



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DEPARTMENT OF INFORMATION SYSTEMS

**MOBILNÍ EDITOR PRO VIZUÁLNÍ PROGRAMOVÁNÍ
IOT ZAŘÍZENÍ**

MOBILE EDITOR FOR VISUAL PROGRAMMING OF IOT DEVICES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

VEDOUCÍ PRÁCE

SUPERVISOR

DAVID ŠKRABAL

Ing. PETR JOHN

BRNO 2025

Zadání bakalářské práce



164742

Ústav: Ústav informačních systémů (UIFS)
Student: **Škrabal David**
Program: Informační technologie
Název: **Mobilní editor pro vizuální programování IoT zařízení**
Kategorie: Webové aplikace
Akademický rok: 2024/25

Zadání:

1. Prostudujte oblast internetu věcí (*Internet of Things*, IoT).
2. Prostudujte principy vizuálního programování a současné nástroje vizuálního programování.
3. Analyzujte požadavky na grafický editor optimalizovaný pro mobilní zařízení, který umožní snadno vytvářet programy pro chytrá zařízení v IoT aplikacích (např. domácí chytré skleníky, chytré domy).
4. Po konzultaci s vedoucím navrhnete grafický editor pro tvorbu jednoduchých programů určených pro chytrá zařízení.
5. Navržený grafický editor implementujte.
6. Výsledný editor otestujte s koncovými uživateli. Vyhodnoťte jeho použitelnost.

Literatura:

- Badii, C., Bellini, P., Difino, A., Nesi, P., Pantaleo, G., & Paolucci, M. (2019). Microservices suite for smart city applications. *Sensors*, 19(21), 4798.
- Hynek, J. a spol. (2023). Služby pro systém řízení a monitoringu vody v retenčních nádržích. Výzkumná zpráva. Vysoké učení technické v Brně.
- Greengard, S. (2015). *The Internet of Things*. MIT Press, ISBN 978-026-2527-736.
- Kuhail, M. A., Farooq, S., Hammad, R., & Bahja, M. (2021). Characterizing visual programming approaches for end-user developers: A systematic review. *IEEE Access*, 9, (pp. 14181-14202).
- Ray, P. P. (2017). A survey on visual programming languages in internet of things. *Scientific Programming*, 2017.
- John, P. (2024). Optimising processes in IoT. Pojednání o tématu disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Tomáš Hruška, CSc.
- Wroblewski, L. (2011). *Mobile First. A Book Apart*. Book Apart. ISBN 978193755702.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 - 4.

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

Vedoucí práce: **John Petr, Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2024
Termín pro odevzdání: 14.5.2025
Datum schválení: 22.10.2024

Abstrakt

S rozvojem technologií internetu věcí (IoT) vzniká pro uživatele potřeba mít nástroj, pomocí kterého by si je mohli sami nastavovat bez znalosti programování. Současné nástroje jsou často složité a nevhodné pro mobilní zařízení. Mohou tak odradit uživatele od snahy jakkoliv do nastavení systémů internetu věcí zasahovat. Cílem této práce je vytvořit nástroj, určený pro mobilní zařízení, který umožní koncovým uživatelům jednoduše programovat IoT technologie pomocí vizuálního programování. Řešení by mělo zároveň umožnit práci s IoT zařízeními různého typu a od různých výrobců. Výsledkem práce je vytvoření knihovny, která implementuje editor, určený pro mobilní zařízení. Jeho výstupem je serializovaný program, který je možné převést na příkazy pro zařízení internetu věcí.

Abstract

With the development of Internet of Things (IoT) technologies, users need a tool that allows them to configure these technologies without requiring programming knowledge. Current tools are often complex and unsuitable for mobile devices. They can discourage users from trying to interfere in any way with the settings of IoT systems. The aim of this work is to design a mobile-friendly tool that enables end users to program IoT technologies using visual programming. The solution should also support working with various types of IoT devices from different manufacturers. The outcome of this work is a library that implements an editor designed for mobile devices. Its output is a serialized program that can be converted into commands for IoT devices.

Klíčová slova

internet věcí, IoT, vizuální programování, vývoj koncovým uživatelem, grafický editor, webová aplikace, Lit, TypeScript, JSON, webové komponenty

Keywords

internet of things, IoT, visual programming, End-User development, graphical editor, web application, Lit, TypeScript, JSON, web components

Citace

ŠKRABAL, David. *Mobilní editor pro vizuální programování IoT zařízení*. Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Petr John

Mobilní editor pro vizuální programování IoT zařízení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Johna. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
David Škrabal
9. května 2025

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Petru Johnovi za jeho pomoc s touto prací, jeho podporu, trpělivost a aktivní přístup. Dále bych rád poděkoval všem, kteří se zúčastnili průzkumu uživatelských požadavků a testování aplikace. Jmenovitě panu Bc. Tomáši Krönerovi za povolení použít jeho náčrt aplikace v této práci.

Obsah

1 Úvod	7
2 Internet věcí	9
2.1 Typy zařízení a komunikací	9
2.2 Architektury	12
2.3 Digitální dvojče	12
2.4 Použití	13
2.5 Smart Cities	15
3 Vizualní programování	18
3.1 Tvorba programů koncovými uživateli	18
3.2 Principy vizualního programování	20
3.3 Typy vizualních programovacích jazyků	22
3.4 Nástroje vizualního programování pro IoT zařízení	27
4 Analýza programování zařízení IoT	32
4.1 Existující editory	32
4.2 Interakce uživatele s mobilní aplikací	33
4.3 Uživatelské požadavky	34
4.4 Požadavky na řešení	36
5 Návrh mobilního editoru pro vizualní programování	38
5.1 Grafický editor programu	38
5.2 Akce příkazů	40
5.3 Proměnné	41
5.4 Nastavení parametrů příkazu	42
5.5 Textový editor programu	43
5.6 Ikonky a jazyk aplikace	43
6 Implementace	45
6.1 Integrace knihovny Lit	45
6.2 Reprezentace programu v knihovně	46
6.3 Komponenty editoru	47
7 Testování	55
8 Závěr	58
Literatura	59

A	Slovní popis ilustračního programu pro automatizaci chytré domácnosti	64
B	Náčrt aplikace jednoho z respondentů	65
C	Obsah externí přílohy	67

Seznam obrázků

2.1	Na obrázku je znázorněn příklad komunikace senzoru a akčního zařízení. Teplotní senzor měří teplotu v místnosti a data o ní pravidelně odesílá centrální jednotce. Ta je analyzuje a na jejich základě posílá příkazy akčnímu členu topení.	10
2.2	Na obrázku je znázorněna třívrstvá architektura. Každá vrstva obsahuje ikonu technologie, která se v ní nachází (převzato z literatury [18]).	13
3.1	Na obrázku je příklad vytváření automatizace pomocí trigger-action programming v aplikaci Xiaomi Home. V první části si uživatel nadefinuje podmínky spuštění, jako například čas a aktuální stav vysavače. Následně pak určí, jaká akce se za takové podmínky má provést. Na obrázku jde o úklid. Jak podmínky, tak akce se přidávají pomocí tlačítka plus. Na konci je tlačítko pro uložení výsledné automatizace.	20
3.2	Grafický editor Blockly v české jazykové mutaci. V pravé části může uživatel provádět úpravy ve vygenerovaném kódu. Zároveň zde může přepínat jazyk aplikace, programovací jazyk, ve kterém je kód vygenerován, a spouštět program. V levé listě je nabídka všech typů komponent. Zde je možné si daný typ rozkliknout a přetáhnout si z něj konkrétní prvek. Prostřední část tvoří prostor pro sestavování programu. Bloky lze spojovat a zanořovat.	23
3.3	Grafický editor Scratch. V pravé horní části obrazovky vidíme postavičku tučňáka, kterou si uživatel vybral v pravé dolní části. V levé části lze postavice nadefinovat chování jako například zvuk, pohyb nebo vzhled. Uživatel si vybere jednu z nabízených komponent a přetáhne ji do programu. Na obrázku je program, který vydá zvuk „pop“, dvacetkrát otočí tučňáka o 15 stupňů a přesune na náhodné místo. Pokud uživatel zmáčkne mezerník, ozve se desetkrát „pop“.	24
3.4	Program vytvořený v aplikaci MicroApp. Ten nám říká, že pokud zazvoní zvonek, spustí se kamera u dveří, která přenáší obraz na televizi a na mobilní zařízení (<i>PlayVideo</i>). Pokud přijde od uživatele správný text, tak se dveře otevřou a zároveň se uloží záznam z kamery (převzato z literatury [14]). . .	25
3.6	Ukázka aplikace vyrobené v nástroji Snap4city. Uživatel zde kliknutím na jedno z tlačítek zašle vstupní data. Na základě nich se vygeneruje událost, která je poté odeslána k provedení na server (převzato z literatury [4]). . .	26
3.7	Na obrázku A vidíme, jak uživatel přidává první komponentu do programu pomocí tlačítka plus. Na snímku B nejdříve vybere, zda chce přidat blok operací, nebo pouze jednu a následně si vybere z nabídky (převzato z literatury [50]).	27

3.8	Ukázka dvou ECA bloků, které oba mají stejnou událost (odemknut zámek 1), ovšem rozdílnou podmínku. V prvním bloku je vyhodnocena podmínka, zda je aktivní presenceSensor. Pokud ano, zapne se světlo a hudba. Druhý ECA blok obsahuje opačnou podmínku. Pokud platí, je aktivován alarm a odeslána zpráva uživateli (převzato z literatury [5]).	28
3.9	Ukázka aplikace Loxone Config, určené pro programování chytré domácnosti od společnosti Loxone. Tato aplikace se skládá z menu, seznamu dostupných periférií a pracovní plochy pro tvorbu programu. Menu se nachází v horní části obrazovky a obsahuje základní funkce programu. Nejvíce prostoru zabírá editor programu. V něm můžeme vidět několik senzorů (nalevo), řídicí bloky programu (uprostřed) a prováděné akce (napravo). Program definuje ovládání osvětlení jídelny.	29
3.10	Aplikace ACADA ve verzi pro desktopová zařízení (vlevo) a mobilní zařízení (vpravo) (převzato z literatury [29]).	30
4.1	Na obrázku lze vidět grafický editor od pana Podvojského spuštěný na desktopovém zařízení. Ten zobrazuje program s jednou podmínkou a dvěma akcemi. Stránka se skládá z menu, grafického a textového editoru.	33
4.2	Ukázka zobrazení nabídky možností v aplikaci Messenger. Nabídka se objeví ve spodní části obrazovky po podržení zprávy.	34
4.3	Zobrazení nabídky možností v aplikaci Telegram. Nabídka se stejně jako u aplikace Messenger zobrazí po podržení zprávy, ovšem zde je umístěna co nejbližší samotné zprávě.	34
4.4	Na obrázku lze vidět poslechové cvičení v aplikaci Duolingo. Uživatel zde skládá větu, kterou předtím slyšel, z bublin se slovy.	35
4.5	Na obrázku je ukázka fronty přehrávaných záznamů z aplikace Můj Rozhlas. Zde je využívána metoda <i>drag and drop</i> . U každé skladby je ikonka s dvěma vodorovnými čarami. Pokud ji uživatel drží (<i>drag</i>), může přesouvat skladbu na libovolnou pozici ve frontě. Pořadí se pak změní podle toho, kde uživatel pustí (<i>drop</i>) komponentu.	35
5.1	Na obrazovce a) je znázorněn vzhled aplikace na začátku tvorby programu. Ve vrchní části je vidět stručný zašedlý pokyn, co by měl uživatel dále dělat. Ve spodní části je potom vidět seznam příkazů, které je možné doplnit. Obrazovka b) zobrazuje situaci, kdy je po vybrání složeného příkazu nutné doplnit jeho podmínku. V seznamu ve spodní části obrazovky jsou podmínky uložené v repozitáři a možnost vytvořit novou podmínku v editoru. Obrazovka c) znázorňuje stav, kdy již složený příkaz má podmínku a je nutné do něj vložit jednotkový nebo složený příkaz. Obrazovka d) ukazuje situaci, kdy již složený příkaz může být korektně uzavřen, nebo rozšiřován dalšími příkazy.	39
5.2	Na obrázku je znázorněn stav, kdy uživatel podržel příkaz. Po podržení se ve spodní části obrazovky zobrazí možnosti další práce s příkazem a samotný příkaz se zvýrazní.	40
5.3	Obrázek ukazuje rozložení obrazovky po rozšíření složeného příkazu na plnou šířku obrazovky.	41

5.4	Na obrázku je znázorněno, jak vypadá obrazovka aplikace poté, co uživatel podržel u komponenty zapni vysavač tlačítko s dvěma čárkami a snaží se tuto komponentu přesunout.	42
5.5	Obrazovka obsahující seznam všech proměnných v programu.	42
5.6	Vyskakovací okno pro vytvoření a editaci proměnné.	43
5.7	Vyskakovací okno pro editaci parametru příkazu.	43
5.8	Ikonky pro jednotlivé základní příkazy. První ikonka symbolizuje splnění zadané podmínky složeného příkazu (if). Další ukazuje složený příkaz, kdy neplatila první podmínka, ovšem druhá ano (elseIf). Následující symbolizuje nesplnění zadané podmínky příkazu (else). Další vybrání jedné z existujících možností (switch). Pátá ikonka příkazů představuje určitý počet opakování (repeat). Další opakování těla příkazu, dokud platí podmínka (while). Předposlední ikonka symbolizuje posláni notifikace na zařízení uživatele (send notification). Poslední označuje zapsání hodnoty do proměnné (set variable).	44
6.1	Hlavní obrazovka editoru spuštěná na počítači. Obrazovku lze rozdělit na tři části: menu, editor programu, nabídka možností. Jelikož má obrazovka dostatečnou šířku, můžeme vedle sebe vidět textový a grafický editor programu.	48
6.2	V levé části obrázku je znázorněn program před zahájením přetahování komponenty. V pravé části je ukázána situace těsně před umístěním bloku na začátek těla bloku if. Místo, kam bude blok vložen, je barevně odlišeno od ostatních.	48
6.3	Obrázek znázorňuje změnu textového editoru v případě vytvoření chyby. V tomto případě jde o chybějící čárku.	49
6.4	Nabídka bloků, které je možné doplnit do programu. Je možné ji filtrovat podle typu příkazu pomocí menu, které se nachází v její horní části. Jednotlivé bloky obsahují ikonky a jsou barevně odlišeny podle svého typu.	50
6.5	Změna vykreslení editoru pro přidání akce zařízení do programu.	50
6.6	Zobrazení příkazu s více argumenty. Horní příkaz ukazuje zobrazení v běžném režimu a dolní příkaz s rozbalením všech argumentů.	51
6.7	Zobrazení okna pro změnu hodnoty parametru příkazu. V horní části se nachází pole pro doplnění hodnoty. Ve spodní části jsou vidět tlačítka pro doplnění některé z existujících proměnných.	51
6.8	Zvýraznění vybraného příkazu a zobrazení nabídky. Vybraný příkaz je zvýrazněn červeným rámečkem, zatímco zbytek editoru je upozaděn. Zároveň se objeví nabídka s možnými akcemi pro daný příkaz, která je filtrována podle dostupných možností.	52
6.9	Rozšíření příkazu v detail módu editoru.	52
6.10	Nastavení editoru. V levé části obrázku se nachází horní polovina obrazovky pro nastavení editoru. V pravé části se nachází spodní část obrazovky, která má spíše informační funkci.	52
6.11	Tabulka se seznamem dostupných uživatelských proměnných. Ke každé proměnné je uvedeno její jméno, typ a hodnota. Ve spodní části se nachází tlačítka pro opuštění obrazovky a vytvoření nové proměnné.	53

6.12	V levé části obrázku se nachází editor proměnné při definování nové hodnoty. Jelikož nejsou vyplněna všechna pole, není možné proměnnou uložit. V pravé části je znázorněna úprava již existující proměnné, u které nelze měnit jméno a typ proměnné, tak aby nevznikaly syntaktické chyby v kódu.	54
6.13	Editor podmínek, který se skládá ze tří hlavních částí. První je prostor pro zadání jména (tato část není při editaci existující podmínky zobrazena. Druhá část obsahuje grafické zobrazení podmínky a poslední tlačítka pro tvorbu podmínky a její uložení.	54
B.1	Jeden z náčrtů vytvořených během výzkumu.	66

Kapitola 1

Úvod

V současné době můžeme čím dál více pozorovat snahu o vzájemné propojování elektronických zařízení. Běžná zařízení jako jsou pračky, vysavače nebo zavlažování jsou vybavována připojením k internetu. Zároveň často obsahují různé senzory. Mohou tak sbírat informace o svém okolí. Tato zařízení jsou součástí konceptu, který se nazývá internet věcí (anglicky *Internet of Things*, zkráceně *IoT*). Ten je postavený na síti elektronických přístrojů schopných spolu komunikovat a kooperovat. Mohou také pracovat s daty ze senzorů, které mohou být samostatné jednotky, nebo jako součást jiných zařízení. Takováto síť nám otevírá zcela nové možnosti. Umožňuje lepší a efektivnější využívání zařízení, které jsou její součástí. Zároveň nám ulehčuje práci s nimi a dává nám zcela nové možnosti jejich využití. Díky tomu můžeme například nastavit vysávání na dobu, kdy jsme v práci. Dalším příkladem může být automatické spuštění zavlažování v případě nízké vlhkosti půdy. Celý koncept se vyvíjí od 90. let minulého století až do dnešních dnů. Za tu dobu se stal součástí nejen běžných domácností, ale i průmyslu, zemědělství, zdravotnictví, dopravy a mnoha dalších oborů. V poslední době už internet věcí přestává být záležitostí pouze odborníků a nadšenců, ale stává se nedílnou součástí životů běžných lidí.

