VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PŘEVOD PROGRAMU ROBOTA Z PYTHON KÓDU DO REPREZENTACE JSON

CONVERSION OF A ROBOT PROGRAM FROM CODE TO AN AR-COMPATIBLE REPREZENTATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ŠIMON KADNÁR**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ZDENĚK MATERNA, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

140504

Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Student: **Kadnár Šimon**
Program: Informační technologie
Specializace: Informační technologie
Název: **Převod programu robota z Python kódu do reprezentace JSON**
Kategorie: Algoritmy a datové struktury
Akademický rok: 2022/23

Zadání:

1. Seznamte se se systémem pro programování robotů ARCOR2 a používanými reprezentacemi programu.
2. Seznamte se s možnostmi využití abstraktních symbolických stromů (AST) pro analýzu/generování kódu, se zaměřením na jazyk Python.
3. Navrhněte algoritmus pro převod programu v rámci ARCOR2 z Python do reprezentace JSON a případné nutné související změny v existujícím kódu nebo některé z reprezentací.
4. Implementujte navržené řešení.
5. Implementujte související jednotkové a integrační testy.
6. Změny publikujte na GitHubu.
7. Vytvořte video prezentující Vaši práci, její cíle a výsledky.

Literatura:

- Agrahari, Vartika, and Sridhar Chimalakonda. "AST [AR]–Towards using augmented reality and abstract syntax trees for teaching data structures to novice programmers." *2020 IEEE 20th International Conference on Advanced Learning Technologies (ICALT)*. IEEE, 2020.
- Wimmer, Christian, and Stefan Brunthaler. "Zippy on truffle: a fast and simple implementation of python." *Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity*. 2013.

Při obhajobě semestrální části projektu je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Materna Zdeněk, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 31.10.2022

Abstrakt

Dopyt po robotoch a ich rozvoj neustále rastie. Spolu s robotmi vznikajú aj rôzne prostredia ktoré ich programovanie uľahčujú. Jedno z takýchto prostredí je aplikácia ARCOR2. Aplikácia má zavedenú funkcionalitu ktorá umožňuje interne reprezentované dáta, riadiace robota, previesť do všeobecne známeho jazyka Python. Uvedený prevod bol zavedený z dôvodu využitia jazyka Python ako interpreta pre program robota v ktorom je možné program spustiť. Cieľom práce je využiť prevod z internej reprezentácie dát do jazyka Python k úpravám kódu robota. Prínosom práce je umožnenie úprav v jazyku Python vďaka spätnému prevodu upraveného kódu do internej reprezentácie dát. Operátor pracoviska vďaka tomu môže vytvoriť program pomocou rozšírenej reality a skúsený programátor v jazyku Python môže následne upraviť vytvorený program. Riešenie umožňuje efektívnu spoluprácu bežných užívateľov a programátorov, pričom každý pracuje s formou programu ktorá je na úrovni jeho schopností.

Abstract

The demand for robots and their development is constantly growing. Along with robots, various environments are being created to facilitate their programming. One such environment is the ARCOR2 application. The application has implemented a functionality that allows internally represented data, which control the robot, to be converted into the widely known Python language. The aforementioned conversion was introduced because Python is used as an interpreter for the robot program in which it is possible to run the program. The aim of the work is to utilize the conversion from the internal representation of data to the Python language for modifying the robot's code. The benefit of the work is enabling modifications in the Python language by converting the modified code back into the internal representation of data. As a result, the workstation operator can create a program using augmented reality, and an experienced programmer can subsequently modify the created program in the Python language. The solution allows for effective collaboration between regular users and programmers, with each working with a program form that corresponds to their abilities.

Kľúčové slová

robot, ARCOR2, prekladač, Python, JSON, AST, projekt, scéna, logické prvky, akčné body, akcie, skript

Keywords

robot, ARCOR2, compiler, Python, JSON, AST, project, scene, logic items, action points, actions, script

Citácia

KADNÁR, Šimon. *Převod programu robota z Python kódu do reprezentace json*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zdeněk Materna, Ph.D.

Převod programu robota z Python kódu do reprezentace json

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Maternu, Ph.D.. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Šimon Kadnár
4. mája 2023

Poďakovanie

Moje poďakovanie patrí pánovi Ing. Zdenkovi Maternovi, Ph.D., vedeckému pracovníkovi Ústavu počítačové grafiky a multimédií Fakulty informačních technologií VUT a členovy výskumnej skupine robotiky Robo Fakulty informačních technologií, ktorý bol vedúcim mojej bakalárskej práce. Ďakujem mu za odbornú pomoc, usmernenia, cenné rady a priateľský prístup počas celého obdobia.

Obsah

Kapitola 1

Úvod

V priemysle postupne prichádza k nasadeniu väčšieho počtu robotov, ktorý nahrádzajú ľudskú pracovnú silu. Roboti sú efektívnejší a môžu byť použitý v miestach ktoré sú rizikové pre človeka. Pred použitým robota do prevádzky je potrebné jeho naprogramovanie. Pod naprogramovaním rozumieme nastavenie činnosti ktorú bude robot vykonávať, alebo opätovné nastavenie v prípadne potreby pre iný typ práce. Naprogramovať algoritmus robota je možné aj pomocou nízkoúrovňového programovacieho jazyka. Pri výrobe s menším počtom zmien je takýto spôsob postačujúci. V prípade firiem v ktorých je zmena procesu výroby často sa opakujúca by takéto riešenie bolo príliš náročné a pomalé. Z tohto dôvodu existujú aplikácie ktoré sa špecializujú na zjednodušenie programovania robotov napr. pomocou grafického rozhrania. Jednou z takýchto aplikácií je ARCOR2[1].

V prípade, že vyššia úroveň riadenia robotov neumožňuje vykonať niektorú operáciu je potrebné vrátiť sa k ručnému programovaniu robota. Dáta ktoré riadia robota majú svoju vlastnú vnútornú reprezentáciu ktorá môže byť pre človeka náročná na čítanie a aj na písanie. Avšak pre stroj je vhodná. Takouto dátovou reprezentáciou je napr. JSON formát ktorý sa využíva aj v aplikácií ARCOR2. Aplikácia ARCOR2 má zavedenú funkcionalitu ktorá umožňuje previesť JSON formát do programovacieho jazyka Python, ktorý je všeobecne známy. Vzhľadom na to, že úprava kódu robota v JSON reprezentácií by bola náročná je možné využiť tento prevod a následne kód upraviť v Python formáte. Aplikácia ARCOR2, ale nedisponuje spätným prevodom.

Cieľom tejto práce je pridať algoritmus ktorý zabezpečí spätný prevod z jazyka Python do JSON reprezentácie. Podobne ako keď sa prekladá slovenský, český alebo iný jazyk do iného napr. anglického jazyka kedy je náročné previesť frázy, ktoré majú zmysel iba v danom jazyku a musia byť preložené spôsobom ktorý sa na prvý pohľad môže zdať zvláštny. Podobne aj preklad medzi dvoma programovacími jazykmi nemusí byť na prvý pohľad tak zrejmý a jasný. Z tohto dôvodu v mojej práci priblížim a vysvetlím reprezentáciu dát v jazykoch Python a JSON v systéme ARCOR2.

Práca je rozdelená do siedmych kapitol, v ktorých je samotná činnosť práce opísaná v piatich kapitolách. V kapitole 2 sú rozpísané základné princípy ako fungujú prekladače a z čoho sa skladajú. V tejto kapitole sú následne rozpísané jazyky medzi ktorými dochádza k prevodu. Kapitola 3 popisuje podrobnejšie architektúru aplikácie ARCOR2 a reprezentáciu dát. V kapitole 4 je opísaný návrh samotného algoritmu, možné problémy a ich riešenia. Kapitola 5 obsahuje popis implemnetovaného algoritmu a jeho jednotlivých častí. Predpo-

[1] Odkaz na ARCOR2: https://github.com/robofit/arcor2

slednou kapitolou bakalárskej práce je kapitola 6. Ide o testovanie navrhnutého algoritmu ktoré dokazuje, že výsledný program pracuje správne.

Kapitola 2

Prekladané jazyky

Podľa [7] je prekladač program, ktorý na vstupe prijíma zdrojový program zapísaný v zdrojovom jazyku, aby k nemu na výstupe generoval funkčne ekvivalentný cieľový program v cieľovom jazyku. Transformácia ktorú prekladač prevádza sa nazýva preklad. Ide o preklad zo zdrojového jazyka do cieľového jazyka. Preklad zdrojového jazyka programu z pravidla prebieha v šiestich hlavných častiach:

- lexikálna analýza,
- syntaktická analýza,
- sémantická analýza,
- generovanie vnútornej formy programu,
- optimalizácia,
- generovanie cieľového programu.

Z uvedených názvov hlavných fáz prekladu sú odvodené aj názvy jednotlivých častí prekladača, ktoré príslušné fázy vykonávajú. Ich názvy sú: lexikálny analyzátor, syntaktický analyzátor a generátor kódu. Lexikálny analyzátor číta zdrojový program a prekladá ho na reťaz lexikálnych symbolov čo sú jednotlivé elementy zdrojového jazyka napr. identifikátory, kľúčové slová a iné. Výstup lexikálneho analyzátora je vstupom pre syntaktický analyzátor, ktorého úlohou je overiť, že zdrojový program je syntakticky správne zapísaný, tzn. či reťaz lexikálnych symbolov patrí do zdrojového jazyka. Ak tomu tak skutočne je, potom výstupom syntaktického analyzátora je tzv. derivačný strom, ktorý reprezentuje syntaktickú štruktúru zdrojového programu. Sémantický analyzátor vykonáva kontrolu najrôznejších sémantických aspektov programu ako napr. deklaratívnosť premenných. Na základe syntaktickej štruktúry zdrojového programu vytvára generátor kódu cieľový program. Generovanie kódu sa pritom z pravidla samo rozpadá, ako bolo vidieť do troch samostatných fáz: generovanie vnútornej formy programu, optimalizácia, generovanie cieľového programu. Syntaktický strom, ktorý je väčšinou výstupom zo syntaktického analyzátora, sa vo väčšine prípadoch preloží na program zapísaný v tzv. vnútornej forme, ktorá umožňuje vykonať pomerne jednoduchým spôsobom optimalizáciu zdrojového programu. Takýto optimalizovaný program vo vnútornej forme je nakoniec preložený generátorom vnútorného programu na funkčne ekvivalentný program v cieľovom jazyku.

Na prekladanie sa používa aj postup pri ktorom hlavnou riadiacou jednotkou je syntaktická časť a nie sú nutne oddelené ďalšie časti prekladu. Tento prístup sa nazýva syntaxou riadený preklad. Spolu so syntaktickou analýzou býva spojená aj sémantická analýza a následne aj generovanie kódu. Dochádza tak ku kontrolovaniu programu z prekladaného jazyka a súčasne generovaniu kódu vo výslednom jazyku.

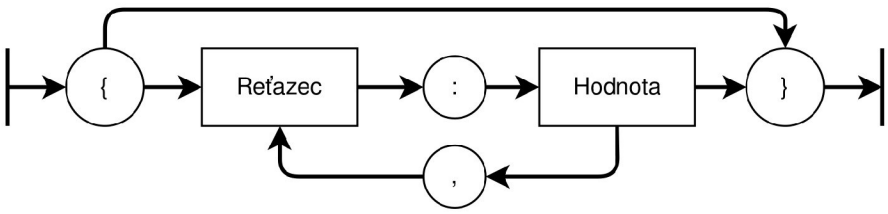
V tejto práci sa budem primárne zaoberať časťou generovania cieľového programu, pričom sa nejedná ani tak o preklad programu do iného jazyka, ale do dátovej reprezentácie. Samotná optimalizácia nie je potrebná z dôvodu, že program vo výslednej reprezentácii ARCOR2, má jedinečne špeciálnu a súčasne jednoduchú štruktúru. To znamená, že nie je moc možností ako ho optimalizovať. O samotnej dátovej reprezentácií v ARCOR2 viac v kapitole 3.

2.1 JSON

JSON, v anglickom jazyku JavaScript Object Notation je odľahčený formát ktorý sa často využíva k tzv. serializácii. Ide o proces konvertovania dátových štruktúr alebo stavov objektov do formátu ktorý môže byť uložený napr. ako textový súbor. Proces konvertovania dátových štruktúr alebo stavov objektov do formátu môže byť prenášaný aj sieťovým procesom. Opačný proces sa nazýva deserializácia - rekonštrukcia hodnoty na ten istý alebo, iný použiteľný aj netextový formát. Ide o vytvorenie sémanticky ekvivalentného klonu pôvodnej dátovej štruktúry [4].

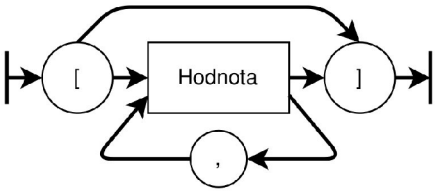
Nasledujúci odstavec bol prevzatý z [2]. Pre používateľa je jednoduché JSON písať, čítať a pre stroje je jednoduché ho analyzovať a generovať. JSON je založený na podmnožine štandardu JavaScript, Standard ECMA262 3rd Edition - December 1999. Ide o textový formát ktorý je nezávislý na jazyku ale používa konvencie ktoré sú známe z rodiny jazykov C (C, C++, C#), Java, JavaScript, Perl, Python a mnoho ďalších. Vďaka týmto vlastnostiam je JSON vhodným jazykom pre výmenu údajov. JSON sa skladá z dvoch štruktúr:

- Kolekcie párov mena a hodnoty.
  V rôznych jazykoch je realizovaný ako objekt, záznam, štruktúra, slovník, hašovacia tabuľka alebo asociatívne pole. Zapisuje sa pomocou množinových zátvoriek { }, pričom obsah zátvoriek je vyplnený pármi, t.j. názvami a hodnotami medzi ktorými je dvojbodka. Tieto páry sú následne oddelené čiarkami.



