

TDK-dolgozat

Fritsi Dániel

Szegedi Tudományegyetem
Informatikai Tanszékcsoport

Egy layout-független módszer és eszköz kidolgozása
Magic/uniPaaS-ban fejlesztett
alkalmazások automatizált GUI tesztelésére

Fritsi Dániel

Programtervező Informatikus (MSc) szakos hallgató

Témavezetők:

Dr. Ferenc Rudolf egyetemi adjunktus

Nagy Csaba doktorjelölt

2012

Tartalomjegyzék

Absztrakt	4
1 Bevezetés	5
2 Bevezetés a Magic világába	7
3 Egy Magic alkalmazás automatikus GUI tesztelése	9
3.1 Magic alkalmazások statikus elemzése	9
3.2 GUI események rögzítése	11
3.3 Rögzített GUI események visszajátszása	15
3.3.1 Események végrehajtása	15
3.3.2 A visszajátszás kiértékelése	17
3.4 A módszer hátrányai	19
3.5 Az eszköz rövid bemutatása	19
3.5.1 Eseményvezérlő kulcsszavak	21
3.5.2 Validáló kulcsszavak	22
4 Kapcsolódó szakirodalom és összehasonlítás más technikákkal	23
4.1 Kapcsolódó szakirodalom	23
4.2 Összehasonlítás más technikákkal	24
4.2.1 Kulcsszó-alapú tesztelés	24
4.2.2 Adat-alapú tesztelés	25
4.2.3 Modul-alapú tesztelés	26
4.2.4 Koordináta-alapú és "white-box" tesztelés	26
5 Eredményeim és jövőbeli tervek	28
6 Irodalomjegyzék	29
Melléklet 1	32
Melléklet 2	33

Absztrakt

Egy jó szoftverfejlesztési folyamat megvalósításához elengedhetetlen egy alapos tesztelési fázis, amely általában drága, de szükséges ahhoz, hogy egy megbízható és jó minőségű szoftverterméket állítsunk elő. Egy alkalmazás grafikus felületének tesztelése rengeteg manuális munkát igényel, viszont gyakran a tesztelés néhány lépése egyébként automatizálható lenne. Ilyen esetekben a tesztautomatizáló eszközök nagy segítségére lehetnek a fejlesztőknek és tesztelőknek, különösen a változások újratestelésénél, azaz a regressziós tesztekénél. Még ezeknek az eszközöknek a használata mellett is sok manuális munka szükséges a tesztesetek fenntartásához. Például, ha egy ablak elrendezése egyszerűen csak megváltozott anélkül, hogy ez kihatással lett volna az alkalmazás funkcionalitására, az ablakhoz tartozó teszteseteket újra rögzíteni kell. Ez a nehéz karbantartási munka jelenti a regressziós tesztek egyik legnagyobb problémáját a grafikus alkalmazásoknál.

Dolgozatomban bemutatok egy olyan megközelítést, amely a grafikus alkalmazások forráskódjából kinyerhető GUI információkat használja az automatikus tesztelés folyamán arra, hogy elrendezés-független teszt szkripteket hozzon létre. Ezzel a technikával a már rögzített teszt szkriptekre nem lesznek hatással a grafikus felületben bekövetkezett kisebb változások. Így sokkal könnyebb karbantartani a nagyon drága regressziós teszteseteket, ahol az is előfordulhat, hogy több ezer tesztesetet kell fenntartania a tesztelő csoportnak. Az ötletet az motiválta, hogy hogyan teszteljünk egy olyan alkalmazást, amely egy negyedik generációs programnyelvben (4GL), a Magic/uniPaaS-ban íródott. Ebben a programnyelvben a grafikus elemek szerkezetének leírása (az ablakok felépítése, pozíciók, méretek, stb.) a kódban van eltárolva, amit statikus elemzéssel össze tudunk gyűjteni. A bemutatott módszerhez a Szoftverfejlesztés Tanszéken egy eszközt is kifejlesztettünk, amit a tanszék egy ipari partnere is használ a tesztelési feladataihoz [1].

Kulcsszavak: Szoftverfejlesztés, GUI tesztelés, tesztautomatizálás, 4GL

1 Bevezetés

Egy alkalmazás felhasználói felületének alapos tesztelése nem könnyű feladat különösen nagy, összetett és sok különböző funkcióval rendelkező rendszerek esetében. A tesztelőknek követniük kell a több ezer tesztet lépéseit és az eredményeket kézzel kell kiértékelniük. Ez a nehéz munka támogatható különböző automatizáló eszközökkel, amelyek képesek követni és rögzíteni a tesztelő által előidézett eseményeket (billentyűzet, egér, stb.) majd visszajátszani őket a tesztelés alatt lévő alkalmazásnak. Ez nagy segítség pl. regressziós tesztek esetén, azonban még mindig sok manuális munka marad így is a tesztelőre. A tesztelőnek továbbra is rögzítenie kell a tesztet a legelső alkalommal és továbbra is karban kell tartani a rögzített tesztet, ahogy az alkalmazás fejlődik és a grafikus felület vagy az alkalmazás implementált funkciói változnak, bővülnek.

A jelenlegi eszközök támogatják a legnépszerűbb 3. generációs nyelveket (pl.: C/C++, Java, C#), de a magasabb szintű programnyelvek, mint például a 4. generációs nyelvek (Magic 4GL, ABAP, Informix) is népszerűek lettek a szoftverfejlesztésben az elmúlt években. A fejlesztők, akik ezekben a nyelvekben fejlesztenek, nem hagyományos értelemben vett forráskódot írnak, hanem az absztrakció egy magasabb szintjén, mondjuk egy alkalmazásfejlesztési környezetet használva írják az alkalmazásokat. Ilyen nyelvekben általában az alkalmazás kódja tárolja a felhasználói felület leírását is (pl.: egy ablak vagy form szerkezete, pozíciók, színek, méretek). Mivel ezeket az információkat ki lehet nyerni a statikus elemzés során, ezért felhasználhatjuk őket a tesztelési folyamat során is. A munkámban ezt az információt arra használom, hogy a grafikus felület automatizált tesztelési folyamatát elrendezés-függetlenné tegyem. Ez azt jelenti, hogy egy teszt szkript¹ nem függ pontos koordinátáktól vagy a grafikus elemek elrendezésétől, és így a teszteteket² újra fel lehet használni, még akkor is, ha a fejlesztők kisebb változásokat eszközölnek az alkalmazás felhasználói felületében (pl.: átrendezik a gombok sorrendjét).

A grafikus alkalmazások regressziós tesztelésével az egyik legnagyobb probléma, hogy előfordulhat, hogy a legkisebb változás eredményeként újra kell írni az összes tesztet. Egy

¹ A szoftvertesztelésben egy teszt szkript nem más, mint olyan utasítások sorozata, amelyeket végre kell hajtani a tesztelt alkalmazáson, hogy leteszteljük, hogy az alkalmazás megfelelően működik-e.

² A tesztet olyan feltételek vagy változók sorozata, amely alapján egy tesztelő meg tudja határozni, hogy a tesztelt alkalmazás megfelelően működik-e vagy sem.

lehetséges megoldásként a dolgozatomban bemutatott technika jelentősen csökkentheti a regressziós tesztek fenntartásának költségét ezzel biztosítva az alkalmazás minőségét.

Legfontosabb eredményeim:

- kidolgoztam egy elrendezés-független GUI tesztautomatizáló módszert,
- ez a módszer egy tanszéki projekt keretében implementálva is lett, ami validálja hatékonyságát.

Dolgozatomat a következő módon építem fel. A 2. fejezetben adok egy rövid bevezetést a Magic 4GL világáról. Ezután a 3. fejezetben leírom a Magic alkalmazások automatikus elrendezés-független tesztelésének módszerét. Itt leírom a grafikus események rögzítésének legfontosabb lépését, bemutatom, hogyan lehet őket visszajátszani a tesztelt alkalmazásnak majd megmutatom, hogy hogyan lehet a visszajátszás eredményeit kiértékelni. A 4. fejezetben bemutatok néhány hasonló munkát és eszközt. Legvégül az 5. fejezetben összefoglalom a dolgozatomat és leírok néhány jövőbeli tervet a kutatásommal kapcsolatban.

2 Bevezetés a Magic világába

A 80-as évek elején a Magic Software Enterprises (MSE) bemutatott egy új 4. generációs programnyelvet, a Magic 4GL-t. A fő elképzelés az volt, hogy az alkalmazást egy magasabb szintű „meta nyelven” állítsuk össze és hagyjuk, hogy a végső alkalmazást egy alkalmazás generátor motor hozza létre. Ezzel a módszerrel az MSE kész megoldásokat tudott szállítani felhasználó kezelésre, nyomtatásra, adatbázis kezelésre, stb., és a fejlesztőknek nem kellett időt pazarolniuk ezeknek a komponenseknek a kidolgozására. Egy Magic alkalmazás elfutott a népszerű operációs rendszereken, mint a DOS vagy a UNIX, így az alkalmazások könnyen hordozhatóak voltak. A Magic gyorsan fejlődött és megjelent egy újabb verzió, a uniPaaS. Az új verzió olyan korszerű technológiákat támogatott, mint a RIA (Rich Internet Applications), SOA (Service Oriented Architecture) és a mobil fejlesztés (például Android).

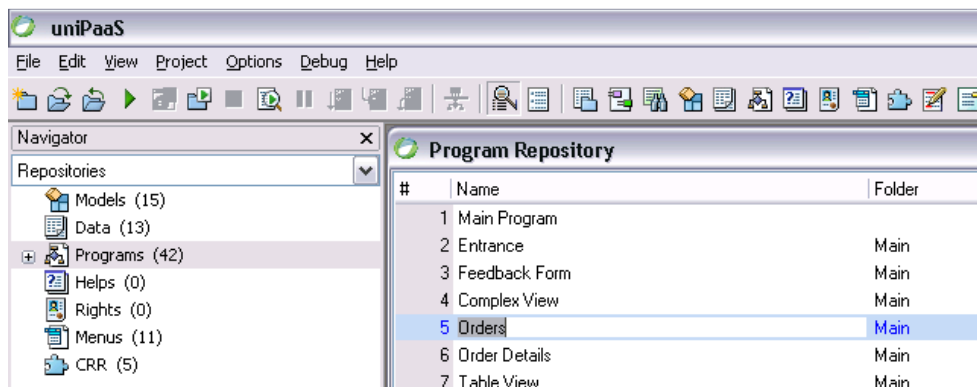
A Magic nyelv egyedi meta modellje magasabb absztrakciós szintű utasításokat tartalmaz, amelyek közelebb állnak az üzleti logikához. Amikor valaki egy Magic-es alkalmazást fejleszt, akkor ő tulajdonképpen a Magic Runtime Application Environment-t (MRE) programozza az említett meta modell segítségével. Ez a meta modell az, ami a Magic-et tulajdonképpen egy RADD (Rapid Application Development and Deployment) eszközzé teszi.