Síť zařízení internetu věcí je komplikovaný systém skládající se z mnoha vrstev. Aby mohl správně fungovat, je třeba vše správně naprogramovat. Nejnižší vrstvou je naprogramování mikrokontrolerů na jednotlivých zařízeních. Dále je například nutné správné nastavení komunikačních protokolů, ukládání a zpracování dat ze senzorů a mnoho dalšího. Jednou z posledních vrstev je rozhraní pro uživatele, pomocí kterého může internet věcí ovládat. Složitosti vývoje často nutí firmy vyrábět systémy na míru, které nejsou schopny spolupracovat se zařízeními od jiných výrobců. Uživatel si tak musí vše pro svou chytrou domácnost koupit u jednoho výrobce, nebo používat více aplikací.

Uživatelská rozhraní pro ovládání zařízení internetu věcí umožňují různou míru kontroly nad celým systémem. Některé aplikace například umožňují uživateli pouze spuštění předdefinovaných akcí a nastavení jejich základních parametrů. Další možností je využití vizuálních programovacích jazyků. Na ten se zaměřuje tato bakalářská práce. Nástroje vizuálního programování umožňují uživatelům tvorbu vlastního chování pro jejich zařízení internetu věcí. Grafické komponenty a další aspekty vizuálního programování umožňují uživatelům snadné definování logiky celého systému i bez znalostí programování.

Cílem této práce je vytvořit grafický editor pro definování chování zařízení internetu věcí. Ten by měl uživatelům pomocí nástrojů vizuálního programování umožnit snadnou tvorbu programů. Výstupem bude serializovaný formát, který lze následně převést do příkazů podporovaných konkrétními zařízeními internetu věcí. Díky tomu bude možné zajistit spolupráci zařízení různých výrobců. Editor by měl být primárně určen pro mobilní zaří-

zení. Ta jsou čím dál více využívána běžnými uživateli i pro provádění složitých operací. Většina současných nástrojů pro ně není přizpůsobena a špatně se na nich ovládají.

Kapitola 2 se věnuje základním poznatkům v oblasti IoT. Kapitola 3 se zaměřuje na využití konceptu vizuálního programování pro uživatelskou správu zařízení IoT. Jsou zde rozebrány některé z používaných nástrojů. V kapitole 4 je popsána analýza interakce uživatele s mobilními aplikacemi, průzkum uživatelských požadavků a jejich rozbor. Kapitola 5 popisuje finální návrh editoru, který vzešel z poznatků z předchozí kapitoly. Poslední kapitola 8 pak shrnuje výsledky práce.

Kapitola 2

Internet věcí

Internet věcí (anglicky *Internet of Things*, zkráceně *IoT*) lze definovat jako síť elektronických zařízení, která jsou schopna spolu komunikovat prostřednictvím internetu [21]. Často se jedná o zařízení, která běžně využíváme. Například mobilní telefon, chytré hodinky, robotický vysavač, lednička nebo rádio. Tato síť obvykle obsahuje zařízení se senzory (například průchodové čidlo) nebo akční prvky (například chytrá žárovka). Nezbytnou částí je pak software, který zajistí vzájemnou komunikaci a sdílení dat mezi jednotlivými zařízeními. Zde jsou využity různé typy síťových technologií, protokolů a standardů [47]. Pro správné fungování je nutné, aby každé zařízení v síti mělo unikátní identifikační číslo (anglicky *unique identification number*) [16]. Díky tomu je možné každé zařízení v síti jednoduše identifikovat. Architektura celé sítě pak může být postavena na vzdáleném centrálním řízení, nebo na vzájemné lokální komunikaci mezi zařízeními.

Jelikož internet věcí je široce využívaný pojem, existuje pro něj více možných definic. Například Evropská unie definuje pojem internetu věcí následovně: Internet věcí (IoT) označuje distribuovanou síť propojující fyzická zařízení, která jsou schopna zachycovat, nebo reagovat na své prostředí a schopná komunikovat mezi sebou, jinými stroji nebo počítači [12].

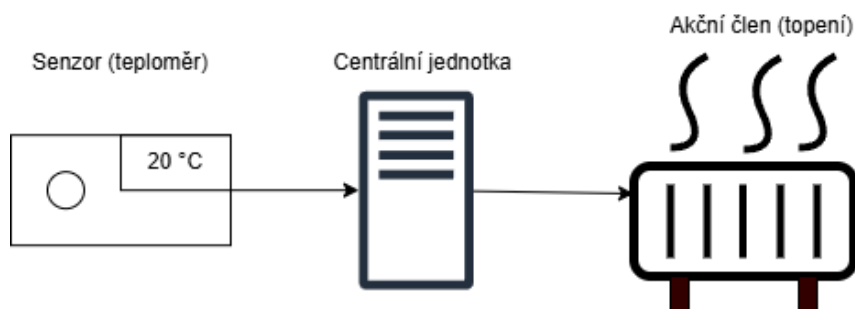
Předpokládá se, že jako první použil pojem internet věcí (anglicky *Internet of Things*) Kevin Aston ve své prezentaci pro společnost Procter & Gamble (P&G) v roce 1999. Ten byl spoluzakladatelem a výkonným ředitelem Auto-ID Center na MIT [3]. Nicméně konceptu propojených zařízení se tato společnost věnovala již od začátku 90. let minulého století. Zkoumali myšlenku vytvoření systému, který by umožnil propojení fyzických zařízení pomocí senzorů a bezdrátových signálů [16]. Následoval vývoj této problematiky, který trvá až dodnes.

S rozvojem internetu věcí přibývá i odvětví, kde našel své uplatnění. Dnes se využívá například ve zdravotnictví, chytrých domácnostech nebo ovládání chytrých měst. S rostoucím počtem zařízení internetu věcí však bohužel roste i množství způsobů jejich napadení a zneužití.

2.1 Typy zařízení a komunikací

Základem všech systémů IoT jsou jednotlivá elektronická zařízení. Ta využívají k vzájemnému propojení různé síťové technologie. Mohou komunikovat jak mezi sebou, tak s centrální řídicí jednotkou. Tato zařízení můžeme rozdělit například podle role v systému na senzory, které získávají informace pro systém, a akční členy, které provádí akce podle pokynů systému [33]. Oba typy mohou být jako samostatná zařízení, nebo součástí jednoho přístroje

(například roboticky vysavač, který má senzory a zároveň pracuje jako akční člen). Příklad zařízení různých typů a jejich spolupráci můžeme vidět na obrázku 2.1.



Obrázek 2.1: Na obrázku je znázorněn příklad komunikace senzoru a akčního zařízení. Teplotní senzor měří teplotu v místnosti a data o ní pravidelně odesílá centrální jednotce. Ta je analyzuje a na jejich základě posílá příkazy akčnímu členu topení.

Senzory jsou zařízení, která jsou schopna snímat svět kolem sebe. Umí měřit různé fyzikální veličiny jako je například vlhkost, teplota nebo tlak [33, 21]. Naměřené hodnoty pak převádějí na signál, který je možné dále přenášet a zpracovávat. Na základě těchto dat pak akční členy vykonávají akce.

Jednou z nejdůležitějších vlastností senzorů je přesnost naměřených dat. Systém potřebuje získávat dostatečně přesná data, aby mohl správně ovládat akční členy. Zároveň by senzory neměly ovlivňovat prostředí, ve kterém se nachází a narušovat tím měření jiných senzorů nebo svá. Jejich další důležitou vlastností je energetická efektivita. Senzory mohou být umístěny i na místech bez přístupu elektrické energie. V takovém případě mohou být napájeny z baterií nebo obnovitelných zdrojů. Je proto třeba, aby využívaly energii co nejefektivněji.

Jedním ze způsobů klasifikace senzorů může být rozdělení podle druhu výstupu, a to na digitální a analogové [33]. Další možností dělení může být na pasivní a aktivní senzory. Pasivní senzory měří a odesílají data automaticky, zatímco aktivní senzory provádí měření pouze pokud je jim zaslán příkaz. Nejčastějším způsobem klasifikace senzorů je podle měřené veličiny. Jako příklad můžeme uvést senzory tlakové, polohové (například systém GPS), teplotní, infra senzory a mnoho dalších.

Akční členy IoT jsou schopny vykonávat určitou činnost. Přijímají od systému data, která předtím získaly senzory. Na jejich základě provedou akci s určitými parametry. Vyžadují ovládací signál a zdroj energie [33]. Využívají elektrickou energii k provedení akcí jako například chlazení, mechanický pohyb, reprodukce zvuku a další. Akční členy, které provádí mechanický pohyb je možné dělit podle způsobu transformace elektrické energie na elektrické, mechanické, hydraulické a pneumatické [33].

Chytrá zařízení využívají pro komunikaci široké spektrum síťových technologií a protokolů. Bezdrátové technologie pro komunikaci můžeme rozdělit podle dosahu na technologie krátkého dosahu (*short-range*), dlouhého dosahu (*long-range*) a celulární (*cellular*) [1].

Bezdrátové technologie krátkého dosahu

RFID (*Radio Frequency Identification*) je jednou z nejdůležitějších technologií pro přenos dat mezi zařízeními IoT. Využívá se hlavně pro komunikaci na krátkou vzdálenost. Skládá se z paměťového zařízení (*tag*) a čtečky. Čtečka vyšle signál pomocí rádiových vln [47]. Ten

je zachycen tagem, který následně pošle čtečce data ze svého uložště. RFID pracuje na různých rádiových frekvencích tak, aby co nejvíce vyhovoval požadavkům aplikace. Tato technologie se často využívá k identifikaci objektů stejně jako čárové kódy.

Na podobném principu jako RDIF je postavena i technologie **NFC** (*Near Field Communication*). Ta se využívá pouze na krátké vzdálenosti (přibližně 4 cm [33]). Na rozdíl od RDIF podporuje i obousměrnou komunikaci.

Další možností je například využití technologie **WiFi** (*Wireless Fidelity*). Ta zajišťuje připojení k internetu nebo k lokální síti pomocí bezdrátových síťových protokolů založených na standardech IEEE 802.11 [53].

Jednou z dalších často využívaných technologií je **Z-Wave**. Její hlavní výhodou je nízká spotřeba. Je optimalizována pro spolehlivou komunikaci malých datových packetů s nízkou latencí [33]. Mezi další podobné technologie patří například Bluetooth a ZigBee.

Bezdrátové technologie dlouhého dosahu

Jednou ze základních technologií pro přenos informací na velké vzdálenosti je **LoRa** (*Long range*). Pro přenos dat využívá rádiovou komunikaci. Sítě LoRa jsou založeny na topologii typu *star-to-star*, kde každé koncové zařízení má přímé připojení k bráně [1]. Mezi její hlavní výhody patří nízká energetická náročnost. Pro zabezpečení a autentizaci používá šifrování AES a IEEE 802.15.4/2006 Annex B [11]. **LoRaWAN** (*Long Range Wide Area Network*) je rozšíření původní technologie o síťový komunikační protokol.

Další podobnou technologií určenou pro posílání krátkých zpráv na velké vzdálenosti je **SigFox**. Ta vyniká nízkou spotřebou energie a dlouhým dosahem, ale má omezenou rychlost přenosu dat [1]. Její výhodou je, že může být použita spolu s jinými typy sítí jako například Wi-Fi, Bluetooth nebo LoraWAN¹.

Celulární bezdrátové technologie

Důležitou součástí rozvoje IoT jsou dnes celulární technologie. Ty umožňují bezdrátový přenos dat pomocí rádiových vln v síti, která je rozdělena na buňky (*cells*). Postupně se vyvíjely od 2G až po dnešní 5G generaci [1]. Nejnovější generace mobilních sítí 5G zahrnuje také podporu pro zařízení internetu věcí.

Komunikační protokoly pro IoT

Mezi nejčastěji využívané komunikační protokoly pro IoT patří MQTT (*Message Queuing Telemetry Transport*) protokol [48]. Ten je založen na architektuře publikování a odebírání (*publish/subscribe architecture*). Ta využívá model klient/server. Sever (broker) se stará o příjem a odesílání zpráv na specifické téma (*topic*) od všech klientů. Ti mohou mít dvě různé role. První je vydavatel (*publisher*), který odesílá zprávy na téma dostupné na brokeru. Druhou je odběratel (*subscriber*), který přijímá zprávy od brokera na dané téma.

MQTT se využívá pro odlehčenou M2M (machine to machine) komunikaci. Umožňuje komunikační možnosti typu one-one, one-to-many a many-to-many [47]. Kvalitu služeb (anglicky *Quality of Service*, zkráceně *QoS*) MQTT můžeme rozdělit do tří kategorií² [48]:

- **QoS 0** – Klient odešle zprávu brokeru a nevyžaduje od něj žádné potvrzení o přijetí.

¹Stránky technologie SigFox: <https://sigfox.com/what-is-sigfox/>

²<https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>

- **QoS 1** – Po odeslání zprávy si klient její kopii uloží. Následně čeká na potvrzení od brokeru. Pokud odpověď nepříjde do určitého časového limitu, zprávu znovu odešle. Kopii zprávy klient smaže, až když obdrží od brokeru potvrzení o doručení.
- **QoS 2** – Na této úrovni proběhne mezi klientem a brokerem *four-part handshake* proces. Ten zajistí, že je každá zpráva doručena právě jednou.

Dalším protokolem využívaným pro komunikaci IoT zařízení je například aplikační protokol CoAP (Constrained Application Protocol). CoAP slouží pro přenos dokumentů a je navržen pro využití na strojích s omezenými výpočetními a energetickými zdroji, jako jsou senzory [47]. Díky těmto vlastnostem je pro IoT vhodnější než HTTP (*Hypertext Transfer Protocol*) protokol. Na HTTP jej lze mapovat pomocí proxy meziprotokolů [47]. Mezi další používané protokoly patří například DDS (Data Distribution Service), XMPP (*Extensible Messaging and Presence Protocol*), AMQP (Advanced Message Queuing Protocol) nebo LLAP (Lightweight Local Automation Protocol) [47, 21].

2.2 Architektury

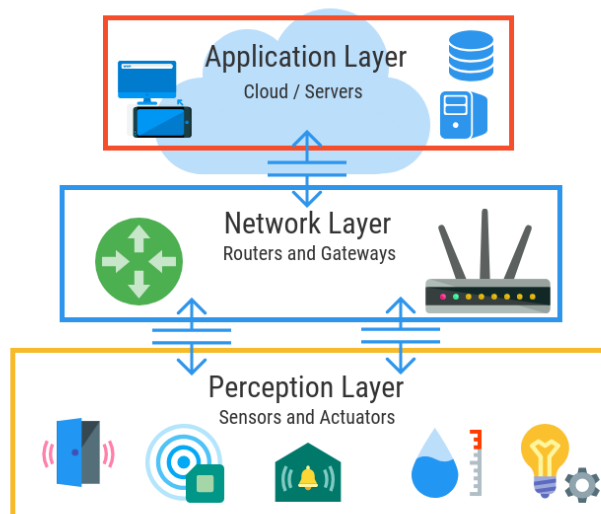
Za dobu vývoje vzniklo mnoho různých architektur IoT. Žádná z nich nebyla do dnešního dne standardizována [33]. Neexistuje tak všeobecně uznávaný standart pro tvorbu IoT systémů. Společnosti proto zavádějí své vlastní standarty a postupy. Z toho důvodu nejsou zařízení od různých výrobců často schopna spolupracovat. Mezi základní vlastnosti všech architektur by měla patřit rozšiřitelnost, škálovatelnost a spolupráce mezi zařízeními v reálném čase [15]. Jejich vývoji se věnují společnosti jako například ITU (*International Telecommunication Union*), IEEE (*Institute of Electrical and Electronics Engineers*) nebo Cisco [17].