Obr. 2.1: Schéma kolekcie

- Usporiadaný zoznam hodnôt.
  Vo väčšine jazykov je realizovaný ako pole, vektor alebo zoznam. Zapisuje sa pomocou hranatých zátvoriek [ ], pričom jednotlivé hodnoty sú oddelené čiarkami.



Obr. 2.2: Schéma zoznamu

Vo vyššie uvedených bodoch ide o univerzálne dátové štruktúry, ktoré všetky moderné jazyky prakticky podporujú v jednej alebo v druhej forme prípadne v oboch formách. Na základe týchto skutočností dáva zmysel, že dátový formát ktorý je zameniteľný s formátmi programovacích jazykov je založený na týchto štruktúrach. Samotný JSON podporuje hodnoty akými sú:

- reťazec, ktorý sa zapisuje do úvodzoviek napr. `{"názov":"reťazec"}`,
- klasické čísla (integer), alebo-typ double napr. `{"názov":42}`,
- null napr. `{"názov":null}`,
- pravda alebo nepravda napr. `{"názov":true}`,
- štruktúry t.j. "Kolekcia párov mena a hodnoty" a "Usporiadaný zoznam hodnôt", ktoré môžu byť vnorené do seba napr. `{"názov":{[]}}`.

Pre programovací jazyk Python existuje modul `json` ktorý umožňuje prevod hodnôt z jazyka Python do JSON zápisu pomocou funkcie `dumps()`.

2.2 Python

Python je interpretovaný, objektovo orientovaný, vysoko úrovňový programovací jazyk s dynamickou sémantikou. Vďaka jeho vysokoúrovňovým vstavaným datovým štruktúram, kombinovaných s dynamickým písaním, je veľmi atraktívny pre rýchly vývoj množstva aplikácií, alebo pre použitie ako skriptovací/lepiaci jazyk na prepojenie existujúcich komponentov. Jednoduchá syntax jazyka Python, ktorá sa dá ľahko naučiť, zdôrazňuje čitateľnosť a znižuje tak náklady na údržbu programu. Interpret Pythonu a rozsiahla štandardná knižnica sú k dispozícii v zdrojovej alebo binárnej forme bez poplatku pre všetky hlavné platformy a možno ich voľne šíriť [1].

2.2.1 Anotácia typov

Jazyk Python disponuje niekoľkými vstavanými dátovými typmi. Podobne ako v iných programovacích jazykoch aj každý dátový typ jazyka Python má svoje značenie a špeciálne vlastnosti.

Medzi základné dátové typy ktoré sú dôležité pre túto prácu patrí:

- `str` - textová hodnota zapisuje sa vždy do úvodzoviek napr. "text"
- číselné typy:
  - `int` - celé číslo napr. číslo 42

  - `float` - číslo s desatinnou čiarkou napr. 1.1

- `bool` - boolovská hodnota (pravda alebo nepravda) napr. `True`
- `list` - zoznam. Podobne ako v JSON formáte ide o "usporiadaný zoznam hodnôt" ktorý využíva hranaté zátvorky a môže obsahovať iba rovnaké dátové typy,
- `dict` - slovník. Ide o mapovací typ ktorý je v JSON formáte reprezentovaný ako "kolekcia párov mena a hodnoty". Využíva množinové zátvorky a umožňuje na základe mena, ktorý slúži ako kľúč, ukladať hodnoty.

K identifikácií typov je možné použiť vstavanú funkciu `type()` ktorá na základe vloženej hodnoty vracia meno daného typu. Programovací jazyk Python tiež umožňuje vytvárať vlastné dátové typy pomocou príkazu `Class` ktoré sa označujú ako triedy.

Danej triede je možne definovať premenné ktoré môžu obsahovať nejakú hodnotu. Takéto premenné ktoré patria iba danej triede a sú s ňou úzko späté sa nazývajú atribútmi triedy. Triedy je tiež možné priraďovať ako atribúty ďalších tried. V triede je možné definovať aj jej vlastné funkcie. Funkcie ktorými disponuje daná trieda sa označujú ako metódy danej triedy. Metóda podobne ako funkcia môže vyžadovať rôzne vstupné hodnoty ktoré sa nazývajú parametre metódy. Tak tiež je možné využiť vlastnosti už existujúcej triedy čím vzniká závislosť nazývaná dedenie prípade viacnásobné dedenie.

Po tom čo je trieda definovaná je možné jej vlastnosti použiť ako náčrt pre vytvorenie objektu s jej vlastnosťami. Vytvorený objekt môže mať ľubovolné meno ktoré ešte nie je použité. Objekt tak disponuje atribútmi a metódami využitej triedy. Kód v ktorom sa pracuje s objektmi a ich metódami sa označuje ako objektovo orientovaný.

Okrem používania objektov a ich metód v kóde je tiež možné analyzovať tieto objekty a ich vlastnosti. Analýzou je myslené prezeranie atribútov, metód ktorými daný objekt disponuje a parametrov jednotlivých metód, atď.. Z dôvodu, že výsledkom tejto práce je prekladač je takáto analýza veľmi nápomocná. Získanie hodnoty atribútu objektu je možné napr. pomocou použitia daného objektu a následne meno atribútu alebo existujúcou vstavanou funkciou, ktorá dokáže vytiahnuť hodnotu atribútu pomenovaného objektu. Funkcia ktorá dokáže získať hodnotu atribútu na základe mena z objektu ku ktorému patrí je `getattr()`. Informácie o objekte je možné tiež získať pomocou funkcií z Python modulu inspect.

Disponuje funkciami ktoré umožňujú napr. preskúmanie obsahu triedy, získanie zdrojového kódu metódy, extrahovať a naformátovať zoznam argumentov pre funkciu [3]. Funkcia ktorá je dôležitá z tohto modulu a využívaná v tejto práci je `getfullargspec()`. Umožňuje získať zoznam parametrov ktoré skúmaná metóda vyžaduje. Zoznamom parametrov sú myslené atribúty ktoré poskytujú rôzne informácie o týchto parametroch metódy, napr. atribút `args` obsahuje zoznam mien parametrov v metóde alebo atribút `annotations` je slovník mapujúci názvy parametrov na ich anotácie t.j. vyžadované typy parametrov.

2.2.2 AST

AST je modul jazyka Python ktorý umožňuje spracovať zdrojový kód napísaný v jazyku Python a previesť ho na abstraktný syntaktický strom. Takýto strom je podobný tomu, ktorý vzniká ako výsledok syntaktickej analýzy. Celý program je rozobraný na elementy ktoré sa nazývajú uzly. Pričom niektoré uzly sú nadradené iným uzlom. Tieto uzly sú uložené v štruktúre nazývanej strom. Na prevod zo zdrojového kódu do abstraktného syntaktického stromu slúži funkcia `parse()`, ktorá prevod vykoná iba v prípade, že zdrojový

kód neobsahuje žiadne syntaktické chyby. Spätný prevod zo stromovej štruktúry do Python kódu umožňuje funkcia `unparse()`.

Modul AST umožňuje okrem samotného prevodu aj vyhľadávanie a navštevovanie jednotlivých uzlov pomocou triedy `NodeVisitor`. Jednotlivé uzly sú uložené v poliach na rôznych úrovniach podľa zanorenia. Čo umožňuje iterovať cez tieto uzly a prechádzať ich dáta. Jednotlivé uzly majú definované svoje vlastné typy podľa operácie ktorú daný uzol reprezentuje. Najdôležitejšie typy uzlov používané v tejto práci sú:

- `Assign` - priradenie napr. `variable = object.method(an="ac1")`,
- `Expr` - výraz napr. `object.method(an="ac1")`,
- `Call` - volaná funkcia alebo metóda niektorého objektu. Uzol je vždy pod uzlom napr. uzla `Expr` -> `object.method`. Samotná časť reprezentovaná uzlom `Call` je len `.method`,
- `Attribute` - atribút niektorého objektu. Uzol je vždy pod uzlom napr. uzla `Expr` -> `object.attribute`. Samotná časť reprezentovaná uzlom `Attribute` je len `.attribute`,
- `If` - podmienka napr. `if variable == value:`,
- `While` - cyklus `while` napr. `while True:`,
- `Compare` - porovnanie napr. `variable == value`,
- `Continue` - príkaz `continue`,
- `Name` - použitie premennej napr. `variable`,
- `Constant` - konštantná hodnota napr. `42`.

```
while True:
    obj.met(42, an='ac1')
    obj.met(var, an='ac2')
```

Výpis 2.1: Python kód

```
Module(
    body=[
        While(
            test=Constant(value=True),
            body=[
                Expr(
                    value=Call(
                        func=Attribute(
                            value=Name(id='obj', ctx=Load()),
                            attr='met',
                            ctx=Load()),
                        args=[
                            Constant(value=42)],
                        keywords=[
                            keyword(
                                arg='an',
                                value=Constant(value='ac1'))])),
                Expr(
                    value=Call(
                        func=Attribute(
                            value=Name(id='obj', ctx=Load()),
                            attr='met',
                            ctx=Load()),
                        args=[
                            Name(id='var', ctx=Load())],
                        keywords=[
                            keyword(
                                arg='an',
                                value=Constant(value='ac2'))]))],
            orelse=[])],
    type_ignores=[])
```

Výpis 2.2: Abstraktný syntaktický strom pre vyššie uvedený Python kód

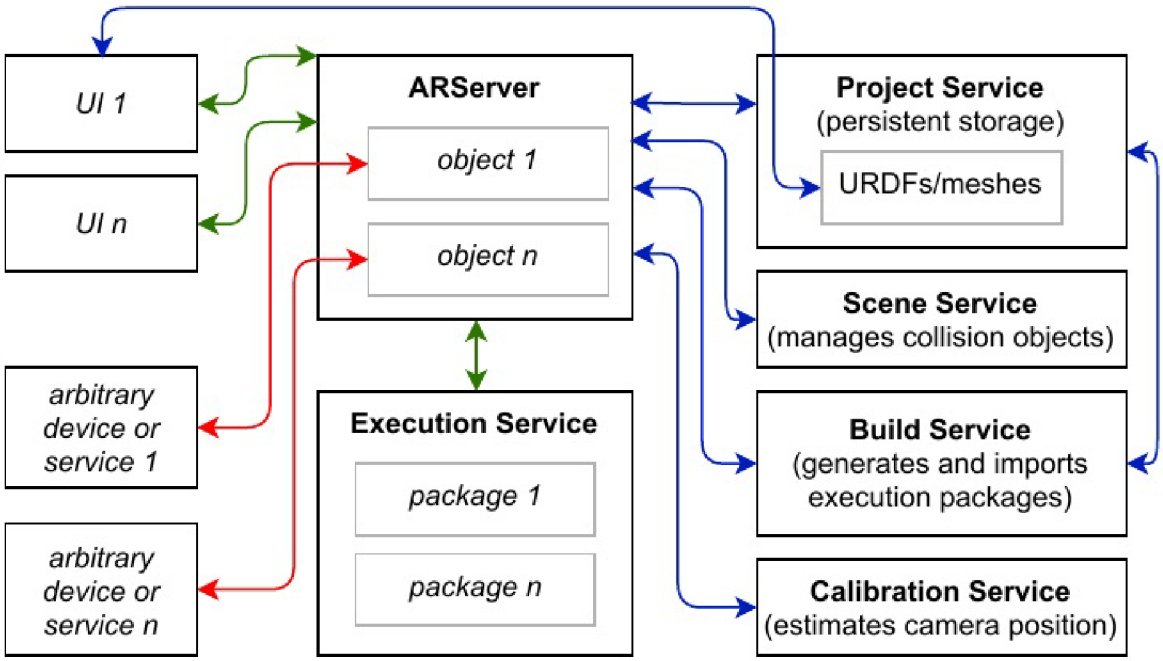
Kapitola 3

ARCOR2

Ide o aplikáciu ktorá vzniká v rámci projektu Test-it-off pre programovanie priemyselných robotov. Je vytvorená predovšetkým v jazyku Python. Jej výhodou je, že poskytuje grafické rozhranie ktoré je jednoduché na pochopenie a ovládanie. Vďaka tomu je tak možné robota naprogramovať prípadne preprogramovať aj užívateľmi ktorí nemusia byť experti na daný typ robota. Spoločnosť v ktorej sa takýto robot nachádza nemusí platiť odborníkov na nastavovanie robotov prípadne zamestnancom spoločnosti zjednoduší prácu.

