Interpretarea workflow-urilor UiPath

Absolvent

Coteanu Vlad

Coordonator științific

Prof. dr. Ștefănescu Alin

București, septembrie 2021

Abstract

RPA (Robotic Process Automation) reprezintă o tehnologie software recent dezvoltată, ce presupune automatizarea proceselor de business cu ajutorul unor roboți software. Roboții sunt capabili să interacționeze cu sisteme digitale și aplicații software asemenea oamenilor. Principalul scop al unui robot software este de a prelua sarcinile repetitive ale unor oameni pentru ca aceștia să se poată concentra asupra altor activități. Unul dintre principalii actori pe piața deschisă de RPA este compania UiPath. Platforma oferită de ei se bazează, în linii mari, pe 3 componente: Studio, Robot și Orchestrator. Cu ajutorul Studio-ului, un RPA developer definește un proces de automatizare (workflow), care va fi executat ulterior de un Robot, execuția putând fi gestionată cu ajutorul platformei Orchestrator.

Dezvoltarea RPA, asemenea oricărui alt tip de dezvoltare software, trebuie să asigure o anumită calitate a produsului, astfel că testarea software este importantă și în acest segment de business. În cazul UiPath, în funcție de workflow-ul ce trebuie testat, există diferite metode de testare manuale, și, recent, metode automate, cu ajutorul platformei Test Suite. Un indicator în testarea automată îl reprezintă noțiunea de „acoperire”, iar o acoperire completă, fie a liniilor de cod (a activităților, în cazul workflow-urilor), fie a ramurilor posibile de execuție poate deveni o sarcină complexă. Automatizarea procesului de generare de date de test pentru un workflow UiPath are nevoie de informații legate de acesta (structură, diferite proprietăți), iar aceste informații nu pot fi obținute atât de ușor datorită formatului „.xaml” sub care este salvat workflow-ul.

Această lucrare de cercetare prezintă un tool de interpretare a workflow-urilor UiPath care expune structura acestora spre a fi folosită de alte tool-uri pentru generarea automată de date de test (tool-uri bazate pe execuție simbolică, pe testare pe bază de căutare, etc.). Tool-ul, denumit RPA2Graph, se bazează pe librării precum Windows Workflow Foundation pentru interpretarea workflow-urilor și extragerea informațiilor și este capabil să le serializeze în format „.json”. RPA2Graph este o aplicație software scrisă în C#, ușor de întreținut și de extins datorită design-ului arhitectural folosit. O capabilitate importantă a RPA2Graph îl reprezintă sistemul de adnotări, ce are ca scop transmiterea directă de informații de către un tester (sau dezvoltator RPA), către orice tool care se va folosi de output acestuia.

RPA (Robotic Process Automation) represents a software technology recently developed, which implies the automation of business processes by using software robots. The robots can interact with digital systems and software applications just as human do. The main goal of a software robot is to execute some of the repetitive tasks that people do, such that they would have time to focus on other relevant activities. One of the biggest actors on the market that RPA created is UiPath. The platform they provide is mainly based on 3 components: Studio, Robot and Orchestrator. By using the Studio, an RPA developer is able to design an automation process (workflow), that is executed by a Robot. The execution can be managed through Orchestrator.

RPA development, as any other type of software development, needs to provide a quality assurance of the product, therefore making software testing an important process for this business area. Speaking of UiPath, depending on the workflow that is to be tested, there are various manual testing methods and, recently, automated ones, by using the Test Suite platform. A relevant indicator for automated testing is represented by the coverage, and to provide a full „lines of code” coverage (activity coverage in our case), a full branch coverage is a complex task. The automated generation of test data for an UiPath workflow needs information about it (structure, properties) and this kind of information cannot be easily obtained due to the format of the files („.xaml”).

This research paper describes a tool that can parse UiPath workflows and exposes their structure to be used by automated test data generation tools (tools that are based on symbolic execution, on search-based testing, etc.). The tool, called RPA2Graph, is based on libraries like Windows Workflow Foundation to parse the workflows and extract the relevant information and is able to serialize them in a „.json” format. RPA2Graph is a software application written in C#, that is easily maintainable and extendable due to its architectural design. One relevant capability of this tool is represented by the annotations system, that aims to provide a channel which a tester (or RPA developer) can use to pass information directly to the consumer of the outputted “. json”.

CUPRINS

I.	Introducere	6
II.	Noțiuni tehnologice.....	9
1.	RPA (Robotic Process Automation)	9
2.	Platforma UiPath.....	11
3.	C#, .Net, NuGet, Git.	14
4.	Windows Workflow Foundation.....	17
III.	RPA2Graph.....	18
1.	Funcționalitate.....	18
1.	Rularea executabilului	18
2.	Interpretarea activităților	20
2.	Arhitectura aplicației.....	23
1.	Arhitectura în linii mari	23
2.	Design pattern-uri folosite	25
3.	Sistemul de adnotări	28
4.	Calități arhitecturale	30
IV.	Concluzii	32
1.	Rezultate obținute. Aprecieri asupra procesului de dezvoltare.....	32
2.	Direcții viitoare de dezvoltare a tool-ului	33
	Bibliografie	34

Listă figuri

Figură II.1 - Workflow modelat sub formă de mașină cu stări finite	12
Figură II.2 – Workflow modelat sub formă de secvență	13
Figură II.3 - Workflow modelat sub formă de flowchart.....	13
Figură II.4 - Exemplu de funcție lambda folosită în RPA2Graph	15
Figură II.5 - Ecranul de gestionare a dependențelor	15
Figură II.6 - Exemplu de pull-request.....	16
Figură III.1 - Instrucțiuni de folosire ale executabilului	18
Figură III.2 - Exemplu de fișier rezultat al RPA2Graph.....	19
Figură III.3 - Exemplu de flow-switch	21
Figură III.4 - Rezultatul interpretării unui flow-switch	21
Figură III.5 - Exemplu de mașină cu stări finite	22
Figură III.6 - Apel de metodă prin Reflection	23
Figură III.7 - Interfața IActivityParser.....	24
Figură III.8 - Interfața IWorkflowReducer	25
Figură III.9 - Interfața IWorkflowSerializer	25
Figură III.10 - Diagramă de clase pentru singleton	26
Figură III.11 - Diagramă de clase pentru Factory	27
Figură III.12 - Diagramă de clase pentru Chain-of-Responsibility	28
Figură III.13 - Exemplu de rezultat pentru adnotarea @Variables	29
Figură III.14 - Exemplu de rezultat pentru adnotarea @Expression	30

I. Introducere

Automatizarea proceselor cu ajutorul roboților software, sau pe scurt, RPA, reprezintă un segment de piață care a evoluat enorm în ultimii ani. Motivația din spatele succesului acestui domeniu este dată de promisiunea automatizărilor cu costuri reduse a proceselor companiilor, care ajută la salvarea de timp și bani [1]. La baza RPA-ului stă noțiunea de robot software. Un robot software reprezintă o entitate capabilă să interacționeze cu sisteme informatice asemenea oamenilor: ei pot înțelege ce se află pe ecran la un moment dat, pot naviga în sistem, pot extrage date și pot executa mult alte sarcini predefinite [2]. Avantajul roboților software este că ei pot executa aceste sarcini foarte rapid, și în marea majoritate a cazurilor nu mai este nevoie de interacțiune umană. Lider pe piața de RPA conform celor mai noi rapoarte [3], compania UiPath pune la dispoziția utilizatorilor săi o platformă performantă pe care orice persoană o poate folosi pentru a configura și executa roboți software, așa cum promite CEO-ul companiei, Daniel Dines [4]. Platforma UiPath presupune 3 componente principale: un IDE pentru descrierea sarcinilor pe care un robot software urmează să le execute, numit Studio, entitatea care execută task-urile, Robotul și o aplicație de gestionare a execuției roboților denumită Orchestrator. Astfel, un utilizator al platformei va folosi Studio-ul pentru dezvoltarea unui proces de automatizare și Orchestratorul pentru gestionare execuției lui.

Dezvoltarea de procese de automatizare pentru roboți se aseamănă cu dezvoltarea oricărui alt produs software, iar pentru asigurarea calității rezultatului, și în RPA este nevoie de testarea produsului final. Totuși, platformele care permit implementarea automatizărilor permit doar testarea manuală, puține dintre ele oferind soluții pentru a realiza acest lucru automat [1]. Recent, UiPath pune la dispoziția utilizatorilor o platformă complexă, numită Test Suite, ce facilitează procesul de testare al workflow-urilor. Ea permite crearea și gestionarea scenariilor de test în soluția Test Manager și conectarea acestor scenarii cu procese ce pot fi pornite din Orchestrator [5]. Un indicator relevant pentru testarea automată este reprezentat de acoperirea pe care o determină testele scrise (în această lucrare, termenul se referă exclusiv la acoperirea căilor din CFG-ul asociat workflow-ului). Astfel, o acoperire cât mai mare minimizează riscul de a scrie cod greșit. Problema cea mai comună ce apare în cazul testării căilor din CFG-uri o reprezintă

posibilitatea generării datelor de test în mod automat [6]. În cazul UiPath, sarcina devine una și mai dificilă, datorită formatului neobișnuit al workflow-urilor („xaml”).

Scopul lucrării este de a descrie un produs software capabil de interpretarea workflow-urilor UiPath, pentru a expune structura acestora spre alți consumatori, capabili de generare automată de date de test (aplicații bazate pe execuție simbolică, pe testare pe bază de căutare, învățare automată, etc.). Aplicația creată se numește RPA2Graph și se bazează pe librăria Windows Workflow Foundation pentru analizarea și extragerea de informații din fișierele de tip „xaml”. Aplicația poate expune informațiile în format „json”, format care este ușor de înțeles de aproape orice limbaj de programare (Python, Java, etc.). RPA2Graph este scrisă în C# iar procesul de dezvoltare a avut drept indicatori de calitate ușurința de întreținere și extindere a produsului.