Jednou z prvních architektur byla třívrstvá architektura, kterou můžeme vidět na obrázku 2.2. Ta je považována za základní architekturu, ze které vychází všechny ostatní. Třívrstvá architektura se skládá ze vnímací vrstvy (*perception layer*), síťové vrstvy (*network layer*) a aplikační vrstvy (*application layer*) [2]. Vnímací vrstva zahrnuje fyzická zařízení v síti. Ta můžeme rozdělit na senzory a akční členy. Senzory shromažďují data o svém okolí (teplota, tlak, vlhkost) a ta následně předávají síťové vrstvě. Akční členy přijímají od síťové vrstvy data ze senzorů a externí příkazy. Na jejich základě následně provedou akci, která ovlivní jejich okolí (klimatizace sníží teplotu v místnosti). Úkolem síťové vrstvy je propojení zařízení ve vnímací vrstvě s aplikační vrstvou a mezi sebou. Využívá k tomu pevného nebo bezdrátového připojení. Mezi její základní vlastnosti by měla patřit bezpečnost dat a schopnost automaticky přidávat a spravovat zařízení v síti [15]. Nejvyšší aplikační vrstva provádí analýzu a zpracování dat od senzorů. Následně provede rozhodnutí o dalším chování sítě a přes síťovou vrstvu odešle příkazy zařízením ve vnímací vrstvě.

Většina novějších architektur vychází ze třívrstvé a rozšiřuje ji o nové vrstvy. Jako je tomu například u čtyřvrstvé architektury SoA (*Service Oriented Architecture*) [15]. Ta přidává vrstvu rozhraní (*interface layer*) sloužící pro komunikaci systému s uživatelem. Další často používanou architekturou je pětivrstvá architektura.

2.3 Digitální dvojče

Digitální dvojče (*Digital twin*) je digitální/virtuální model nebo simulace skutečného subjektu nebo objektu [45]. Jednou z jeho hlavních vlastností je obousměrné sdílení dat s jeho



Obrázek 2.2: Na obrázku je znázorněna třívrstvá architektura. Každá vrstva obsahuje ikonu technologie, která se v ní nachází (převzato z literatury [18]).

fyzickou předlohou. Má stejné chování jako jeho fyzická předloha a je schopen se dostávat do stejných situací. K tomu využívá historická data a informace o měnících se podmínkách svého okolí.

Jedním z možných dělení digitálních dvojčat je na tři úrovně podle hierarchie [45]. Nejnižší je jednotková úroveň (*Unit level*). Jde o nejmenší možné jednotky (část zařízení). Jejich spojením dohromady vzniká Systémová úroveň (*System level*), která obsahuje informace o propojenosti a spolupráci mezi jednotkami. Poslední vrstvou je systém systémů (*System of systems*), která spojuje dohromady digitální dvojčata systémové úrovně.

Digitální dvojče se využívá se například k simulaci a analýze chování fyzického zařízení, testování jeho bezpečnosti a spolehlivosti nebo k jeho optimalizaci [32, 45]. Jednou z hlavních výhod digitálního dvojčete je urychlení vývoje fyzického zařízení pomocí simulací různých scénářů. Zároveň jeho tvorba není tak nákladná jako provádění testů na fyzickém zařízení, jeho následné přepracování a opětovné testování. Zároveň vyšší množství otestovaných scénářů může pomoci s tvorbou optimálnějšího a bezpečnějšího zařízení.

Dnes se využívá především v průmyslu 4.0. V oblasti internetu věcí se používá k tvorbě digitálních verzí zařízení IoT. Pomocí nich se provádí simulace v reálném čase na datech od senzorů. Jejich výsledky lze využít k rozhodování o dalším chování systému.

2.4 Použití

Zařízení internetu věcí se v posledních letech začínají čím dál více využívat. V roce 2024 jich bylo na světě přes 18 miliard a odhaduje se, že v roce 2030 by jich mohlo být až 40 miliard³. Tato technologie nachází uplatnění v celé řadě oblastí.

Chytrá domácnost – jde o jednu ze známějších oblastí využití IoT. Jejím hlavním cílem je automatizace domácnosti, optimalizace každodenních činností a úspora energií [13].

³Analýza počtu IoT zařízení: <https://www.demandsage.com/number-of-iot-devices/>

Mezi její další výhody patří například monitorování systému uživatelem, vyšší bezpečnost domácnosti a ochrana životního prostředí. Skládá se z běžných zařízení, která musí být připojena k internetu. Příkladem mohou být robotické vysavače, klimatizace, chytré žárovky, rolety nebo zámky u dveří. Běžná chytrá domácnost dále může obsahovat senzory jako například teploměry, pohybová čidla nebo měřiče vlhkosti. Ty dodávají data, na základě kterých spotřebiče vykonávají akce [33]. Všechna zařízení jsou většinou připojena k centrální jednotce, která slouží k jejich nastavení a ovládání. Dobrým příkladem fungování chytré domácnosti může být její uvedení do nočního stavu. Chytrý termostat sníží večer teplotu, aby se uživateli lépe spalo. Dále se zamkne automatický zámek u vchodových dveří a aktivuje se pohybové čidlo. Zatáhnou se rolety, aby uživatele při spaní nerušila světla pouličních lamp. Nakonec se vypnou běžící spotřebiče jako například žárovky a rádio⁴.

Chytrý skleník – jeho hlavním cílem je upravovat prostředí ve skleníku tak, aby měly rostliny co nejlepší podmínky a šetřilo se energiemi [42]. Pro získání dat o aktuálních podmínkách využívá senzory teploty, vlhkosti nebo intenzity světla. Příkladem fungování může být úprava teploty a osvětlení na základě dat o aktuálních podmínkách ve skleníku. Systém navíc většinou umožňuje uživateli sledovat aktuální stav skleníku prostřednictvím chytrého zařízení. V neposlední řadě může systém také pomoci s včasnou detekcí škůdců.

Sport – IoT se využívá také při sportu. Příkladem jsou chytré hodinky, které nám během aktivity měří tep, rychlost pohybu, počet kroků nebo EKG [40]. Na základě těchto informací nám pak jsou schopny vytvořit vhodný tréninkový plán, vybrat sport nebo nás upozornit na dodržování pitného režimu. Dalším využitím IoT ve sportu je například rozhodování sporných situací. Příkladem může být posuzování ofsaidu na mezinárodních fotbalových šampionátech⁵. Ty jsou posuzovány na základě dat ze senzorů v míči a záběrů kamer umístěných kolem hrací plochy. Využívá se také například ve volejbale, gymnastice a skocích do vody.

Zdravotnictví – zde se technologie IoT využívají především k monitoringu stavu pacienta [6]. Ten může být vybaven senzory pro měření životních funkcí, tlaku nebo cukru v krvi. Data z těchto senzorů pak mohou lékaři vyhodnocovat v reálném čase. Zároveň je mohou využít chytré dávkovače léků. Dalším pozitivem je zjednodušení komunikace mezi lékařem, pacientem a jeho rodinou.

Průmysl – využívají se zde například pro monitoring závad systémů, sledování výkonu výroby nebo stavu výrobních zdrojů. Využití těchto technologií umožňuje také vyšší flexibilitu výroby [19].

Nebezpečí

Spolu s rostoucím počtem zařízení IoT roste i počet s nimi spojených rizik. Musíme je brát v úvahu a snažit se jim předcházet. Jedním z možných problémů je špatná identifikace zařízení v síti. Pokud síť není schopna odlišit od sebe některá zařízení, může dojít k poruše celého systému. Je nutné, aby každé zařízení v síti mělo své jedinečné ID, pomocí kterého bude snadno identifikovatelné [16]. Systém by zároveň měl být schopen přiřadit ID i novým zařízením, která jsou přidána do systému tak, aby bylo možné jej v budoucnu rozšiřovat.

Velmi vážným nebezpečím jsou pro IoT systémy kybernetické útoky. Výrobci se často snaží navrhnout zařízení tak, aby co nejvíce zvýšili jejich energetickou efektivitu [47]. Ta potom často mají malou výpočetní kapacitu a minimální zabezpečení. Toho mohou využít

⁴<https://www.loxone.com/cscz/chytry-dum/chytra-domacnost/>

⁵<https://ncscicentrail.org/science-blog/how-artificial-intelligence-has-enhanced-judging-at-the-paris-olympics/>

hackeři jako slabého místa a získat přístup do systému. Například pomocí vložení škodlivého kódu do zařízení IoT [10]. Zde pak mohou získávat citlivé informace jako například šifrovací klíč nebo přerušit komunikaci se zařízeními. Mohou také využívat rušičky signálu k narušení bezdrátové komunikace se zařízeními jako například bezpečnostní kamery. Hackeři rovněž mohou provést DDoS útok na tato zařízení s cílem celý systém vyřadit z provozu [47]. Příkladem může být napadení systému stavebního řízení ČR, které vedlo k částečnému výpadku služeb⁶.

Vzhledem k malé výpočetní kapacitě některých zařízení nejsou data v síti vždy dostatečně šifrována, což může vést k jejich odcizení [47]. Následně může dojít k jejich zneužití například pro personalizovanou reklamu. Odcizená data mohou rovněž obsahovat osobní údaje, které jde následně ilegálně prodávat⁷. Naše data může rovněž využívat provozovatel IoT systému, který k nim má přístup. Ten je může použít pro personalizovanou reklamu a analýzu našeho chování [16].

Mezi další problémy v práci se zařízeními IoT patří lidský faktor. Ten může způsobit problém už při navrhování systému. Vývojáři se při tvorbě snaží ošetřit co nejvíce kritických situací, které mohou nastat. Ne všechny hypotetické scénáře se však podaří pokrýt. Když se systém do takové situace dostane, může způsobit velké škody [16]. Některé systémy proto umožňují uživateli převzít kontrolu v případě nečekané situace. Nicméně uživatel nemusí být schopen včas a adekvátně zasáhnout, jelikož je zvyklý, že systém pracuje sám a nevěnuje mu proto pozornost. Běžně dostupná zařízení lze využít nejen k ulehčení práce, ale dají se také zneužít. Teroristé mohou například využívat běžně dostupné drony k útokům na infrastrukturu.

V neposlední řadě je nutné zmínit i socioekonomické problémy, které jsou s využíváním IoT spojené. Mohou například způsobit rozdělení společnosti a znevýhodnění těch, kdo nebudou schopni se s touto novou technologií naučit pracovat. Mohou také v mnoha oborech nahradit lidskou práci a způsobit rušení pracovních pozic. Jejich časté využívání může také oslabit naši paměť a naše kognitivní dovednosti [16].

2.5 Smart Cities

Pro pojem chytré město (*Smart city*) existuje mnoho různých definic. Žádná z nich není obecně přijímána a považována za jedinou správnou. Jednou z nich je definice od Washburn a spol. [52]. Ta definuje Smart city jako využití chytrých výpočetních technologií (*Smart Computing*) pro zlepšení kritické infrastruktury a služeb města. Mezi ně patří městská správa, vzdělání, zdravotní péče, veřejná bezpečnost, doprava a komunální služby.

Jde o koncept využívání moderních technologií ve městech pro zefektivnění jejich fungování a podporu udržitelného rozvoje. Často se zde využívají technologie internetu věcí. Příkladem může být chytré odpadové hospodářství. Odpadkové koše ve městě jsou vybaveny senzory a popeláři je tak vyváženou pouze v případě, že jsou plné. Jednotlivé systémy chytrého města pak mohou být navzájem propojeny a spolupracovat jako jeden organismus zajišťující jeho fungování. Často také využívají pomoci občanů, kteří mohou například hlásit problémy ve veřejném prostoru.

V poslední době se začíná stávat problémem přesídlování lidí z venkova do velkých měst (urbanizace). Podle dat světové zdravotnické organizace žije nyní v městských oblastech více

⁶Článek o napadení systému stavebního řízení: https://www.irozhlaz.cz/zpravy-domov/system-stavebniho-rizeni-celi-hackerskemu-utoku-hlasi-ministerstvo-cast-je-mimo_2503201951_hof

⁷<https://policie.gov.cz/clanek/zneuziti-osobnich-dat.aspx>

než 55 % světové populace⁸. Do roku 2050 by toto číslo mohlo stoupnout až na 68 %. Zvláště rychlý průběh urbanizačních procesů lze pozorovat v rozvojových zemích Afriky a Asie [54]. Města často nejsou připravena na tak velký nárůst obyvatel a trpí přelidněností, která může vést ke zhoršení podmínek k životu. Obyvatelé tak mohou trpět špatným ovzduším, znečištěním prostředí, vody a snížením hygienických nároků, což může vést k šíření nemocí. Dalšími možnými problémy jsou špatná dopravní obslužnost, nízká kvalita bydlení a zvýšená kriminalita v jejich okolí. Koncept *Smart city* se těmito problémům snaží zabránit. Aby mohly *Smart cities* správně fungovat, měly by být splněny některé podmínky [9]:

- Efektivní organizace a řízení všech systémů chytrého města.
- Využití moderních technologií k fungování města (podpora inovací systémů).
- Zajištění spolupráce mezi všemi zúčastněnými subjekty.
- Tvorba legislativy a nařízení podporujících rozvoj chytrých technologií ve městě.
- *Smart cities* by měly zohledňovat potřeby a přání obyvatel.
- Vybudování dostatečné síťové infrastruktury, která zajistí rychlý a bezpečný přenos dat a bude připravena na budoucí rozvoj systému.
- *Smart cities* by měly také sloužit k ochraně životního prostředí a přírodních zdrojů ve městě.

E-government označuje strategické a koordinované využívání informací a komunikačních technologií ve veřejné správě a politickém rozhodování [51]. Zahrnuje například online komunikaci s úřady, digitální podávání žádostí a digitální portál občana. Jejich cílem je zefektivnění fungování úřadů. *Smart cities* často využívají data, která úřady uchovávají v digitální podobě. Může jít například o otevřená data, která úřady zveřejňují (například data o počtech obyvatel ve městě). Ta pak může *Smart city* využít k efektivnějšímu fungování. Chytrá města mohou také využít data o občanech k jejich informování o mimořádných událostech a o změnách ve fungování města (například dopravní uzavírky). Občané pak mohou využívat portál občana k získání dat, které o jejich městě schraňuje systém *Smart city*.

Použití

Chytré zemědělství (*Smart agriculture*) – jeho cílem je co nejvíce optimalizovat výnosy, pracovní postupy a umožnit zemědělcům vyšší flexibilitu [38]. Zároveň vede k omezení pesticidů a lepšímu využití vodních zdrojů, čímž pomáhá chránit životní prostředí. Ke sběru dat se využívají senzory pro měření vlhkosti, teploty a půdní senzory. Dále se využívají například drony nebo satelitní snímky [37]. Získaná data pak mohou sloužit k přesnějšímu zavlažování a úspoře vody. Snímky z dronů mohou sloužit k odhadu nejvhodnějšího času sklizně a pomoci s její automatizací.

Odpadové hospodářství – slouží hlavně k efektivnějšímu svozu odpadu ve městě. Kontejnery mohou být vybaveny senzory, které monitorují jejich plnost⁹. Kontejner je následně vyvážen pouze v případě, že je plný. Na základě analýzy získaných dat lze taky kontejnery lépe rozmístit. Některé senzory jsou také schopné identifikovat poruchy nebo

⁸https://www.who.int/health-topics/urban-health#tab=tab_1

⁹<https://www.o2.cz/podnikatele-a-firmy/reference/mesto-kolin>

poškození kontejnerů. Moderní popelnice jsou také schopné odpad lisovat tak, aby nebylo nutné je tak často vyvážet¹⁰.

Doprava – zde jsou pro získání informací často používány dopravní kamery, sčítače projíždějících aut, nebo vozidel na parkovišti. Data z těchto senzorů se následně využívají k lepší koordinaci semaforů na křižovatkách nebo ke směrování aut na volná parkoviště [39]. Systém také může detekovat dopravní nehodu a následně upravit proměnné značení tak, aby se netvořily kolony. Informaci o havárii také může předat do dopravních aplikací, které o události informují své uživatele a upraví jim trasu jízdy.

Bezpečnost – policie běžně využívá radarová zařízení pro měření rychlosti vozidel, na jejichž základě jsou následně automaticky generovány pokuty za překročení stanoveného limitu¹¹. Města jsou dnes také vybavena velkým množstvím kamer. Data z nich jsou využívána k identifikaci trestných činů a přestupků (například jízda na červenou) [25]. Záznamy z kamer pak mohou sloužit jako důkazní materiál.