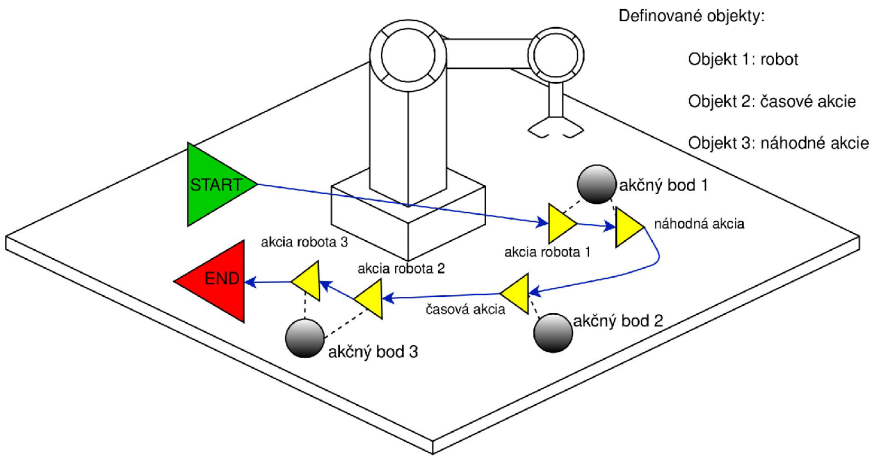
Architektúra tohto systému (obr. 3.1) je zložená zo sady nezávislých služieb (backend) a užívateľského rozhrania (frontend). Hlavnou službou systému je ARServer, ktorý funguje ako centrálny bod medzi užívateľským rozhraním a ostatnými službami ktoré sprostredkováva [6]. Komunikácia s týmito službami funguje na základe žiadostí (request). Z toho dôvodu existuje trieda `ARServer` ktorá disponuje metódou `call_rpc()` pomocou ktorej je možné komunikovať s jednotlivými službami. Pre priamu komunikáciu pomocou adresy url a štandartných metód ako sú PUT, POST, GET slúži funkcia `call()`. Obidve funkcionality využívajú vo finálnej časti testovania. Služba ktorá je v tejto práci rozšírená o spätný preklad je služba "Stavať" (ďalej len "Build").

Rozšírenie sa týka iba spätného prekladu z dôvodu, že služba už obsahuje funkcionalitu, ktorá umožňuje prevod z JSON reprezentácie do Python kódu. Tento prevod zaobstaráva funkcia `program_src()`. Na základe vstupov a výstupov tejto funkcie je možné odsledovať rôzne prípady ako má daný preklad vyzerať, čo je predovšetkým využité v sekcii testovania. Samotná služba umožňuje generovať a importovať tzv. exekučné balíčky, ktoré obsahujú informácie o tom ako sa má daný robot správať. Viac o exekučných balíkoch v podkapitole 3.4.



Obr. 3.1: ARCOR2 architektúra prevzaté z [6]

Kód naprogramovaného robota je interne reprezentovaný v JSON formáte, ktorý sa prevádza do jazyku Python. Tento prevod sa využíva ak je potreba program robota spustiť. Ide o dve štruktúry kotré sa označujú projekt a scéna (obr. 3.2). Preložená JSON reprezentácia do programovacieho jazyka Python sa označuje ako skript a súbor v ktorom sa vyskytuje je `script.py`. Keďže JSON formát môže obsahovať rôzne dáta, v nasledujúcich častiach sú priblížené a vysvetlené jednotlivé štruktúry v systéme ARCOR2. Pre lepšie pochopenie prevádzaného kódu, aj keď ide o program napísaný v jazyku Python, za zmienku stojí štruktúra kódu uložená v súbore `script.py`.

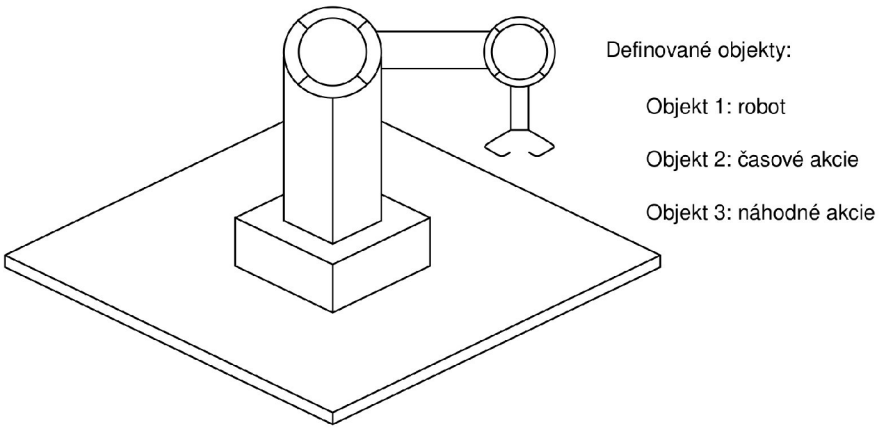


Obr. 3.2: Vizuálne reprezentovaný program robota

3.1 Scéna

Scéna definuje rozloženie pracovného priestoru s ktorým robot pracuje. Je to súbor objektov a priestorových vzťahov [6]. Skladá sa z nasledujúcich atribútov:

- `created` - dátum a čas vytvorenia,
- `modified` - dátum a čas poslednej úpravy,
- `name` - názov scény,
- `id` - jednoznačný identifikátor v rámci celého programu,
- `description` - slovný popis,
- `objects` - objekty s ich názvami a identifikátormi pod ktorými sa používajú v danom programe robota.



Obr. 3.3: Scéna

Scénu je možné použiť ako vstupný parameter do triedy `CachedScene`. Trieda disponuje dátami použitej scény a ponúka rôzne metódy pre prácu s dátami.

3.1.1 Objekt Scény

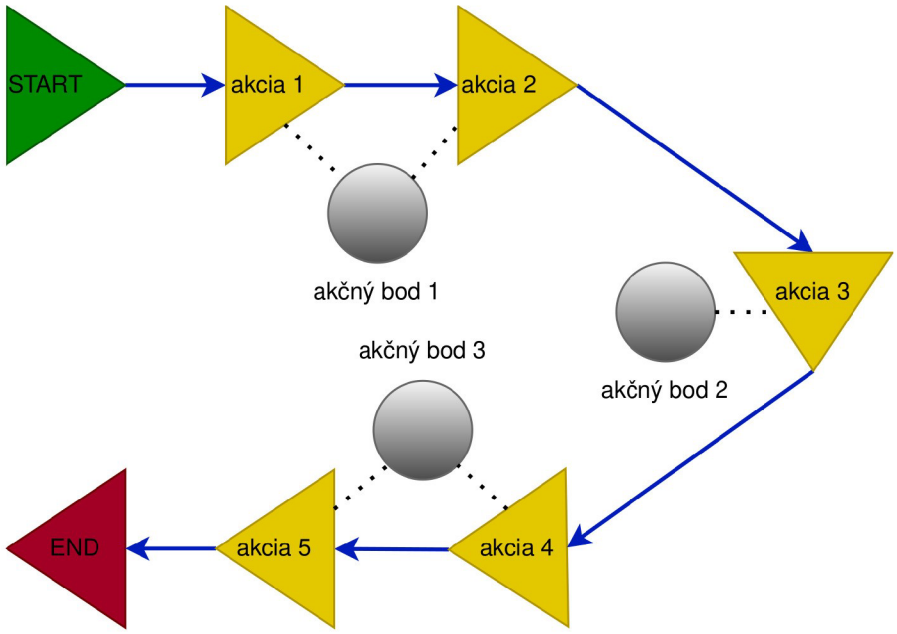
Ide o reprezentáciu ktorá poskytuje integráciu s konkrétnym typom objektu z reálneho sveta, akým je napr. určitý typ robota alebo virtuálny objekt ako "cloudové API". Je napísaný v jazyku Python vďaka čomu môže využívať viacnásobné dedenie s cieľom rozšírenia alebo zdieľania funkčnosti. Každý objekt má svoj vlastný identifikátor pod ktorým je definovaný a ktorý sa využíva pri volaní jeho metód. Metódy objektu so špeciálnou anotáciou sa môžu stať akciami ktoré sú priradené k akčným bodom [6].

3.2 Projekt

Projekt slúži ako úložisko príkazov pre program robota v JSON reprezentácii. Podobne ako scéna obsahuje informácie akými sú:

- `created` - dátum a čas vytvorenia,

- `modified` - dátum a čas poslednej úpravy,
- `name` - názov projektu,
- `id` - identifikátor projektu,
- `description` - slovný popis,
- `scene_id` - identifikátor scény s ktorou je úzko spätý,
- `has_logic` - atribút ktorý podáva informáciu o tom či projekt obsahuje logiku alebo nie,
- `parameters` - atribút ktorý obsahuje preddefinované premenné v ktorých sa môžu nachádzať hodnoty,
- `action_points` - akčné body,
- `logic` - logika.



Obr. 3.4: Projekt

Projekt je možné použiť ako vstupný parameter do triedy `CachedProject`. Trieda disponuje dátami použitého projektu a ponúka rôzne metódy pre prácu s dátami.

3.2.1 Akčné body

Ide o kontajner pre nastavenie orientácie, konfigurácie kĺbov robota a akcie [6]. Akčné body obsahujú dáta:

- `pose` - pozícia v 3D priestore,
- `orientations` - orientácia,

- `joints` - nastavenie kĺbov robota.

Tieto dáta je možné použiť ako parameter v metóde, ktorá by ich vyžadovala. Atribúty `orientations` a `joints` obsahujú veľa vlastných dát. V tejto práci sú použité nasledovné dáta:

- `id` - identifikátor orientácie alebo nastavenia kĺbov robota,
- `name` - meno orientácie alebo nastavenia kĺbov robota.

Ďalšie atribúty akčných bodov sú:

- `name` - meno akčného bodu,
- `id` - identifikátor akčného bodu,
- `actions` - zoznam akcií patriace k danému akčnému bodu.

Akčný bod je medzikrokom medzi projektom a samotnými akciami. Akčný bod tiež zhromažďuje akcie, ktoré dáva logický zmysel mať pri sebe, napr. ak má robot niečo zdvihnúť v nejakej polohe ktorú definuje daný akčný bod a následne sa volá metóda, ktorá nastavuje rýchlosť pohybu robota, je zmysluplné tieto volania metód dať k jednému akčnému bodu.

3.2.2 Akcie

Sú to kroky programu do ktorých je robot navádzaný pomocou logických prvkov. Ide o volanie metódy objektu s minimálne jedným parametrom. Akcia obsahuje niekoľko základných atribútov:

- `id` – identifikátor,
- `name` – jedinečný názov danej akcie v rámci celého programu. Je určený hodnotou `an` v skripte, ktorá je jedným z povinných parametrov volanej metódy napr. `objekt.metóda(an="akcia_číslo_1")`,
- `type` – samotný objekt a jeho metóda. Objekt je zapísaný pod jeho identifikátorom zo scény spolu s použitou metódou napr. `identifkátor_objektu/metóda`.

Akcia ďalej môže obsahovať ďalšie dva atribúty:

- `flows` - vyskytujú sa len pri priradení hodnoty do premennej. Označuje premennú do ktorej má byť vložená hodnota.
- `parameters` - ide o jednotlivé parametre, ktoré vyžaduje volaná metóda.

Parametre akcií

Každý parameter má svoje meno, hodnotu a typ. Meno parametra je odvodené od názvu parametra z pôvodnej metódy ktorá je napísaná v jazyku Python. Podobne aj typ akcie parametru je odvodený od typu parametru ktorý očakáva použitá metóda v jazyku Python. Zápis označenia typov parametru pre akciu nie je však totožný s obecným označením typov v jazyku Python. ARCOR2 má pre JSON reprezentáciu definované vlastné názvy pre jednotlivé typy parametrov v jazyku Python. Pričom hodnoty parametrov je potrebné pri prevode z JSON reprezentácie do Python kódu a späť, prevádzať špecifickým spôsobom

na základe ich typu. Tento prevod z časti umožňuje Python modul json, ale ARCOR2 pracuje s niektorými dátovými typmi inak ako je bežné a má tiež definované aj vlastné dátové typy. Z tohto dôvodu v ňom existuje prevádzacia trieda `ParameterPlugin`, ktorá po získaní typu hodnoty z Python alebo JSON reprezentácie je schopná previesť danú hodnotu do opačnej reprezentácie. Prípadne poskytnúť informácie o danej triede akými sú napr. označenie danej triedy v Python alebo JSON reprezentácii. Funkcie ktoré sú schopné získať túto triedu a ktoré sú využité v tejto práci, sú predovšetkým `plugin_from_type_name()` a `plugin_from_type()`. V ARCOR2 sa vyskytuje päť druhov parametrov:

- Konštanta
  Ide o klasickú statickú hodnotu akou je napr. číslo, reťazec, pravda alebo nepravda.

- Atribút triedy
  Ide o prípad kedy je použitý atribút triedy ako hodnota v metóde, napr. ak by trieda `MoveType` mala definovaný atribút `JUMP` ktorý predstavuje nejakú hodnotu, napr. reťazec `"jump"` je možné túto triedu s jej atribútom použiť ako parameter v metóde. Do hodnoty parameteru akcie by potom bola zapísaná hodnota `"jump"`.

- Hodnota akčného bodu
  V skripte ide o hodnotu triedy ktorá je väčšinou označovaná ako `aps`. Ako bolo spomenuté v oddiele 3.2.1 použitou hodnotou môže byť atribút tohto bodu ako je pozícia (`pose`), orientácia (`orientations`) a nastavenie kĺbov robota (`joints`). Rozdiel medzi hodnotou triedy a akčného bodu je v tom, že hodnota parametru akcie v prípade akčného bodu sa zapisuje ako:

  – identifikátor bodu - ak ide o atribút `pose`,

  – identifikátor orientácie - ak ide o atribút `orientations`,

  – identifikátor nastavenia kĺbov robota - ak ide o atribút `joints`,

- Parameter projektu
  Ide o preddefinovanú premennú z projektu ktorá je použitá ako hodnota v metóde. V skripte je možné nájsť definíciu premennej nad nekonečným cyklom. Viac o štruktúre kódu skriptu v podkapitole 3.3. V tomto prípade sa neodvádza typ parametra akcie na základe parametra použitej metódy. Ak ide o parameter projektu typ parametru sa označuje hodnotou `ActionParameter.TypeEnum.LINK`. Na rozdiel od hodnoty triedy sa nezapisuje do hodnoty parametra hodnota premennej ale sa zapisuje identifikátor parametra z projektu.