A Magic alaphoz rengeteg képernyőt és riport szerkesztőt tartalmaz, mivel elsősorban azért találták ki, hogy adatkezelési és jelentési vállalati alkalmazásokat fejlesszenek benne. A Magic legfontosabb elemei a különböző üzleti logikai entitás típusok, vagyis az adattáblák. Egy táblának vannak oszlopai, amelyeket különböző programok (és altaskok) módosíthatnak, amelyek különböző formokhoz, menükhöz és súgó képernyőkhöz vannak kötve. Ezek az elemek ugyancsak üzleti logikát valósítanak meg különböző logikai utasításokon (például változók (virtuális vagy tábla oszlop) kiválasztása, változók frissítése, feltételes utasítások) keresztül. Az **1. ábra** a uniPaaS fejlesztői környezetet szemlélteti.

A uniPaaS, mint 4. generációs programozási nyelv legfőbb összetevői:

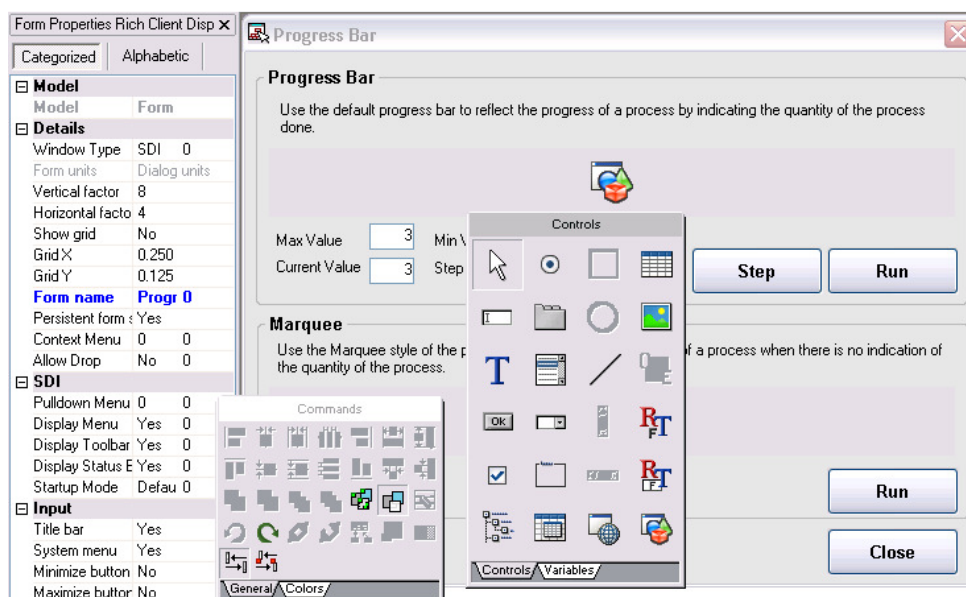
- **Adatobjektumok.** Ezek lényegében az adatbázistáblák leírásai. Ahogyan a táblák és az oszlopaik, valamint az elsődleges és külső kulcsok definiálva vannak az adatbázisban, hasonló módon tudják ezeket definiálni a uniPaaS-ban is.
- **Programok.** Az alkalmazás logikája valósul meg ezeken az elemeken keresztül. A programok legfelsőbb szintű taskok, különböző altaskokkal alattuk. Egy task mindig valamilyen adatobjektumon hajt végre valamilyen műveletet. Megadhatjuk, hogy

mely adatbázistáblákat használja a task és, hogy milyen műveleteket hajtson végre rajtuk.



1. ábra – A uniPaaS fejlesztői környezet

- **Menük.** Az alkalmazásban használhatunk különböző főmenüket és pop-up menüket, amelyek itt vannak definiálva.
- **Form-ok.** A uniPaaS saját, grafikus szerkesztő felületet biztosít az ablakok elrendezésének (menük, kontrolok, stb) és különböző tulajdonságainak (pl.: cím, méret, pozíció) meghatározására. Egy grafikus ablak nem más, mint egy uniPaaS form. A uniPaaS fejlesztői környezetben rendelkezésünkre áll rengeteg beépített kontrol, de létrehozhatunk saját, egyedi kontrolokat is. Egy form mindig egy adott taskhoz van definiálva. A uniPaaS ablakszerkesztője látható a 2. ábrán.

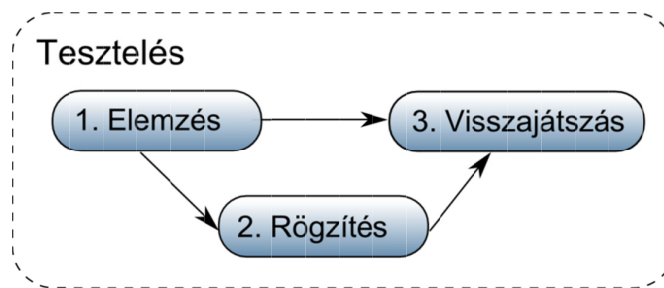


2. ábra – A uniPaaS ablakszerkesztője

3 Egy Magic alkalmazás automatikus GUI tesztelése

Egy tanszéki projekt keretében egy Magic Teszt Automatizáló eszköz fejlesztésében vettem részt, amely lehetővé teszi a Magic/uniPaaS-ban fejlesztett alkalmazások automatikus GUI tesztelését. Egy Magic alkalmazás automatikus tesztelésének 3 főbb lépése van (lásd **3. ábra**).

1. **A Magic alkalmazás elemzése.** Itt egy statikus elemzést hajtunk végre, hogy kinyerjünk minden szükséges információt az alkalmazás grafikus elemeinek felépítéséről.
2. **GUI események rögzítése.** Ez az a lépés, ahol a billentyűzet és egér eseményeket figyeljük és rögzítjük, és arra használjuk őket, hogy elrendezés-független teszt szkripteket hozzunk létre.
3. **A rögzített GUI események lejátszása.** Az elrendezés-független teszt szkripteket arra használjuk, hogy egér és billentyűzet eseményeket szimuláljunk a tesztelt alkalmazáson.



3. ábra – Egy Magic alkalmazás automatikus GUI tesztelésének főbb lépései

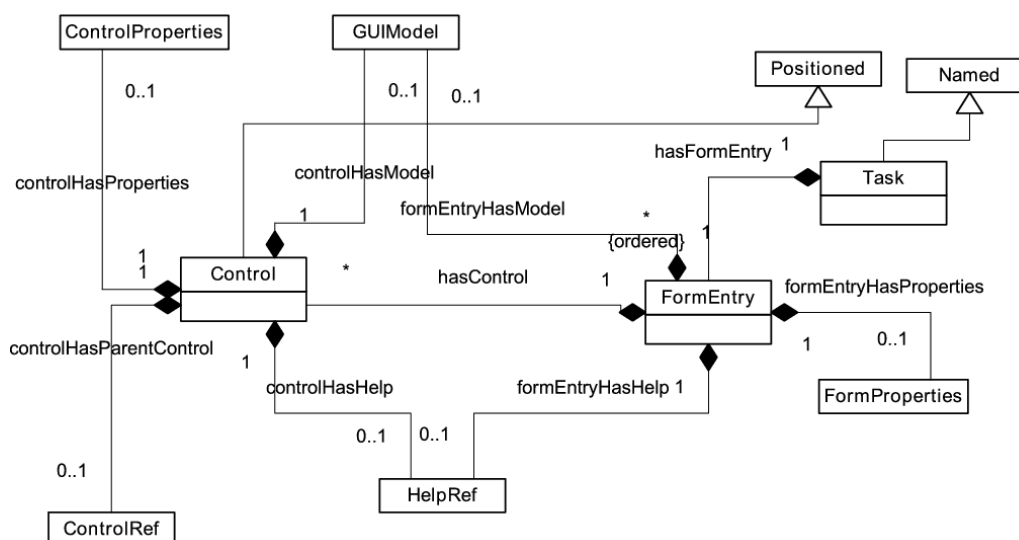
Ezeket a lépéseket a további alfejezetekben részletesebben is bemutatom.

3.1 Magic alkalmazások statikus elemzése

Egy Magic alkalmazás statikus elemzése során információkat nyerünk ki az alkalmazás futtatása nélkül, annak legalacsonyabb szintű leírásából, a forráskódjából. Az elemzés eredményeként egy, az alkalmazás szerkezetét leíró gráf keletkezik, amit Absztrakt Sematikus Gráfnak (ASG) nevezünk. Egy ilyen gráf felépítése a Columbus Magic Sémán alapul. A sémát egy UML osztály diagrammal definiáljuk, amely tartalmazza a Magic nyelv fontosabb építőelemeit (lásd **4. ábra**).

Egy Magic alkalmazásnak nincsen „hagyományos értelemben” vett forráskódja, hanem az aktuális modelljének egy mentése írja le. A régebbi Magic verziókban ez a mentés egy

strukturált szöveges fájl, az újabb verziókban, mint például a uniPaaS-ban pedig egy XML fájl. A régi DOS-alapú Magic verziók nem tartalmaztak bonyolult grafikus elemeket, mint az újabb verziók, ezért a Magic Sémát úgy kellett megtervezni, hogy az összes Magic verzió sajátosságait kezelni tudja.