Aplicația este menită să ruleze ca parte într-un proces automat de generare de date de test. Inputul procesului este reprezentat de un workflow UiPath. Acesta va fi interpretat cu ajutorul RPA2Graph care va genera un fișier „json” cu detaliile procesului de automatizare. Fișierul va fi preluat de o aplicație ce va genera un fișier „csv” cu datele de test, pe baza principiilor de execuție simbolică. Cu ajutorul arhitecturii producator-consumator pe care o implementează Orchestratorul, rezultatele pot fi încărcate într-o coadă, de unde vor fi preluate și executate de un robot software.

Tool-ul este inovativ în raport cu rezultatele deja existente în domeniu. O aplicație similară este descrisă în lucrarea „Towards Automated Testing of RPA Implementations”, care precizează că până la momentul publicării acesteia, nu exista vreo soluție care să transforme un workflow într-un format apropiat de UML sau BPMN [1]. Astfel, autorii au realizat o implementare recursivă în C# care se bazează pe interpretarea nodurilor XML. Aplicația scrisă de aceștia era destinată workflow-urilor scrise cu ajutorul platformei UiPath. Tehnica este ușor de aplicat în cazul workflow-urilor simple, care au o ierarhie ușor de înțeles, dar, în practică, există ierarhii destul de complexe de activități. De asemenea, firul execuției poate trece de la un workflow la altul prin activități specifice, apărând astfel nevoia unei soluții mai performante. RPA2Graph a plecat de la aceeași bază de cod ca soluția descrisă în articolul amintit mai sus, dar observând dificultatea procesului, am luat decizia de a încerca o altă abordare. Astfel, cu ajutorul librăriei Workflow Foundation, am reușit să încarc ierarhia de activități în memoria aplicației, acestea fiind, de fapt, clase de C#. Procesul de dezvoltare a devenit unul rapid și independent de modul de scriere a datelor în fișierul „xaml”, iar soluția poate fi folosită pentru interpretarea de ierarhii complexe:

mașini cu stări finite care conțin secvențe, workflow-uri care sunt invocate din alte workflow-uri, etc.

În următoarele capitole ale lucrării vor fi abordate:

1. Noțiuni tehnologice – capitolul va introduce mai larg noțiunea de RPA, vor fi prezentați principalii actori pe piață și evoluțiile lor tehnologice din ultima perioadă. Se va insista pe platforma UiPath cu cele 3 servicii de bază (Studio, Robot și Orchestrator) și vor fi prezentate tehnologiile folosite în realizarea aplicației RPA2Graph.
2. RPA2Graph – capitolul va prezenta evoluția design-ului arhitectural al aplicației, precizând problemele care au dus la schimbările acestuia de-a lungul procesului de dezvoltare. Vor fi evidențiate calitățile arhitecturale ale produsului, de ce sunt acestea necesare și cum sunt asigurate în cod și se va prezenta una dintre cele mai importante funcționalități ale acestuia, sistemul de adnotări.
3. Concluzii – capitolul va conține idei retrospective asupra rezultatelor obținute, asupra procesului de dezvoltare și va preciza direcțiile în care trebuie continuată dezvoltarea.

II. Noțiuni tehnologice

1. RPA (Robotic Process Automation)

Automatizarea a devenit o arie de interes din în ce mai populară, în ultimii ani, mai ales datorită contextului pandemic. Noțiunea de RPA acoperă soluțiile care interacționează cu interfețele aplicațiilor în același mod în care ar interacționa un om. RPA își propune să înlocuiască resursa umană prin automatizarea sarcinilor repetitive într-o manieră „outside-in”, manieră diferită de cea clasică: „inside-out” [7]. Cu alte cuvinte, sistemul informatic automatizat rămâne neschimbat. Avantajul principal stă în executarea acelorași sarcini, mai rapid și cu o rată de eroare mult mai scăzută, resursele umane având mai mult timp pentru îndeplinirea altor activități. Conform unui studiu intern realizat de compania UiPath în 2021, 67% din angajați sunt de părere că au de executat sarcini repetitive, pierzând aproximativ 4 ore și jumătate pe activități ce pot fi automatizate. Majoritatea persoanelor care au răspuns ar dori mai mult timp pentru implicarea în alte activități, în afara rutinei lor zilnice. Studiul arată că principalele sarcini ce ar trebui automatizate sunt: trimiterea de emailuri, introducerea de date și programarea de întâlniri [8].

Putem analiza în continuare un exemplu de muncă repetitivă, apropiat de spațiul academic. Să presupunem că un profesor de informatică le-a cerut studenților săi să redacteze o aplicație ce poate fi rulată în linia de comandă. Aplicația primește un input, îl procesează și afișează un rezultat. Nota temei este dată de procentul de teste la care aplicația răspunde cu rezultatul așteptat. Aplicația trebuie trimisă pe email, iar în interiorul email-ului să fie precizate numele și grupa studentului care a l-a trimis. Să presupunem cazul optimist în care email-urile sunt deja stocate într-un folder separat pe baza unor reguli. Procesul corectării temei unui singur student cuprinde următoarele sarcini: descărcarea aplicației atașată la email, rularea aplicației cu testele prestabilite, analizarea rezultatului și trimiterea răspunsului către student pe email. Dacă ar trebui să aproximăm durata acestui proces, acesta ar consuma între 5 și 10 minute din timpul profesorului. Putem considera, din nou, cazul optimist, în care verificarea completă a unei teme durează 5 minute. Să presupunem că sunt 30 de studenți care trimit acea temă, deci, timpul total, fără pauze, este de 150 de minute. În funcție de experiența dezvoltatorului RPA, implementarea scenariului descris mai sus poate varia ca durată. Pentru un începător, dezvoltarea ar putea dura aproximativ 120 de minute, iar

pentru o persoană familiarizată cu platforma în care lucrează, dezvoltarea ar putea dura chiar și 30 de minute. Timpul în care un robot poate analiza un singur email este foarte mic, dar vom folosi o valoare aproximativă de un minut. În cel mai rău caz, implementarea automatizării și rularea acesteia poate dura la fel de mult ca atunci când sarcina este executată de un om. Totuși, avantajul automatizării trebuie analizat pe termen lung: ce se întâmplă în cazul în care tema este destinată unei serii întregi de studenți (aprox. 180) și nu doar unei grupe de 30, cum evoluează statisticile dacă apare o a doua temă, în care aplicația și testele se schimbă, dar procesul rămâne același? În perspectivă, automatizarea acestui proces duce la salvarea unei cantități de timp considerabilă, iar dacă este executată într-un mod neasistat, profesorul poate folosi aceeași mașină (laptop, PC) pentru realizarea altor activități ce nu pot fi automatizate. Platformele de RPA sunt capabile de integrarea mai multor servicii, pentru care nu pot fi scrise automatizări în maniere clasice (automatizările clasice sunt scrise în interiorul aplicațiilor și nu pot interacționa în afara acestora).

O platformă de RPA matură include cel puțin trei componente: o componentă care descrie sau modelează sarcinile roboților, o componentă care rulează roboții software și o componentă care le orchestrează activitatea, adică îi stochează (în Cloud, de cele mai multe ori), le programează execuția și o monitorizează [1]. Mai nou, platformele de RPA implementează noțiunea de „hyper-automatizare”, care presupune integrarea de componente de inteligență artificială (NLP, învățare automată, CV), minare de procese, analiză și alte tool-uri avansate [9].

Piața de RPA este încă în perioada de formare, estimându-se că abia în 2024 aceasta va ajunge la o oarecare maturitate. În prezent, ea este încă fragmentată, întrucât companii din arii adiacente încep să ofere și ele servicii de RPA. Conform unui studiu realizat în iulie 2021, există 4 lideri pe această piață, ordonați după nota primită: Blue Prism, Microsoft, Automation Anywhere și UiPath. Produsul companiei Blue Prism se numește Intelligent Automation Platform și are aproximativ 168.000 de utilizatori activi. Printre cele mai noi capabilități ale acestei platforme se numără: procesarea inteligentă a documentelor, minarea de procese și acceleratori ERP. Printre categoriile unde soluția celor de la Blue Prism nu a excelat, se află capabilitatea de înțelegere a pieței, viteza de răspuns și prețurile ușor ridicate pentru serviciile oferite. Microsoft a intrat recent pe piața de RPA prin îmbunătățirea semnificativă a soluției deja existente Power Automate, care include și o variantă desktop: Power Automate Desktop. Avantajele Microsoft sunt existența unui ecosistem de clienți puternic dezvoltat și posibilitatea integrării facile cu orice alt serviciu Microsoft. Ca

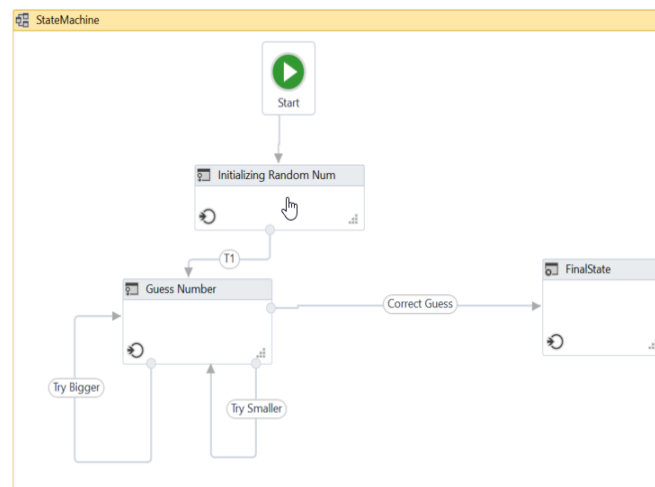
dezavantaje, soluția Microsoft este puternic dependentă de Windows și experiența utilizatorilor poate fi confuză în cazul navigării între Microsoft Teams, Power Automate Desktop și Power Automate. Produsul companiei Automation Anywhere se numește Automation 360, o platformă puternică, ce permite stocarea automatizărilor în Google Cloud sau AWS. Avantajele lor stau în platforma ce integrează componente importante de inteligență artificială (AARI, AiSense), orientarea puternică către serviciile cloud și strategia de prețuri avantajoasă. Dezavantajele includ dificultatea trecerii de la vechea lor platformă, la Automation 360, rezultatele deficitare în domeniul de procesare a documentelor și scăderea vânzărilor în Europa. UiPath este lider pe piața de RPA, punând la dispoziția utilizatorilor o platformă inteligentă și bine dezvoltată, care a reușit să creeze o comunitate de aproximativ un milion de utilizatori. Printre dezavantajele platformei se află lipsa unei soluții de automatizare web (totuși, există în plan ideea de construire a unei soluții bazată web) și strategia complexă de prețuri [3].