- Premenná
  Ide o premennú ktorá je definovaná počas behu programu. Typ takého parametra akcie je označovaný hodnotou `ActionParameter.TypeEnum.PROJECT_PARAMETER`. Do hodnoty parametra sa zapisuje identifikátor akcie v ktorej bola premenná deklarovaná spolu s hodnotou `/default/0`. Ak by bola premenná deklarovaná ako výstup akcie napr. s identifikátorom `id_ac_1`, parameter metódy v ktorej by bola použitá táto premenná by obsahoval hodnotu `id_ac_1/default/0`.

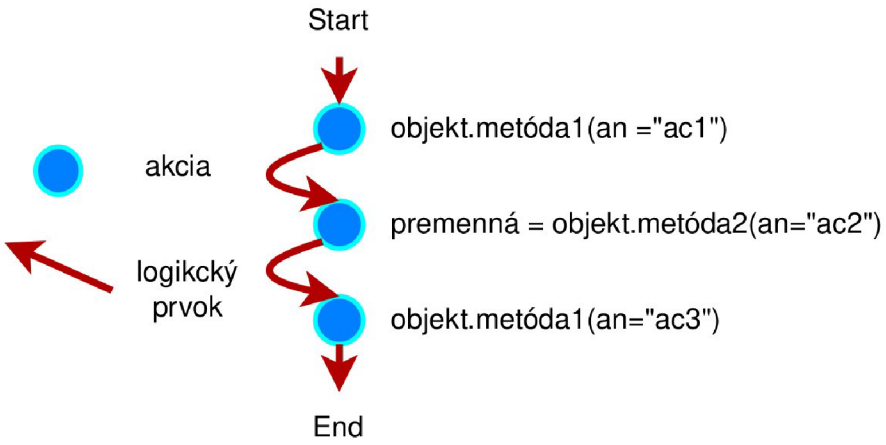
3.2.3 Logika

Logika je úzko spätá s akciami. Vzhľadom na to, že akcie sú jednotlivé kroky programu podľa ktorých má daný robot niečo spraviť, je potrebné poznať nejakú informáciu o tom

v akej postupnosti tieto kroky idú za sebou a za akých podmienok. Túto informáciu poskytuje logika. Je to zoznam logických prvkov ktoré na seba nadväzujú na základe identifikátorov akcií. Logické prvky obsahujú tri základné atribúty:

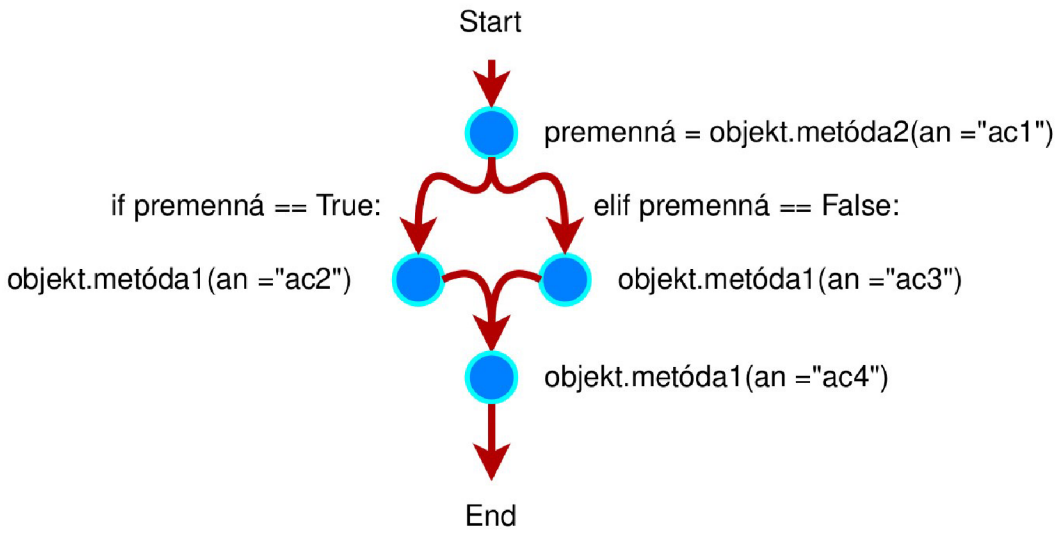
- `id` – identifikátor logického prvku ktorý je automaticky generovaný pri inicializácii,
- `start` – miesto odkiaľ logický prvok začína,
- `end` – miesto v ktorom logický prvok končí.

Miestom v ktorom logický prvok začína môže byť identifikátor akcie alebo hodnota `LogicItem.START`, ktorá reprezentuje špeciálnu štartovnú hodnotu. Túto štartovnú hodnotu obsahuje vždy len jeden logický prvok ktorý je štartovným logickým prvkom. Miestom v ktorom logický prvok končí môže byť opäť identifikátor akcie alebo hodnota `LogicItem.END`, ktorá reprezentuje špeciálnu konečnú hodnotu. Na rozdiel od štartovnej hodnoty, konečnú hodnotu môže mať viacero logických prvkov v rámci jedného programu. V prípade prázdneho programu by logika obsahovala jeden logický prvok ktorý začína v štartovnej hodnote a končí v konečnej hodnote.



Obr. 3.5: Schéma logiky bez podmienky

Logický prvok môže obsahovať ešte jeden atribút, ktorý sa používa ak ide o podmienky (`if/elif`). Nazýva sa `condition` a obsahuje dva atribúty `what` a `value`. Atribút `what` nesie ľavú časť podmienky t.j. meno premennej. Atribút `value` je pravá časť podmienky a v súčasnosti môže nadobúdať iba dve hodnoty pravdu a nepravdu. Vďaka týmto atribútom (`what` a `value`) je možné odkazovať sa z jednej akcie na dve rôzne prípadne aj viac akcií, podľa pravdivosti podmienky. Taktiež to platí opačne a to tak, že keď sa podmienka ukončuje môže sa viacero logických prvkov odkazovať na jednu konkrétnu akciu. Je celkom dôležité spomenúť, že atribút `condition` reprezentuje celú podmienku a je súčasťou logického prvku ktorý ide z jednej akcie (metódy) do druhej akcie (metódy). Samotný atribút `condition` reprezentuje len jednu podmienku. Z tohto dôvodu neexistuje situácia kedy nasledujú dve podmienky za sebou a súčasne nie je medzi nimi žiadne volanie metódy. Takýto prípad nemá reprezentáciu v ARCOR2. Väčšinou kód obsahuje príkaz `if` nasledovaný len jedným `elif` príkazom.



Obr. 3.6: Schéma logiky s podmienkou

3.3 Skript

Spojením dát zo scény a projektu, ktorý musí obsahovať logiku v JSON formáte je možné vytvoriť kód v jazyku Python pomocou vstavanej funkcionality ARCOR2. Samotný kód je silne objektovo orientovaný to znamená, že sa v ňom nenachádzajú žiadne volania funkcií. Každú operáciu je nutné vykonať pomocou metódy objektu, ktorý je importovaný do programu. Samotný skript má vždy rovnakú štruktúru:

- časť v ktorej sa zo súborov importujú triedy,
- definíciu funkcie `main` ktorá je zložená z dvoch častí:
  - miesta v ktorom sú definované objekty s ich identifikáciou zo scény, alebo prípadne premenné ktoré sa používajú ako parametre,
  - nekonečného cyklu (`while`), ktorý obsahuje hlavný kód pre riadenie robota vo forme podmienok a objektov s metódami,
- volanie definovanej funkcie `main`.

```python
from object_types.test import Test
from action_points import ActionPoints
from arcor2_runtime.resources import Resources
from arcor2_runtime.exceptions import print_exception

def main(res: Resources) -> None:
    aps = ActionPoints(res)
    test_name: Test = res.objects['obj_test']
    while True:
        bool_res = test_name.test(an='ac1')
        if bool_res == True:
            test_name.test(an='ac2')
        elif bool_res == False:
            test_name.test(an='ac3')
        test_name.test(an='ac4')

if __name__ == '__main__':
    try:
        with Resources() as res:
            main(res)
    except Exception as e:
        print_exception(e)
```

Výpis 3.1: script.py

Kód tela cyklu je jediná časť, ktorú je povolené upravovať. V prípade upravovania kódu iných častí môžu vznikať rôzne chyby. Príkazy ktoré sa môžu vyskytovať v časti cyklu sú:

- klasické volania metód objektov s možným priradením výsledku do premennej,
- podmienky ktorých telá môžu obsahovať ďalšie podmienky a volania metód. V takomto prípade musí platiť, že po podmienke nesmie nasledovať hneď ďalšia podmienka, ale musí byť medzi nimi aspoň jedna metóda.
- príkaz `continue` - ktorý vráti program na začiatok tohto cyklu. V logike je reprezentovaný ako predčasný koniec a to tak, že hodnota `Logic.END` je v mieste kde logický prvok končí.

3.4 Exekučný balík

Vstupnými a výstupnými dátami s ktorými služba Build pracuje sú exekučné balíky vytvorené aplikáciou ARCOR2 ktoré môžu byť následne pozmenené užívateľom.

Ide o samostatnú spustiteľnú snímku projektu, ktorá sa používa na otestovanie programu robota, alebo na vydanie programu do produkčného prostredia [5]. Obsahuje informácie akými sú:

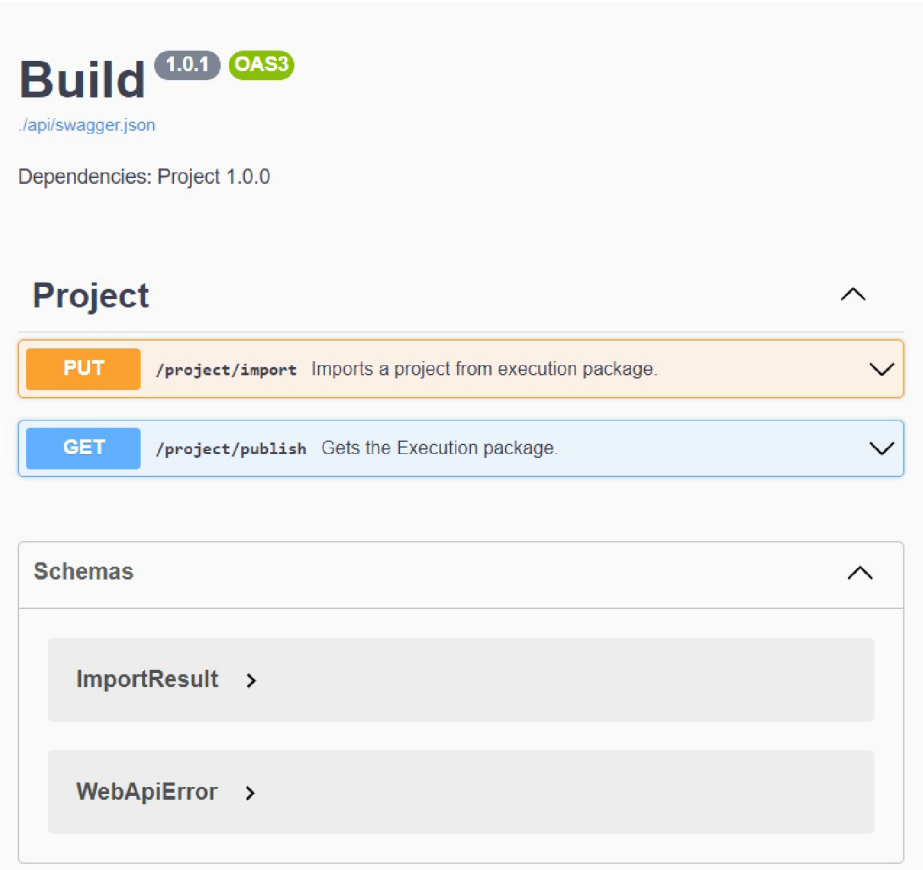
- `package.json` - obsahuje informácie ako názov, o tom kedy bol daný balík vytvorený a kedy bol balík naposledy spustený,
- `script.py` - súbor obsahujúci logiku programu vygenerovanú na základe projektu. Súbor môže byť v prípade potreby upravený ručne a potom spätne importovaný pomocou služby Build [6]. V prípade tejto práce je potrebné súbor skontrolovať a preložiť.
- `action_points.py` - akčné body ktoré sa používajú v danom programe a v uvedenom `script.py`. Súbor slúži na sprehľadnenie akčných bodov.

- priečinok `data/` - obsahuje `project.json`, `scene.json` a priečinok `models`, ktorý obsahuje JSON súbory odpovedajúce každému typu objektu spojeného s daným modelom,
- priečinok `objects_types/` - obsahuje Python súbory pre všetky typy objektov, ktoré sa používajú v danom programe (odkazované v scéne priamo, alebo nepriamo) pre každý typ objektu je jeden súbor.

Dôležité je spomenúť, že súbor `script.py` sa celý neupravuje a tiež sa celý neprevádza. Užívateľ má možnosť upravovať ho v samotnom rozhraní kompletne celý ale nie je to dovolené. Upravovaním nepovolených častí môžu vznikať rôzne chyby a použitia nepovolených operácií, ale ošetrenia takýchto prípadov neboli predmetom tejto práce a je možné ich spraviť v budúcom vývoji samotného systému ARCOR2. V podkapitole 3.3, je skript rozdelený na niekoľko častí. Časť ktorá sa môže upravovať je obsah nekonečného cyklu t.j. kód ktorý riadi správanie samotného robota prípadne viac robotov. Takými príkazmi napr. sú volanie metód, podmienky alebo príkaz `continue`. Keďže ide len o úpravu tejto časti je tak možné pracovať s časťami scény a projektu pri ktorých sa nemôže stať, že by mal pozmenený súbor script.py nejaký efekt napr. importované objekty, preddefinované premenné.