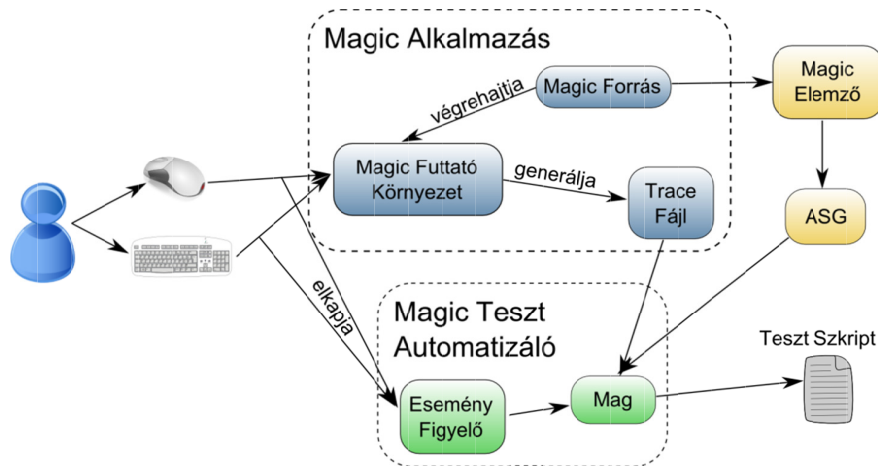
4. ábra – A Magic Séma egy egyszerűsített része, amely a Taskokat, Formokat, Controlokat és a közöttük lévő kapcsolatokat írja le.

Az ASG egy pontja egy elemet reprezentál a forráskódban. Minden ilyen pont a megfelelő forráskódbeli elem egy példánya. Két pont között kétféle kapcsolat jöhet létre: aggregáció és asszociáció. Az aggregációt összetett nyelvtani elemek (a szintaxis fa egy élének) leírására lehet használni, az asszociációval pedig szemantikai részleteket (pl.: azonosító referenciák) tudunk leírni. A gráfot egy statikus elemző eszköz hozza létre, amely leelemezi a tesztelt alkalmazás leíró fájlját, létrehozza a pontokat, majd egyesíti őket az ASG-ben.

A 4. ábra a Magic Séma egy egyszerűsített része látható, amely az automatizálás számára fontos GUI elemeket emeli ki. Az ábra bemutatja a grafikus elemek és a Magic Taskok közötti kapcsolatot. A *FormEntry* egy grafikus ablaknak felel meg, amely egy *Task* része, de egy Taskhoz több *FormEntry* is tartozhat. Egy *FormEntry* tulajdonságait (cím, méret, stb.) a *FormProperties* pont tárolja. A *hasControl* élen keresztül juthatunk el a *FormEntry*knél lévő grafikus elemekig. Minden grafikus elemet egy *Control* pont ír le, és a grafikus elemek tulajdonságait (felirat, típus, pozíció, stb.) a *ControlProperties* pontok tárolják. A Magic alkalmazások visszatervezéséről bővebben egy másik munkában olvashatunk [2].

3.2 GUI események rögzítése

Egy alkalmazás grafikus felhasználói felületének manuális tesztelésénél a tesztelő egyszerűen végigmegy egy tesztet bizonyos lépésein (kattintás valahova, egy szöveg beírása egy szöveges mezőbe, stb.). Ezután pedig leellenőrzi, hogy a tesztelt alkalmazás az elvárt viselkedést produkálja-e. Annak érdekében, hogy ezt a manuális tesztelési folyamatot automatizálni tudjuk, először is definiáltam egy teszt szkript formátumot, amelyben le lehet írni azokat a GUI eseményeket, amelyeket a tesztelt alkalmazáson végre kell hajtani. Kézzel megírni egy ilyen szkript fájlt lehetséges, de igencsak nehézkes, hiszen egy tesztet sok bonyolult GUI eseményt tartalmazhat. Ezért a Magic Teszt Automatizáló nem csak lejátszani képes az ilyen szkript fájlokat, hanem rögzíteni is. Az **5. ábra** a GUI események rögzítésének folyamatát szemlélteti.



5. ábra – A GUI események rögzítésének folyamata

Mielőtt teszt szkripteket rögzítünk vagy lejátszunk, elemeznünk kell a tesztelt alkalmazást, hogy megkapjuk a hozzá tartozó ASG-t. Ezt a lépést csak egyszer kell végrehajtani az alkalmazás minden verziójánál, és miután létrehoztuk az ASG-t, azt fel tudjuk használni az összes rögzítéshez és visszajátszáshoz.

A GUI események rögzítésének az alapja, hogy a Windows lehetővé teszi számunkra, hogy figyeljük az egér és billentyűzet eseményeket, továbbá, hogy információt nyerjünk ki arról a grafikus elemről, amin az esemény történt. A GUI események megfigyelésének a módja, hogy a Windows API tartalmaz egy *SetWindowsHookEx* nevű függvényt, amely a megfigyelés módját várja paraméterül. Ennek a függvénynek a segítségével nem csak egér vagy billentyűzet eseményeket figyelhetünk meg, hanem figyelni tudjuk az ablakok közötti

üzenetváltásokat és a shell eseményeket is. A függvény továbbá paraméterül vár egy előre megadott prototípusú (*HookProc*) callback függvényt is. Amikor meghívjuk a *SetWindowsHookEx* függvényt, a Windows eltárolja a callback függvény címét, és amikor egy egér vagy billentyűzet esemény történik, akkor a Windows meghívja az eltárolt címen keresztül ezt a callback függvényt és adatokat ad át rajta keresztül az aktuális eseményről. A **6. ábra** a *SetWindowsHookEx* és a *HookProc* prototípusát szemlélteti.

```
HHOOK WINAPI SetWindowsHookEx(    LRESULT CALLBACK HookProc(
    __in int idHook,              __in int code,
    __in HOOKPROC lpfn,           __in WPARAM wParam,
    __in HINSTANCE hMod,         __in LPARAM lParam
    __in DWORD dwThreadId        );
);
```

6. ábra – A *SetWindowsHookEx* és a *HookProc* prototípusa

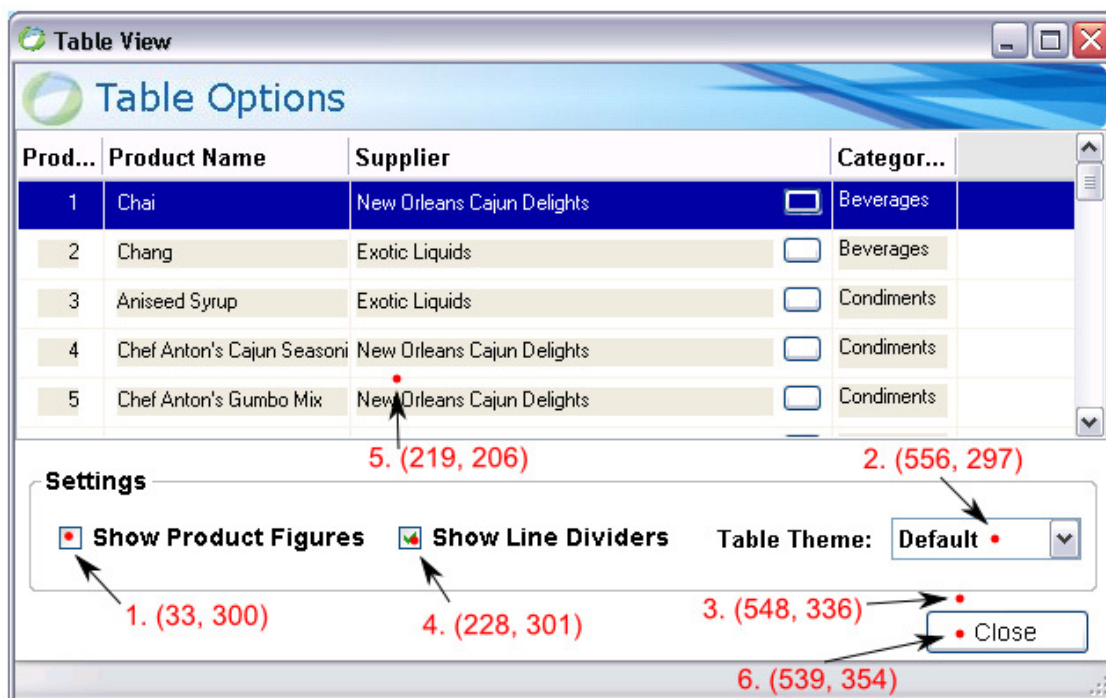
Egy teszt szkript rögzítésének az első lépése, hogy a tesztelt alkalmazást el kell indítani a Magic Futtató Környezetben (MRE). Ezt a futtató környezetet be lehet úgy állítani, hogy generáljon egy trace fájlt, ami dinamikus futási információkat tartalmaz. A trace fájlból ki lehet nyerni, hogy melyik task indult el vagy záródott be, továbbá ennek az eseménynek a pontos idejét. Ezt az információt arra használok, hogy meghatározzam, melyik task futott és melyik form volt aktív abban a pillanatban, amikor egy felhasználói esemény bekövetkezett.

A felhasználói események figyelését a Magic Teszt Automatizáló egy külön komponense, a Hook DLL végzi. A tesztelő elindítja a Magic alkalmazást a MRE-ben, majd elindítja a Magic Teszt Automatizáló eszközt, ahol a felvétel módot választja. Ettől kezdve a Hook DLL aktiválódik és figyelni kezd minden GUI eseményt, amely a tesztelt alkalmazáson történik. A tesztelő elkezdheti az egerét vagy billentyűzetét használni és a Windows minden eseményt elküld a Hook DLL-nek, amely pedig továbbítja őket a Magic Teszt Automatizáló eszköz magjának. A Teszt Automatizáló eszköz figyelni fogja a trace fájlt, és amikor egy GUI esemény érkezik, akkor meghatározza a Windows kontrollt, amin az esemény történt. Ezután meghatározza azt az ASG-beli pontot, amihez a kontroll tartozik. A GUI eseményhez tartozó Windows kontroll meghatározása szintén a Windows API-n keresztül történik, mert a GUI esemény tartalmazza annak a kontrollnak az azonosítóját, amin az esemény történt. Ezt az azonosítót felhasználva információt (méret, pozíció, felirat, szülő kontroll, stb.) nyerhetünk ki a kontrollról például a *GetWindowRect*, *GetWindowText* és *GetParent* függvényekkel.

Miután összegyűjtöttünk minden szükséges információt a Windows kontrollról, a következő feladat, hogy összekössük a hozzá tartozó ASG-beli ponttal. Korábban említettem,

hogy egy FormEntry mindig egy Task-hoz tartozik. Tehát kiolvassuk a trace fájlból, hogy abban a pillanatban, amikor a GUI esemény történt melyik Task volt aktív, majd végigmegyünk az ASG-n és megkeressük benne ezt a Task-ot. Miután meghatároztuk ezt az ASG-beli Task pontot, lekérdezzük a hozzá tartozó FormEntry-ket. A Windows az egy ablakon lévő kontrolokat egy fában tárolja, ahogy a Magic ASG is. Elkezdünk végigmenni a Windows kontrolhoz tartozó fán és az ASG-ben lévő kontrolhoz tartozó fán, kezdve az aktív FormEntry-től. Összehasonlítjuk a Windows kontrol és az ASG-beli kontrol pont tulajdonságait (pl.: méret, pozíció, felirat, stb.), és ezzel a módszerrel beazonosítjuk az ASG-beli Control pontot, ami a megadott Windows kontrolhoz tartozik.