2. Platforma UiPath

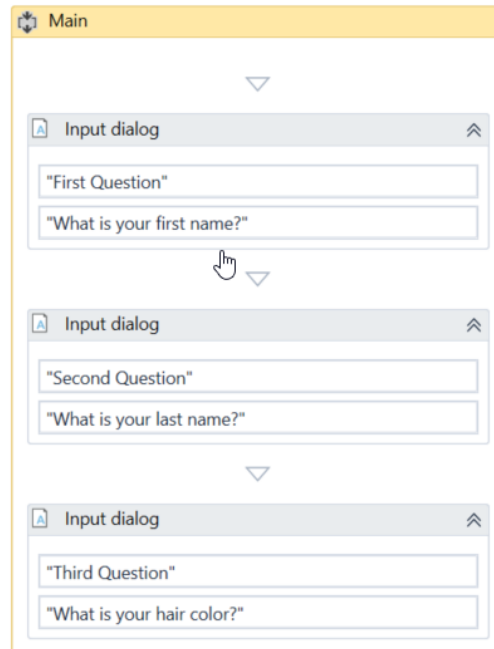
Platforma UiPath este una complexă și prin implementarea paradigmei „hyper-automation” include un set de servicii complementare ce duc la soluții RPA calitative. Prin procedee de minare de procese și de task-uri, platforma este capabilă să identifice sarcinile repetitive pe care o persoană le execută. Pe baza acestor date, un dezvoltator RPA va folosi Studioul pentru implementarea unei soluții care să înlocuiască efortul uman. Soluția poate folosi librării puse la dispoziție de UiPath, dar și de alți utilizatori care le-au publicat pe „Marketplace”. Soluția este apoi urcată în Cloud, adică în Orchestrator. Aceasta poate fi testată cu ajutorul serviciului Test Suite și poate fi executată de roboți. După executarea task-urilor, vor fi publicate rezultate și statistici care pot fi ulterior analizate pentru îmbunătățirea automatizărilor. La baza platformei stau 3 componente principale: Studioul, Robotul și Orchestratorul.

UiPath Studio este componenta prin care un dezvoltator RPA poate descrie procese de automatizare. Acesta este un IDE care se bazează pe Microsoft Workflow Designer și alte componente din Microsoft Visual Studio pentru a modela grafic workflow-urile sub formă de secvențe, flowchart-uri sau mașini cu stări finite [10]. Cea mai granulară entitate când vine vorba

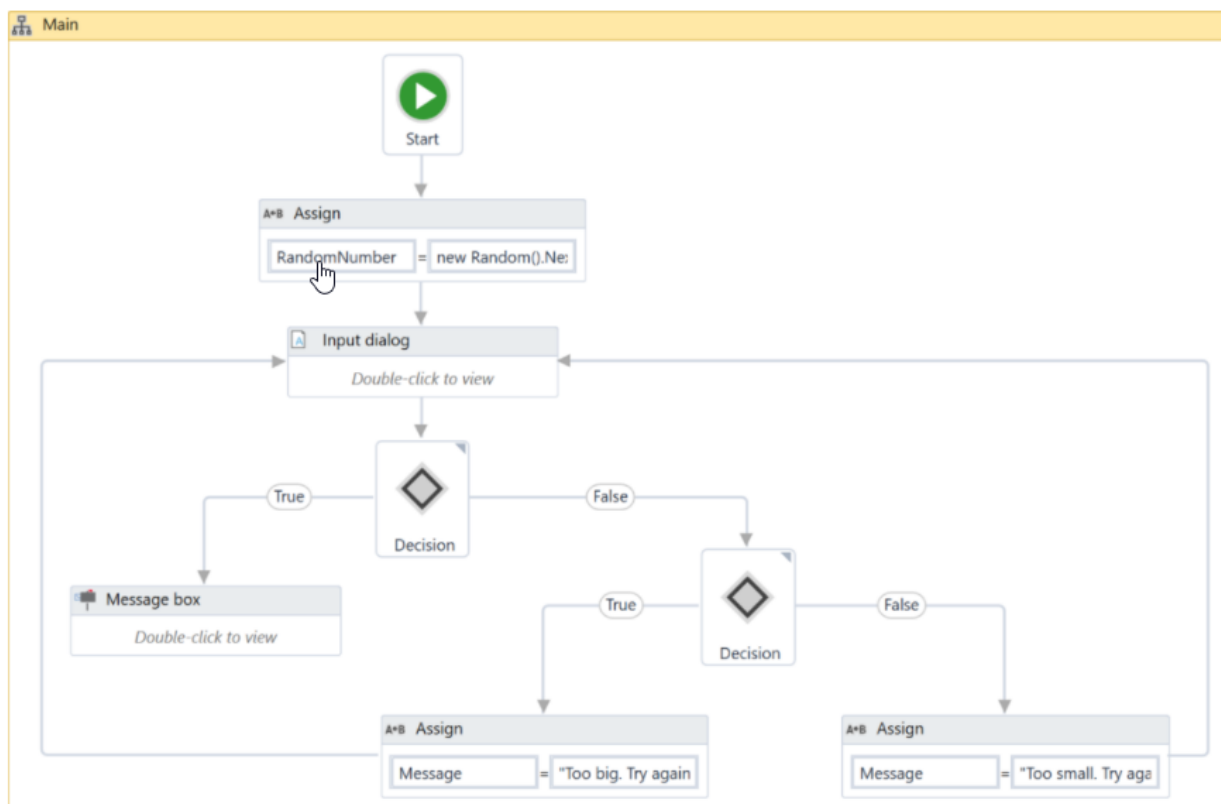
de un workflow este o activitate. Activitățile fac parte din librării și în funcție de scopul acestora, descriu anumite capabilități. Spre exemplu, în librăria UiPath.UiAutomation.Activities există activități pentru interacțiunea UI: activități de Click, de scriere de date, de extragere de date, etc. Activitățile pot fi simple, ca cele din librăria amintită anterior sau complexe, precum activități care se conectează la servicii de email, la baze de date. O secvență conține activități, modelând o listă simplu înlănțuită. Ordinea de execuție a activităților este de la prima până la ultima. Un flowchart conține noduri (simple sau condiționale) și legături orientate între acestea. În cazul acestui mod de structurare a unui workflow, execuția pleacă de la nodul inițial și continuă în ordinea indicată de conexiuni. Nodurile simple pot conține secvențe de activități. Modelarea unui workflow sub formă de mașină cu stări finite presupune o stare inițială și niciuna sau mai multe stări finale. Execuția pleacă de la starea inițială și continuă în funcție de cum sunt evaluate legăturile între stări. Atât stările cât și conexiunile pot conține secvențe de activități. Există și o variantă de Studio destinată utilizatorilor neexperimentați, sau „business useri” cum sunt ei denumiți pe pagina oficială UiPath [11].



Figură II.1 - Workflow modelat sub formă de mașină cu stări finite



Figură II.2 – Workflow modelat sub formă de secvență



Figură II.3 - Workflow modelat sub formă de flowchart

Roboții UiPath sunt entități software capabile să execute automatizări descrise prin workflow-uri. Există trei tipuri de roboți puși la dispoziție de platforma UiPath: roboți neasistați, care lucrează independent în background și se ocupă cu executarea unor sarcini de lungă durată, roboți asistați care colaborează cu resursa umană pe mașinile acestora, acționând ca un asistent personal și roboți hibridi, adică o combinație între cele două categorii de mai sus.

Orchestratorul reprezintă componenta prin care sunt gestionate soluțiile de RPA. Prin acesta, un dezvoltator poate alocă roboți, îi poate urca pe anumite mașini, îi poate porni, monitoriza și urmări. Acest lucru poate fi realizat fie din versiunea web a aplicației, fie din versiunea ei mobilă. Tot Orchestratorul este folosit în cadrul serviciului de Test Suite pentru a executa teste și a analiza rezultatele acestora. Există mai multe metode prin care cineva poate porni execuția unui robot. O modalitate este de a programa execuția la un anumit moment în viitor. Un alt mod de declanșare a execuției unui robot este bazat pe relația producător-consumator. Astfel, există noțiunea de coadă unde se pot încărca date, atât manual prin interfața Orchestratorului, cât și automat prin activități specifice. Atunci când se află în coadă un element neprocesat, există posibilitatea executării unor roboți consumatori care să se folosească de datele din coadă.

3. C#, .Net, NuGet, Git.

Aplicația RPA2Graph este scrisă integral în limbajul C#. Acesta este un limbaj de programare ușor de înțeles, dezvoltat de Microsoft care implementează paradigma programării orientate pe obiecte. Limbajul își are rădăcinile în limbajul C, dar se aseamănă mult cu Java datorită existenței unor funcționalități specifice: interfețe, „garbage collector” (GC), tipuri ce pot fi nule, etc. O astfel de funcționalitate notabilă și folosită în proiect este reprezentată de funcțiile lambda. Acestea pot fi folosite pentru definirea anonimă a unei secvențe de cod ce poate fi executată mai târziu. În exemplul de mai jos, funcția lambda este stocată cu ajutorul clasei „Action”, echivalentă funcțiilor care nu întorc niciun tip de date (funcții void). Pentru stocarea unei funcții lambda care întoarce un rezultat, se poate folosi clasa „Func”.