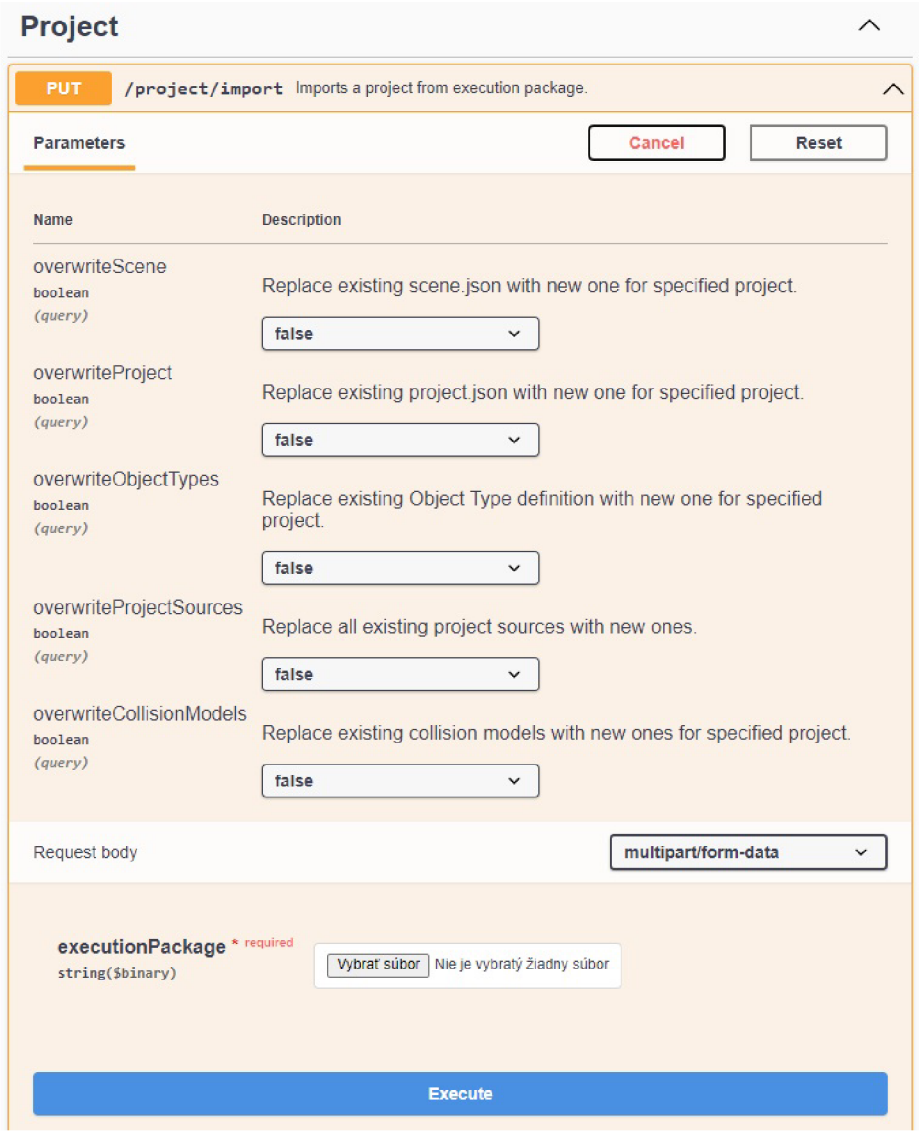
3.5 Build služba

Build služba disponuje dvoma zakladanými funkcionalitami: nahrávanie a vydávanie exekučných balíkov. Pre nahrávanie balíkov slúži funkcia `project_import()`. Pre vydávanie balíkov slúži funkcia `project_publish()`. Build služba úzko pracuje so službou “Projekt“ (ďalej len “Project“) a službou “Majetok“ (ďalej len “Asset“) ktoré slúžia ako úložisko dát.



Obr. 3.7: Služba Build

Pri nahrávaní exekučného balíka je možné zvoliť rôzne nastavenia na základe toho ako sa má pracovať s nahratými dátami.



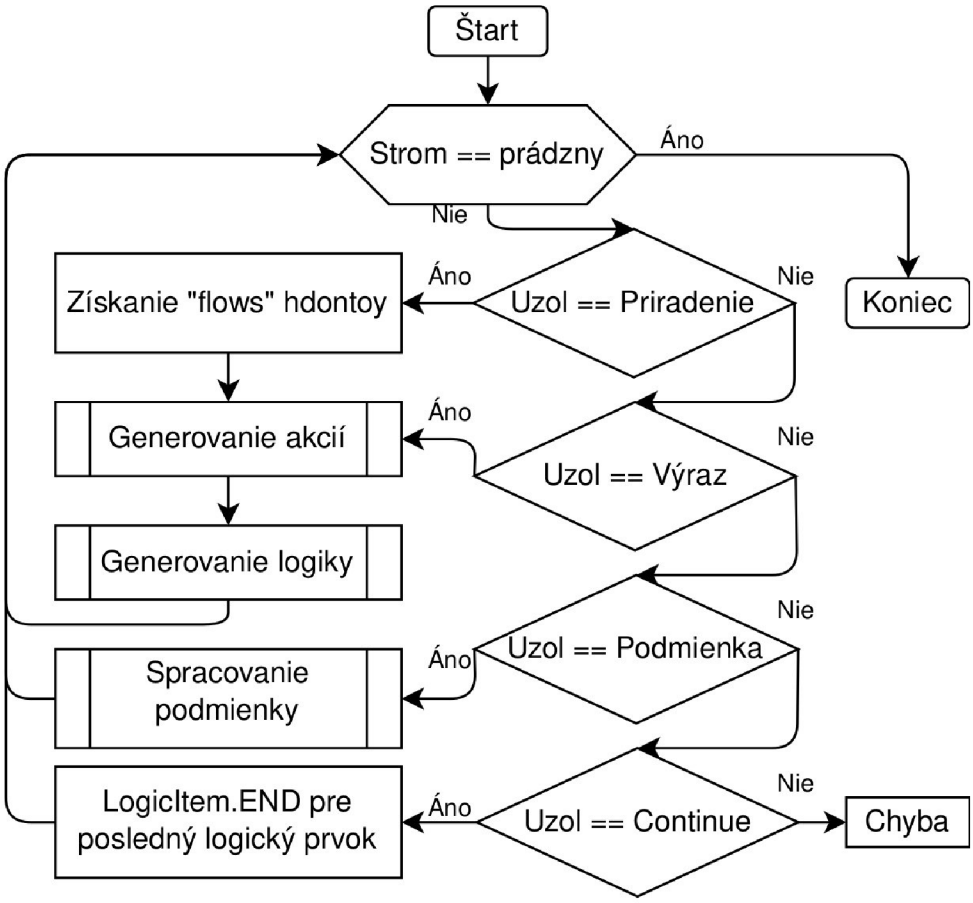
Obr. 3.8: Nastavenie importovania exkučného balíčka

Kapitola 4

Návrh

Keďže vstupom programu sú exekučné balíky, je potrebné ich na začiatku načítať a získať z nich všetky potrebné údaje. Rozšírenie systému ARCOR2 vo forme prekladača sa integruje priamo do systému a to konkrétne do funkcie ktorá už spracovanie vstupu zaobstaráva. Z uvedeného dôvodu nie je potrebné vytvárať spracovanie vstupu a všetky potrebné parametre je možné si priamo zaslať. Potrebnými parametrami sú: Projekt, Scéna, Skript vo forme reťazca a importované objekty do programu.

Ako je uvedené v kapitole 2, pred samotným generovaním kódu je potrebné urobiť sadu kontrol aby bolo zaistené, že zdroj ktorý je prekladaný bol správne napísaný a t.j. aby neobsahoval žiadne chyby. Časť syntaktickej kontroly skriptu sa vykoná pomocou modulu AST pre jazyk Python, ktorý zároveň prevedie skript do abstraktného syntaktického stromu. Tento prevod pokrýva len syntaktické chyby v jazyku Python, ale nepokrýva prípady výskytu operácií ktoré nemajú reprezentáciu v ARCOR2. Z tohto dôvodu sa tieto chyby odchytia počas generovania samotného kódu. Bližšie o chybách v podkapitole 4.5. Výstup vo forme abstraktného syntaktického stromu sa použije pre následné generovanie kódu. Jednotlivé uzly tohto stromu sú postupne prechádzané v cykle a na základe inštancie uzla sa rozhodne o ďalšom postupe generovaní akcií a logiky.



Obr. 4.1: Prechádzanie stromu

4.1 Spracovanie akcií

Akcie sa generujú v dvoch prípadoch:

- Volanie metódy bez návratovej hodnoty - zavolá sa funkcia pre vygenerovanie akcie z informácií ktoré obsahuje daný uzol. Informáciami ktoré sú potrebné pre generovanie sú:
  - meno ktoré sa nachádza v uzle v atribútu `keywords`,
  - typ ktorý je možné získať z objektov scény. Typ je potrebné nájsť a to tak, že sa porovná meno použitého objektu z uzlu s menami objektov v scéne. V prípade, že sa zhodujú mená, je možné použiť identifikátor tohto objektu zo scény a spojiť ho s názvom použitej metódy,
  - parametre metódy je možné získať z daného uzlu v ktorom sa metóda nachádza. Bližšie o parametroch v podkapitole 4.3.
- Volanie metódy s návratovou hodnotou - podobne ako v predchádzajúcom prípade dôjde k vygenerovaniu akcie s jej atribútmi. Rozdiel je v tom, že uzol stromu je inštancie `Assign`. Na základe tejto inštancie je možné povedať, že ide o priradenie a daná akcia bude obsahovať atribút `flows`. Tento atribút sa nachádza v danom uzle

pod atribútom `targets`. Po vygenerovaní akcie je dôležité si uložiť meno premennej ktorá vznikla a identifikátor akcie v ktorej k tomu prišlo, pričom pod menom premennej bude uložený identifikátor akcie. K tomuto držaniu informácií je vhodné použiť slovník. V prípade, že daná premenná bude použitá ako parameter bude možné ju nájsť a použiť identifikátor akcie ku ktorej patrí.

4.2 Spracovanie logických prvkov

Pri spracovaní-generovaní logických prvkov vzniká niekoľko problémov. Generovať akcie a súčasne k nim príslušné kompletné logické prvky nie je možné a to z dôvodu, že logické prvky potrebujú informáciu o tom aká je ich štartovná a konečná hodnota. Zároveň na jednu akciu sa môže odkazovať viacero logických prvkov ktoré opäť potrebujú vedieť ich štartovné a konečné pozície. Uvedený problém je možné riešiť viacerými spôsobmi napr.:

- Dvojitým priechodom stromu, pričom pri prvom priechode dôjde k uloženiu identifikátorov všetkých vygenerovaných akcií a pri druhom priechode je tak možné logickým prvkom poskytnúť informáciu o tom aká je ich štartovná a konečná pozícia.
- Spätným pozeraním sa. Keď dôjde k vygenerovaniu logického prvku ako konečná hodnota sa použije identifikátor najnovšej vygenerovanej akcie a jeho štartovný bod sa vyhľadá v zozname akcií ktoré sú vygenerované.
- Logickými prvkami ktoré nie sú úplne dokončené. Po vygenerovaní akcie sa vygeneruje aj nový logický prvok, ktorý si identifikátor akcie zapíše ako štartovný bod a predošlý logický prvok si zapíše identifikátor akcie ako konečný bod. Tým bude spôsobené, že vždy bude aspoň jeden logický prvok, ktorý nebude mať určený koniec. Z tohto dôvodu nie je potrebné vyhľadávať identifikátory akcií. Po skončení prechádzania stromom je nutné skontrolovať vytvorené logické prvky a doplniť cieľové body ktoré sú prázdne hodnotou `LogicItem.END`.

Z dôvodu úspornosti počtu operácií programu vo svojej práci je použité tretie riešenie - "logické prvky ktoré nie sú úplne dokončené". Počas generovania logických prvkov pre podmienky vznikajú nové dva problémy:

- Pri nájdení podmienky, logický prvok vie z ktorej akcie ide. V prípade ak pôjde o pokračovanie podmienky (`elif`), je potrebný identifikátor akcie ktorá bola pri prvej podmienke (`if`). Tento problém je možné vyriešiť uložením identifikátora akcie ktorá je pred podmienkou.
- Druhý problém je opačný a to keď prvá podmienka (`if`) končí a nasleduje za ňou druhá podmienka (`elif`). V tomto prípade nie je isté aký je identifikátor akcie ktorá nasleduje za podmienkami a na ktorú sa má logický prvok naviazať. Túto situáciu je možné riešiť tým, že v podmienkach sa opäť vynechá cieľové miesto pri posledných logických prvkoch. Po ukončení všetkých podmienok sa nastaví príznak na to, aby identifikátor prvej akcie, ktorá bude vygenerovaná sa použil na doplnenie cieľových miest logických prvkov ktorým chýba.