Az ASG-beli pontok azonosítására egyedi scope-olt azonosítót használunk. Minden ASG-beli pontnak egyedi scope-olt azonosítója van, amely a szülő pontjának a típusát és nevét, valamint egy elválasztó szekvenciát használ. Például a *Pj~Test::Pr~Main::Fe~Settings::Ct~Close* azonosító egy *Close* nevű Controlhoz (*Ct*) tartozik, ami egy *Settings* nevű FormEntry-n (*Fe*) van rajta, és ez a FormEntry a *Main* nevű Program (*Pr*) része, amit pedig a *Test* nevű Projekt (*Pj*) tartalmaz. A legtöbb esetben ez az azonosító egyedi marad a tesztelt alkalmazás különböző verziói között.



7. ábra – Egy uniPaaS demó alkalmazás egy ablaka példa tesztlépésekkel a teszteléséhez.

Egy teszt szkript rögzítésének az utolsó lépése, hogy elmentsük az összegyűjtött információt egy szkript fájlba, amikor a felhasználó leállítja a rögzítést. A szkript fájl mentésekor két formátum közül választhatunk:

- A Magic Teszt Automatizáló saját formátuma. Az ilyen típusú szkript fájlok kevésbé bővíthetők, de a Teszt Automatizáló közvetlenül kezeli őket.
- Python szkript formátum. Ha ezt a formátumot választjuk, akkor a Teszt Automatizáló a szkript fájlt egy Python értelmezőn keresztül kezeli, viszont itt a Python nyelv minden elemét ki lehet arra használni, hogy fejlettebb szkript fájlokat írjunk.

A 7. *ábra* egy uniPaaS demó alkalmazás egy ablakát mutatja, amely azt mutatja be, hogy hogyan lehet használni a táblázat kontrolokat. Ezen az ábrán azt szemléltetem, hogy egy tesztelő milyen lépéseket hajtana végre, amikor ezt az ablakot teszteli. Rögzítettem az illusztrált lépéseket a Magic Teszt Automatizáló segítségével és elmentettem az eredményt Python szkript formátumban. A 8. *ábra* ezt a szkript fájlt ábrázolja.

```
from testrunner import *
from testrunner_ext import *

def runScript():
    assert waitFor(3014, findWindow(741, 379, "Table View")) == True
    mouseClickAtControl("Fe~Table View::Ct~Divider Box", MouseButton.LEFT, 11, 23)
    mouseClickAtControl("Fe~Table View::Ct~Theme Combo", MouseButton.LEFT, 44, 14)
    mouseClickAtControl("Fe~Table View::Ct~Theme Combo", MouseButton.LEFT, 47, 33)
    mouseClickAtControl("Fe~Table View::Ct~Show Figures Box", MouseButton.LEFT, 13, 17)
    mouseClickAtControl("Fe~Table View::Ct~Table", MouseButton.LEFT, 218, 129)
    mouseClickAtControl("Fe~Table View::Ct~Close Button", MouseButton.LEFT, 36, 22)
```

8. ábra – A 7. ábrán látható lépésekhez tartozó elrendezés-független Python szkript

Látható, hogy a Magic Teszt Automatizáló eszköz összekapcsolta a Magic forrást az ASG-vel és a kigyűjtött azonosítók segítségével legenerálta a szkript fájlt. A Magic Teszt Automatizálót nem Magic-es alkalmazások tesztelésére is lehet használni, így amikor nem áll rendelkezésre az ASG, az eszközt egy általános tesztautomatizáló eszközként is használhatjuk, ahol a szkript fájlban található GUI eseményeket koordináták alapján generáljuk le. Újra végrehajtottam az előző tesztlépéseket ezzel a módszerrel és újra elmentettem a Python szkript fájlt. Az eredmény a 9. *ábrán* látható.

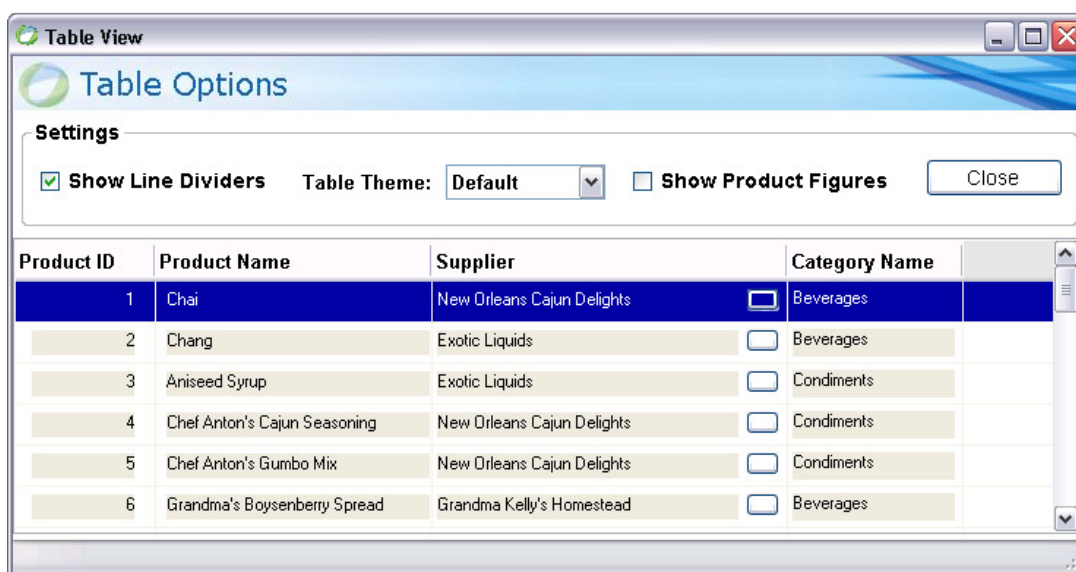
Látható, hogy mindkét szkript azonos mennyiségű utasítást tartalmaz. Amikor visszajátsszuk a két szkriptet, akkor ugyanazt az eredményt fogják produkálni, de mi történik,

ha átrendezzük az ablakot? Illusztrációként a **10. ábrán** egy példa látható az átrendezett ablakra.

```
from testrunner import *
from testrunner_ext import *

def runScript():
    assert waitFor(3014, findWindow(741, 379, "Table View")) == True
    mouseClickAXY(MouseButtons.LEFT, 33, 300)
    mouseClickAXY(MouseButtons.LEFT, 556, 297)
    mouseClickAXY(MouseButtons.LEFT, 548, 336)
    mouseClickAXY(MouseButtons.LEFT, 228, 301)
    mouseClickAXY(MouseButtons.LEFT, 219, 206)
    mouseClickAXY(MouseButtons.LEFT, 539, 354)
```

9. ábra – A 7. ábrán látható tesztlépésekhez tartozó pozíció-alapú Python szkript.



10. ábra – A 7. ábrán látható példa ablak átrendezett verziója

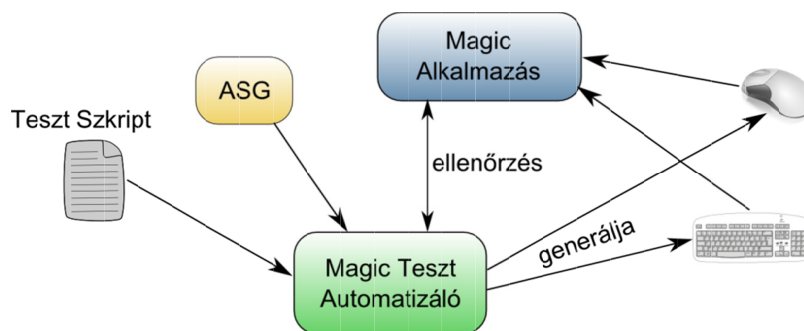
Az alkalmazás ugyanúgy működik, ahogy eddig, csak a kontrolok vannak más helyen. Ha végrehajtjuk az elrendezés-független Python szkriptet, akkor az eredmény ugyanaz lesz, mint korábban, mert a Magic Teszt Automatizáló újraszámítja a koordinátákat az egyedi azonosítók alapján. Ezzel szemben, ha a pozíció-alapú Python szkriptet futtatjuk, akkor az eredmény hibás lesz, mert a kontrolok már nem ugyanazon a pozíción vannak, mint korábban.

3.3 Rögzített GUI események visszajátszása

3.3.1 Események végrehajtása

Ahhoz, hogy visszajátsszunk egy rögzített szkript fájlt, először is szükségünk van magára a

fájltra, és az ASG-re, hogy a benne lévő egyedi azonosítókat össze tudjuk kötni a megfelelő Windows kontrollokkal. Visszajátszás során el kell indítanunk a Magic alkalmazást a Magic Futtató Környezetben és be kell töltenünk a szkript fájlt a Magic Teszt Automatizáló eszközbe. A szkript fájl tartalmazza a felvett egér és billentyűzet eseményeket, amelyeket a Teszt Automatizáló először értelmez, majd végrehajt. Ennek a menetét szemlélteti a **11. ábra**.



11. ábra – Felvett GUI események visszajátszása

Az események értelmezése során kikeressük az aktuális eseményben található ASG pontot az egyedi azonosítója alapján. Ezután meghatározzuk az ehhez az ASG ponthoz tartozó Windows kontrolt. Ez a beazonosítás néha elég nehézkes lehet, ugyanis a kontrol alacsony szintű implementációja teljesen más lehet, mint egy egyszerű Magic kontrollé. Például, ha egy fa kontrolról vagy egy groupboxról van szó, amelyek több apró kontrolt tartalmazhatnak. Hogy kiküszöböljük ezt a beazonosítási problémát, összegyűjtök minden információt (méret, pozíció) az ASG-ből, amire szükség van ahhoz, hogy beazonosítsak egy GUI elemet, de ez még mindig nem elég, hiszen az alkalmazás egyszerre több ablakot, sőt ablakot tartalmazó ablakot is megjeleníthet. Ezért szükségünk van minden információra az adott kontrol szülőelemeiről is. Ez alapján meghatározhatjuk azt az ablakot, amin az adott kontrol található. A Windows API segítségével meghatározhatjuk ezt az ablakot és az elmeit a fejléc szövegek, pozíciók és a szülő ablakok azonosítói alapján. Az ablak azonosítóját ily módon a *FindWindow* és *FindWindowEx* Windows API függvényekkel kaphatjuk meg a fejléc szövegek és egyéb információk alapján, amiket az ASG-ből nyerek ki. Az aktuális GUI elem ablakhoz viszonyított relatív koordinátáit is könnyen meghatározhatjuk a Windows API segítségével. Mivel a GUI elem benne lehet több GUI elembe is, például egy groupboxban, ezért a Windows kontrol fa alján kezdjük keresni és végigsétálok a fa tetejéig és folyamatosan újraszámolom a relatív koordinátát, amíg el nem jutok a keresett elemig.