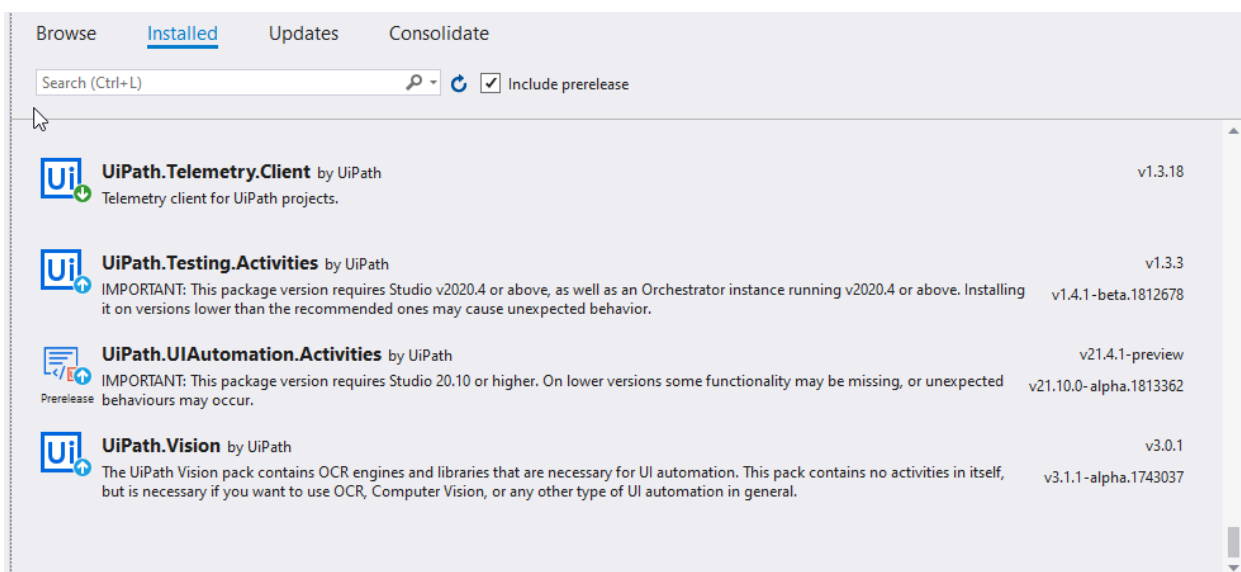
```

/// Connect the states
Action<Node, Node> createSimpleTransition = (source, destination) =>
{
    Common.Transition t = new Common.Transition();
    t.Source = source;
    t.Destination = destination;
    t.Expression = "";
    t.ExpressionValue = Common.Transition.TRUE_TRANSITION_VALUE;
    graph.Transitions.Add(t);
};

```

Figură II.4 - Exemplu de funcție lambda folosită în RPA2Graph


Artefactul rezultat prin asamblarea proiectului este un executabil ce poate fi rulat și parametrizat din linia de comandă. Aplicația se bazează pe platforma .Net care asigură integrarea ușoară a librăriilor dependente. RPA2Graph trebuie să fie conștientă de activitățile existente în fișierul „.xaml” ce urmează să fie interpretat. Librăriile din care activitățile fac parte sunt specificate în primul rând din fișierele “.xaml”, iar, la momentul încărcării ierarhiei de activități în memorie, soluția trebuie să poată asocia nodurile dintr-un fișier cu clase cunoscute. Astfel, librăriile din care fac parte activitățile trebuie să fie adăugate ca dependențe ale aplicației RPA2Graph. Aceste librării sunt stocate pe platforma NuGet: <https://www.nuget.org/>. IDE-ul Visual Studio integrează un client al platformei NuGet, care este capabil de gestionarea dependențelor proiectului.




Figură II.5 - Ecranul de gestionare a dependențelor

Proiectul este versionat cu ajutorul aplicației „open-source”, Git, el fiind stocat pe platforma Github. Git introduce o serie de noțiuni și un mod de lucru diferit când vine vorba de dezvoltarea unui produs software. Proiectul este stocat într-un „repository”, adică un container ce poate fi stocat în Cloud, pe diferite platforme. Atunci când un dezvoltator lucrează într-un repository, atunci fie creează unul nou, fie clonează unul deja existent. Un repository conține mai multe „branch-uri” adică mai multe fire de dezvoltare a proiectului, ce pot avea un istoric comun. Git nu stochează integral versiunile diferite de proiecte, ci diferențele de la o versiune la alta. Cea mai granulară formă de stocare a diferențelor o reprezintă un „commit”. Deoarece proiectul RPA2Graph face parte dintr-un repository la care lucrează o echipă cu mai multe persoane, atunci, pentru claritate, dezvoltarea a fost realizată folosindu-mă de sistemul de „branching” amintit mai sus. Există un branch principal, numit „main”, iar fiecare dezvoltator ce aduce modificări trebuie să o facă printr-un branch separat. Astfel, atunci când am adăugat funcționalități noi, am creat branch-uri cu prefixul „feature/”, urmat de descrierea funcționalității, iar când am fixat probleme în cod, am folosit prefixul „fix/”. După implementarea funcționalității sau fixarea problemei, am deschis un „pull-request”, adică am cerut părerea altor persoane din echipă asupra codului scris. După rezolvarea eventualelor probleme sau sugestii ce pot apărea, „pull-request”-ul a fost închis, și codul a fost adăugat în branch-ul principal.

Add support for repetitive activities #77

 Merged vlad-coteanu-uni... merged 2 commits into master from feature/repetitive-structures 23 days ago

Conversation 0 Commits 2 Checks 0 Files changed 15

 vlad-coteanu-unibuc commented 24 days ago

Implement support for:

- While
- Do While
- ForEach
- ForEachRow

Added wf's to test plus json parsed examples.

Figură II.6 - Exemplu de pull-request

4. Windows Workflow Foundation

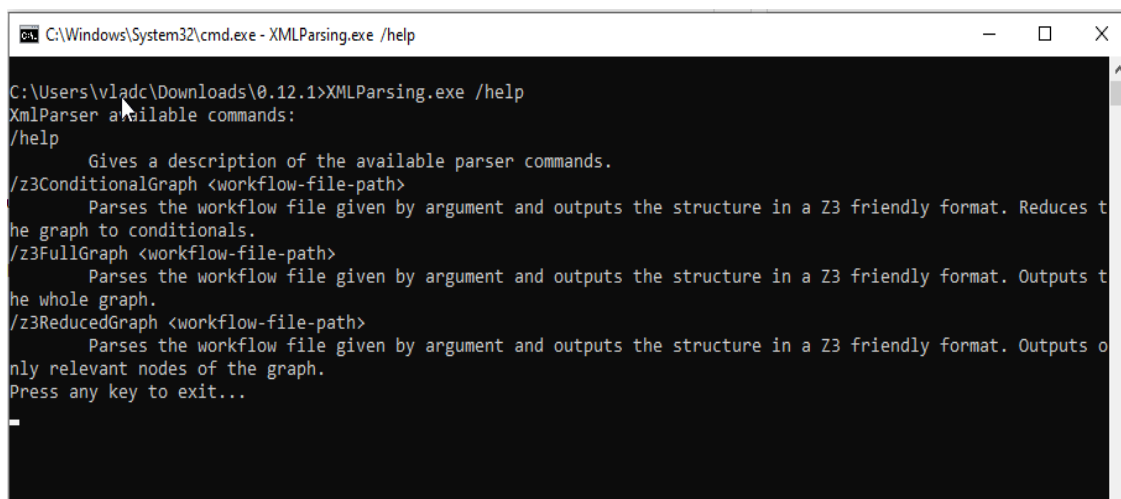
Motivația alegerii grupului de tehnologii prezentate anterior este existența în această sferă a librăriei Windows Workflow Foundation. Aceasta permite dezvoltatorilor să definească workflow-uri, în mod programatic, asemenea cum o fac dezvoltatorii de RPA prin platforma UiPath Studio. Astfel, pe baza clasei „Activity Builder” și a numelui fișierului workflow ce urmează a fi interpretat, RPA2Graph încarcă implementarea în memorie. La baza claselor interpretate stă clasa „Activity”, definită în namespace-ul ”System.Activities”. Activitățile întâlnite în procesul de dezvoltare fac parte din implementarea „NativeActivity” a clasei abstracte „Activity”. O activitate nativă poate conține diferite proprietăți și poate fi executată într-un anumit context. Conform Workflow Foundation, activitățile pot fi organizate în secvențe, flowchart-uri sau mașini cu stări finite, forme ce pot fi interpretate cu succes de RPA2Graph. Un astfel de workflow poate fi rulat programatic folosind clasa „Workflow Invoker” sau clasa „Workflow Application”. Cea din urmă permite definirea unui context printr-un obiect de tip dicționar care este specificat la momentul construirii obiectului.