V prípade generovania akcií a logiky v tele podmienky je možné použiť funkciu, ktorá prechádzala strom a využiť jej vlastnosti pre generovanie. Týmto postupom je zabezpečené generovanie akcií a logických prvkov ktoré sú naviazané na podmienky pri postupnom

prechádzaní stromu. Pri príkaze `continue` nie je potrebné generovať akcie. Potrebné je priradiť poslednému logickému prvku do cieľového atribútu hodnotu `LogicItem.END`, aby bolo zrejmé že dochádza k ukončeniu v tomto bode. V prípade, že by za príkazom `continue` boli ďalšie príkazy je potrebné tento prípad ošetriť tak, aby neprišlo k prepisu cieľovej hodnoty z LogicItem.END na identifikátor novej akcie. Použitie príkazu `continue` v takomto prípade je ale skôr chybné, pretože nedáva logický zmysel dávať akékoľvek príkazy za `continue` lebo k nim nikdy nedôjde.

4.3 Spracovanie parametrov

V oddiele 3.2.2 je uvedené, že môže ísť až o päť rôznych typov hodnôt. Z tohto dôvodu je potrebné vytvoriť päť rôznych prístupov pre tieto hodnoty. Na začiatku spracovania parametrov sa určí o akú metódu ide a následne sa získajú názvy a typy jej parametrov a potom hodnoty, ktoré sú z daného uzlu. Taktiež získanie týchto informácií o metóde je dôležité pre kontrolu a to z toho dôvodu, že ten kto pozmenil skript, mohol zadať do metódy hodnoty nesprávneho typu a počtu. Pre takúto analýzu metódy je vhodné importovať použité objekty a ich metódy z programu robota (v skripte) priamo do programu prekladača a pracovať s nimi počas generovania JSON reprezentácie. Keďže objektov môže byť veľké množstvo, je veľmi užitočné uložiť si ich do dátovej štruktúry akou je slovník vo forme: názov objektu a daný objekt. Z toho dôvodu jednou z vyžadovaných hodnôt pre tento program je slovník importovaných objektov. V prípade, že bude nejaký objekt potrebný, bude možné tento objekt ihneď vyhľadať. Po nájdení objektu je potrebné získať informácie o metóde objektu. K uvedeným informáciám je možné dopracovať sa pomocou vstavanej Python funkcie `getattr()` a funkcie `getfullargspec()` z Python modulu inspect. Program tak môže nahliadnuť na parametre metódy a zapísať si meno parametru a jeho typ. Tento typ ktorý metóda očakáva je ale typovo Pythonový a z tohto dôvodu je potrebné typ previesť do JSON podoby v ARCOR2. K tomuto je možné použiť funkciu `plugin_from_type()` ktorá vracia triedu `ParameterPlugin` v ARCOR2. Z triedy `ParameterPlugin` je možné získať meno typu parametra v JSON reprezentácii. Hodnotu parametra je možné získať z daného uzlu metódy. Pričom pre každý z piatich typov parametrov platí iný prístup:

- Konštanta - na základe typu hodnoty previesť hodnotu z uzlu do hodnoty reprezentovanej v JSON.
- Atribút triedy - je možné opäť použiť funkciu `getattr()` a získať tak požadovanú hodnotu.
- Hodnota akčného bodu - vzhľadom na to, že akčných bodov môže byť ľubovolný počet je potrebné v uzle najskôr nájsť použitý bod podľa mena a následne zistiť či ide o hodnotu `postion`, `orientations` prípadne `joints`. Následne je možné nájsť identifikátor tejto hodnoty v zozname akčných bodov projektu.
- Premenná - na základe mena premennej z uzlu vyhľadať identifikátor akcie v slovníku, kde sú uložené identifikátory akcií pod názvom premennej a tento identifikátor akcie použiť ako hodnotu daného parametra.
- Preddefinovaná premenná v projekte - na základe mena premennej z uzlu vyhľadať jej meno v parametroch projektu a uložiť si jej identifikátor ako hodnotu parametra.

4.4 Integrácia do systému

K využitiu tejto funkcionality dochádza v prípade že, exekučný balík obsahuje kód skriptu ktorý neodpovedá JSON reprezentácii a užívateľ by tak chcel využiť prekladaciu funkciu. Z toho dôvodu na použitie tejto funkcie sa pridá možnosť aktualizácie projektu na základe zmien v skripte do služby Build.

4.5 Chybné vstupy

Na základe vyššie uvedeného, syntax skriptu v jazyku Python sa overení pomocou AST. V niektorých prípadoch nie je overenie syntaxe pomocou AST postačujúce. Ide o prípad keď užívateľ zadá operáciu ktorá nie je podporovaná v systéme ARCOR2, napr. matematické operácie, vnorenie volaní metód a funkcií do seba, alebo použitie viacerých podmienok za sebou bez toho aby bolo medzi nimi volanie metódy. Miesta kde je možné detektovať tieto prípady sú pri prechádzaní inštancií volania metód, hľadaní hodnôt v podmienkach a pri prechádzaní inštancií parametrov v metódach. Všetky tieto situácie sú podobné v tom, že hľadajú inštanciu. Stačí pridať prípad kedy hľadaná hodnota sa nezhoduje ani s jednou hľadanou hodnotu a nastaviť prípad ako chybný. Tým sú ošetrené všetky nepodporované typy operácií. Okrem syntaktických chýb môže prísť ešte k inému druhu chýb ktorými sú sémantické chyby. Ide o prípady ako sú napr. použitia nedefinovaného objektu, metódy a premennej, atď.. Tieto prípady je možné ošetriť pridaním kontroly či použitý objekt, metóda alebo premenná existuje. V prípade chybných parametrov metódy sa porovná typ vloženej hodnoty do metódy s očakávaným typom.

Kapitola 5

Implementácia

Výsledný prekladač je implementovaný formou funkcie `python_to_json()`. Uvedená funkcia vyžaduje štyri základné parametre: projekt, scénu, skript vo forme reťazca a slovník objektov. Všetky tieto parametre sa používajú vo väčšine funkcií v tomto programe. Na začiatku programu sa vytvorí abstraktný syntaktický strom zo skriptu tým následne príde ku kontrole syntaxe vstupného programu. Ak vytvorenie stromu prebehlo úspešne je potrebné navštíviť uzol `while`, ktorým sa bude postupne prechádzať. K navštíveniu uzla bola implementovaná pomocná funkcia `find_While()` ktorá vráti potrebný uzol. Časť kódu tejto funkcie je prevzatý z funkcie `find_asserts()` ktorá sa v systéme nachádzala ešte pred zadaním práce. Nasleduje vyčistenie akcií a logiky pôvodného projektu, aby uvolnili miesto novo-vygenerovaným akciám a logickým prvkom. Následne je vytvorený slovník `variables`, ktorý uchováva premenné vytvorené počas behu programu robota. Pod názvom premennej je uložený identifikátor akcie, z ktorej bola priradená hodnota do premennej. Scéna sa prevedie do triedy `CachedScene`. Takto pripravené dáta je možné poslať do funkcie `evaluate_nodes()`.

5.1 Spracovanie stromu

Samotná funkcia `evaluate_nodes()` v cykle prechádza jednotlivé uzly zvyšku stromu, ktoré jej boli zaslané. Funkcia obsahuje dve premenné ktoré ovplyvňujú generovanie logiky. Premenná `ac_id`, ktorá si uchováva identifikátor poslednej vygenerovanej akcie a premenná `after_if`, ktorá indikuje či posledný navštívený uzol nebol inštanciou podmienky (`if`). Uzol ktorý sa aktuálne analyzuje je uložený v premennej `node` a na základe jej inštancií sa následne generujú akcie a logické prvky. Ide o tri podmienky s jednotlivými inštanciami:

- `Expr` (výraz) alebo `Assign` (priradenie)
  V prípade inštancie priradenia, do premennej `flows` sa uloží zoznam priradení ktoré patria k metóde. Tento zoznam sa nadobudne z atribútu `targets` daného uzlu. Následne sa uzol zníži na úroveň `Call` pomocou funkcie `find_Call()` a nasleduje vygenerovanie akcie na základe zvyšných hodnôt z daného uzlu pomocou funkcie `gen_actions()`. Časť kódu funkcie `find_Call()` bol prevzatý z funkcie `find_function()` ktorá bola v systéme pred zadaním bakalárskej práce. Po jej vygenerovaní príde k uloženiu identifikátora akcie do premennej `ac_id`, ktorý sa následne použije pre vygenerovanie logického prvku. Ak premenná `after_if` obsahuje nepravdivú hodnotu dochádza k obyčajnému generovaniu logiky pomocou funkcie `gen_logic()`. V prípade ak obsahuje pravdivostnú hodnotu, posledný uzol je inštancie `If` a vygenerované lo-

gické prvky obsahujú prázdne miesta ktoré je potrebné doplniť. Z toho dôvodu sa použije funkcia `gen_logic_after_if()`, ktorá doplní identifikátor poslednej vygenerovanej akcie na prázdne miesta logických prvkov a následne vygeneruje nový logický prvok pre danú akciu. Po vygenerovaní nastaví premennú `after_if` na nepravdivú hodnotu.

- `If` (podmienka)
  V prípade podmienky dôjde k použitiu funkcie `evaluate_if()`, ktorá spracúva podmienky. Po jej skončení sa premenná `after_if` nastaví na pravdivostnú hodnotu. V prípade, že by premenná `after_if` obsahovala pravidvostnú hodnotu ešte pred volaním funkcie `evaluate_if()` program skončí s chybou z dôvodu, že ide o nepodporovanú kombináciu podmienky `if` a `elif`.

- `Continue`
  Pri tejto inštancii príde k ukončeniu logického prvku nahraním hodnoty `LogicItem.END` do posledného logického prvku. Po ukončení logického prvku nasleduje príkaz `break` ktorý zamedzí ďalšiemu generovaniu akcií a logických prvkov na danej úrovni stromu.

V prípade, že uzol nespadá ani do jednej podmienky ide o nepodporovanú operáciu a program končí s chybou.

5.2 Vyhodnotenie podmienok

Spracovanie a vyhodnotenie podmienok sa nachádza vo funkcii `evaluate_If()`. Na začiatku sa určí či v podmienke bolo použité porovnanie. Ak by podmienka nebola porovnávacieho typu program končí s chybou, pretože podmienka nemôže byt iného formátu. Následne sa pomocou `find_Compare()` navštívi uzol podmienky z ktorého sa budú spracovávať jednotlivé hodnoty podmienky. Časť kódu funkcie `find_Compare()` je prevzatý z funkcie `find_function()`.

Následne sa nadobudne ľavá časť podmienky z uzlu, pomocou atribútu `left`, ktorou je meno premennej v podmienke. Meno sa použije k tomu aby sa našiel identifikátor akcie v slovníku `variables` ku ktorej premenná patrí. V prípade, že sa premenná nenájde ide o použitie nedeklarovanej premennej a program situáciu vyhodnotí ako chybu. Ak sa premenná nájde uloží sa jej identifikátor do premennej `what`. Následne sa spracuje pravá časť podmienky. Najskôr sa určí typ hodnoty z uzlu, na pozícii `comparators[0].value` pomocou funkcie, `type()`. Tento typ sa použije vo funkcii `plugin_from_type()`, čo umožní prevedie akejkoľvek hodnoty z pravej časti podmienky do typu ARCOR2. Prevedená hodnota sa uloží do premennej `value`.

Nasleduje generovanie logiky pre danú podmienku. Ak je funkcia `evaluate_if()` volaná z `evaluate_nodes()`, príde k uloženiu posledného identifikátora akcie, ktoré sa nachádza v zozname logických prvkov ako štartovný bod posledného prvku. Vďaka tomu ako sú generované logické prvky nie je potrebné pri prvom volaní funkcie žiadne generovanie logického prvku. V prípade, že táto funkcia bola volaná sama sebou, identifikátor akcie pred samotnou podmienkou je uložený a je možné vygenerovať nový logický prvok so štartom v tomto identifikátore. Nadobudnuté hodnoty v premenných `what` a `value` sa priradia poslednému logickému prvku do atribútu `condition`.

Na záver je potrebné vygenerovať akcie a logiku pre telo podmienky. To sa vykoná zavolaním funkcie `evaluate_nodes()` s daným uzlom podmienky. V prípade, že by

za podmienkou nasledovala ďalšia podmienka (`elif`) je možné ju detektovať pomocou atribútu `orelse` v danom uzle. Tak sa postupne prechádzajú nadväzujúce podmienky v cykle, pričom funkcia volá sama seba vďaka čomu sú pokryté všetky prípady kombinácií podmienok.

5.3 Generovanie akcií

Generovanie akcií pozostáva z nadobúdania informácií o jednotlivých atribútoch akcií ktorými sú: meno, typ, výstup akcie a parametre akcie (prezeranej metódy). Následne sa tieto informácie použijú pre vygenerovanie akcie.

Meno sa získa z daného uzlu pomocou sady atribútov uzla ktoré odpovedajú umiestneniu názvu akcie. V prípade, že uzol neobsahuje celú sadu atribútov program skončí s chybou pretože uzol neobsahuje kľúčové meno akcie. V prípade, že uzol obsahuje sadu atribútov uloží sa meno akcie. Po uložení mena dôjde k vyhľadaniu tohto mena vo vygenerovaných akciách projektu. V prípade nájdenia tohto mena program opäť končí s chybou, pretože sa jedná o opätovné použitie mena akcie.

Ak by do funkcie bola zaslaná hodnota `flows` ktorá reprezentuje výstup metódy, bude táto hodnota použitá ako výstup akcie.