Informování občanů – prostřednictvím aplikace připojené k Smart city může občan získávat aktuální informace o svém městě¹². Zároveň do ní může například zadávat informace o závadách, které by mělo město opravit¹³.

¹⁰<https://www.komunalniekologie.cz/info/olomouc-testuje-chytre-kose-na-odpad>

¹¹https://olomoucky.denik.cz/zpravy_region/radar-olomouc-prerovska-chvalkovice-chomoutov-2024.html

¹²<https://opendata.ostrava.cz>

¹³Portál města Olomouc pro hlášení závad: <https://hlaseni.olomouc.eu>

Kapitola 3

Vizuální programování

Pro správné fungování zařízení internetu věcí (IoT) je nutné, aby spolu komunikovala a tvořila dohromady jeden celek. Jako příklad můžeme uvést systém pro klimatizaci a obměnu vzduchu. Ten se skládá z několika senzorů: teploměr, čidlo pro měření oxidu uhličitého ve vzduchu, snímač spotřeby elektrické energie v budově. Dále obsahuje klimatizační jednotku, která je schopna měnit vzduch v místnosti a ochlazovat jej.

Klimatizace začne ochlazovat prostor pouze v případě, že teploměr ukazuje zvýšenou teplotu v pokoji, mění se hladina oxidu uhličitého (místnost není prázdná) a spotřeba elektřiny v budově nepřekračuje danou hranici (nedojde k přetížení sítě). Vzduch se začne měnit, pokud procento oxidu uhličitého v místnosti stoupne nad stanovenou hranici.

Nadefinování chování a propojení celého systému si většinou koncový uživatel chce provést sám podle svých potřeb a možností. Zároveň je vhodné, aby mohl provádět v případě potřeby změny. Proto byly vytvořeny nástroje, které mu umožní programovat si zařízení internetu věcí sám. Koncový uživatel ovšem většinou neumí programovat a napsat kód v některém z klasických programovacích jazyků pro něj může být náročné [26]. Proto tyto nástroje využívají různé postupy, jak umožnit vývoj i bez znalostí programování. Jedním z nejvyužívanějších přístupů je využití vizuálního programování.

3.1 Tvorba programů koncovými uživateli

Problematice tvorby softwaru koncovými uživateli se věnuje obor *End-User development* (EUD). Ten se zabývá metodami, technikami a nástroji, které umožňují uživatelům softwaru, kteří nejsou profesionální vývojáři, tvořit, upravit nebo rozšířit počítačový program [26]. Využití těchto technik nabízí řadu výhod nejen pro koncové uživatele, ale i pro softwarové vývojáře.

Výhody

Mezi hlavní výhody patří možnost tvorby programu lidmi se znalostí oblasti, kde bude software nasazen [35]. Ve chvíli, kdy je potřeba vytvořit nový program, musí budoucí uživatel vývojáři přesně nadefinovat své požadavky. Tento proces může být komplikovaný. Profesionálnímu vývojáři často chybí znalosti dané domény a běžně využívaných konceptů. Naopak koncoví uživatelé nejsou vždy schopni přesně specifikovat své požadavky a principy, které využívají. Následně může dojít k tomu, že nově vytvořený software není pro zadavatele vyhovující. Metody EUD umožňují koncovému uživateli vytvořit si program sám na základě

znalostí, které o dané problematice má. Výsledný program je tak často vhodnější, jelikož koncoví uživatelé znají svou vlastní doménu a potřeby lépe než kdokoli jiný [24].

Dalším pozitivem je zrychlení celého vývojového cyklu [35]. V průběhu času se uživateli mění požadavky a je nutné podle nich upravit výsledný program. Aby vývojář mohl změny správně implementovat, musí od zákazníka získat nové parametry softwaru. Tento proces je popsán v předchozím odstavci. Pokud je koncový uživatel schopen zanést změny do softwaru sám, dojde k urychlení celého procesu. Zároveň je možné dělat změny častěji a lépe tak reagovat na nově vzniklé požadavky.

Tyto nástroje také mohou pomoci vývojářům v jejich práci. Lze je například využít při společném vývoji (*co-development*) s uživatelem [35]. Ten může pomocí některého z nástrojů EUD spolupracovat při návrhu systému. V některých případech může být zapojen i do jeho kódování. Vývojář tak získá lepší povědomí o požadavcích na výsledný program i o celkové problematice.

Podobory a metody

EUD je rozsáhlé odvětví informatiky, které můžeme rozdělit do podoborů. Mezi ně patří například End-User programming (EUP), který se zaměřuje na tvorbu vlastních programů uživatelem. Dalším podoborem je End-user Software Engineering (EUSE), který se zaměřuje na kvalitativní parametry výsledného softwaru jako jsou opětovná použitelnost a bezpečnost [24]. EUD je také možné rozdělit podle technik, které jsou při vývoji programu koncovým uživatelem využívány. Mezi ně patří například vizuální programování, které je popsáno v kapitole 3.2.

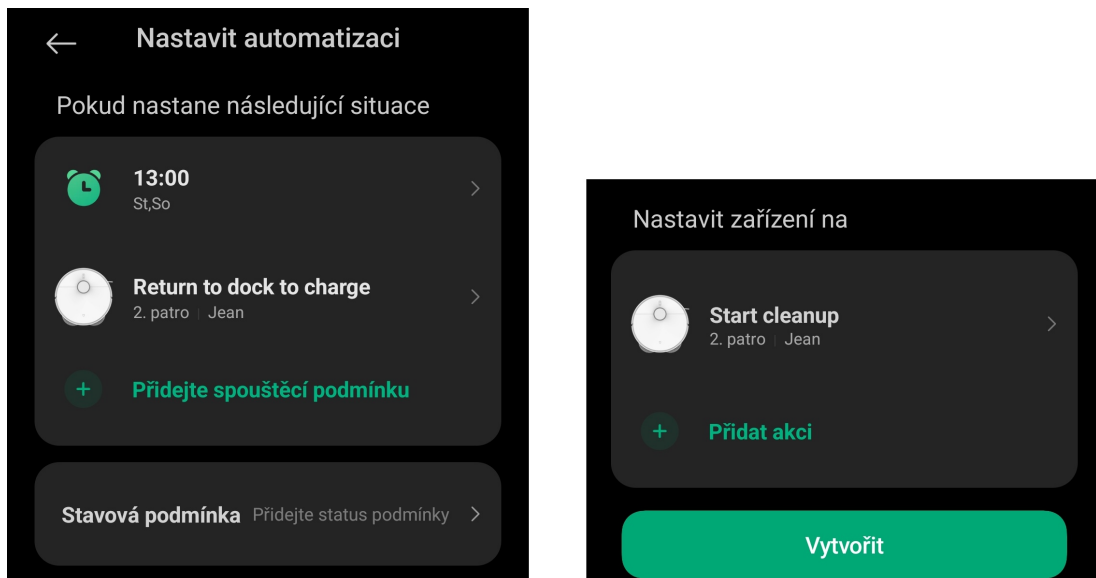
Jednou z dalších technik je tvorba designu pomocí umísťování komponent. Ta se uplatňuje například ve vizuálních WYSIWYG editorech (*What You See Is What You Get*). Uživatel v takovýchto aplikacích vidí na začátku prázdnou plochu, která představuje aktuální vzhled webové stránky. Do této oblasti může umísťovat přetažením z nabídky různé komponenty (například tlačítka, textová pole, nadpisy). Těm může upravovat barvu, velikost textu a další parametry. Postupně tak z těchto komponent skládá finální vzhled webové stránky. Podobně se postupuje i v aplikacích pro tvorbu designu uživatelského rozhraní nebo *mockup*. Díky tomu se těmito editory webových stránek dobře pracuje nejen programátorům, ale například i webovým designérům nebo manažerům. Na základě rozmístění komponent a jejich parametrů editor vygeneruje kód webové stránky (například v programovacích jazycích HTML a CSS). S tím pak může dále pracovat programátor a upravit ho do finálního stavu. Některé nástroje také vygenerují napojení na frontend. Jedním z těchto nástrojů je například Elementor¹, který se využívá při tvorbě webových stránek v aplikaci Wordpress.

Další často používanou technikou je trigger-action programming (TAP). Při programování pomocí této metody uživatel definuje chování pomocí dvou komponent událost (*trigger*) a akce [49]. Trigger obsahuje podmínky, které specifikují danou událost. Pokud je splněn, provede se akce definovaná ve druhé části. Jinými slovy, pokud nastane událost A (například senzory detekují zvýšení teploty), provede se akce B (spuštění klimatizace). Mezi její hlavní výhody patří jednoduchost a intuitivnost pro koncové uživatele. Tato technika je často využívána při programování IoT zařízení v chytré domácnosti. Využívá ji například webová služba IFTTT (*if this, than that*)². Mezi její hlavní výhody patří propojení s různými dalšími aplikacemi (například sociálními sítěmi) a široká podpora zařízení chytré domácnosti

¹Domovská stránka nástroje Elementor: <https://elementor.com/>

²Domovská stránka IFTTT: <https://ifttt.com/>

od různých výrobců. IFTTT tak umožňuje programovat dohromady zařízení od různých společností. Trigger-action programming využívá také mobilní aplikace Xiaomi Home. Ta je určená pro řízení chytré domácnosti obsahující zařízení od společnosti Xiaomi. Na vstupní obrazovce nabízí seznam všech připojených IoT přístrojů. Po rozkliknutí kteréhokoliv z nich se nám zobrazí možnosti jeho ovládání. TAP je zde využito pro nastavení automatizace, jejíž příklad můžeme vidět na obrázku 3.1. Trigger-action programming využívá například také webová aplikace Blink určená pro správu a integraci zařízení internetu věcí.



Obrázek 3.1: Na obrázku je příklad vytváření automatizace pomocí trigger-action programming v aplikaci Xiaomi Home. V první části si uživatel nadefinuje podmínky spuštění, jako například čas a aktuální stav vysavače. Následně pak určí, jaká akce se za takové podmínky má provést. Na obrázku jde o úklid. Jak podmínky, tak akce se přidávají pomocí tlačítka plus. Na konci je tlačítko pro uložení výsledné automatizace.

3.2 Principy vizuálního programování

Pro zjednodušení vývoje softwaru koncovým uživatelem se často využívá vizuální programování (VP). U této techniky uživatel definuje program ve dvou a více rozměrném prostoru [55]. Nevyužívá se zde psaní programu v textové podobě, ale práce s grafickými komponentami [22]. Tento přístup je pro mnoho lidí srozumitelnější, protože vizualizace problému tvoří základ jejich kreativního myšlení. Zároveň odbourává nutnost převádět vizuální nápady do čistě textové podoby. Na této myšlence jsou postaveny vizuální programovací jazyky (VPL). Většina nástrojů využívajících VP je určena pro průměrné koncové uživatele [24]. Tento koncept se začal více rozvíjet v 80. letech 20. století a jeho vývoj trvá dodnes. VP se snaží dosáhnout tří základních cílů: lepší pochopení sémantiky (významu) programových struktur, pochopení vytvořených programů v kontextu specifických situací a zjednodušení dodržování syntaktických pravidel [43].

Vizuální programování obvykle poskytuje uživatelům vysokou míru abstrakce [24]. Složitě programové konstrukce jsou pro ně skryty za grafickými prvky, se kterými pracují (například je mohou přetahovat, nebo vybírat z menu). Tyto elementy jsou pro člověka

snadnější na porozumění. Ten má díky tomu při využití vizuálního programování pozitivnější uživatelský zážitek, než při práci s některým z textových programovacích jazyků [7]. Díky abstrakcím se stává vizuální programování více dostupným i pro uživatele bez znalostí programování [24]. Ti mohou snáze pracovat a přesněji definovat svůj záměr. Díky porozumění významu samotných prvků se uživateli lépe chápe program jako celek a může ho poté lépe využít v konkrétních situacích.

Zároveň i přes vysokou míru abstrakce si VP zachovává základní principy programování. Ty jsou podobné ve většině programovacích jazyků. Proto se nástroje VP často využívají k tomu, aby se začátečníci seznámili a pochopili základy programování [22]. Jsou zde odstíněny složité konstrukce klasických programovacích jazyků a student se tak může soustředit pouze na pochopení elementárních pravidel.

Vizuální programování se také snaží zamezit tvorbě syntaktických chyb uživatelem [24]. Syntax jsou pravidla, která definují, jak můžeme sestavit jazykové komponenty, aby byl výsledný program funkční. Pokud nejsou dodržena, výsledný program se nepodaří spustit. Aby se zamezilo syntaktickým chybám, je práce s grafickými elementy ve VP často omezena tím, že syntaxe příkazů je již implementována do vizuálních tvarů příkazů [22]. Uživateli je povoleno provádět pouze akce (například propojení prvků), které jsou syntakticky správně. Snižuje se tak pravděpodobnost vzniku syntaktických chyb.

Další výhodou VP je uživatelská přívětivost. Obvykle jsou všechny potřebné komponenty pro tvorbu programu předdefinované [24]. Uživatel je tak pouze vybírá z nabídky a nemusí si je hledat, nebo sám vytvářet. Zároveň jsou nástroje využívající vizuální programování často vybaveny nápovědou, jelikož jsou primárně určené pro uživatele, kteří nemají moc zkušeností s programováním. Většina nástrojů VP zároveň umožňuje uživateli se v práci zdokonalovat a tvořit komplexnější software. Na druhou stranu omezené množství nástrojů a nemožnost vytvořit si vlastní může tvorbu komplexnějších programů komplikovat.

Mezi problémy VP patří například téměř nemožná optimalizace [41]. Grafické prvky většinou obsahují složité programové konstrukce. Ty se snaží zahrnout všechny možné úkony, které může uživatel udělat, a ošetřit potenciální chyby. Vzhledem k jejich komplexnosti a složitosti je těžké je optimalizovat.

Vizuální programování se využívá například v oblasti vzdělávání, počítačových her, simulacích systémů a robotice [41]. V problematice internetu věcí se využívá k propojení různých objektů pomocí jednotné platformy, kterou je schopen ovládat koncový uživatel.

Vizuální programovací jazyky

Vizuální programovací jazyk (*visual programming language*, zkráceně *VPL*) je druh programovacího jazyka, který umožňuje tvorbu programu pomocí práce s grafickými elementy. Ty lze například prostorově uspořádat, vzájemně propojovat šipkami nebo zanořovat do sebe [41]. Tyto komponenty může uživatel většinou přetahovat z menu, kde jsou znázorněny textově, nebo graficky. Vizuální programovací jazyky mají různé vlastnosti. Zda a do jaké míry daný VPL splňuje danou vlastnost většinou závisí na účelu a skupině koncových uživatelů, pro které je určen. Mezi důležité rysy, které můžeme pozorovat patří [41, 4]:

- **Rozšiřitelnost** – jde o jeden z důležitých aspektů pro použití v IoT. Mnoho VPL má pouze omezený počet prvků, se kterými může uživatel pracovat. U programování zařízení internetu věcí se předpokládá, že do systému může být přidán člen s novou funkcionalitou. O tu je následně nutné rozšířit VPL tak, aby s ní uživatel mohl pracovat. Dále může uživatel narazit na okrajové případy, které je složité vyřešit pomocí běžných komponent a je potřeba pro ně vytvořit nové prvky VPL se specifickou ope-

rací. Zároveň by měl uživatel mít možnost vytvořit si své vlastní funkce. Ty může využít ke zjednodušení komplexních programů tak, aby ho vizuální programovací jazyk neomezoval při práci.

- **Rychlost generování kódu** – v systému IoT může být zároveň připojeno mnoho členů, které musí spolu komunikovat. Je proto nutné, aby počet vyhodnocených zpráv za určitý čas byl co nejvyšší a nedocházelo k přetížení systému. U VPL je náročné zajistit vysoký výkon, jelikož mají často vysokou úroveň abstrakce a je obtížné je optimalizovat.
- **Úroveň abstrakce** – VPL často využívají vysokou úroveň abstrakce, aby s nimi mohli pracovat i lidé, kteří nemají znalosti v programování. Zároveň malá míra abstrakce může vést ke vzniku komplexních VPL programů.
- **Integrace** – VPL by měl být vytvářen spolu s grafickým editorem, ve kterém se bude používat, aby spolu co nejlépe pracovali.
- **Podpora nasazení v cloudu a na různých zařízeních** – často nevíme, na jakém zařízení bude koncový uživatel s VPL pracovat. Je proto nutné zajistit fungování na strojích s různými operačními systémy, které mohou mít různý výkon a zdroje. Vše by také mělo fungovat v cloudovém prostředí.
- **Použitelnost** – u VPL je nutné zvážit, pro jakou oblast bude využíván a jakými koncovými uživateli. Jaké mají vzdělání, znalosti v dané oblasti a zkušenosti s programováním.