Nem mindig elég azt tudni, hogy melyik Windows kontrol tartozik a megfelelő ASG ponthoz, hanem néha azt is tudnunk kell, hogy az adott kontrolon belül pontosan hova

kattintsunk. Egy gomb esetében nem fontos, hogy a gomb belsejére vagy szélére kattintunk, de egy fa kontrol esetében ez már nem mindegy. Tudnunk kell a klikkelés pontos helyét az adott kontrolon belül. A Magic Teszt Automatizáló oly módon generálja le a szkript fájlt, hogy eltárolja az esemény relatív koordinátáit a beazonosított Magic kontrolhoz képest.

Miután meghatároztuk annak a GUI elemnek a Windows beli azonosítóját, amin egy egér vagy billentyűzet eseményt kell végrehajtani, és amint megvannak a pontos relatív koordináták, elkezdünk felfele sétálni a Windows kontrol fában és összeadjuk ezeket a relatív koordinátákat, amíg el nem jutunk a fa tetejéig. Miután megvannak az abszolút koordináták, a Windows API segítségével legeneráljuk a megfelelő billentyűzet vagy egér eseményt és elküldjük a tesztelt alkalmazásnak.

Ezek a lehetőségek azonban még mindig nem elegendőek ahhoz, hogy egy manuális GUI tesztelést szimuláljunk, hiszen rengeteg olyan alkalmazás van, amelyknél például, ha beírnunk egy szöveget egy szöveges mezőbe a gépelés során elindul a háttérben egy folyamat (pl.: keresés), amely közben eredményeket jelenít meg egy másik GUI elemben. A következő GUI eseményt pedig csak azután lehet végrehajtani, hogy a háttérfolyamat befejeződött. Manuális tesztelésnél ez nem jelent problémát, hiszen egy ember látja, hogy mikor ért véget a háttérben futó folyamat. Egy tesztautomatizáló eszköz segítségével támogatni tudjuk azt, hogy késleltetéseket helyezünk a szkriptbe, amiken a futtatás csak egy megadott idő vagy egy bizonyos esemény után lép tovább. Például megmondhatjuk egy tesztautomatizáló eszköznek, hogy várjon pontosan 10 másodpercet vagy várjon addig, amíg egy grafikus elem elveszti vagy megkapja a fókuszot, vagy várjon addig, amíg egy grafikus elem aktiválódik.

3.3.2 A visszajátszás kiértékelése

Egy visszajátszás eredményeit kiértékelni szintén nagyon fontos feladat. Ennek a kiértékelésnek néhány része automatikusan is meghatározható futás közben (pl.: legyen megnyitva egy adott ablak), de néhány olyan vizsgálatot, amire szükség van (pl.: egy szöveges mező a megfelelő szöveget tartalmazza-e), a tesztautomatizáló eszköz nem tudja automatikusan meghatározni, ezért általában szükség van a rögzítés után a szkript fájl manuális módosítására, hogy ellenőrző eseményeket szűrassunk be bizonyos lépések után. A tesztelő ezt úgy teheti meg, hogy egyszerűen ellenőrző (assert) eseményeket szűr be a megfelelő helyekre. A Magic Teszt Automatizáló a következő ellenőrző lehetőségeket támogatja:

- Ellenőrizni, hogy az alkalmazás kontrol fájában, vagy egy adott ablakban van-e olyan GUI elem, amely tartalmaz egy adott szöveget, vagy van-e olyan ablak, ami a megadott fejléccel rendelkezik.
- Egy adott GUI elem szövegének összehasonlítása egy megadott szöveggel.
- Ellenőrizni, hogy egy adott GUI elem fókuszban van-e.
- Ellenőrizni, hogy egy adott GUI elem engedélyezve van-e.
- Ellenőrizni, hogy egy checkbox vagy radiobutton be van-e kattintva.

A Magic Teszt Automatizáló megvizsgálja és kiértékeli az eredményeket és ennek megfelelően jelenti le a teszt szkript végrehajtásának az eredményét.

Ezeknek az ellenőrző függvényeknek az előnye, hogy ezeket már a korábban említett késleltető függvényekben is használni lehet. Például tudunk olyan ellenőrzést mondani, hogy várjunk addig, amíg egy checkbox be nem kattintódik, vagy amíg egy szöveges mező nem tartalmazza a megadott szöveget. Sőt, Python szkriptek segítségével komplexebb teszt szkripteket is létrehozhatunk. Például tudunk olyat csinálni, hogy ha egy szöveges mező egy megadott szöveget tartalmaz, akkor hajtsunk végre bizonyos lépéseket, ha pedig nem, akkor más lépéseket hajtsunk végre.

```
from testrunner import *
from testrunner_ext import *

def runScript():
    assert waitFor(3014, findWindow(741, 379, "Table View")) == True
    mouseClickAtControl("Fe~Table View::Ct~Divider Box", MouseButton.LEFT, 11, 23)
    assert validate(checkState("Fe~Table View::Ct~Divider Box", CheckStates.Checked)) == True
    mouseClickAtControl("Fe~Table View::Ct~Theme Combo", MouseButton.LEFT, 44, 14)
    mouseClickAtControl("Fe~Table View::Ct~Theme Combo", MouseButton.LEFT, 47, 33)
    assert validate(compareText("Fe~Table View::Ct~Theme Combo", "Rose")) == True
    mouseClickAtControl("Fe~Table View::Ct~Show Figures Box", MouseButton.LEFT, 13, 17)
    assert validate(checkState("Fe~Table View::Ct~Show Figures Box", CheckStates.Checked)) == True
    mouseClickAtControl("Fe~Table View::Ct~Table", MouseButton.LEFT, 218, 129)
    assert validate(checkFocus("Fe~Table View::Ct~Show Figures Box", False)) == True
    mouseClickAtControl("Fe~Table View::Ct~Close Button", MouseButton.LEFT, 36, 22)
    assert validate(findWindow(741, 379, "Table View")) == False
```

12. ábra – Ellenőrző példák egy Python szkriptben

A 12. ábra a 8. ábrán található Python szkriptet ábrázolja, kiegészítve ellenőrző függvényekkel. Miután a checkbox-ra kattintunk, van egy *checkState* függvény, ami leellenőrzi, hogy a checkbox valóban bekattintódott-e. Miután kiválasztottunk egy elemet a combobox-ból, van egy *compareText* függvény, ami megnézi, hogy a combobox a megfelelő szöveget tartalmazza-e, valamint miután a táblázatban klikkeltünk, leellenőrizzük, hogy fókuszban van-e a "Fe~Table-View::Ct~Show Figures Box" azonosítójú GUI elem. Legvégül

pedig az `"Fe~Table View::Ct~CloseButton"` azonosítójú gombra kattintás után megnézzük, hogy az ablak tényleg bezáródott-e. Az **1. melléklet** egy hasonló elrendezés-független Python szkriptet tartalmaz, amely egy másik ablak funkcionalitását teszteli.

3.4 A módszer hátrányai

Számos előnyét bemutattam az elrendezés-független automatikus tesztelési technikának, amiből a legfontosabb, hogy a felvett teszt szkript érzéketlen lesz a grafikus felületben fellépő kisebb változtatásokra. Van azonban néhány hátránya is a módszernek, amiről itt szeretnék beszélni.

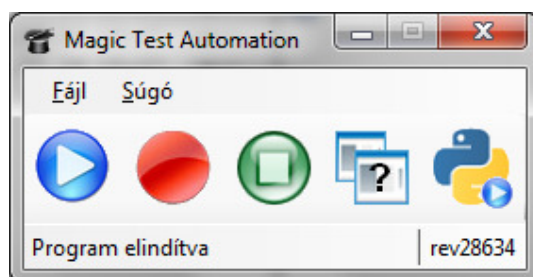
Először is, a grafikus felületben bekövetkező kisebb változtatásoknak azokat tartom, amikor egyszerűen átrendezik az ablak grafikus elemeit és nem drasztikusan módosítják a struktúráját. A módszerem a kontrollt az ASG-beli egyedi azonosítója alapján ismeri fel, majd a kontrol fában haladva párosítja hozzá a megfelelő Windows kontrollt. Ha megváltozik a kontrolok hierarchiája, akkor az eszköz nem lesz képes újra felismerni ugyanazt a kontrollt.

Egy másik fontos hátrány, hogy ez a módszer nagyban a kontrolon belüli relatív koordinátákon alapul. Ezek a koordináták erősen függenek a kontrol belső elrendezésétől. Például, ha egy fa kontrolban az elemek sorrendje megváltozik futtatásonként, akkor a Magic Teszt Automatizáló nem lesz képes követni az új sorrendet.

Egy másik hibalehetőség, hogy a tesztelő futás közben átméretezi, vagy áthelyezi valamelyik kontrollt. Mivel ezeket az információkat az ASG-ből nyerem ki, ezért ez a módszer addig fog működni, amíg a kontrolok mérete és pozíciója nem változik a futtatás alatt.

3.5 Az eszköz rövid bemutatása

A **13. ábra** a Magic Teszt Automatizáló eszköz indításakor megjelenő felületét ábrázolja.