III. RPA2Graph

1. Funcționalitate

1. Rularea executabilului

Aplicația RPA2Graph este o aplicație ce poate fi executată cu anumiți parametri în linia de comandă. Funcționalitatea ei principală este de a interpreta un workflow stocat într-un fișier „.xaml” și a-l expune într-o formă ușor de înțeles de consumatori scriși în diferite limbaje de programare. Soluția suportă la momentul scrierii 4 comenzi ce pot sau nu să aibă nevoie de un parametru adițional. Prima comandă, „/help” este destinată obținerii de informații despre capabilitățile programului. A doua comandă, „/z3ConditionalGraph <workflow-file-path>” presupune interpretarea fișierului transmis prin parametru și expunerea doar a nodurilor condiționale din structura obținută. Această comandă a fost folosită doar în etapele inițiale ale proiectului. A treia comandă, „/z3FullGraph <workflow-file-path>”, presupune interpretarea fișierului transmis prin parametru și expunerea întregii structuri de noduri. Ultima comandă, „/z3ReducedGraph <workflow-file-path>”, include, pe lângă interpretarea workflow-ului, și reducerea acestuia pentru eliminarea nodurilor ne semnificative, care doar cresc dimensiunea fișierului rezultat și nu aduc nicio informație în plus.



```
C:\Windows\System32\cmd.exe - XMLParsing.exe /help

C:\Users\vladc\Downloads\0.12.1>XMLParsing.exe /help
XmlParser available commands:
/help
    Gives a description of the available parser commands.
/z3ConditionalGraph <workflow-file-path>
    Parses the workflow file given by argument and outputs the structure in a Z3 friendly format. Reduces the graph to conditionals.
/z3FullGraph <workflow-file-path>
    Parses the workflow file given by argument and outputs the structure in a Z3 friendly format. Outputs the whole graph.
/z3ReducedGraph <workflow-file-path>
    Parses the workflow file given by argument and outputs the structure in a Z3 friendly format. Outputs only relevant nodes of the graph.
Press any key to exit...
```

Figură III.1 - Instrucțiuni de folosire ale executabilului

Rezultatul rulării programului RPA2Graph pentru un fișier „.xaml” este un fișier „.json” care este stocat în același director cu documentul primit ca input. Acesta conține, la momentul scrierii lucrării și executând ultima versiune disponibilă, următoarele informații: nodul de început al workflow-ului principal, variabilele, argumentele de intrare și de ieșire, denumirea și nodul de început al tuturor workflow-urilor ce ajung să fie invocate în execuția automatizării principale și structura de noduri (în imagine, sunt eliminate majoritatea nodurilor pentru a putea observa mai multe detalii). Dacă în workflow, nodurile sunt activități, iar firul execuției trece de la una la alta, aici există un număr de noduri cel puțin la fel de mare cu numărul de activități, iar direcția de parcurgere este dată de tranziții. Unele noduri condiționale pot avea expresii, și tranzițiile se execută în funcție de valoarea rezultatului acelei expresii.

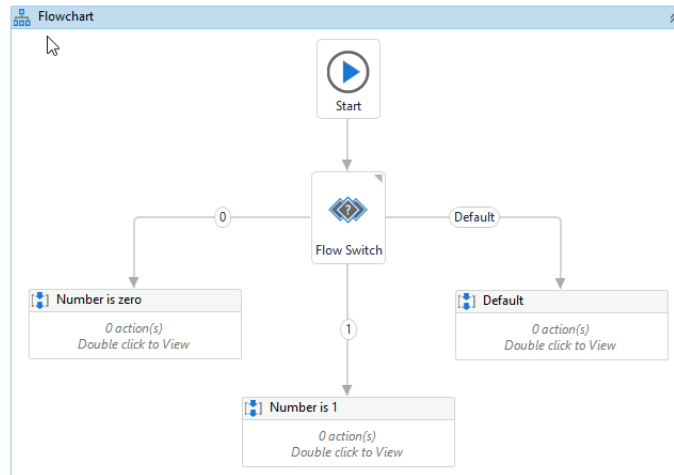
```
{
  "workflows": [
    {
      "variables": {
        "InitialGuess": {
          "Type": "Int32"
        },
        "RandomNumber": {
          "Type": "Int32"
        }
      },
      "inputArguments": [],
      "displayName": "FirstStateMachine_1",
      "fullPath": "C:\\UiPath\\rpa-testing\\C#Models\\CSharpLoans_Consumer\\FirstStateMachine.xaml",
      "invokedBy": "",
      "startNode": "FirstStateMachine_1:Initializing_Random_Number_4"
    }
  ],
  "graph": {
    "FirstStateMachine_1:Guess_Number_1": {
      "transitions": [
        {
          "value": "True",
          "destination": "FirstStateMachine_1:InputDialog_3"
        }
      ]
    },
    "FirstStateMachine_1:Tl_15": {
      "transitions": [
        {
          "value": "True",
          "destination": "FirstStateMachine_1:Guess_Number_1"
        }
      ]
    },
    "FirstStateMachine_1:Initializing_Random_Number_Virtual_End_16": {
      "transitions": []
    }
  },
  "startNode": "FirstStateMachine_1:Initializing_Random_Number_4"
}
```

Figură III.2 - Exemplu de fișier rezultat al RPA2Graph

2. Interpretarea activităților

În momentul actual aplicația suportă interpretarea unui număr destul de mare de activități, într-un mod generic, prin extragerea numelui și tranzițiilor lor, și un număr restrâns de activități prin cod scris special pentru acestea. Plecând de la ideea că o activitate poate cuprinde în interiorul ei una sau mai multe activități, am stabilit o regulă pentru asigurarea integrității rezultatului: interpretarea fiecărei activități va duce la un nod de început și un nod de final din structura rezultată. De exemplu, pentru interpretarea unei activități de tip „If”, vom avea un nod de start ce conține expresia condiționalului, două noduri copii corespondente acțiunilor ce se vor executa pe cazurile posibile și un nod virtual, ce va uni cele două fire de execuție diferite. Secțiunea curentă va prezenta câteva astfel de activități și soluția programatică găsită pentru interpretarea lor.

Un prim exemplu îl reprezintă interpretarea flowchart-urilor. Un flowchart conține 3 tipuri de noduri: noduri simple, ce pot conține secvențe de activități, noduri condiționale și noduri de tip switch. Nodurile simple sunt interpretate ușor, iar nodurile condiționale într-o manieră asemănătoare cu cea prezentată pentru activitatea de tip „If”. Problema cea mai mare a fost întâmpinată în cazul nodurilor „switch”. Formatul final presupune ca nodurile să aibă sau nu o expresie, și să aibă o tranziție când expresia este adevărată și una când expresia este falsă. În cazul unui switch cu mai multe ramuri posibile, este nevoie de o procesare specială pentru a ajunge la rezultatul propus. Astfel, să presupunem că switch-ul nostru are trei cazuri, cazul A, cazul B și cazul „default”, iar variabila în discuție este „elementValue”. În structura finală, vor apărea următoarele noduri și tranziții: un nod corespunzător switch-ului, fără nicio expresie, cu o tranziție către un nod virtual ce reprezintă cazul A. Acest „nodA”, conține expresia „elementValue == A” și are o tranziție pentru cazul „true” către nodul destinație al acestui caz din „flow-switch” și o tranziție pentru cazul false către nodul virtual „nodB”. „nodB” conține, la fel, o expresie „elementValue == B”, o tranziție pentru cazul „true” către nodul destinație al acestui caz și o tranziție pentru cazul „false” către nodul destinație al cazului „default”. În cazul în care clauza „default” lipsea, atunci ar fi avut loc o tranziție către un nod virtual ce semnifică finalul execuției flowchart-ului.



Figură III.3 - Exemplu de flow-switch

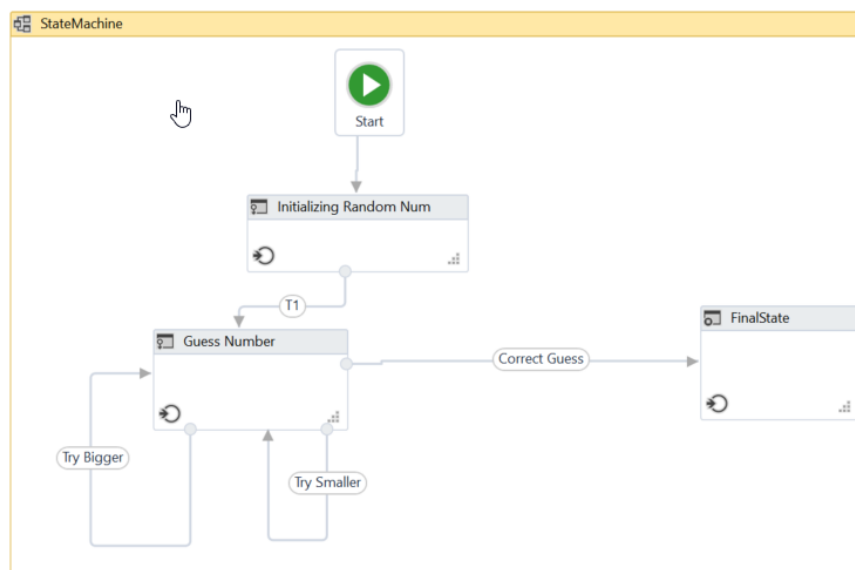
```

{
  "workflows": [
    {
      "variables": {
        "nr": {
          "Type": "Int32",
          "Default": "0"
        }
      },
      "inputArguments": [],
      "displayName": "Test_1",
      "fullPath": "C:\\UiPath\\rpa-testing\\C#Models\\CSharpExampleWithManyDataTypes\\Test.xml",
      "invokedBy": "",
      "startNode": "Test_1:Flow_Switch_1"
    }
  ],
  "graph": {
    "Test_1:Flow_Switch_1": {
      "transitions": [
        {
          "value": "True",
          "destination": "Test_1:Flow_Switch_Case_0_6"
        }
      ]
    },
    "Test_1:Flow_Switch_Case_0_6": {
      "expression": "nr == 0",
      "transitions": [
        {
          "value": "True",
          "destination": "Test_1:Number_is_zero_2"
        },
        {
          "value": "False",
          "destination": "Test_1:Flow_Switch_Case_1_7"
        }
      ]
    },
    "Test_1:Flow_Switch_Case_1_7": {
      "expression": "nr == 1",
      "transitions": [
        {
          "value": "True",
          "destination": "Test_1:Number_is_1_4"
        },
        {
          "value": "False",
          "destination": "Test_1:Default_3"
        }
      ]
    }
  ]
},

```

Figură III.4 - Rezultatul interpretării unui flow-switch

O altă structură de activități mai complexă ce a avut nevoie de cod special pentru interpretare o reprezintă mașinile cu stări finite. Acestea conțin stări, iar stările pot avea activități interne și tranziții, care sunt evaluate într-o ordine prestabilită. O tranziție presupune o condiție, un cod intern de executat, și o destinație. Spre exemplu, putem analiza un caz de mașină cu stări finite prezentată în documentația oficială de la UiPath [12]. Workflow-ul descrie jocul „Guess the number” în care utilizatorul trebuie să ghicească un număr ales aleator, prin mai multe încercări. Problema aici este transformarea celor 3 tranziții posibile din starea „Guess Number” în stările destinație. Aceasta transformare se bazează pe algoritmul prezentat la exemplul anterior cu „flow-switch”. Astfel, se va crea un nod pentru starea „Guess Number” care nu conține nicio expresie, iar apoi, se creează câte un nod pentru fiecare tranziție. Pe condiția „true” a fiecărei tranziții, se continuă cu activitățile prezente pe tranziție și apoi cu nodul destinație, iar pe condiția „false”, se analizează următoarea tranziție.