3.3 Typy vizuálních programovacích jazyků

Vizuální programovací jazyky lze podle podle Kuhail a spol. [24] rozdělit do čtyř základních skupin. Ty se liší hlavně typem grafických prvků, na kterých je jazyk založen. Koncepty jednotlivých typů vizuálních programovacích jazyků lze kombinovat a VPL tak nemusí patřit pouze do jedné skupiny. Následující sekce popisuje jednotlivé kategorie spolu s příklady jejich využití.

Blokové programovací jazyky

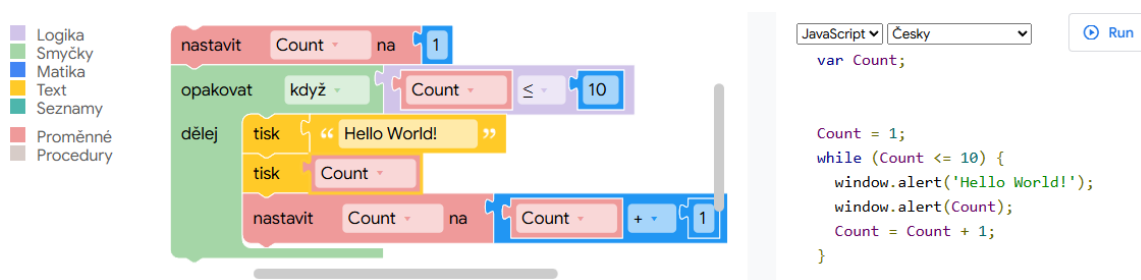
Blokové programovací jazyky (*Block-based languages*) jsou založeny na grafických prvcích ve formě bloků [24]. Ty se většinou nachází v nabídce, odkud je uživatel může přetahovat na pracovní plochu. Tam je pak může spojovat dohromady, nebo zanořovat do sebe. Možnosti propojování bloků jsou přesně definovány pravidly, která zabraňují vzniku syntaktických chyb. Díky tomu jsou méně komplexní a vhodné pro koncové uživatele. Ti se nemusí soustředit na detaily syntaxe a mohou se více věnovat logice programu jako celku.

Jedním z nástrojů využívajících tento typ jazyka je knihovna Blockly od společnosti Google. Ta slouží k přidání grafického editoru pro vizuální programování do webových a mobilních aplikací³. Kód knihovny je přístupný zdarma pod svobodnou licencí Apache 2.0. Editor využívá blokový programovací jazyk Blockly. Výsledný program je pak schopen exportovat v jednom z podporovaných programovacích jazyků. Mezi ně patří například JavaScript, Python, PHP, XML. Blockly je kompatibilní s hlavními webovými prohlížeči

³<https://github.com/google/blockly>

a pracuje jak ve webovém rozhraní, tak na mobilních zařízeních (*works on both web and mobile*)⁴.

Editor umožňuje vytvářet programy pomocí přetahování bloků z nabídky na pracovní plochu, jak je ukázáno na obrázku 3.2. Tyto komponenty mají předdefinovanou funkcionalitu a mají barvu podle skupiny, do které patří (například logické bloky, smyčky, textové bloky, proměnné). Pokud blok obsahuje nějaké parametry (například typ cyklení), je pro ně v bloku vyhrazeno prázdné místo. Do něj je lze doplnit výběrem z nabídky, nebo textovým vyplněním hodnoty. Jednotlivé bloky mají výstupky různého tvaru. Aby bylo možné bloky spojit, musí do sebe výstupky zapadat. Díky tomu lze propojovat bloky pouze tak, aby nevznikaly syntaktické chyby. Výsledný program je možné v editoru spustit, nebo zobrazit v některém z podporovaných programovacích jazyků.

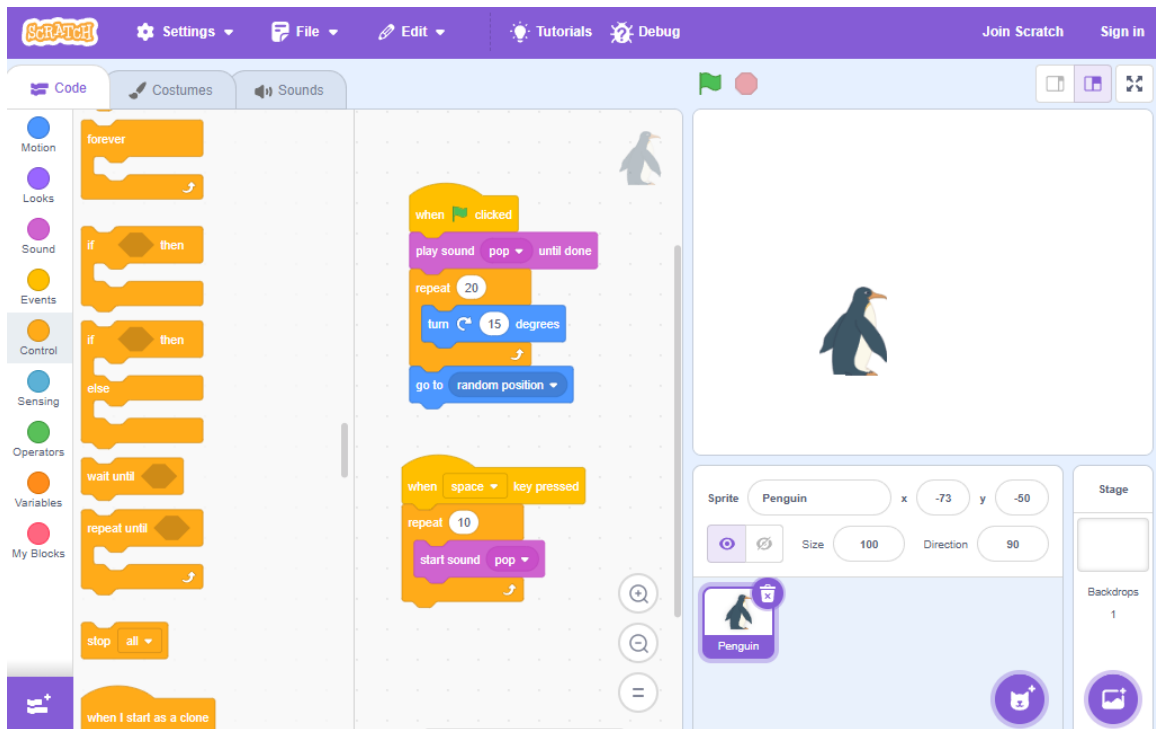


Obrázek 3.2: Grafický editor Blockly v české jazykové mutaci. V pravé části může uživatel provádět úpravy ve vygenerovaném kódu. Zároveň zde může přepínat jazyk aplikace, programovací jazyk, ve kterém je kód vygenerován, a spouštět program. V levé listě je nabídka všech typů komponent. Zde je možné si daný typ rozkliknout a přetáhnout si z něj konkrétní prvek. Prostřední část tvoří prostor pro sestavování programu. Bloky lze spojovat a zanořovat.

Knihovna Blockly je využívána v mnoha nástrojích určených pro vzdělávání. Patří sem například Scratch, Code, Open Roberta nebo Wonder Workshop. Využívá se také v nástroji Smart Block určeném pro programování IoT zařízení [5].

Jedním z VPL, které využívají knihovnu Blockly je Scratch, se kterým lze pracovat ve webovém editoru. Umožňuje uživateli vytvářet vlastní animace, hry a interaktivní umění pomocí skládání programů z grafických bloků [31], jak můžeme vidět na obrázku 3.3. Je určen hlavně pro děti ve věku 8 – 16 let a lidi, kteří začínají s programováním [44]. Je dostupný ve více než 70 jazycích. Jeho cílem je hravou formou podpořit počítačové myšlení a naučit uživatele základní principy programování. VPL Scratch si zachovává principy programovacích jazyků, jako jsou práce s proměnnými, využívání cyklů a tvorba funkcí. Podle Repenning [43] patří právě jejich pochopení k jednomu ze základních cílů vizuálního programování. Uživatel prací s tímto jazykem pochopí tyto techniky a naučí se je využívat. Zároveň se naučí dělit problémy na podproblémy, které je schopen řešit. Díky tomu je pro něj následně jednodušší začít pracovat v některém z klasických programovacích jazyků, jako je například Java. Mezi nástroje využívající blokový programovací jazyk patří například také mobilní aplikace Puzzle.

⁴Stránka nástroje Blockly: <https://developers.google.com/blockly>



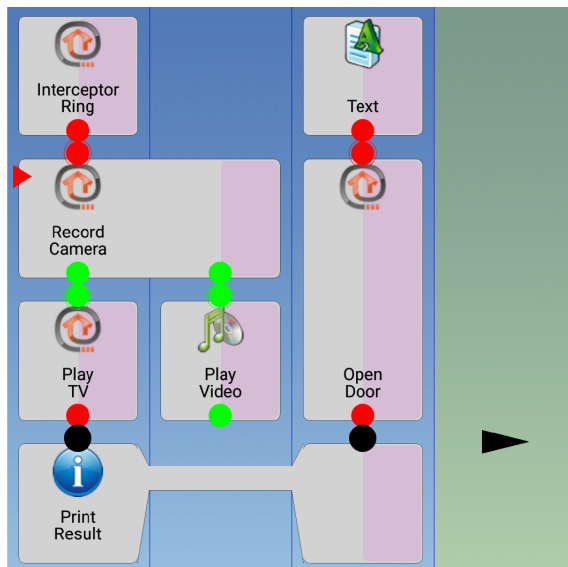
Obrázek 3.3: Grafický editor Scratch. V pravé horní části obrazovky vidíme postavičku tučňáka, kterou si uživatel vybral v pravé dolní části. V levé části lze postavičce nadefinovat chování jako například zvuk, pohyb nebo vzhled. Uživatel si vybere jednu z nabízených komponent a přetáhne ji do programu. Na obrázku je program, který vydá zvuk „pop“, dvacetkrát otočí tučňáka o 15 stupňů a přesune na náhodné místo. Pokud uživatel zmáčkne mezerník, ozve se desetkrát „pop“.

Ikonové programovací jazyky

Základním prvkem ikonových programovacích jazyků (*Icon-based languages*) jsou ikony a grafické symboly, které reprezentují různé předměty a akce [24]. Ikony můžeme rozdělit do dvou skupin. První jsou jednoduché ikony. Ty představují jednu akci (například vyfotografovat) nebo objekt (například kamera). Z těchto jednoduchých ikon pak lze sestavit složité ikony, které již mají komplexnější funkcionalitu. Práce s ikonami je intuitivní a tyto VPL jsou tak vhodné pro koncové uživatele bez znalostí programování.

Příkladem tohoto typu VPL je MicroApp, který slouží k vytváření jednoduchých aplikací [14]. Hlavními komponentami jazyka jsou zaoblené obdélníky obsahující ikony. Tento jazyk je založen na přístupu skládání služeb dohromady, a proto všechny ikony představují služby [14]. Ikony v komponentách reprezentují kategorii objektu, kterou daná služba zpracovává. Na okrajích ikon jsou značky, které symbolizují vstupy a výstup dané služby. Značky pro vstupy jsou ve vrchní části ikony a pro výstup na její spodní hraně. Barva těchto značek udává typ vstupu/výstupu. Například žlutá barva symbolizuje e-mail. Tyto symboly mohou mít tvar kolečka nebo trojúhelníku. Kolečka znamenají dynamický vstup/výstup, který je přiřazen až za běhu programu. Zatímco trojúhelníky značí statický vstup nastavený před spuštěním. Ikony uživatel pak skládá na pracovní ploše do předpřipravených řádků a sloupců, jak je ukázáno na obrázku 3.4. Vstupy a výstupy ikon do sebe musí sedět,

aby je bylo možné propojit. VPL umožňuje uživateli v případě potřeby vytvořit most přes prázdné pole, nebo jinou ikonu pro propojení dvou komponent.



Obrázek 3.4: Program vytvořený v aplikaci MicroApp. Ten nám říká, že pokud zazvoní zvonek, spustí se kamera u dveří, která přenáší obraz na televizi a na mobilní zařízení (*PlayVideo*). Pokud přijde od uživatele správný text, tak se dveře otevřou a zároveň se uloží záznam z kamery (převzato z literatury [14]).

Diagramové programovací jazyky

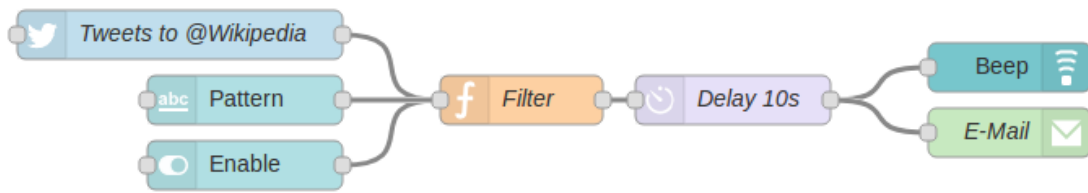
Pro Diagramové programovací jazyky (*Diagram-based visual programming languages*) se také používají pojmy *diagrammatic* a *data flow languages*. Jsou postaveny na konceptu propojování grafických objektů za pomoci šipek nebo čar, které představují vztahy mezi danými objekty [24]. Například jakým směrem v programu proudí informace. Výsledný program je reprezentován diagramem, ve kterém lze vidět posloupnost propojených grafických objektů. Jejich funkcionalitu je pak možné postupně vykonávat podle pořadí v diagramu.

Jedním z nástrojů, které využívají diagramový programovací jazyk je Node-RED od společnosti IBM [4]. Je určen k propojení hardwarových zařízení, API a online služeb [34]. Součástí nástroje je webový grafický editor, který je na obrázku 3.5. Je určen hlavně na použití na desktopovém zařízení.

Program vytvořený v Node-RED se skládá z jednotlivých uzlů, které jsou propojeny cestami. Jednotlivé uzly mají danou funkcionalitu a zpracovávají data ze svého vstupu na výstup. Cesty definují toky (*flows*) dat v programu. Spojují výstupy uzlů se vstupy jiných. Určují jak budou data procházet programem a postupně modifikována. Nástroj umožňuje uživateli uložit si funkce a programy.

Node-RED má velkou komunitu, která sdílí vytvořené programy a je neustále vyvíjen. Využívá se často pro správu IoT zařízení. Je použit například v nástroji od společnosti Go-IoT nebo Sence Technic [34]. Také ho můžeme najít v nástrojích pro řízení chytrých měst jako například Snap4City [4].

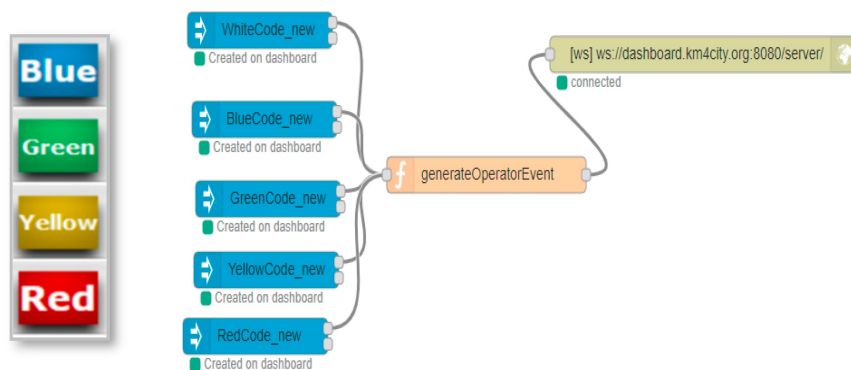
⁶Převzato z: <https://en.wikipedia.org/wiki/Node-RED>



Obrázek 3.5: Ukázkový program vytvořený v Node-RED, kde na vstupu jsou tweety pro @Wikipedia, vzor pro třídění (*Pattern*) a přepínač na zapnutí programu. Pokud je přepínač ve stavu povoleno, vezmou se příchozí zprávy a podle vzoru se přefiltrují. Následně se počká deset sekund a poté se odešle e-mail a spustí se zvukové upozornění⁶.

Snap4City je platforma pro správu a řízení chytrých měst. Slouží ke sběru, analýze a vizualizaci dat z IoT zdrojů ve městech. Využívá model řízený daty (*data-driven model*). Umožňuje získávat údaje ze senzorů, předvídat nečekané události a reagovat na ně nebo provádět analýzu získaných dat [46]. Pomáhá k efektivnímu využití zdrojů, prevenci kriminality a ochraně životního prostředí. Jelikož ve městech pracuje s velkým množstvím dat, využívá různé datové modely pro jejich efektivní ukládání (například RDF) [4].