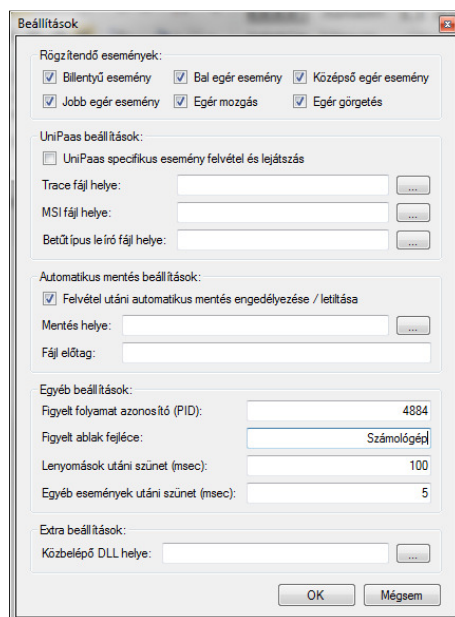


13. ábra – A Magic Teszt Automatizáló eszköz

Az eszközt úgy terveztem, hogy minél felhasználóbarátabb legyen, ezért a főképernyő

csak a főbb funkcionalitásoknak megfelelő gombokat tartalmazza. A főképernyő 5 gombot tartalmaz, amelyek balról jobbra rendre a következők:

- **Lejátszás.** Ezzel a gombbal lehet elindítani a korábban betöltött saját Teszt Automatizáló szkriptformátumában megírt teszt szkriptet. A gombra kattintva a program előtérbe hozza a tesztelt alkalmazást, majd elindítja a szkript fájlt.
- **Felvétel indítása.** A gombra kattintva elindul a tesztelt alkalmazáson történő billentyűzet és egér események figyelése és rögzítése. Ilyenkor a tesztelt alkalmazás szintén előtérbe kerül, hogy a tesztelő megkezdhesse annak tesztelését.
- **Felvétel leállítása.** A gombra csak akkor lehet kattintani, ha a felhasználó korábban elindította a felvételt. Ha a tesztelő végzett egy teszt szkript rögzítésével, akkor erre a gombra kattintva leállíthatja azt, majd elmentheti a szkriptet.
- **Folyamat kiválasztása.** Ezzel a gombbal lehet a tesztelt alkalmazást kiválasztani. Erre azért van szükség, hogy a billentyűzet és egér események figyelésekor be tudjuk azonosítani, hogy a megfelelő folyamat azonosítóval (process id) rendelkező, azaz a tesztelt alkalmazástól érkezett-e az esemény.
- **Python szkript futtatása.** Az utolsó gombbal tudjuk a Python programozási nyelvben megírt szkript fájlok lejátszását elindítani.



14. ábra – Az eszköz beállítási lehetőségei

Az eszközbe természetesen implementáltam néhány olyan beállítási lehetőséget, amely egy tesztautomatizáló eszköztől elvárható. Ezeket a beállításokat a Fájl menü Beállítások

almenüjével hozhatjuk elő. Ezt szemlélteti a **14. ábra**. Az ábrán jól látható, hogy ki lehet választani, milyen típusú eseményeket akarunk rögzíteni, továbbá az is látható, hogy a Magic/uniPaaS specifikus teszt szkript rögzítést és visszajátszást ki és be lehet kapcsolni. Továbbá be lehet állítani bizonyos automatikus mentési opciókat, illetve lehetőség van arra is, hogy valaki egy előre megadott prototípusú függvényeket tartalmazó dll-t implementáljon és azt mint egy plugin-t használhassa felvétel közben.

3.5.1 Eseményvezérlő kulcsszavak

Ebben a fejezetben azt szeretném bemutatni, hogy a szkript fájlokban milyen eseményvezérlő kulcsszavak érvényesek, és azok mire használhatóak. Ezt szemlélteti az **1. táblázat**.

Kulcsszó	Leírás
keyPress keyDown keyUp	Ezekkel a kulcsszavakkal lehet egy egyszeri billentyűzet leütést, lenyomást, illetve felengedést szimulálni.
systemKeyPress systemKeyDown systemKeyUp	Ezekkel a kulcsszavakkal speciális billentyűk (alt, funkcióbillentyűk, stb.) egyszeri leütését, lenyomását, illetve felengedését lehet szimulálni.
typeText	Ha egyszerre egy egész szöveget szeretnénk begépelni valahova, azt ennek a kulcsszónak a használatával tehetjük meg.
mouseToXY	Használatával a kurzor a megadott koordinátára mozog.
mouseWheelAtXY	Az egér görgőjét irányíthatjuk ezzel a kulcsszóval.
mouseClickAtXY mouseDownAtXY mouseUpAtXY mouseDoubleClickAtXY	Ezekkel a kulcsszavakkal egyszeri egér kattintást, lenyomást, felengedést, illetve dupla kattintást idézhetünk elő.
mouseToControl	Ezt a kulcsszót elrendezés-független lejátszásnál használhatjuk. Hatására az egér a megadott egyedi azonosítójú GUI elem felé mozog.
mouseWheelAtControl	Az egér görgőjét irányíthatjuk a megadott azonosítójú GUI elem felett elrendezés-független szkriptek lejátszásakor.
mouseClickAtControl mouseDownAtControl mouseUpAtControl mouseDoubleClickAtControl	Elrendezés-független szkriptek lejátszásánál a megadott azonosítójú GUI elemen idézhetünk elő egyszeri egér kattintást, lenyomást, felengedést, illetve dupla kattintást.

1. táblázat – A Magic Teszt Automatizálóban használható eseményvezérlő kulcsszavak

3.5.2 Validáló kulcsszavak

Ahogy korábban említettem, a tesztelés automatizálásához nem elég az, hogy képesek legyünk eseményeket rögzíteni illetve visszajátszani, hanem az is szükséges, hogy ezeknek az eseményeknek a hatását, kimenetelét ellenőrizni tudjuk. A **2. táblázat** azt szemlélteti, hogy a Magic Teszt Automatizálóban milyen ellenőrző kulcsszavak használhatóak.

Kulcsszó	Leírás
wait (<i>ellenőrző függvény</i>)	Ennek a kulcsszónak a használatával megadott ezredmásodpercnyi késleltetést idézhetünk elő.
waitFor (<i>ellenőrző függvény</i>)	A kulcsszó használatával azt érhetjük el, hogy a szkript futtatása addig nem folytatódik, amíg egy bizonyos esemény be nem következik.
validate (<i>ellenőrző függvény</i>)	Azt ellenőrzi le, hogy egy bizonyos esemény bekövetkezett-e.
findText (<i>esemény függvény</i>)	Megnézi, hogy a tesztelt alkalmazásban fellelhető-e a megadott szöveg.
findTextInCaption (<i>esemény függvény</i>)	Megnézi, hogy a tesztelt alkalmazásnak van-e olyan ablaka, amely a megadott fejléccet tartalmazza.
findTextInCurrentWindow (<i>esemény függvény</i>)	Megnézi, hogy a tesztelt alkalmazás aktuálisan aktív ablakában fellelhető-e a megadott szöveg.
findWindow (<i>esemény függvény</i>)	Megnézi, hogy van-e olyan ablak, amely a megadott méretekkel és fejléccel rendelkezik.
focusChanged (<i>esemény függvény</i>)	Megnézi, hogy változott-e a fókusz.
compareTextInCurrentWindow (<i>esemény függvény</i>)	Megnézi, hogy a megadott GUI elem a megadott szöveget tartalmazza-e.
checkEnabledInCurrentWindow (<i>esemény függvény</i>)	Megnézi, hogy a megadott GUI elem engedélyezve van-e.
checkStateInCurrentWindow (<i>esemény függvény</i>)	Megnézi, hogy a megadott GUI elem be van-e kattintva.

2. táblázat – A Magic Teszt Automatizálóban használható validáló kulcsszavak

4 Kapcsolódó szakirodalom és összehasonlítás más technikákkal

4.1 Kapcsolódó szakirodalom

Az automatikus szoftvertesztelés manapság egy nagyon fontos témakör. Ennek köszönhetően rengeteg könyv és cikk jelent meg ebben a témakörben [3] [4] [5] [6] [7], köztük olyanok is, amelyek tapasztalati példákon keresztül mutatják be, hogy hogyan érdemes a tesztelést gyorsítani és hogy milyen lehetőségek vannak az automatizálásra [8] [9] [10]. Ebben a fejezetben részletesen foglalkozok a kapcsolódó szakirodalommal, összefoglalva az olyan eszközökre, amelyek közel állnak az én munkámhoz.

A tesztautomatizáló keretrendszereket általában 5 generációra lehet osztani [11] [12]. Az első generációba az úgynevezett felvétel/lejátszás (record/playback) eszközök tartoznak, amelyek egyszerű teszt szkriptekkel dolgoznak, ahol egy teszt szkript egy tesztesetnek felel meg. A második generációs eszközök már fejlettebb teszt szkriptekkel dolgoznak, amelyek például függvényeket, ciklusokat is kezelnek. A harmadik generációs (data-driven) keretrendszerek adatokat vesznek ki a teszt szkriptből és így egy teszt szkriptet újra lehet futtatni különböző bemeneteken. Ezt nevezzük adat-alapú tesztelésnek [5] [13]. Egy másik elgondolás szerint a negyedik generációs tesztelés a kulcsszó-alapú tesztelés, ahol a tesztlétrehozási folyamat egy magasabb szintű tervezési és egy alacsonyabb szintű implementációs részre osztható, ahol különböző kulcsszavak vannak magasabb szinten definiálva [5] [14] [15]. Az új technikák a tesztautomatizálást még magasabb szintre emelik (ötödik generáció), ahol az automatikus teszteseteket inkább a fejlesztők hozzák létre, nem pedig a tesztelők [16].

Az én megközelitésem felfogható egy harmadik generációs megközelítésnek, mert gondosan megtervezett teszt szkripteknél az adatokat külön lehet választani magától a végrehajtási folyamattól. A kulcsszó-alapú tesztelés ötlete szintén hasonló az enyémhez, de az én teszteseteim inkább alacsonyabb szintűek, implementáció-közelibbek. Az ötlet, hogy támogassuk a tesztesetek felvételét és visszajátszását a forráskódban lévő GUI információk segítségével a 4GL környezetekben, a legjobb tudomásom szerint eléggé újszerű megközelítés. Azonban a statikus elemzés más megközelítésekénél [17] [18] [19] [20] egy bevett módszer az automatizált GUI tesztelések esetében, például a teszt szkriptek létrehozásánál. Az automatikus GUI tesztelés terén léteznek olyan megközelítések is, ahol a teszt szkriptek nem csak szöveges, hanem grafikus elemeket is tartalmaznak (pl.: kattintsunk

egy gombra, ahol a gomb nem azonosítóval vagy koordinátákkal van leírva, hanem egy kép van megadva hozzá) [21].