Figură III.5 - Exemplu de mașină cu stări finite

2. Arhitectura aplicației

1. Arhitectura în linii mari

Proiectul RPA2Graph este structurat în 3 categorii de clase: modele, servicii și utilitare. Prima categorie reprezintă noțiunile cu care lucrează aplicația, cuprinzând clase pentru stocarea informațiilor despre workflow-uri, pentru modelarea grafului rezultat, a nodurilor și a tranzițiilor acestora. În funcție de complexitatea activității modelate, nodul poate fi extins pentru a include și alte date. Spre exemplu, există noduri speciale pentru: activitățile de tip „Invoke Workflow” care stochează numele fișierul workflow-ului invocat, pentru activitățile de tip „ReadCSV” care stochează numele fișierului ce urmează a fi citit sau pentru activități de tip „Assign” unde se stochează operandii implicați în activitate. Utilitarele conțin funcționalități pentru interpretarea expresiilor, pentru extragerea unor informații de pe activități, pentru modelarea unor excepții și pentru folosirea librăriei „Reflection” de la Microsoft. Tehnica implementată de această librărie ajută la extragerea informațiilor de pe obiecte, atunci când programul nu poate stoca obiectul într-un tip de date cunoscut. Spre exemplu, avem de extras numele de pe un obiect reținut printr-o interfață, știm cum se numește câmpul în implementarea interfeței, dar nu avem acces la aceasta. Atunci, vom obține numele prin „Reflection”, într-o manieră similară ca în următoarele linii de cod:

```
try
{
    MethodInfo methodInfo = source.GetType().GetMethod(name);
    return methodInfo.Invoke(source, parameters);
}
catch (Exception)
{
    // Console.WriteLine(e.Message);
}
```

Figură III.6 - Apel de metodă prin Reflection

Zona serviciilor este una complexă și conține o clasă ajutătoare pentru modelarea inputului transmis în linia de comandă și o serie de interfețe și implementări care, prin asociere, formează

opțiunile posibile de executare ale aplicației. Astfel, o opțiune este formată dintr-un „Parser”, maxim un „Reducer” și un „Serializator”.

„Parserele” sau interpretoarele implementează interfața „IActivityParser”, ale cărei implementări sunt create printr-o clasă „Factory”. La momentul actual, există mai mult de 10 interpretoare specifice pe activități și un interpretor generic, folosit în cazul în care nu există unul specific. Ele sunt construite în jurul ideii amintite anterior de folosire a unui nod de intrare și a unui nod de ieșire din acea activitate. Folosind această tehnică, este foarte ușor de integrat rezultatul interpretării unei activități în graful rezultat. Un interpretor primește activitatea din care trebuie să extragă informații, graful rezultat unde vor fi înregistrate noduri și tranziții și informațiile workflow-ului curent. Nodurile rezultate din activități vor avea un identificator unic și un nume prefixat cu numele workflow-ului din care ele fac parte.

```
namespace XMLParsing.Services
{
    13 references | Vlad Coteanu, 53 days ago | 1 author, 3 changes
    interface IActivityParser
    {
        /**
         * Method should return a tuple formed by starting and ending node of the activity
         */
        11 references | Vlad Coteanu, 53 days ago | 1 author, 2 changes
        Tuple<Node, Node> ParseActivity(Activity activity, Graph graph, WorkflowData workflowData);
    }
}
```

Figură III.7 - Interfața IActivityParser

Un „reducer” este o clasă menită să elimine noduri din graful rezultat, după anumite reguli, cu scopul reducerii cantității de date exportate. În proiect, aceste clase implementează interfața „IWorkflowReducer” și primesc ca input graful ce trebuie modificat.


```

namespace XMLParsing.Services
{
    1 reference | Vlad Coteanu, 67 days ago | 1 author, 2 changes
    interface IWorkflowReducer
    {
        2 references | Vlad Coteanu, 67 days ago | 1 author, 1 change
        void ReduceWorkflow(Graph graph);
    }
}

```

Figură III.8 - Interfața IWorkflowReducer

„Serializatoarele” sunt clase ce au ca scop redactarea datelor într-un obiect de tip „TextWriter”, folosind diferite tehnici. În proiect, ele implementează interfața „IWorkflowSerializer” și primesc ca input graful ce trebuie scris și obiectul în care urmează să fie scris acesta. „Serializatorul” corespunzător opțiunii „z3FullGraph” va scrie informații despre toate workflow-urile ce ajung să fie invocate, despre structura grafului rezultat și despre variabilele și argumentele implicate.

```

namespace XMLParsing.Services.Serializers
{
    1 reference | Vlad Coteanu, 67 days ago | 1 author, 2 changes
    interface IWorkflowSerializer
    {
        3 references | Vlad Coteanu, 67 days ago | 1 author, 1 change
        void SerializeWorkflow(Graph graph, TextWriter textWriter);
    }
}

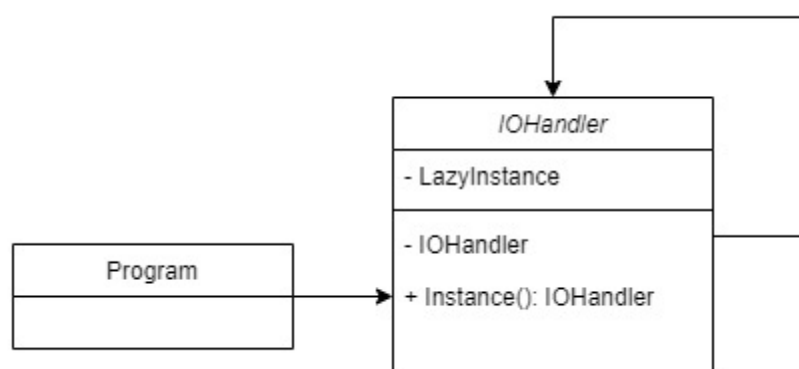
```

Figură III.9 - Interfața IWorkflowSerializer

2. Design pattern-uri folosite

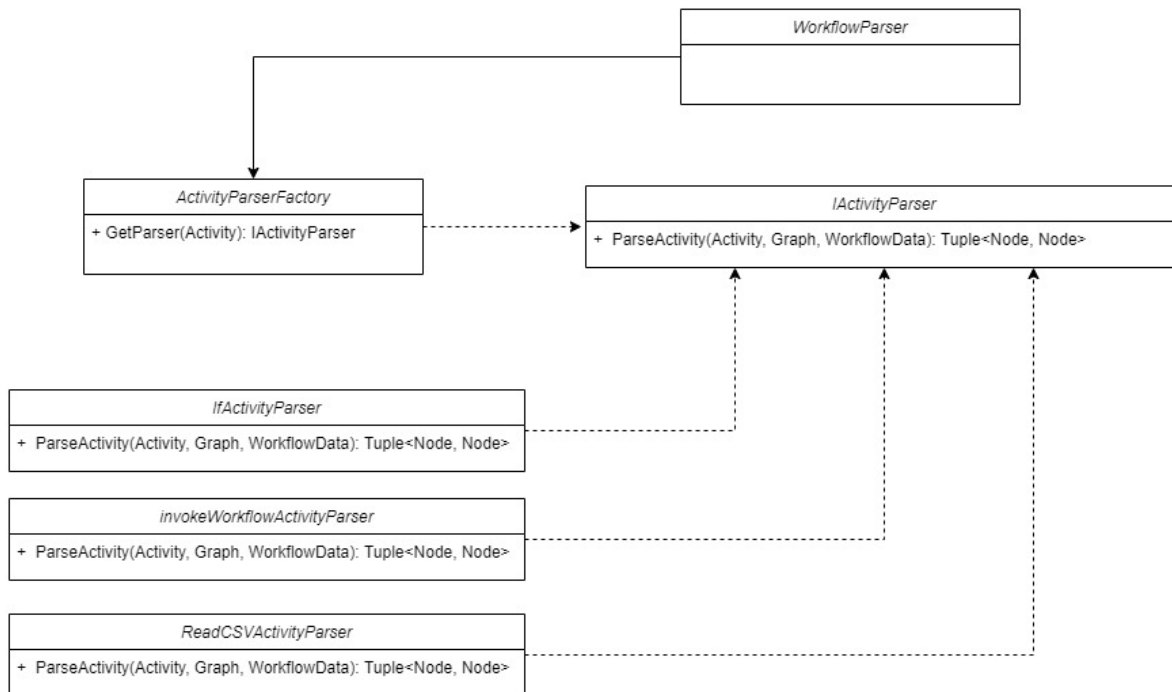
Proiectul RPA2Graph implementează câteva design pattern-uri ce facilitează procesul de dezvoltare și ajută la creșterea calității produsului final: Singleton, Factory și Chain-of-responsibility. Aceste design pattern-uri ajută la îndeplinirea principiilor SOLID, care ajută la scrierea unui cod ușor de înțeles, flexibil și ușor de întreținut.