Nástroj umožňuje tvorbu vlastních IoT aplikací. K tomu je využívána rozšířená verze vizuálního programovacího jazyka Node-RED a jeho grafický editor. Tento přístup umožňuje jednoduchou tvorbu a nasazení aplikací i uživatelům bez programátorských znalostí. Příklad můžeme vidět na obrázku 3.6. Jako uzly jsou používány základní uzly Node-RED a Snap4City MicroServices [4]. Snap4City MicroServices jsou předdefinované komponenty, které reprezentují služby dostupné na platformě (například správa a propojení IoT zařízení, přístup k městským entitám a jejich vztahům). Snap4City obsahuje více než 150 Microservices [4]. Pro aplikaci IoT si také může uživatel vytvořit uživatelské rozhraní ve formě řídicího panelu (*dashboard*). Uživatel si ho může sestavit pomocí výběru datových zdrojů a widgetů, nebo vygenerovat podle diagramu aplikace, který si vytvořil pomocí Node-RED.

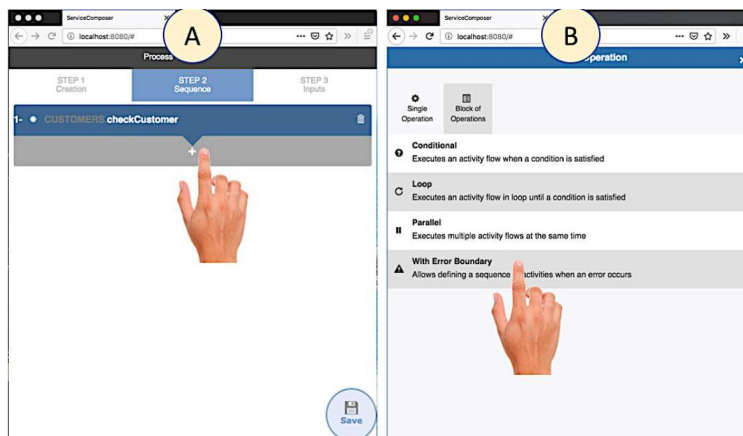


Obrázek 3.6: Ukázka aplikace vyrobené v nástroji Snap4city. Uživatel zde kliknutím na jedno z tlačítek zašle vstupní data. Na základě nich se vygeneruje událost, která je poté odeslána k provedení na server (převzato z literatury [4]).

Formulářové programovací jazyky

Programy u formulářových programovacích jazyků (*Form-based languages*) jsou založeny na postupném doplňování formuláře, který reprezentuje výsledný program [24]. Přidávání prvků je řešeno výběrem nebo přetažením z nabídky. U některých jazyků tohoto typu je možné doplnit některé informace textově. Například uživatel může textově do komponenty doplnit podmínku jako matematický vzorec, který se odkazuje na jiné komponenty.

Příkladem je mobilní aplikace EUCalipTool, která využívá formulářový programovací jazyk DSML (*Domain Specific Modeling Language*) [50]. Je určena k intuitivnímu vytváření mikroslužeb pro mobilním zařízení. Příklad tvorby programu můžeme vidět na obrázku 3.7.

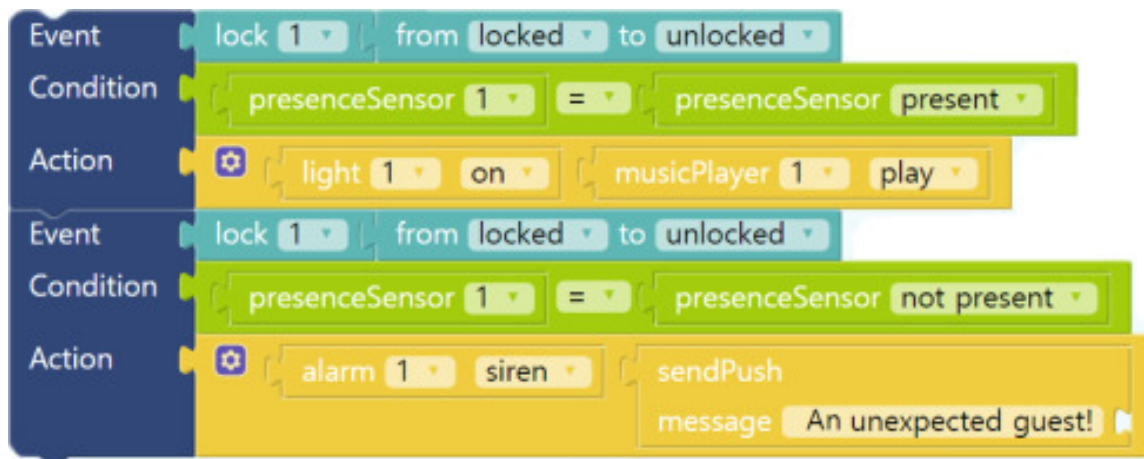


Obrázek 3.7: Na obrázku A vidíme, jak uživatel přidává první komponentu do programu pomocí tlačítka plus. Na snímku B nejdříve vybere, zda chce přidat blok operací, nebo pouze jednu a následně si vybere z nabídky (převzato z literatury [50]).

Vizuální komponenty se do výsledného programu přidávají postupně pod sebe pomocí tlačítka plus. Dělíme je na jednotlivé operace a fragmenty *single operations, fragments* [50]. První jmenované jsou samotné služby využité v programu. Fragmenty reprezentují struktury využívané v programování, jako jsou například smyčky a větvení. Tento typ komponent může navíc obsahovat parametry (například podmínku), které specifikují jeho chování. Výsledný program je transformován do spustitelných BPMN (*Business Process Model and Notation*) specifikací. Během generování je program rozdělen na BPMN kousky, které obsahují mikroslužby, které se vzájemně podporují [50].

3.4 Nástroje vizuálního programování pro IoT zařízení

Existuje mnoho druhů IoT zařízení od různých výrobců, které využívají širokou škálu technologií. Z toho důvodu se pro ně využívají různé programovací jazyky jako například C, C++, Python, Java. Aby spolu mohla jednotlivá zařízení tvořit systém, je třeba jim nastavit chování. Jádrem mnoha přístrojů internetu věcí (IoT) jsou mikrokontrolery nebo vývojové desky (např. Arduino nebo Raspberry Pi). Ty je nutné naprogramovat tak, aby bylo zajištěno jejich správné chování. Pro programování IoT zařízení byly vytvořeny nástroje, které umožňují koncovým uživatelům definovat jejich chování pomocí vizuálních programovacích jazyků bez znalostí programování. Některé z těchto nástrojů byly zmíněny v předchozí kapitole. V této části se zaměříme na další běžně používané možnosti.



Obrázek 3.8: Ukázka dvou ECA bloků, které oba mají stejnou událost (odemknut zámek 1), ovšem rozdílnou podmínku. V prvním bloku je vyhodnocena podmínka, zda je aktivní presenceSensor. Pokud ano, zapne se světlo a hudba. Druhý ECA blok obsahuje opačnou podmínku. Pokud platí, je aktivován alarm a odeslána zpráva uživateli (převzato z literatury [5]).

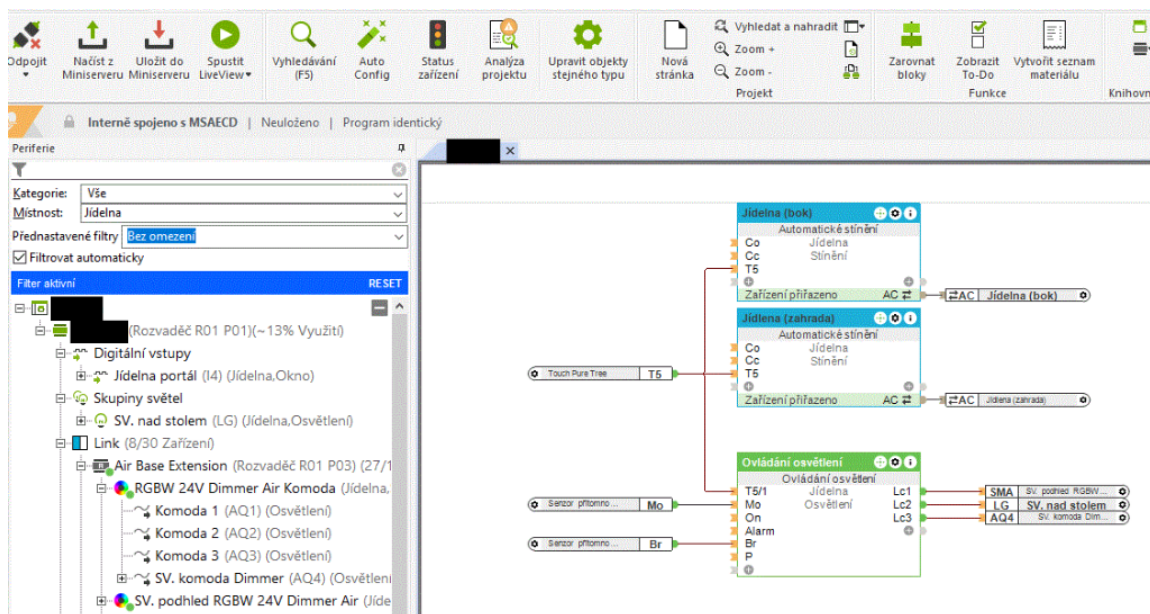
Smart Block

Smart Block je vizuální blokový programovací jazyk, který je možné použít pro aplikace postavené na platformě *SmartThings* od společnosti Samsung [5]. V jazyce je možné sestavit program, který je následně přeložen do programovacího jazyka Groovy⁷. Ten je využíván zařízeními platformy SmartThings. Smart Block tak zjednodušuje programování zařízení SmartThings a zpřístupňuje ho i běžným uživatelům. Implementace Smart Block je založená na nástroji Blockly, který rozšiřuje o vlastní ECA bloky. ECA bloky definují pravidla, která jsou vkládána do výsledného programu. Zkratka ECA znamená událost-podmínka-akce (*event-condition-action*). Z názvu tedy vyplývá, že každé ECA pravidlo je složeno ze 3 částí [5]. Pokud je vyvolána událost (například je aktivován senzor pohybu), je zkontrolována druhá část pravidla podmínka. Pokud je splněna, provede se akce, definovaná ve 3. části ECA bloku. Systém ECA pravidel je pro uživatele bez technického vzdělání snadno pochopitelný a umožňuje jim tvořit programy bez znalostí programování.

Loxone Config

Loxone Config je software, který umožňuje konfiguraci chytré domácnosti využívající zařízení společnosti Loxone [30]. Tato inteligentní instalace je řízena centrální jednotkou Miniserver. Aplikace Loxone Config slouží k nastavení chování domácnosti a nahrání výsledného programu na Miniserver. Můžeme ji vidět na obrázku 3.9. Také umožňuje simulaci a vizualizaci chování celého systému i bez připojení k centrální jednotce. Pro nastavení chování je využíván diagramový programovací jazyk. Základem jsou předpřipravené funkční bloky, kterým lze nastavovat parametry. Tyto komponenty lze následně propojovat pomocí hran. Software má také možnost automatického programování. Zde uživatel nastaví místnosti a jaké chování zařízení je v nich požadováno. Následně je vytvořen základní program, který lze dále upravovat.

⁷Programovací jazyk Groovy <https://groovy-lang.org/>



Obrázek 3.9: Ukázka aplikace Loxone Config, určené pro programování chytré domácnosti od společnosti Loxone. Tato aplikace se skládá z menu, seznamu dostupných periférií a pracovní plochy pro tvorbu programu. Menu se nachází v horní části obrazovky a obsahuje základní funkce programu. Nejvíce prostoru zabírá editor programu. V něm můžeme vidět několik senzorů (nalevo), řídicí bloky programu (uprostřed) a prováděné akce (napravo). Program definuje ovládání osvětlení jídelny.

ACADA

ACADA (*Asset Control and Data Acquisition*) je cloudová platforma pro správu IoT zařízení a sběr jejich dat [29]. Byla vytvořena společností Logimic. Zajišťuje připojení nových zařízení ke cloudu, jejich nastavení, vzdálené ovládání, monitorování přenášených dat a jejich analýzu. Je schopna předvídat možné problémy a informovat o nich uživatele. Ten může pomocí platformy zařízeními zasílat příkazy a včas tak zasáhnout.

Uživatel komunikuje s platformou pomocí webové aplikace, která je přizpůsobena pro mobilní i desktopová zařízení, jak můžeme vidět na obrázku 3.10. Informace je možné v aplikaci zobrazit pro jednotlivé objekty nebo pro skupinu. Skupina spojuje různá zařízení, která řeší jednu problematiku (například řízení retenčních nádrží).

ACADA umožňuje práci s IoT zařízeními od různých výrobců, které využívají odlišné technologie. Díky tomu může uživatel jednoduše zajistit komunikaci mezi více systémy internetu věcí. Využívá se například v chytrých městech, průmyslu nebo zdravotnictví.

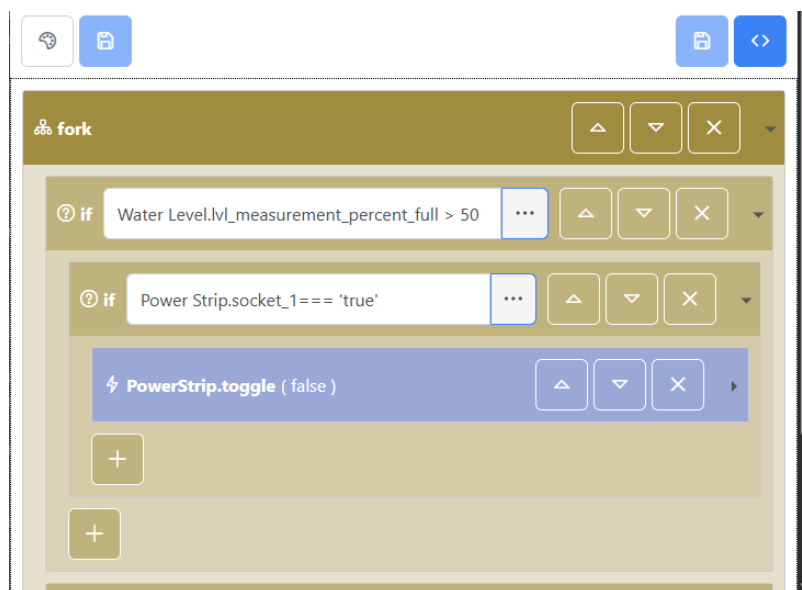
Do platformy byl přidán demo modul určený pro programování systémů IoT zařízení [20]. Ten obsahuje grafický editor, který pracuje s formulářovým programovacím jazykem. Program vytvořený ve VPL je převeden do JSON formátu a ten je využit při implementaci programu na jednotlivá zařízení.

Ukázku grafického editoru můžeme vidět na obrázku 3.11. Uživatel postupně přidává do pracovní plochy komponenty příkazů a tím vytváří výsledný program. K vkládání nových prvků slouží tlačítka plus. Ta jsou v programu zobrazena na místech, kde je možné je rozšířit. VPL rozděluje příkazy na jednotkové a složené. Do složených příkazů je možné vnořovat další komponenty a mohou obsahovat podmínku. V jejich těle jsou tedy zapouzdřeny pří-



Obrázek 3.10: Aplikace ACADA ve verzi pro desktopová zařízení (vlevo) a mobilní zařízení (vpravo) (převzato z literatury [29]).

kazy. Obvykle se jedná o abstraktní reprezentaci klasických programovacích konstrukcí jako například větvení, nebo cyklení. Jednotkové příkazy mohou obsahovat parametry reprezentované výrazy, které definují jejich chování. Příkazy je možné přesouvat a mazat pomocí tlačítek, která jsou součástí každé komponenty.



Obrázek 3.11: Vizuální editor pro aplikaci ACADA zobrazený na mobilním zařízení. V horní části obrazovky je menu s akcemi uložit, přepnout do textového editoru a dalšími. Ve spodní části je zde jednoduchý program, který ukazuje rozvětvení. První větví je blok se dvěma podmínkami (*if*) a jednou akcí. U každé komponenty jsou tlačítka pro přesouvání a mazání⁹.

⁹Převzato z: <https://v1.pocketix.org/>

Výzkumná zpráva [20] zmiňuje některé problémy, které má současná demo verze aplikace. Editor není upraven pro mobilní zařízení a nemusí na nich být přehledný. Pro uživatele tak může být komplikované na nich editor používat, zejména při tvorbě komplexnějších programů. Dalším problémem je práce s výrazy. Ty jsou zde vytvářeny v textové podobě. Tento přístup ovšem není příliš uživatelsky přívětivý a není vhodný pro tvorbu komplexnějších výrazů. Zpráva dále zmiňuje možnost rozšíření editoru o znovupoužitelné procedury a uživatelsky definované proměnné [20].