A szoftverfejlesztők számára rengeteg automatikus GUI tesztelő eszköz áll rendelkezésre. Csak hogy néhány példát említsek, a Selenium³ egy GUI tesztelő eszköz webes alkalmazások számára. Hasonlóan, mint egy alkalmazás tesztelésénél, a Selenium is lehetőségeket kínál fel a teszt szkriptek egyszerűsítésére azzal, hogy a weboldal html kódjában lévő kontrolok azonosítóit használja. Ez a megközelítés hasonló a miénkhöz, webes alkalmazások számára. A TestComplete⁴ a SmartBear Software terméke, amit szintén széles körben használnak a Magic világában. A HP szintén a Quality Center által biztosít lehetőségeket az automatikus GUI tesztelésre, és a Quick Test Professional (QTP) eszközük szintén széles körben használt a 4GL nyelvekben fejlesztett alkalmazásoknál. A Microsoft szintén biztosít lehetőségeket az automatikus GUI tesztelésre, például a .NET keretrendszer automatikus tesztelésére.

4.2 Összehasonlítás más technikákkal

4.2.1 Kulcsszó-alapú tesztelés

Egy kulcsszó a legegyszerűbb formában egy atomi tesztlépés vagy többi atomi tesztlépés aggregációja. Egy műveletet ír le, amit végre kell hajtani, így a kulcsszó-alapú tesztelést sokszor művelet-alapú tesztelésnek is nevezik. A legtöbb esetben a kulcsszó-alapú tesztelés két részre osztható [22]:

- tervezési fázis
- implementációs fázis

A tervezési szakaszban a tesztmérnökök meghatározzák a lépéseket a tesztesetekhez (pl.: egy szöveg beírása egy mezőbe, egy gombra kattintás, stb.). Később az implementációs fázisban a tervezők egy keretrendszert használva leírják a korábban megtervezett teszt szkripteket olyan formátumban, amit a keretrendszer értelmezni képes. Egy egyedi tesztelés alatt lévő rendszernek szüksége lehet különálló műveletekre, amelyeket fontos, hogy a tesztelési keretrendszer támogasson, ezért a tesztelők/fejlesztők sokszor maguknak fejlesztenek egy olyan automatizáló keretrendszert, amely kielégíti minden igényeiket [23] [24] [25].

Esetemben a tesztautomatizáló eszközöm egyben egy keretrendszer is, amely képes

³ <http://seleniumhq.org/>

⁴ <http://smartbear.com/products/qa-tools/automated-testing/>

megérteni alacsony szintű kulcsszavakat (pl.: mouseToXY), és támogatja a teszt szkripteket, mint Python szkript vagy egy szkript a saját tesztautomatizáló szkript formátumomban. Néhány esetben a tervezési és az implementációs fázist egyesíteni lehet egy lépéssé, ahol a tesztelők közvetlenül a keretrendszer formátumában tervezik meg a teszteseteket.

Az én tesztautomatizáló eszközüm használható kulcsszó-alapú tesztelésre, hiszen ez egy olyan eszköz, aminek van egy saját szkript nyelve, amely képes megérteni bizonyos kulcsszavakat és lefordítani őket egér, billentyűzet, vagy más eseménnyé. A mi eszközünkhez hasonlóan a Selenium is egy felvevő/visszajátszó eszköz, amit webes alkalmazások tesztelésére használnak. Olyan kulcsszavai vannak, mint Goto WEBOLDAL vagy Enter "felhasználónév", stb. A TestComplete szintén egy tesztautomatizáló eszköz, ami kulcsszavakat használ különböző események szimulálására. A TestComplete-tal továbbá képesek vagyunk kulcsszó-alapú teszt szkriptek rögzítésére, amit később kézzel is módosíthatunk. Egy másik példa a kulcsszó-alapú tesztautomatizáló eszközre a TestArchitect⁵, amit a LogiGear fejlesztett ki.

4.2.2 Adat-alapú tesztelés

Az adat-alapú tesztelés a tesztadatok és a végrehajtás különválasztásán alapul. A tesztelő bemeneteket és elvárt kimeneteket határoz meg egy teszt szkript számára, így a teszt szkriptet többször is végre lehet hajtani különböző bemenő adatokon. Az adat-alapú tesztelést legtöbbször egy alkalmazás valamely űrlapjának tesztelésére használják. A tesztelőnek ilyenkor meg kell adnia bemenő adatokat, amelyet a tesztelő keretrendszer automatikusan beír a tesztelt űrlapba, majd összehasonlítja az eredményt a korábban definiált elvárt eredménnyel. A legnagyobb különbség az adat-alapú és a kulcsszó-alapú tesztelés között, hogy a kulcsszó-alapú teszt szkripteknél az adatok rögzítve vannak a szkriptben (pl.: IrdBeHogy "teszt szöveg" egy szöveges mezőbe), és ha valaki ugyanazt a szöveges mezőt más adatokkal is le akarja tesztelni, akkor létre kell hoznia egy másik teszt szkriptet.

Pythont használva az én eszközümet is lehet használni adat-alapú tesztelésre. A Python segítségével az adatokat el lehet tárolni különböző változóban, amiket inicializálni lehet, akár egy külön fájlban, így a végrehajtást és a tesztadatokat teljesen külön lehet választani. Emellett tömböket használva a bemenő adatok és az elvárt eredmények tárolására, ciklusokat is tudunk használni arra, hogy egy teszt szkriptet többször végrehajtsunk különböző adatokon.

⁵ <http://www.testarchitect.com/>

Ilyen módon a szkriptekbe beégetett adatokat ki lehet küszöbölni. Egy példa adat-alapú tesztelésre a Magic Teszt Automatizálóval a **2. mellékleten** látható. Sok tesztautomatizáló keretrendszer támogatja a kulcsszó-alapú és az adat-alapú tesztelés kombinálást [26]. Például a TestComplete is képes bemenő adatok specifikálásra a teszt szkriptek számára, így a TestComplete-et is lehet használni adat-alapú tesztelésre. Különböző kiegészítőkkel a Selenium is képes ugyanazt a teszt szkriptet különböző adatokon futtatni.

4.2.3 Modul-alapú tesztelés

A modul-alapú tesztelésnél kicsi, különálló teszteseteket szoktak írni a tesztelt alkalmazás minden moduljához, csomagjához és funkciójához. Ezeket a kisebb teszt szkripteket később arra használják, hogy nagyobb tesztek hozzanak létre, amelyek egy tesztesetnek felelnek meg [27] [28]. Például, ha valaki az admin felhasználó egy bizonyos funkcionálisát akarja tesztelni, akkor írnia kell egy szkriptet a bejelentkezés tesztelésére és írnia kell egy külön szkriptet magának a funkciónak a tesztelésére. Ezután egy nagyobb teszt szkriptben meghívódik a bejelentkezést tesztelő szkript, majd ha az sikeresen lefutott, akkor meghívódik a funkciót tesztelő szkript. Ennek a módszernek az egyik nagy előnye, hogy egy modulban vagy funkcionálisban bekövetkezett változás esetén csak az adott modulra/funkcionálisra vonatkozó teszt szkriptet kell módosítani, és a többi teszt szkript érintetlen marad.

A Python segítségével a modul-alapú tesztelés szintén lehetséges az eszközzel, azzal, hogy valaki különálló automatizált Python szkripteket ír, amiket később összekapcsol egy nagyobb teszt szkriptben azzal, hogy beimportálja őket.

4.2.4 Koordináta-alapú és "white-box" tesztelés

Az automatikus GUI tesztelésnek az egyik legalapvetőbb módja a koordináta-alapú tesztelés, mert ilyenkor a tesztelő keretrendszernek semmit sem kell tudnia a tesztelt alkalmazásról. A koordináta-alapú tesztelés lényegében a kulcsszó-alapú tesztelés egyik fajtája. Általában egy kulcsszó egy koordinátát és egy felhasználói műveletet tartalmaz, amit el kell végezni a megadott koordinátán. A koordináta-alapú tesztelésnek két fajtája létezik:

- Abszolút koordináták használata az alkalmazás ablakán, ahol a koordináták relatívak általában a képernyő bal felső sarkához. Ez a módszer nem tűnik túl hasznosnak, azonban sok alkalmazásban az ablak helyzete nem befolyásoló tényező

- Az aktuálisan aktív ablak bal felső sarkához viszonyított relatív koordináták használata.

A koordináta-alapú tesztelés ésszerű megoldás, ha nem áll rendelkezésünkre semmilyen információ a tesztelt alkalmazásról. Azonban a koordináta-alapú teszt szkripteket nehéz fenntartani, mert könnyen változhat, hogy mi van egy adott koordinátán az alkalmazás következő futtatásakor.

Ha hozzáférésünk van a tesztelt alkalmazás forráskódjához vagy valamilyen dokumentációjához a tesztelés során és ezeket az információkat fel is használjuk a teszt szkriptek létrehozásánál, vagy a teszt szkriptek valamilyen struktúra szerint épülnek fel, akkor azt "white-box" („fehér-doboz”) tesztelésnek nevezzük. Lényegében az én módszerem is egy "fehér-doboz" tesztelés, mert az alkalmazás elrendezésének leírását használom a teszt szkriptek létrehozásához és futtatásához.

A **3. táblázat** a bemutatott technikák néhány szempont szerinti összehasonlítását szemlélteti. Az én módszeremre ráillenek ezek a technikák, így kombinálja az összes előnyüket és hátrányukat.

	Kulcsszó alapú	Adat alapú	Modul alapú	Koordináta alapú	"White-box" alapú
nincs szükség programozói tudásra	X	X			
nincsenek beágyazott adatok		X			
kombinálható teszt szkriptek			X	X	X
nincs szükség forráskódra	X	X	X	X	
kezeli az ablakokon történt átrendezéseket					X

3. táblázat – A különböző automatizálási technikák előnyei és hátrányai

5 Eredményeim és jövőbeli tervek

A dolgozatomban bemutattam egy módszert az automatikus elrendezés-független GUI tesztelésre az alkalmazás felhasználói felületének forráskódban lévő leírását használva. Statikus kódelemzést használok, hogy kigyűjtsem a felhasználói felület leírását, továbbá ezt a leírást kombinálom a felvétel során a futás közben dinamikusan generálódó trace adatokkal. Az eredményként keletkező teszt szkript csak elrendezés-független adatokat tartalmaz, amit később vissza lehet játszani az alkalmazásnak még akkor is, ha kisebb változás történt a felhasználói felületben. Ez a módszer drasztikusan csökkentheti a regressziós tesztek fenntartásának költségét, ahol a tesztelőknek és fejlesztőknek több ezer tesztet kell fenntartaniuk. Ez a módszer nem csak Magic/uniPaaS-ra alkalmazható, hanem általánosítható minden olyan programnyelvre, ahol rendelkezésünkre áll statikus információ az alkalmazás grafikus felületének elrendezéséről.