Primul și cel mai comun design pattern este singleton-ul, care, prin gestionarea instanțelor unui anumit obiect se încadrează în categorie design pattern-urilor creaționale [13]. Acest principiu ne asigură că la un moment dat există cel mult o singură instanță a unei clase și aduce siguranță și eficiență când vine vorba de resursele implicate în proiect. Siguranța este asigurată prin menținerea unui singur punct de accesare a resursei printr-o metodă publică. În RPA2Graph, clasa ce permite interacțiunea în linia de comandă, numită „IOHandler” este un singleton și este folosită din clasa principală a aplicației („Program.cs”). Instanța este învelită în utilitara „Lazy” care ne asigură alocarea optimă a resursei, astfel, el nu va fi creat decât în momentul primei utilizări.



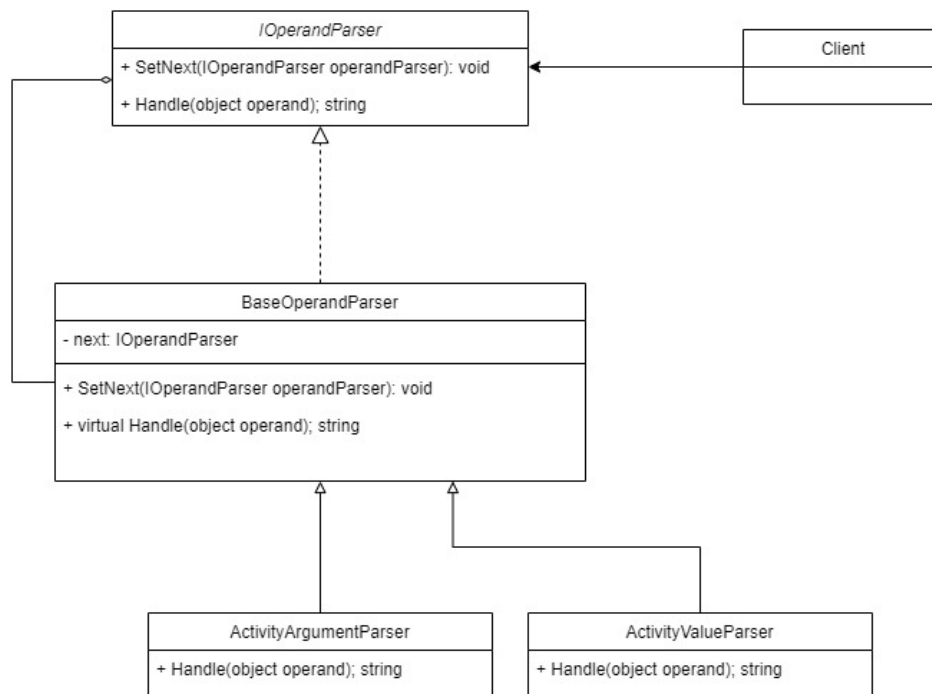
Figură III.10 - Diagramă de clase pentru singleton

Un alt design pattern folosit în proiect este „Factory”. Pentru integrarea ușoară a interpretoarelor de activități, acestea sunt nevoite să implementeze o interfață precizată anterior: „IActivityParser”. Atunci când un client dorește să interpreteze o activitate, el se folosește de clasa „ActivityParserFactory” care întoarce obiectul corespunzător interpretării aceluia tip de activitate. Astfel, la apelul metodei „GetFactory”, pe baza tipului clasei activității primite, clasa „Factory” va întoarce implementarea corectă a interfeței „IActivityParser”.



Figură III.11 - Diagramă de clase pentru Factory

Principiul „Chain-of-Responsibility” este în general folosit la gestionarea request-urilor care ajung la un API al unui server dintr-o aplicație web, sau pentru gestionarea evenimentelor care ajung într-un sistem de procesare a evenimentelor [13]. În cazul RPA2Graph, acest design pattern este folosit pentru interpretarea expresiilor ce se pot afla fie pe activități condiționale, fie pe orice alt tip de activități ce suportă expresii ca input. La bază se află interfața „IOperandHandler” care cuprinde două metode: „SetNext” responsabilă cu conectarea clasei curente de procesare a expresiilor de următoarea și „Handle” responsabilă cu procesarea expresiei. Clasa de bază care implementează această interfață este „BaseOperandParser” care cuprinde un cod general. El implementează metoda „SetNext” prin stocarea următoarei clase de procesare și metoda virtuală „Handle” care asigură trecerea la următorul pas, dacă clasa curentă nu a reușit procesarea. Există mai multe clase care extind această clasă de bază, specifice pentru diferite tipuri de operanzi. Procesarea se termină fie când o implementare a interpretorului de expresii reușește să extragă rezultatul cu succes, fie când au fost executate toate nodurile din lanț.



Figură III.12 - Diagramă de clase pentru Chain-of-Responsibility

3. Sistemul de adnotări

Una dintre cerințele apărute pe parcursul dezvoltării produsului a fost de a asigura un mod de comunicare direct între dezvoltatorul de RPA sau testerul care se ocupa de fișierul „.xaml” și consumatorul rezultatului aplicației RPA2Graph. Această cerință s-a născut din incompatibilitatea între librăriile și limbajul C# folosit în workflow și limbajul Python în care era scris un consumator care s-a folosit de rezultatul acestei soluții. Printre exemple de situații incompatibile se numără: tipul de date „Int32” care este reprezentat ca „int” în Python, tipurile „Dictionary”, etc. Soluția tehnică găsită a fost implementarea unui sistem de adnotări, bazat pe comentariile pe care utilizatorii UiPath Studio le pot adăuga pentru fiecare activitate. Există două tipuri de adnotări pe care RPA2Graph știe să le interpreteze: @Variables și @Expression.

@Variables este o adnotare menită să specifice informații adiționale despre variabilele și argumentele dintr-un workflow. Pentru a folosi această adnotare, un dezvoltator trebuie să specifice un obiect JSON valid, având drept chei nume de variabile și drept valori, proprietăți


adiționale. Cheile vor fi folosite pentru asocierea cu variabilele și argumentele workflow-ului, iar proprietățile vor fi copiate așa cum apar ele în fișierul „.json” rezultat. Cel mai bun loc pentru plasarea unei astfel de adnotări este activitatea „top-level”. Un exemplu de astfel de adnotare și cum este ea tradusă în rezultat este:

Input:

@Variables

```
{ "local_number_retries": { "min": 0, "max": 3 }, "local_test_data_expected": { "path": "pin_mocked_data.csv" }, "local_test_data_actual": { "path": "pin_real_data.csv" }, "actual_pin_values": { "bounds": 10, "min": 0, "max": 9999, "userInput": "True" } }
```

Output:



```
{ "variables": { "out_pin_check_successful": { "Type": "Boolean" }, "local_number_retries": { "Type": "Int32", "Default": "0", "Annotation": { "min": 0, "max": 3 } }, "local_pin_test": { "Type": "Boolean" }, "expected_pin": { "Type": "Int32" }, "actual_pin_values": { "Type": "Int32[]", "Annotation": { "bounds": 10, "min": 0, "max": 9999, "userInput": "True" } }, "local_test_data_expected": { "Type": "DataTable", "Annotation": { "path": "pin_mocked_data.csv" } }, "local_test_data_actual": { "Type": "DataTable", "Annotation": { "path": "pin_real_data.csv" } } }, }
```

Figură III.13 - Exemplu de rezultat pentru adnotarea @Variables

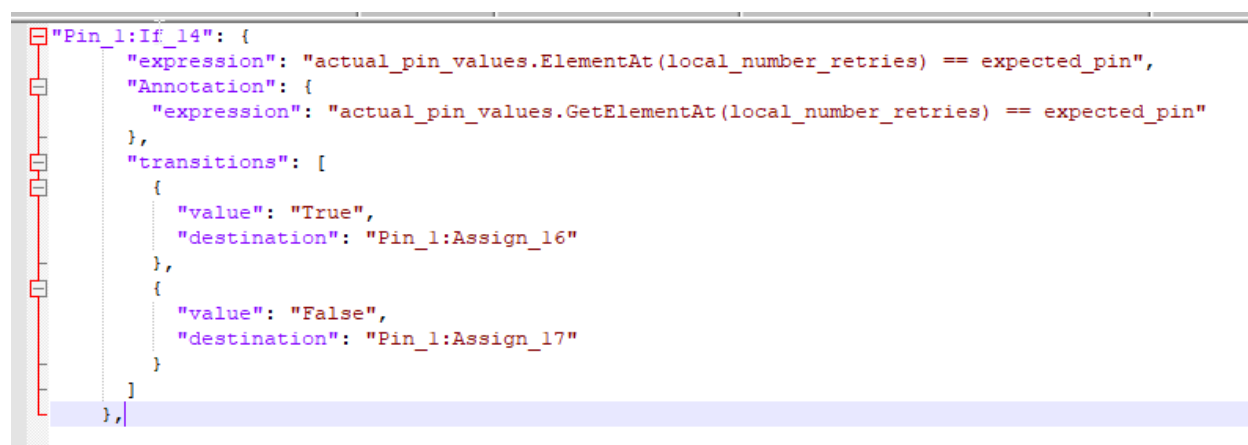
Al doilea tip de adnotare pe care poate să-l interpreteze RPA2Graph este @Expression. Aceasta ajută la rescrierea unei expresii într-un format dorit de consumator. Adnotarea trebuie să fie urmată de un obiect JSON valid, care să aibă drept cheie cuvântul „expression” și ca valoare, expresia rescrisă. Adnotarea se plasează la nivelul activității a cărei expresie trebuie simplificată. Un astfel de exemplu este:

Input:

@Expression

```
{"expression": "actual_pin_values.GetElementAt(local_number_retries) == expected_pin"}
```

Output:



```
{
  "Pin_1:If_14": {
    "expression": "actual_pin_values.ElementAt(local_number_retries) == expected_pin",
    "Annotation": {
      "expression": "actual_pin_values.GetElementAt(local_number_retries) == expected_pin"
    },
    "transitions": [
      {
        "value": "True",
        "destination": "Pin_1:Assign_16"
      },
      {
        "value": "False",
        "destination": "Pin_1:Assign_17"
      }
    ]
  }
},
```

Figură III.14 - Exemplu de rezultat pentru adnotarea @Expression

4. Calități arhitecturale

Calitățile software reprezintă cerințe non-funcționale pe care un produs trebuie să le îndeplinească. Printre cele mai cunoscute se numără performanța, securitatea, ușurința de întreținere a codului, testabilitatea etc [14]. Având în vedere cerințele aplicației și contextul în care aceasta va funcționa, ea trebuie să fie ușor de întreținut și de extins și eficientă. „Maintainability”, așa cum este standardizat conceptul, reprezintă ușurința de întreținere a codului și cuprinde mai multe concepte,

precum: modularitate, ușurința de înțelegere a codului, ușurința de extindere a lui, etc. În aplicația RPA2Graph, codul este unul modular, clasele fiind împărțite, în funcție de responsabilitate, în 3 zone: Modele, Servicii și Utilitare. Folosind soluții arhitecturale bine documentate, adică design pattern-urile amintite în secțiunea anterioară, soluția este ușor de înțeles, chiar și pentru dezvoltatorii începători. Sistemul prin care se adaugă suport pentru activități este unul ușor extensibil, fiind nevoie de o singură modificare în codul deja existent, adică de înregistrarea noului interpretor în clasa „Factory”. Suportul pentru operanzi și expresii noi se adaugă, de asemenea, foarte ușor, prin introducerea unui nou procesator în lanțul acestora din clasa „ExpressionUtils.cs”. În perspectivă, aplicația este menită de a fi rulată fie de un robot, fie de o altă aplicație software, deci ea trebuie să poată fi folosită ușor, din linia de comandă.

IV. Concluzii

1. Rezultate obținute. Aprecieri asupra procesului de dezvoltare.

Aplicația RPA2Graph este o continuare a inițiativei începute de articolul „Towards Automated Testing of RPA Implementations” [1], reușind să rezolve problemele întâlnite de prototipul descris în acesta. Astfel, noua soluție este capabilă de interpretarea ierarhiilor complexe de activități (secvențe de activități existente în nodurile unui flowchart, secvențe de activități existente în stările și pe tranzițiile dintr-o mașină cu stări finite, etc.) și de înțelegerea activităților de tip „InvokeWorkflow” care sunt capabile de trecerea firului de execuție de la un workflow la altul. RPA2Graph a plecat de la aceeași bază de cod ca în articolul de mai sus, dar, prin studierea problemelor întâmpinate în acel stadiu, am ajuns la concluzia că este nevoie de o abordare diferită. Astfel, am luat decizia de a încerca să încarc întreaga ierarhie în memorie cu ajutorul librăriei Windows Workflow Foundation. Astfel, am căpătat acces facil la structura internă a workflow-ului și la variabilele și argumentele din acesta. Totodată, procesul a fost unul mai eficient datorită existenței documentației oficiale de la Microsoft, care a ajutat la identificarea proprietăților utile ale activităților. Prima versiune a soluției, bazată pe Workflow Foundation era capabilă de interpretarea unui singur workflow, bazat pe secvențe și care nu știa să înțeleagă suficiente informații ale activităților. A urmat adăugarea suportului pentru flowchart-uri, mașini cu stări finite și suport specific pentru înțelegerea altor activități („If”, „While”, „DoWhile”, etc.). Recent a fost adăugat și suport pentru sistemul de adnotări, pentru a facilita integrarea cu clienți scriși în diferite limbaje de programare. Soluția curentă este capabilă de obținerea ierarhiei fișierului workflow, expunând-o sub formă de graf orientat, de variabilele și argumentele implicate și de identificarea tuturor workflow-urilor ce ajung să fie executate. Se poate observa că aplicația a evoluat mult, comparând punctul de plecare, dar mai are nevoie de modificări și îmbunătățiri pentru a ajunge un proiect software livrabil.

2. Direcții viitoare de dezvoltare a tool-ului

Așa cum este precizat în subcapitolul precedent, există numeroase direcții de dezvoltare spre care se poate îndrepta soluția curentă. În primul rând, datorită arhitecturii ușor de extins și de întreținut, o direcție este dată de îmbunătățirea suportului pentru alte activități, precum: „TryCatch”, „InvokeCode”, „MultipleAssign”, etc.. Prin scrierea de cod specific pentru acestea, putem obține informații care să crească calitatea rezultatelor clienților. Spre exemplu, în cazul unei aplicații bazate pe execuție simbolică care trebuie să genereze automat date de test pentru un workflow UiPath, orice informație legată de modificarea variabilelor sau a firului de execuție este vitală. Astfel, obținerea structurii din „TryCatch” ajută la eficientizarea testării unui sistem de gestionare a excepțiilor. „InvokeCode”, fiind o activitate capabilă să modifice variabilele din program, trebuie de asemenea interpretată, la fel și cea de „MultipleAssign”, care poate modifica valorile mai mult variabile și argumente din proiect în același timp. Pe lângă extinderea suportului pentru activități, o altă zonă unde poate fi îmbunătățit proiectul este cea de gestionare a excepțiilor. În acest moment, suportul este unul destul de restrâns, iar pentru a fi o soluție ușor de folosit, acesta trebuie dezvoltat mult mai mult. Motivația stă în importanța identificării rapide a expresiilor sau adnotărilor ce nu au putut fi interpretate sau în identificarea rapidă a activităților necunoscute. O altă direcție de dezvoltare este dată de îmbunătățirea suportului de adnotări, care, în acest moment, știe să înțeleagă doar două adnotări: @Variables și @Expression.

În acest moment, versiunea aplicației RPA2Graph este 0.12.1-alpha, aflându-se în stadii premature de dezvoltare. Modul în care poate fi rulată aplicația este prin folosirea unui executabil, în linia de comandă, acest lucru făcând-o potrivită pentru orice mediu în care va fi integrată.

Bibliografie

- [1] M. Cernat, A. N. Staicu și A. Ștefănescu, „Towards Automated Testing of RPA Implementations,” în *11th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, Decembrie 2020.
- [2] UiPath, „Robotic Process Automation (RPA),” UiPath, 1 Aprilie 2020. [Interactiv]. Available: <https://www.uipath.com/>. [Accesat 13 Septembrie 2021].
- [3] Gartner, „Magic Quadrant for Robotic Process Automation,” Gartner, 26 Iulie 2021. [Interactiv]. Available: <https://www.gartner.com/doc/reprints?id=1-26Q65VFT&ct=210706&st=sb>. [Accesat 13 August 2021].
- [4] A. Konrad, „From Communism To Coding: How Daniel Dines Of \$7 Billion UiPath Became The First Bot Billionaire,” 30 Septembrie 2019. [Interactiv]. Available: <https://www.forbes.com/sites/alexkonrad/2019/09/11/from-communism-to-coding-how--daniel-dines-of-7-billion-uipath-became-the-first-bot-billionaire/?sh=4e9a8b29206e>. [Accesat 15 August 2021].
- [5] UiPath Doc, „Introduction Test Suite,” UiPath, 1 Aprilie 2020. [Interactiv]. Available: <https://docs.uipath.com/test-suite/docs/introduction-test-suite>. [Accesat 13 August 2021].
- [6] G. Neelam, P. M. Aditya și M. L. Soffa, „Automated test data generation using an iterative relaxation method,” în *ACM SIGSOFT Software Engineering Notes*, Noirembrie 1998.
- [7] W. M. P. van der Aalst, M. Bichler și A. Heinzl, „Robotic Process Automation,” *Business & Information Systems Engineering*, p. pages269–272, 2018.
- [8] Businesswire, „New Study Finds Majority of Global Office Workers Crushed by Repetitive Tasks, Stifled From Pursuing More Fulfilling Work,” Businesswire, 20 Mai 2021. [Interactiv]. Available: <https://www.businesswire.com/news/home/20210520005067/en/New-Study-Finds-Majority-of-Global-Office-Workers-Crushed-by-Repetitive-Tasks-Stifled-From-Pursuing-More-Fulfilling-Work>. [Accesat 13 Septembrie 2021].
- [9] UiPath, „Hyperautomation,” UiPath, 1 April 2020. [Interactiv]. Available: <https://www.uipath.com/rpa/hyperautomation>. [Accesat 2021 August 2021].
- [10] M. Cernat, A. N. Staicu și A. Ștefănescu, „Improving UI Test Automation using Robotic Process Automation,” în *15th International Conference on Software Technologies*, Ianuarie 2020.
- [11] UiPath, „StudioX,” UiPath, 1 Aprilie 2020. [Interactiv]. Available: <https://www.uipath.com/product/studiox>. [Accesat 13 August 2021].

- [12] UiPath, „State machines,” UiPath, 1 Aprilie 2020. [Interactiv]. Available: <https://docs.uipath.com/studio/docs/state-machines>. [Accesat 16 August 2021].
- [13] E. Gamma, R. Helm, R. Johnson și V. John, Elements of Reusable Object-Oriented Software, Oxford University Press, Inc., 1994.
- [14] ISO, „Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models,” în *ISO*, 2011.