Shrnutí existujících nástrojů

Jednou z možností řešení této práce bylo upravení některého z již existujících nástrojů pro vizuální programování. Tyto programy ovšem většinou nejsou responzivní. Jejich zobrazení na mobilním zařízení je často nepřehledné. Uživatel je tak nucen využívat aplikaci pouze na počítači. Výzkum Kuhail a spol. [24] zjistil, že z VPL presentovaných ve vybraných studiích bylo 56 % určeno pro webové prohlížeče, 23 % pro mobilní platformy a 20 % pro stolní počítače. Další nevýhodou je, že jsou tyto nástroje často až příliš omezující. Nabízejí uživateli snadno pochopitelné a intuitivní prostředí, ovšem neumožňují mu rozvíjet se. Ten tak nemá možnost tvořit komplexnější programy. V případech, kdy tuto možnost má, jsou pak programy často nepřehledné a jejich tvorba je zdlouhavá. To může uživatele od dalšího používání aplikace odrazovat. Příkladem může být knihovna Blocky v jazyce JavaScript. Dle mého názoru není tento nástroj pro řešení tohoto problému vhodný. Programy skládané z bloků mohou poměrně rychle růst do šířky, což není vhodné pro mobilní zařízení. Zároveň si myslím, že nástroj není vhodný pro tvorbu komplexnějších programů. Využívání pouze jednoduchých předdefinovaných bloků, které mají mnoho různých barev, což vede u složitějších programů k nepřehlednosti. Pro systém Blocky neexistuje nativní aplikace pro Android a iOS a je zde kladen důraz na používání webové aplikace. Některé další aplikace měly naopak velmi komplexní a složité rozhraní, které vyžadovalo od uživatele znalosti programování a elektrotechniky. Příkladem je aplikace společnosti Loxone 3.4. Nativní aplikace pro mobilní zařízení (Loxone App) nabízí uživateli pouze základní možnosti ovládání zařízení internetu věcí (IoT). Pro složitější konfiguraci je nutné využít desktopovou verzi aplikace. Ta je velmi obsáhlá a nabízí uživateli komplexní řešení problémů. Ovšem očekává od uživatele základní znalosti programování a elektrotechniky, což může být omezující.

Kapitola 4

Analýza programování zařízení IoT

Čím dál více domácností si pořizuje chytrá zařízení, aby jim usnadnila práci. Ta tak přestávají být záležitostí počítačových nadšenců, ale čím dál více je využívají i lidé bez technického vzdělání. Díky tomu stoupají požadavky na aplikace, které lze využít k naprogramování chytré domácnosti. Abych správně pochopil potřeby uživatelů, rozhodl jsem se provést analýzu jejich potřeb. Aplikace by měla být přizpůsobena pro mobilní zařízení. Proto jsem se jako první krok rozhodl prozkoumat některé populární mobilní aplikace a zjistit, na co jsou u nich uživatelé zvyklí.

V kapitole 3.4 jsem prozkoumal existující nástroje a rozhodl se, na jakých principech bude můj návrh postaven. Následně jsem provedl průzkum mezi uživateli formou rozhovoru. Na základě zjištěných informací jsem sepsal požadavky, na jejichž základě jsem vytvořil návrh mobilního editoru pro vizuální programování IoT zařízení.

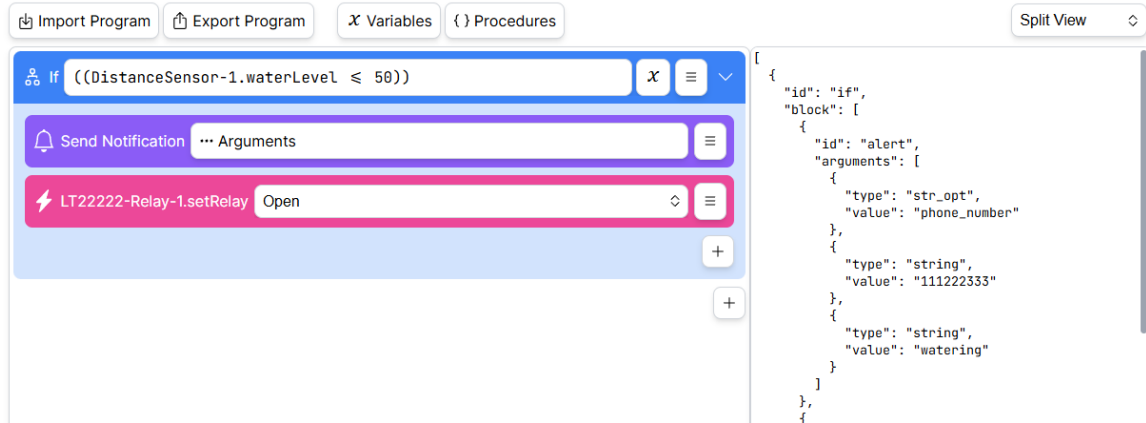
4.1 Existující editory

Na začátku bylo nutné vybrat, na jakých principech existujících nástrojů bude výsledné řešení postaveno. Správnost tohoto rozhodnutí byla následně ověřována během průzkumu uživatelských požadavků, který je popsán v sekci 4.3. Jako základ pro svůj návrh jsem si následně vybral editory, které vznikly v rámci projektu Pocketix¹. Cílem tohoto projektu je automatizace chytrých zařízení na mobilních telefonech. Pocketix navazuje na modul pro programování systémů IoT zařízení, který je součástí platformy společnosti Logimic [20]. V rámci informačního voucheru Pocketix vznikly editory Pocketix React a Pocketixing. Tyto editory jsou určeny pro uživatele bez znalostí programování. Pro své fungování využívají vizuální programovací jazyk Pocketix, který je založen na principech blokových a formulářových programovacích jazyků. Tento jazyk je možné rozšiřovat o příkazy a parametry připojených zařízení.

Rozšíření těchto editorů se věnuje bakalářská práce pana Lukáše Podvojského [36]. Jejím výsledkem je knihovna, kterou můžeme vidět na obrázku 4.1. Ta řeší některé nedostatky předchozích řešení. Zároveň je rozšiřuje o uživatelské proměnné, procedury a přidává editor výrazů. Grafický editor implementovaný touto knihovnou má ovšem také nedostatky. Je určen hlavně pro zobrazení na desktopových zařízeních a jeho zobrazení na mobilních telefonech tak není vždy optimální. Zároveň neobsahuje možnost *drag and drop*, která je u mobilních zařízení často využívána. U textové části editoru se v případě chyby nezabaruje pouze oblast, kde je pravděpodobně chyba, ale celý text. Tento editor rovněž využívá

¹<https://v1.pocketix.org/>

vizuální programovací jazyk Pocketix, který zde byl rozšířen o práci s výrazy. Stejný jazyk byl převzat i v mém řešení dané problematiky. Byly v něm provedeny úpravy, jako například odstranění typu `boolean_expression` a přidání podrobnějších názvů u bloků složených příkazů.

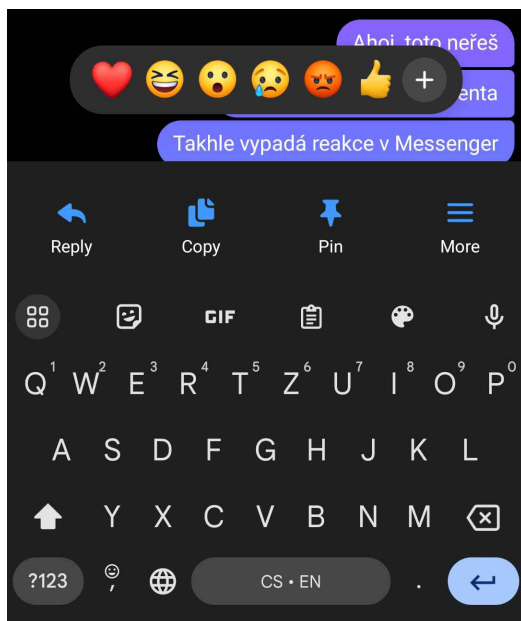


Obrázek 4.1: Na obrázku lze vidět grafický editor od pana Podvojského spuštěný na desktopovém zařízení. Ten zobrazuje program s jednou podmínkou a dvěma akcemi. Stránka se skládá z menu, grafického a textového editoru.

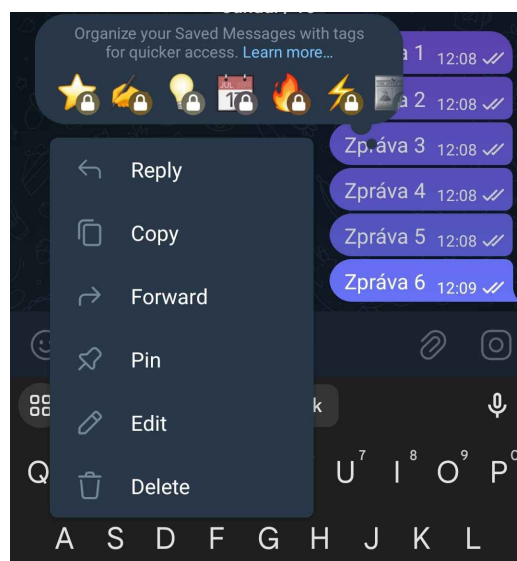
4.2 Interakce uživatele s mobilní aplikací

V rámci kapitoly 3.4 jsem se seznámil s některými nástroji, které jsou dnes využívány pro vizuální programování. Většina z nich byla určena pro desktopová zařízení. Variantu pro mobilní zařízení buď vůbec neměli, nebo nebyla příliš podporována. Z tohoto důvodu jsem se rozhodl v rámci analýzy prozkoumat některé populární mobilní aplikace. Na těchto programech pracují velké společnosti, které provádí podrobnou analýzu uživatelského chování a potřeb. Zároveň na způsoby interakcí s těmito aplikacemi jsou uživatelé zvyklí a jsou pro ně přirozené.

Vizuální programovací jazyky často obsahují bloky, ze kterých se skládá výsledný program. S každým z těchto bloků je třeba provádět akce jako například smazání, editování, přesunutí atd. U editorů pro desktopová zařízení lze tyto akce znázornit pomocí tlačítek u každého z bloků. Jelikož mobilní zařízení mají mnohem menší obrazovku než počítače, bylo nutné vymyslet způsob, jak co nejlépe zredukovat počet těchto tlačítek. Ta totiž zabírají spoustu místa a tvoří aplikaci méně přehlednou. Inspiraci pro řešení tohoto problému jsem našel v chatovacích aplikacích. Ty mají velice podobnou situaci u zobrazování akcí pro jednotlivé příspěvky. Pokud chce uživatel provést operaci s určitou zprávou (odpověď, smazání, přeposlání), musí ji podržet. Následně se objeví lišta s možnými akcemi. Její umístění se u různých aplikací liší. Aplikace Telegram umísťuje možnosti přímo vedle příspěvku, jak je vidět na obrázku 4.3. Naopak obrázek 4.2 ukazuje aplikaci Messenger od společnosti Meta, která tyto možnosti zobrazuje na konci stránky. Tato varianta je dle mého názoru pro můj návrh vhodnější, jelikož nezakrývá část obsahu stránky.



Obrázek 4.2: Ukázka zobrazení nabídky možností v aplikaci Messenger. Nabídka se objeví ve spodní části obrazovky po podržení zprávy.



Obrázek 4.3: Zobrazení nabídky možností v aplikaci Telegram. Nabídka se stejně jako u aplikace Messenger zobrazí po podržení zprávy, ovšem zde je umístěna co nejbližší samotné zprávě.

Dále jsem řešil, jak minimalizovat nutnost využití klávesnice. Ta totiž u mobilních zařízení zabírá téměř polovinu obrazovky. Zde jsem se inspiroval aplikací Duolingo, která slouží k učení se cizích jazyků. Jsou zde cvičení, kde uživatel poslouchá nahrávku a následně musí doplnit, co slyšel. Odpověď nedoplňuje psaním textu na klávesnici. Vkládá sem postupně bubliny se slovy, které se zobrazí ve spodní části obrazovky, jak můžeme vidět na obrázku 4.4. Tento přístup by u aplikace pro vizuální programování pravděpodobně řešil i další problém. Mohl by zmenšit množství syntaktických a sémantických chyb, jelikož uživatel může doplnit pouze komponenty, které jsou v nabídce.

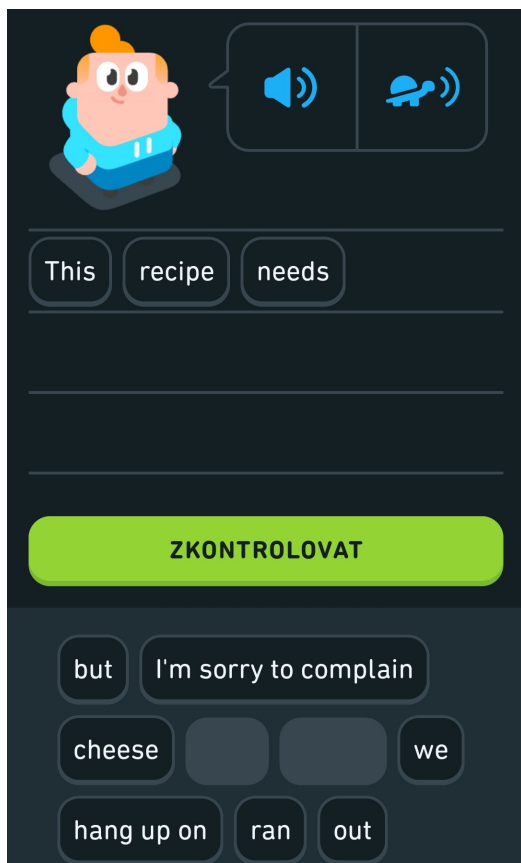
Jako poslední bych rád zmínil problematiku změny posloupnosti bloků vizuálního programovacího jazyka. Zde jsem se inspiroval aplikacemi pro přehrávání hudby. V nich je podobná situace u změny pořadí skladeb ve frontě pro přehrávání. V nich se pro tento účel využívá metoda drag and drop, která je na obrázku 4.5.

4.3 Uživatelské požadavky

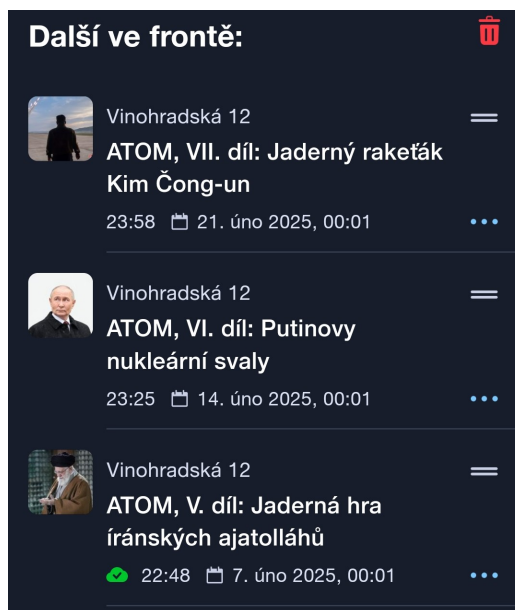
Před zahájením tvorby návrhu jsem provedl průzkum mezi potenciálními uživateli. Šetření probíhalo formou rozhovoru. Ten mi umožnil doptávat se na podrobnosti a pozorovat mentální pochody uživatelů. Zároveň jsem měl možnost doplnit informace, které z otázek nemusely být zřejmé.

Výběr respondentů

Výsledný editor by měl být určen pro co nejširší spektrum uživatelů mobilních zařízení. Tak aby si svou chytrou domácnost mohli naprogramovat jak lidé bez technického vzdělání, tak i ti, kteří mají základní znalosti informatiky. Jedinou skupinou, pro kterou editor není



Obrázek 4.4: Na obrázku lze vidět poslechové cvičení v aplikaci Duolingo. Uživatel zde skládá větu, kterou předtím slyšel, z bublin se slovy.



Obrázek 4.5: Na obrázku je ukázka fronty přehrávaných záznamů z aplikace Můj Rozhlas. Zde je využívána metoda *drag and drop*. U každé skladby je ikonka s dvěma vodorovnými čarami. Pokud ji uživatel drží (*drag*), může přesouvat skladbu na libovolnou pozici ve frontě. Pořadí se pak změní podle toho, kde uživatel pustí (*drop*) komponentu.

určen jsou programátoři. Pro ty je totiž jednodušší si program pro svou domácnost napsat v některém z klasických programovacích jazyků. Respondenty jsem proto vybíral s různých skupin s různou mírou technických znalostí. Výzkumu se celkem zúčastnilo pět respondentů, konkrétně dva muži a tři ženy. Věkové rozložení bylo následující: dva účastníci byli ve věku mezi 20 až 30 lety, jeden respondent spadal do kategorie 40–50 let a zbývající dva byli ve věku 50–60 let. Z hlediska dosaženého vzdělání měly tři osoby vysokoškolské a dvě středoškolské vzdělání. Co se týče odborného zaměření, dva respondenti uvedli technické vzdělání, jeden přírodovědné a další dva oblast sociálních věd. Dva účastníci zároveň uvedli, že mají základní znalosti programování. Stejný počet byl obeznámen s pojmem internet věcí.