Pre typ akcie bola implementovaná pomocná metóda `get_object_by_name()` triede `CachedScene`. Metóda cez cyklus hľadá v scéne identifikátor objektu na základe mena objektu a volanej metódy. Po nájdení mena objektu v scéne príde k zámene mena objektu za identifikátor objektu a k zámene bodky za lomítko, z dôvodu ARCOR2 reprezentácie. Po nájdení a zámene príde k návratu tejto hodnoty. Ak by sa daný objekt nenašiel metóda vyhodnotí situáciu ako chybnú pretože ide o použitie neexistujúceho objektu. Meno objektu a názov metódy sa nadobudne ešte pred volaním tejto funkcie z atribútu uzlu `func` a uloží sa do premennej `object_method` ktorá je zaslaná do tejto metódy.

Pre parametre bola implementovaná funkcia `get_parameters()` ktorá vracia zoznam všetkých parametrov z metódy. Pred zavolaním samotnej funkcie príde k oddeleniu mena objektu a metódy z premennej `object_method`. Po ich oddelení sa získajú informácie o danej metóde a to tak, že zo slovníka ktorý obsahuje importované objekty, sa nadobudne daný objekt na základe mena. Následne sa nadobudne metóda pomocou vstavanej funkcie `getattr()`, ktorá z daného objektu a mena volanej metódy získa danú metódu. Na nadobudnutie informácií o parametroch tejto metódy sa použije funkcia `getfullargspec()` z modulu `inspect` do ktorej sa vloží daná metóda. Takto prezretá metóda sa odošle do zmienenej funkcie `get_parameters()`. Po nadobudnutí zoznamu o parametroch metódy sa všetky tieto hodnoty spoja a vytvoria novú akciu ktorá sa pridá k poslednému známemu akčnému bodu. V prípade, že akcia má výstupnú hodnotu, uloží sa názov premennej do ktorej je hodnota priradená spolu s identifikátorom akcie do slovníka `variables`. Následne funkcia vráti identifikátor novej akcie, posledný známy akčný bod a slovník deklarovaných premenných.

5.4 Generovanie parametrov

Na začiatku funkcie `get_parameters()` príde k inicializácii zoznamu `parameters` do ktorého sa postupne ukladajú jednotlivé parametre vygenerované pre akciu. Následne sa v cykle prechádzajú jednotlivé parametre metódy, ktoré sú z uzlu metódy. Na začiatku cyklu dôjde k uloženiu mena parametra. Meno parametra sa získa zo zoznamu parametrov

prezeranej metódy na pozícii ktorá odpovedá číslu parametra v uzle. Získané meno parametra sa následne uloží do premennej `param_name`. Potom na základe inštancie uzla sa rozhodne o aký typ parametra ide a ako sa z neho nadobudnú potrebné informácie:

- `Constant` - konštantná hodnota
  Na začiatku spracovania príde k získaniu typu daného parametra v metóde, pomocou funkcie `plugin_from_type()`. Následne sa porovná tento typ s typom hodnoty v uzle. Ak by neboli zhodné ide o chybu. Ak sú zhodné príde k uloženiu typu do premennej `param_type`. Hodnota uzla sa prevedie do JSON podoby pomocou metódy `value_to_json()` a uloží sa do premennej `param_value`.

- `Attribute` - atribút triedy
  Na začiatku sa opäť získa typ parametra metódy. Vzhľadom na to, že akčný bod spadá pod inštanciu atribútu triedy je potrebné následne overiť či ide o hodnotu akčného bodu alebo hodnotu nejakej triedy.

  – Ak by sa daná trieda volala `aps` je zrejmé, že ide o akčný bod. Následne je potrebné nájsť tento bod v zozname bodov projektu. Hľadanie je vykonané tak, že sa v cykle porovná meno bodu z uzlu s názvami bodov projektu. Ak by sa nenašiel bod alebo jeho hodnota program vyhodnotí situáciu ako chybnú. Ak sa nájde je potrebné zistiť o ktorú z troch hodnôt ide.
    Ak ide o tri po sebe nasledujúce inštancie typu `Attribute` je zrejmé, že ide o hodnotu `orientations` alebo `joints`, pretože ich zápis je dlhší oproti zápisu hodnoty `postion`. Následne sa zistí z druhého atribútu uzlu či ide o hodnotu `orientations` alebo `joints`. Na základe určenej hodnoty sa začne v cykle prezerať atribút akčného bodu projektu, pričom sa hľadá použité meno hodnoty z prvého atribútu uzlu. Po jeho nájdení sa uloží identifikátor hodnoty tohto bodu do premennej `param_value`.
    Ak prvá podmienka nie je splnená musí ísť o dve po sebe nasledujúce inštancie typu `Attribute` a to z dôvodu porovnania mena bodu z uzla s menami bodov projektu. Ak sa tento názov zhoduje s menom bodu z projektu jeho identifikátor sa uloží do premennej `param_value`.
  – Ak sa meno triedy nezhoduje s menom triedy akčných bodov, je potrebné nájsť túto triedu a jej hodnotu. Zoberie sa daná metóda a na základe mena parametru v `param_name` sa nájde daný parameter v zozname `annotations` prezeranej metódy. Pomocou funkcie `getattr()` sa nadobudne objekt parametra. Následne je možné určiť hodnotu tohto objektu pomocou atribútu `value`, ktorá sa prevedie do JSON podoby pomocou `value_to_json()`. Hodnota sa opäť uloží do premennej `param_value`.

- `Name` - premenná
  Na začiatku príde k hľadaniu mena premennej v slovníku uložených premenných. V prípade, že sa premenná nájde uloží sa typ hodnoty ako `ActionParameter.TypeEnum.LINK` a identifikátor akcie kde bola premenná deklarovaná sa prevedie do JSON typu. V prípade, že sa premenná nenájde príde k prehľadávaniu parametrov projektu na základe mena premennej. Ak sa premenná nájde, typ hodnoty sa uloží ako `ActionParameter.TypeEnum.PROJECT_PARAMETER` a uloží sa identifikátor odpovedajúcej premennej v JSON formáte. V prípade že sa premenná nenájde ani v parametroch projektu, ide o použitie nedeklarovanej premennej.

Na konci cyklu sa vždy nadobudnuté hodnoty z premenných `param_name`, `param_type`, `param_value` spoja a vytvoria jeden parameter ktorý sa pridá do zoznamu parametrov `parameters`. Po skončení prechádzania parametrov metódy funkcia vráti tento zoznam.

5.5 Generovanie logiky

Samotné generovanie logických prvkov je zložené z troch funkcií:

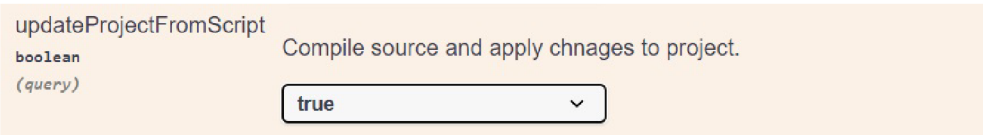
- `gen_logic()` - generovanie logiky pre obyčajné volanie metód.
  Poslednému logickému prvku sa priradí cieľové miesto ako identifikátor akcie, ktorá bola poslaná do funkcie. Následne sa vytvorí nový logický prvok pomocou triedy `LogicItem` s tým, že jeho štartovný bod začína opäť v identifikátore akcie ktorá bola zaslaná. Následne sa nový logický prvok vloží do zoznamu logických prvkov projektu.

- `gen_logic_for_if()` - generovanie logiky pre podmienku.
  Funkcia je takmer totožná s funkciou `gen_logic()`. Rozdiel je v tom, že nepriraďuje identifikátor akcie ako konečný bod. Vytvorí len nový logický prvok a do štartovného bodu vloží identifikátor danej akcie.

- `gen_logic_after_if()` - generovanie logiky po podmienke.
  Po skončení podmienky je potrebné prejsť všetky logické prvky a doplniť identifikátor akcie ako konečný bod logickým prvkom ktoré majú konečný bod prázdny. To sa uskutoční pomocou cyklu kde sa hľadajú logické prvky s prázdnym koncom. Po ich nájdení, sú doplnené konečné body hodnotou identifikátora akcie. Po skončení tohto dopĺňania koncov sa opäť ako vo funkcii `gen_logic()` vytvorí nový logický prvok so štartom v danej akcii.

5.6 Integrácia

Integrovanie tejto funkcie bolo založené na pridaní dát do žiadosti (request) vo funkcii `project_import()`. Pridané dáta informujú o tom či má dôjsť k prepísaniu projektu na základe zmien v kóde skriptu t.j. použitie prekladacej funkcie `python_to_json()`. V prípade, že by bola použitá prekladacia funkcia pri importovaní balíka, ale skript by neobsahoval žiadne zmeny voči projektu, príde k preloženiu skriptu do aktuálneho projektu. Pričom tento prepis by nemal žiaden efekt na výslednú reprezentáciu dát a to z toho dôvodu, že ide o tú istú reprezentáciu ako pred preložením skriptu.



Obr. 5.1: Nastavenie aktualizovania projektu na základe zmien v skripte

5.7 Generovanie skriptu

Z dôvodu predchádzania možných chýb, ktoré by užívateľ mohol vytvoriť upravením iných častí ako `while` v skripte, bol pridaný varovný komentár do skriptu pri jeho generovaní

funkciou `program_src()`. Varovný komentár informuje užívateľa o tom, že nie je povolené upravovať iné časti ako telo cyklu.

Kapitola 6

Testovanie

Testovanie sa uskutočnilo v niekoľkých fázach. Prvá časť testovania pozostávala z testovania jednotlivých pomocných funkcií programu (ďalej len "jednotkové testy"). Následne sa testovala samotná prekladacia funkcia `python_to_json()` ako celok. Na záver sa uskutočnilo testovanie zaintegrovania tejto funkcie do systému vo funkcii `project_import()`. Všetky testy boli z veľkej časti zložené z vytvárania projektu, scény a reťazca t.j. z častí skriptu ktoré mali byť preložené. Z dôvodu zaistenia správnosti testov, boli časti testov overené tým, že boli zadané ako vstupné parametre do funkcie `program_src()`.

Funkcia `program_src()` bola v systéme ARCOR2 vytvorená pred zadaním témy bakalárskej práce. Z tohto dôvodu výstup tejto funkcie som považoval za vzorový. Na základe výstupu funkcie `program_src()` boli vytvorené jednotlivé časti testov.

6.1 Jednotkové testy

Jednotkové testy boli najväčšou testovacou sadou v tejto práci. Testy pokrývajú rôzne chybové a správne stavy. Každý test predstavoval samostatný prípad ktorý by mohol nastať počas priebehu programu. Testy sú rozdelené do troch typov: bezchybné prípady, testy so sémantickými chybami a nepodporované operácie ktoré by sa mohli pokúsiť užívatelia použiť.

6.2 Prekladové testy

Prekladové testy sú zamerané na rôzne komplexné prípady použitia. Vytváranie testov bolo systematické t.j. od najjednoduchších prípadov po zložitejšie prípady. V prekladových testoch sa nenachádzajú chybové stavy a to z toho dôvodu, že k ich otestovaniu prišlo počas jednotkových testov. Na začiatku súboru sú definované reťazce, ktoré sa v skripte vyskytujú najčastejšie a sú rovnaké pre každý test:

- `head` - import objektov do programu,
- `main` - definície premenných a deklarácie objektov,
- `call_main` - volanie funkcie `main`.

Po definovaní reťazcov v súbore nasledujú definície použitých testovacích tried. Definície sú nasledovné:

- `Test` - obsahuje metódy, ktoré majú rôzne vstupné parametre, tak aby bolo možné otestovať všetky prípady vstupov,
- `TestEnum` - slúži ako testovacia trieda pre vstupný parameter do metódy.

Každý test pozostáva z vytvorenia:

- projektu ktorý sa skladá z logických prvkov a akcií,
- scény a definície jej objektov,
- odpovedajúceho skriptu v reťazcovom formáte. Skript je zložený zo štyroch častí ktorými sú `head`, `main`, program robota odpovedajúceho JSON reprezentácii a `call_main`.

Po vytvorení testovacieho prípadu nasleduje volanie funkcie `check_python_to_json()`, ktorá overí zhodu jednotlivých častí medzi pôvodným a preloženým projektom.

6.2.1 Overenie zhody

Overenie zhody medzi pôvodným a preloženým projektom bolo implementované vo funkcii `check_python_to_json()`.

Táto funkcia vychádza z myšlienky, že ak v preloženom projekte existuje logický prvok ktorý začína "v hodnote `LogicItem.START` alebo v identifikátore nejakej akcie" a súčasne končí "v hodnote `LogicItem.END` alebo v identifikátore nejakej akcie", tak takýto prvok musí existovať aj v pôvodnom projekte. Vzhľadom na uvedené súčasne platí, že ak bodmi sú identifikátory akcií, akcie sa musia nachádzať nielen v pôvodnom projekte, ale aj v preloženom projekte. Pričom porovnávané akcie musia mať rovnaké všetky atribúty okrem identifikátora.

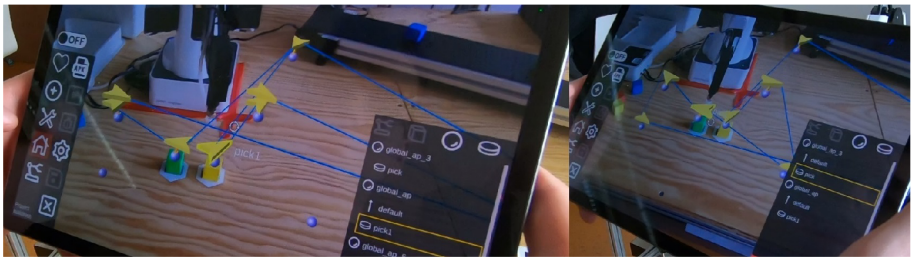
Na začiatku overovacieho procesu sa vytvorí kópia projektu pomocou funkcie `deepcopy()` z Python modulu `copy`. Následne sa nahrá projekt, scéna, skript vo forme reťazca a testovacia trieda `TEST` do funkcie `pyhton_to_json()` ktorá vráti preložený projekt späť do overovacej funkcie. Po preložení sa prevedie pôvodný aj preložený projekt na triedu `CachedProject`. Následne sa skontroluje či pôvodný projekt a preložený projekt obsahujú rovnaké počty logických prvkov a akcií. Po kontrole počtu prvkov projektov sa začnú prechádzať logické prvky preloženého projektu v cykle.