Egy tanszéki projekt keretében implementáltuk is ezt a módszert egy speciális 4GL környezet, a Magic/uniPaaS számára és az eszközt jelenleg is használja egy ipari partnerünk, ahol Magic-ben fejlesztenek több mint egy évtizede. A partnerünk olyan nagykereskedelmi termékeket szállít, ahol a szállított szoftver minősége az elsődleges prioritású, amihez alapos tesztelési folyamatok szükségesek. A módszeremet hasznosnak találta az ipari partnerünk a regressziós tesztelési folyamataik során.

A forráskódban tárolt GUI információkat használni automatikus teszteléshez a legjobb tudomáson szerint egy új megközelítés a Magic 4GL számára. Ahogy említettem, a módszer könnyen általánosítható olyan más nyelvekre (pl.: Delphi, vagy C# alkalmazások [29]), ahol a felhasználói felületről információkat lehet kinyerni a forráskódból statikus elemzéssel. Azonban ez a módszer olyan nyelvekre, ahol a GUI általában dinamikusan generálódik (pl.: Java), nehezen alkalmazható.

Jövőbeli terveimként javítani és fejleszteni akarok a validációs módszereken, és a tesztelést automatikus teszt szkript és teszt adatok generálásával akarom támogatni a Magic alkalmazások számára.

6 Irodalomjegyzék

- [1] Dániel Fritsi, Csaba Nagy, Rudolf Ferenc, and Tibor Gyimóthy, "A Layout Independent GUI Test Automation Tool for Applications Developed in Magic/uniPaaS," *SPLST' 11*, 2011.
- [2] Csaba Nagy, László Vidács, Rudolf Ferenc, and Tibor Gyimóthy, "Solutions for Reverse Engineering 4GL Applications, Recovering the Design of a Logistical Wholesale System," in *15th European Conference on Software Maintenance and Reengineering (CSMR)*, 2011, pp. 343-346.
- [3] Elfriede Dustin, Thom Garrett, and Bernie Gauf, *Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality.*: Addison-Wesley Professional, 2009.
- [4] Elfriede Dustin, Jeff Rashka, and John Paul, *Automated software testing: introduction, management, and performance*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.
- [5] Mark Fewster and Dorothy Graham, *Software test automation: effective use of test execution tools*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999.
- [6] Kanglin Li and Mengqi Wu, *Effective GUI Test Automation*. Alameda, CA, USA: SYBEX Inc., 2005.
- [7] Daniel J. Mosley and Bruce A. Posey, *Just Enough Software Test Automation.*: Prentice Hall PTR, 2002.
- [8] Dorothy Graham and Mark Fewster, *Experiences of Test Automation: Case Studies of Software Test Automation.*: Addison-Wesley Professional, 2012.
- [9] Lisa Crispin and Janet Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams.*: Addison-Wesley Professional, 2008.
- [10] Q. Nguyen Hung, Michael Hackett, and K. Whitlock Brent, *Happy About Global Software Test Automation: A Discussion of Software Testing for Executives.*, 2006.
- [11] Edward Kit, "Integrated Effective Test Design and Automation Software Development," *Software Development online*, 1999.
- [12] Cem Kaner, *Architectures of Test Automation*, 2000.

- [13] Richard Strang, "Data Driven Testing for Client/Server Applications," in *Proceedings of the Fifth International Conference on Software Testing, Analysis and Reliability (STAR'96)*., 1996, pp. 395-400.
- [14] Hans Buwalda, "Automated Testing with Action Words, Abandoning Record and Playback," in *Proceedings of the EuroStar Conference*., 1996.
- [15] Hans Buwalda and Maartje Kasdorp, "Getting Automated Testing under Control, Software Testing and Quality Engineering," *STQE magazine, division of Software Quality Engineering*, 1999.
- [16] Jeff Hinz and Martin Gijssen, *Fifth Generation Scriptless and Advanced Test Automation Technologies*, 2009.
- [17] Svetoslav R. Ganov, Chip Killmar, Sarfraz Khurshid, and Dewayne E. Perry, "Test generation for graphical user interfaces based on symbolic execution," in *Proceedings of the 3rd international workshop on Automation of software test*. Leipzig, Germany: ACM, 2008, pp. 33-40.
- [18] Yongzhong Lu, Danping Yan, Songlin Nie, and Chun Wang, "Development of an Improved GUI Automation Test System Based on Event-Flow Graph," in *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 02*.: IEEE Computer Society, 2008, pp. 712-715.
- [19] Atif M. Memon, Martha E. Pollack, and Mary Lou Soffa, "Hierarchical GUI test case generation using automated planning," *IEEE Transactions on Software Engineering*, pp. 144-155, 2001.
- [20] Jan Peleska, Helge Löding, and Tatiana Kotas, "Test Automation Meets Static Analysis," in *GI Jahrestagung (2)*., 2007, pp. 280-290.
- [21] Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller, "GUI testing using computer vision," in *Proceedings of the 28th international conference on Human factors in computing systems*. New York, NY, USA: ACM, 2010, pp. 1535-1544.
- [22] Hans Buwalda. (2011) Key Success Factors for Keyword Driven Testing. <http://www.logigear.com/resources/articles-presentations-templates/389--key-success-factors-for-keyword-driven-testing.html>.
- [23] Kanglin Li, Mengqi Wu, and Sybex, *Effective Software Test Automation: Developing an Automated Software Testing Tool*., 2004.

- [24] Jie Hui, Lan Yuqing, Luo Pei, Guo Shuhang, and Gao Jing, "LKDT: A Keyword-Driven Based Distributed Test Framework," in *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 02*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 719-722.
- [25] Izzat Alsmadi, *Building a GUI Test Automation Framework Using the Data Model: Introduction, Related Work, The GUI Modeler, Test Case Generation, Selection & Prioritization, GUI Test Harness Execution & Verification*. Saarbrücken, Germany, Germany: VDM Verlag, 2008.
- [26] Bin Mu, Mingkui Zhan, and Lanfang Hu, "Design and Implementation of GUI Automated Testing Framework Based on XML," in *Proceedings of the 2009 WRI World Congress on Software Engineering - Volume 04*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 194-199.
- [27] Mark Utting and Bruno Legeard, *Practical Model-Based Testing: A Tools Approach.*, 2006.
- [28] Jonathan Jacky, Margus Veanes, Colin Campbell, and Wolfram Schulte, *Model-Based Software Testing and Analysis with C#.*, 2007.
- [29] James Dr. McCaffrey, *.NET Test Automation Recipes: A Problem-Solution Approach.*, 2006.

Melléklet 1

Az alábbi melléklet egy elrendezés-független Python típusú szkript fájl, amely a RIA Demo⁶ alkalmazás Complex View nevű formjának egy részét teszteli le. A szkript rákattint a listában egy-egy listaelemre, majd leellenőrzi, hogy jobb oldalt a megfelelő elemek töltődtek-e be. A szkriptben megfigyelhetők az elrendezés-független módszer és a koordináta-alapú módszer nyomai is, hiszen a menüre kattintáskor csak koordináták alapján tudunk dolgozni.

```
# coding=ISO-8859-2

from testrunner import *
from testrunner_ext import *

# opens the Complex View window
def openMenu():
    mouseClickedAtXY(MouseButtons.LEFT, 109, 38)
    mouseClickedAtXY(MouseButtons.LEFT, 108, 55)

def runScript():
    assert waitFor(1000, findWindow(956, 745, "RIA Demo")) == True,
        'Assertion error (1)!'
    openMenu()
    assert waitFor(1000, findWindow(865, 585, "Complex View")) == True,
        'Assertion error (2)!'

    mouseToClickAtControl("Pj~RIA Demo::Pr~Complex View::Fe~Complex
View::Ct~Orders", MouseButtons.LEFT, 5, 15)
    assert validate(compareText("Pj~RIA Demo::Pr~Complex View::Fe~Complex
View::Ct~Contact Name", "Maria Anders")) == True, 'Assertion error (3)!'
    assert validate(compareText("Pj~RIA Demo::Pr~Complex View::Fe~Complex
View::Ct~Address Street", "Obere Str. 57")) == True, 'Assertion error (4)!'

    mouseToClickAtControl("Pj~RIA Demo::Pr~Complex View::Fe~Complex
View::Ct~Orders", MouseButtons.LEFT, 5, 114)
    assert validate(compareText("Pj~RIA Demo::Pr~Complex View::Fe~Complex
View::Ct~Contact Name", "Martín Sommer")) == True, 'Assertion error (5)!'
    assert validate(compareText("Pj~RIA Demo::Pr~Complex View::Fe~Complex
View::Ct~Address Street", "C/ Araquil, 67")) == True, 'Assertion error
(6)!'

runScript()
```

⁶ <http://riademo.magicsoftware.com/>

Melléklet 2

Az alábbi melléklet szintén egy elrendezés-független Python típusú szkript fájl, amely a Windows számológép összeadás funkcióját teszteli. Látható, hogy a bemeneti adatok és az elvárt eredmények el vannak választva magától a teszt függvényről, így az összeadás funkciót több bemeneti adaton is tesztelni lehet.

```
# coding=ISO-8859-2

from testrunner import *
from testrunner_ext import *

inputs = [
    ['123', '456', '789'],
    ['1', '2', '3'],
    ['391', '110', '75', '39']
]

expectedOutputs = ['1368', '6', '615']

def runScript():
    assert waitFor(1000, findWindow(423, 388, "Számológép")) == True,
        'Assertion error (1)!'

    for loop, input in enumerate(inputs):
        for index, number in enumerate(input):
            typeText(number)

            if index < len(input) - 1:
                keyPress(Keys.Add, False)
            else:
                keyPress(Keys.Enter, False)

        assertionString = 'Assertion error (2, loop: ' + str(loop + 1) + ')!'

        assert validate(findTextInCurrentWindow(expectedOutputs[loop])) ==
            True, assertionString

runScript()
```