Z logického prvku upraveného projektu sa zoberie štartovný bod logického prvku. Ak štartovným bodom je identifikátor akcie, akcia sa začne hľadať v preloženom projekte pomocou funkcie `find_action()`. Funkcia `find_action()` využíva metódu projektu `get_by_id()` k hľadaniu akcie. Po nájdení akcie v preloženom projekte sa vezme jej meno na základe ktorého sa hľadá v pôvodnom projekte pomocou implementovanej metódy `action_from_name()`. Po nájdení akcie v oboch projektoch sa porovnajú všetky jej atribúty okrem identifikátora. Tento proces hľadania a porovnávania je opakovaný aj pre koncový bod logického prvku. V prípade, že štartovný bod začína v hodnote `LogicItem.START`, alebo cieľový bod končí v hodnote `LogicItem.END` tento poznatok je uložený pre ďalší proces overenia. Posledná časť overenia je na základe vyššie nadobudnutých informácií, založená na hľadaní logického prvku v pôvodnom projekte.Toto hľadanie sa uskutoční pomocou implementovanej metódy `find_logic_start_end()`. Po nájdení logického prvku sa porovná atribút `condition` z pôvodného projektu s atribútom preloženého projektu. V prípade, ak sa všetky tieto informácie zhodujú a súčasne prišlo aj k overeniu akcií ktoré prvok obsahoval dá sa s určitosťou tvrdiť, že ide o rovnaký logický prvok a teda preklad prebehol správne.

6.3 Integračné testy

Overenie správneho zaintegrovania výsledného programu do aplikácie ARCOR2 sa skladalo z dvoch častí:

- Automatický integračný test.
  ARCOR2 má sadu testov ktoré testujú celkovú funkcionalitu služby ARServer. Pri spustení testov príde k spusteniu jednotlivých služieb ako podprocesov. Z toho dôvodu bol integračný test pridaný do tejto sady testov a využíva spustené služby k testovaniu pridanej funkcionality. Automatický integračný test pozostáva z vytvorenia exekučného balíka ktorý je uložený v dočasnom úložisku. Pre toto vytvorenie balíka bola implementovaná funkcia `create_test_package()`. Kód funkcie bol z veľkej časti prevzatý z funkcie `_publish()` ktorá je súčasťou služby Build pri vydávaní balíkov. Balík obsahuje nasledovné časti: scénu, projekt, skript ktorý má pozemné dáta voči projektu, objekty použité v skripte, akčné body použité v skripte a súbor `package.json`. Balík je následne zaslaný na url adresu služby Build s nastavením aktualizovania projektu na základe zmien v skripte. Toto zaslanie je vykonané pomocou funkcie `call()`. Posledná časť testu vyžiada nahratý projekt a porovná zmeny ku ktorým v ňom prišlo pomocou prekladu. K získaniu dát bola použitá trieda `ARServer` ktorá vyžiadala dáta od služby Project. Ak sa zmeny projektu zhodujú s očakávanými je možné považovať výsledné pridanie funkcie do systému ako úspešné.
- Manuálne otestovanie funkcionality.
  Manuálne testovanie prebiehalo priamou prácou s aplikáciou ARCOR2. Pre skúšku rozšírenia vo forme prekladača, bola nahratá rozšírená verzia tejto aplikácie na server ktorý slúžil ako ARServer. Následne bol niekoľko krát vytvorený program robota, pomocou rozšírenej reality t.j. užívateľského rozhrania, ktorý bol vydaný ako exekučný balík. Balík bol následne stiahnutý, upravený a nahratý späť do systému aby prišlo k preloženiu skriptu. Po spustení upraveného programu bolo pozorované správanie robota. Pozorovaním bolo overené či sa robot správa tak ako by mal na základe zmien. Z testovania je vytvorený video záznam ktorý súčasne demonštruje prácu na tejto aplikácii. Video záznam je dostupný na tomto odkaze video.



Obr. 6.1: Program robota po preložení a pred preložením

6.4 Zistené chyby

Počas vytvárania testov nastal prípad ktorý neodpovedal požadovanej reprezentácii po zadaní vstupu do funkcie `program_src()`. V uvedenom prípade išlo o výskyt samotného príkazu `if` bez nadväzujúceho príkazu `elif`. Z tohto dôvodu bol pridaný test do testovacej sady pre túto funkciu. Test bol označený pomocou značenia `@pytest.mark.xfail()`.

V budúcom vývoji bude možné takýto prípad ošetriť alebo prípadne rozhodnúť o tom, že uvedený prípad nemá mať reprezentáciu.

Pri manuálnom testovaní bol nájdený vedľajší efekt funkcionality prekladača. V prípade, že užívateľ v rozhraní aplikácie vytvorí akčný bod a v bode akciu ktorá nevyžaduje hodnotu akčného bodu, prekladač túto akciu preloží k poslednému akčnému bodu ktorý sa využíva ako parameter v niektorej z akcií. V rozhraní aplikácie bolo po preklade možné vidieť body ktoré nemali v sebe žiadne akcie a v iných bodoch bolo viac akcií než bolo pôvodne. Tento efekt je možné vidieť na obrázku 6.1. Na výsledný algoritmus robota zmena nemala žiaden vplyv. Z dôvodu, že takéto akčné body slúžia ako držitelia pre akcie a akcie tohto typu môžu byť súčasťou ľubovolných akčných bodov. V tomto prípade nešlo o presný preklad a bola potrebná oprava programu. Táto chyba bola opravená pridaním informácie o pôvodnom umiestnení akcií v akčných bodoch projektu. Na začiatok funkcie `python_to_json()` bolo pridané vytvorenie kópie akčných bodov projektu pomocou funkcie `deepcopy()` z Python modulu `copy`. Následne je vždy pri generovaní akcie vyhľadané meno akcie v zozname pôvodných akčných bodov pomocou implementovanej funkcie `action_point_in_list()`. Ak sa meno akcie nájde v zozname akčných bodov, funkcia vráti identifikátor tohto bodu. Následne je tento identifikátor vyhľadaný v zozname bodov projektu pomocou implementovanej metódy `find_action_point()`. Vygenerovaná akcia je potom pridaná k nájdenému akčnému bodu. Ak sa meno akcie nenájde akcia ja pridaná k poslednému videnému akčnému bodu. Pre tento prípad boli vytvorené testovacie prípady ktoré boli zahrnuté do jednotkových a prekladových testov. Taktiež bola upravená funkcia `check_python_to_json()` aby porovnala počty akcií v každom akčnom bode medzi preloženým a pôvodným projektom.

Kapitola 7

Záver

Cieľom tejto práce bolo doplniť funkcionalitu spätného prevodu pre aplikáciu ARCOR2. Na začiatku práce bolo potrebné sa oboznámiť s aplikáciou ARCOR2, ako funguje, aké sú jej vnútorné závislosti a ako sú reprezentované dáta. Tieto poznatky som nadobudol štúdovaním dokumentov o systéme ARCOR2, kódoch aplikácie na platfomre github a štúdiom exekučných balíkov ktoré obsahovali nielen Python kód, ale aj JavaScript Object Notation (ďalej len"JSON"), ktorý je reprezentáciou prekladaného Python kódu. Práca tak obsahuje podrobný popis reprezentácie dát v aplikácií ARCOR2.

Pre vytvorenie návrhu algoritmu prekladača bolo nutné poznať štruktúru prekladačov na základe ktorej fungujú. Tieto poznatky som mal vďaka predmetu "Formálne jazyky a prekladače". V návrhu a aj vo výslednej implementácii sa využíva Python modul AST t.j. abstraktný syntaktický strom na základe ktorého je riadený celý preklad. Následne som sa zameral na jednotlivé prípady ktoré je potrebné pri preklade spracovať. K analyzovaniu parametrov akcií bol použitý modul inspect. Keďže vstupný kód môže obsahovať rôzne chyby bolo potrebné sa zamerať aj na tieto prípady a ošetriť ich.

Výsledné riešenie bolo niekoľko krát otestované nielen jednotkovými, prekladovými a integračnými testami ale aj samotnou prácou na rozšírení. Finálnym testom rozšírenia bolo jeho otestovanie v praxi. Všetky tieto testy overili funkčnosť výsledného programu. Výsledky implementácie programu a testov boli postupne nahrávané do "pull-requestu" ktorý bol následne pridaný do oficiálneho úložiska aplikácie ARCOR2 na platforme github. Všetky vytvorené a upravené súbory je možné nájsť na odkaze.[1]. Finálnou častou práce bolo vytvorenie video záznamu ktoré demonštruje výslednú práce a jej funkčnosť. Tento video záznam je dostupný na adrese[2].

Výsledná implementácia algoritmu na základe ktorého pracoval prekladač bola pre mňa novou skúsenosťou počas ktorej som mohol pracovať na väčšom projekte. Zaujímavé bolo pre mňa spoznať nové postupy a vlastnosti samotného programovania. Pracovať na pozadí veľkého projektu a vidieť ako doplnená funkcionalita aplikácie vo výsledku funguje bolo pre mňa veľmi zaujímavé.

Pokračovanie vo vývoji prekladača by bolo možné pridaním logiky na spracovanie cyklov, pre ktoré už existujú náčrty samotnej JSON reprezentácie, ale zatiaľ nie sú kompletné a o ich pridaní sa uvažuje. Ďalší vývoj systému ARCOR2 by bol možný pridaním rozšírenia pre podmienky pri ktorých by rozšírenie umožňovalo užívateľovi písať viacero podmienok za sebou bez akcií, ktoré ich v súčastnosti musia spájať. Toto rozšírenie by spočívalo úp-

[1] Odkaz: https://github.com/robofit/arcor2/commit/c3e17bac8d0d3e04e765df6434089c3e2c9389d8

[2] Odkaz: https://drive.google.com/file/d/1e6a-9B0M1IDGwTp2p6Bk_WCWGbUD5S4L/view?usp=share_link

ravou atribútu `condition` na zoznam v logických prvkoch. Jeden logický prvok by mohol obsahovať viac podmienok, ktoré by logickému prvku povoľovali prístup k akciám. Na záver by som uviedol, že pridaním kontroly sémantiky mimo časť `while`, by bolo možné predísť chybám vytvorených užívateľom v skripte.

Literatúra

[1] Python. *What is Python? Executive Summary* [online]. 2006 [cit. 2023-2-04]. Dostupné z: https://www.Python.org/doc/essays/blurb/.

[2] JSON. *Introducing JSON* [online]. 2008 [cit. 2022-12-10]. Dostupné z: https://www.json.org/json-en.html.

[3] inspect. *Inspect live objects* [online]. 2012 [cit. 2023-2-12]. Dostupné z: https://docs.python.org/3/library/inspect.html.

[4] Burget, R. *Vizualizace a serializace Způsoby reprezentace strukturovaných dat* [online]. 2020 [cit. 2023-2-1]. Dostupné z: https://moodle.vut.cz/pluginfile.php/457031/mod_resource/content/1/p06_Vizualizace_serializace.pdf.

[5] Kapinus, M., Materna, Z., Bambušek, D., Beran, V. a Smrž, P. *ARCOR2: Framework for Collaborative End-User Management of Industrial Robotic Workplaces using Augmented Reality* [online]. 2021 [cit. 2022-12-20]. Dostupné z: https://vutbr-my.sharepoint.com/personal/xmater02_vutbr_cz/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fxmater02%5Fvutbr%5Fcz%2FDocuments%2FSoubory%20z%20chatu%20aplikace%20Microsoft%20Teams%2FARCOR2%5F%5FEnd%5FUser%5FManagement%5Fof%5FRobotic%5FWorkplaces%5Fusing%5FAR%2Epdf&parent=%2Fpersonal%2Fxmater02%5Fvutbr%5Fcz%2FDocuments%2FSoubory%20z%20chatu%20aplikace%20Microsoft%20Teams&ga=1.

[6] Materna, Z. *Wiki ARCOR2* [online]. 2021 [cit. 2022-12-20]. Dostupné z: https://github.com/robofit/arcor2/wiki.

[7] Meduna, A. a Lukáš, R. *Formální jazyky a překladače IFJ Studijní opora* [online]. Verze: 1.2006+revize 2009-2015. 2006 [cit. 2022-11-23]. Dostupné z: https://www.vut.cz/www_base/priloha_fs.php?dpid=227112&skupina=dokument_priloha.

Príloha A

SD Obsah

Pamäťové médium obsahuje nasledujúci obsah:

- `/program/`
  - `/arcor2-compiler/*` - kompletná aplikácia s pridaným rozšírením
  - `zmeny.txt` - popis zmien vykonaných na aplikácií ARCOR2 s odkazom na vykonaný "commit"
- `/manual/`
  - `manual.pdf` - manuál pre spustenie aplikácie
  - `arcor2-decompiler.mp4` - demonštračné video
  - `pkg_test.zip` - testovací balík
- `/tex/*` - zdrojové kódy práce
- `bp_xkadna00.pdf` - finálna verzia práce v pdf formáte