Obsah průzkumu

Rozhovor s účastníky průzkumu se skládal ze tří částí. V první jsem se o nich snažil zjistit nějaké obecné informace. Zeptal jsem se, do které generace patří, jaké mají vzdělání a jestli umí programovat. V této části jsem se zároveň zeptal, zda znají pojem internet věcí (IoT). Pokud respondent pojem neznal, tak jsem mu ho vysvětlil.

Ve druhé části jsem nejprve dotazovaným předložil slovní popis ilustračního programu pro automatizaci chytré domácnosti (příloha A). Následně jsem se zeptal, jak by s daný

text zpracovali, aby informace z něj mohli zadat například do mobilní aplikace. Sledoval jsem a vyptával se, jak účastníci průzkumu nad zadanou úlohou přemýšlí. Například zda si úkol rozkládají na podproblémy a pokud ano, tak na jaké. Poté jsem dal respondentům papír na kterém bylo několik rámečků o velikosti telefonu. Chtěl jsem, aby se do těchto rámečků pokusili nakreslit, jak by si představovali mobilní aplikaci, ve které by byli schopni zadaný problém naprogramovat. Jeden z náčrtů můžeme vidět v příloze B. Průběžně jsem se jich doptával na potenciální problémy, které jsem v návrhu viděl a sledoval jsem, jak návrh následně upravovali. Zároveň jsem se je snažil vést, aby aplikaci nenavrhovali čistě pro ilustrační příklad, ale spíše obecně.

V poslední části jsem respondentům ukázal několik fotek z jedné z již existujících aplikací. Ptal jsem se, jak by zde řešili problém, který byl dříve zadán. Sledoval jsem, jak rychle jsou schopni se zde orientovat, zda je tato aplikace zaujala a zda jim vyhovuje typ použitého vizuálního programovacího jazyka. Rovněž jsem se zeptal, zda by ji preferovali před svým návrhem. Ukazoval jsem fotky z návrhu aplikace popsáném ve výzkumné zprávě pro firmu Logimic [20]. Ten jsem si vybral, jelikož jsem chtěl svůj návrh postavit na podobných základech. V této části jsem si ověřoval, na jakých principech bych měl svůj návrh postavit. Na konci rozhovoru jsem se zeptal, jaké další funkce by měla aplikace mít.

Výsledky

V rámci druhé části výzkumu jsem zjistil, že způsob uvažování nad zadaným ilustračním problémem závisel na vzdělání dotazovaných. Respondenti s technickým vzděláním se snažili si úkol rozložit na podproblémy a neřešili moc pořadí informací v textu. Ostatní se naopak snažili postupně jít sekvenčně větu po větě a získávat z nich podstatné informace. Dále jsem v rámci této části analyzoval návrhy, které dotazovaní nakreslili. Čtyři z pěti návrhů ukazovaly první obrazovku jako seznam všech zařízení v domácnosti. Z tohoto seznamu se pak dalo dostat na obrazovky, kde se dala zadat podmínka spuštění konkrétního zařízení a nastavit jeho vlastnosti. Nicméně, když jsem poté upozornil na některé možné problémy návrhu, respondenti jej nebyli schopni upravit tak, aby je vyřešili. Většina návrhů tak vedla do slepé uličky, nebo začala být příliš komplikovaná. Ovšem obsahovaly některé zajímavé myšlenky, které jsem pak zvažil při návrhu aplikace:

- Aplikace i vizuální programovací jazyk by měly mít českou mutaci.
- Využívání barev pro lepší orientaci v aplikaci.
- Mít možnost kdykoliv přerušit aktuálně spuštěný program.
- Na začátek dát k ikonám popisky pro snadnější prvotní orientaci.

Ve třetí části výzkumu jsem zjistil, že by se v ukázané aplikaci všichni respondenti zorientovali. Většina z nich se zorientovala velmi rychle a intuitivně věděli, jak by měli postupovat. Dotazovaní zároveň upozorňovali, že skládání bloků s příkazy pod sebe preferují, jelikož si jej snadno spojují s po sobě jdoucími větami v textu. Většina respondentů rovněž řekla, že by preferovali představenou variantu před tou svojí. Proto jsem se svůj návrh rozhodl postavit na stejných principech, jako byla ukazovaná aplikace.

4.4 Požadavky na řešení

Jako řešení této práce jsem se rozhodl vytvořit grafický editor pro vizuální programování IoT zařízení. Před zahájením tvorby návrhu editoru jsem se v sekci 3.4 seznámil s aktu-

álně používanými nástroji pro programování IoT zařízení. Z nich mě nejvíce zaujal grafický editor popsáný v bakalářské práci pana Lukáše Podvojského [36], který vychází z návrhu prezentovaného ve výzkumné zprávě společnosti Logimic [20]. Na tomto editoru mě zaujalo skládání bloků s jednotlivými příkazy pod sebe. Výsledný program tedy roste převážně směrem dolů. Nikoliv do šířky, jak tomu bylo u jiných nástrojů. Tento přístup je dle mého názoru vhodný pro mobilní zařízení, pro která by mé řešení mělo být primárně určeno. Ta mají totiž většinou užší podlouhlý display a uživatelé jsou zvyklí se na něm pohybovat hlavně směrem nahoru a dolů. Další výhodou editoru pana Podvojského byla možnost vytvořit obecné řešení pro různé typy zařízení internetu věcí. Tuto vlastnost jsem na základě studia IoT v kapitole 2 a vizuálního programování v kapitole 3 považoval za jeden z důležitých požadavků svého řešení.

Následně jsem se rozhodl provést průzkum mezi uživateli. Jeho cílem bylo ověřit, zda principy, na kterých jsem chtěl svůj návrh postavit, jsou uživatelsky přívětivé. Zároveň jsem se snažil zjistit, jaké prvky a funkce by podle respondentů řešení mělo mít. Ve stejné době jsem se věnoval analýze populárních mobilních aplikací. Cílem této analýzy bylo lépe pochopit rozdíly mezi aplikacemi pro desktopová a mobilní zařízení. Rovněž jsem v aplikacích hledal některé zajímavé prvky, na které jsou uživatelé zvyklí a které bych mohl využít ve svém řešení. Nakonec jsem si stanovil základní požadavky, které by měl můj editor splňovat:

- **Editor určený primárně pro mobilní zařízení** – uživatelé stále více preferují mobilní zařízení před desktopovými. Editor by tak měl být navržen tak, aby byl primárně určen pro mobilní zařízení. Zároveň by měl mít responsivní design, aby jej bylo možné používat i na desktopových zařízeních.
- **Generické řešení pro různé typy IoT zařízení** – výsledný program by měl být schopen pracovat s různými typy IoT zařízení (různá funkcionalita, různé počty parametrů). Měl by být obecným řešením pro různé systémy Internetu věcí (např. zabezpečovací systém domu nebo systém úpravy travnatých ploch).
- **Uživatelská přívětivost** – editor by měl být snadno použitelný pro uživatele. Neměl by vyžadovat znalosti programování nebo technické vzdělání. Měl by být co nejintuitivnější tak, aby se v něm uživatel byl schopen rychle zorientovat a mohl se soustředit na řešení problémů.
- **Ukládání výrazů** – uživatel by měl mít možnost pojmenovat a uložit si výrazy, aby se v nich mohl lépe orientovat a opakovaně je použít (např. jako součást nějakého komplexnějšího výrazu nebo pro vyhodnocení opakující se podmínky).
- **Práce s chybami** – pokud editor narazí na chybu v programu uživatele, měl by na ni upozornit, nebo zabránit jejímu vzniku. Hlášení o chybě by mělo uživateli dát co nejvíce informací, aby ji zvládl opravit. Měla by být označena pouze část kódu, kde editor narazil na chybu. Zároveň by kontrola chyb měla probíhat průběžně, aby uživatel byl upozorněn co nejdříve. Uživatel by měly být omezeny akce tak, aby se některým chybám předcházelo.
- **Implementace editoru jako knihovny** – tento přístup umožňuje využití komponent editoru v jiných nástrojích pro programování IoT zařízení.

Kapitola 5

Návrh mobilního editoru pro vizuální programování

Na základě studia vizuálního programování v kapitole 3 a analýzy problémů a potřeb uživatelů v kapitole 4 jsem se rozhodl vytvořit mobilní editor pro vizuální programovací jazyk Pocketix. Tento editor by měl umožnit, aby si byl běžný uživatel schopen, bez znalostí programování, nastavit a upravovat fungování svých IoT zařízení prostřednictvím svého mobilního telefonu nebo tabletu. Řešení bude implementováno jako samostatná knihovna, která bude moci být použita v existujících webových aplikacích jako je například systém ACADA společnosti Logimic, RIoT a další. Mé řešení je inspirováno návrhem editoru pro vizuální programování prezentovaným v bakalářské práci pana Lukáše Podvojského [36]. Tato práce se věnuje zjednodušení tvorby programů pro zařízení internetu věcí (IoT). Existující návrh byl použit jako základ pro vytvoření nového rozhraní určeného primárně pro mobilní zařízení.

Podle studie dnes 61 %¹ lidí primárně využívá mobilní zařízení. Toto procento se neustále zvyšuje a objevuje se potřeba, aby uživatelé byli schopni na svém telefonu udělat stejně složité operace jako na svém počítači. Dobrým příkladem je internetové bankovníctví v telefonu, kde u většiny bank je dnes možné vyřídit vše potřebné. Je tedy nutné, aby aplikace byla zejména přizpůsobena na použití na mobilních zařízeních. Díky tomu bude uživatel zároveň schopen dělat rychlé a flexibilní úpravy programu bez nutnosti zapínat počítač. Zároveň by řešení mělo být schopno dobře a intuitivně fungovat i jako webová aplikace tak, aby měli uživatelé možnost tvořit komplexnější programy na větší obrazovce. Výsledné řešení se skládá ze dvou částí: grafického a textového editoru. Uživatel mezi nimi může libovolně přepínat a změny provedené v jednom z nich se automaticky propisují do druhého. V případě, že má zařízení dostatečně širokou obrazovku, může si uživatel zobrazit oba editory vedle sebe.

5.1 Grafický editor programu

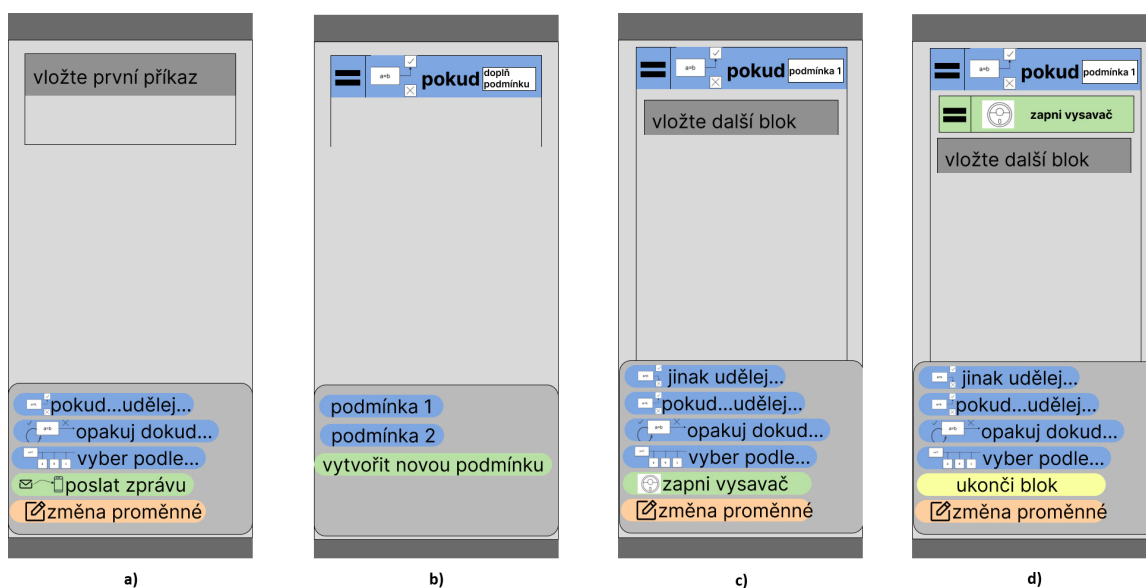
Grafický editor umožňuje tvorbu programů pomocí komponent příkazů a podmínek, které uživatel postupně přidává do programu. Možnosti, které jde aktuálně přidat, jsou uživateli zobrazeny v nabídce. Program se postupně rozšiřuje shora dolů. Na obrázku 5.1 je znázorněna postupná tvorba programu.

¹<https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/>

Příkazy a podmínky

Příkazy jsou grafické komponenty, které mají své vlastnosti (název, ikonku, barvu). Existují dva typy příkazů: složený a jednotkový. Složený příkaz můžeme vidět na obrazovce c) na obrázku 5.1. Do složeného příkazu je možné vnořovat jiné příkazy (složené i jednotkové). Zároveň obsahuje podmínku, která specifikuje, za jakých okolností se má vykonat. Složený příkaz je nutné ukončit tlačítkem pro uzavření bloku, jak je znázorněno na obrázku 5.1 d). Jednotkové příkazy neobsahují podmínku, ale je možné jim nastavit parametry (sekce 5.4). Mezi jednotkové příkazy patří i funkce, vytvořené uživatelem.

Podmínky je možné do programu vložit pouze ve chvíli, kdy je potřeba definovat podmínku složeného příkazu. Použití podmínky je znázorněno na obrázku 5.1 b). Uživatel může využívat dříve vytvořené podmínky, které jsou uloženy v repozitáři podmínek. Nové podmínky je možné vytvářet v editoru podmínek.



Obrázek 5.1: Na obrazovce a) je znázorněn vzhled aplikace na začátku tvorby programu. Ve vrchní části je vidět stručný zašedlý pokyn, co by měl uživatel dále dělat. Ve spodní části je potom vidět seznam příkazů, které je možné doplnit. Obrazovka b) zobrazuje situaci, kdy je po vybrání složeného příkazu nutné doplnit jeho podmínku. V seznamu ve spodní části obrazovky jsou podmínky uložené v repozitáři a možnost vytvořit novou podmínku v editoru. Obrazovka c) znázorňuje stav, kdy již složený příkaz má podmínku a je nutné do něj vložit jednotkový nebo složený příkaz. Obrazovka d) ukazuje situaci, kdy již složený příkaz může být korektně uzavřen, nebo rozšiřován dalšími příkazy.

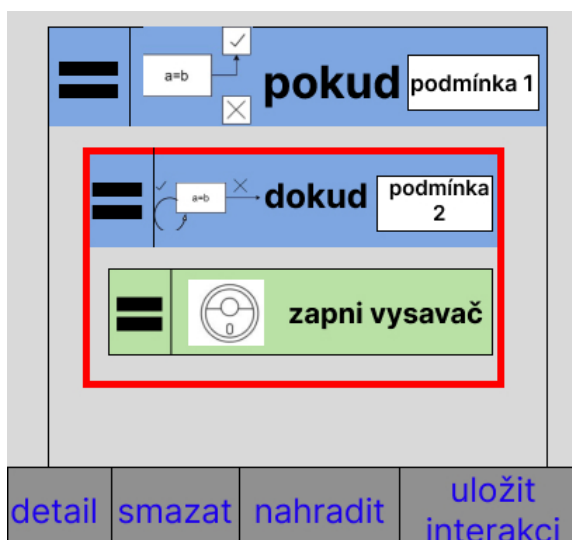
Problematika editoru podmínek byla tématem bakalářské práce pana Lukáše Podvojského [36]. Jeho práce byla dostatečně podrobná, a proto jsem jí do své práce využil pouze s drobnými úpravami.

Editor umožňuje vytvořenou podmínku pojmenovat a uložit do repozitáře podmínek. V případě, že uživatel nevyplní název, je podmínka použita na daném místě, ovšem není uložena do repozitáře a existuje pouze v kódu.

Uživatel vybírá komponenty (příkazy, podmínky, proměnné, ukončení složeného příkazu) z nabídky ve spodní části obrazovky. Její zobrazení je inspirováno aplikací Duolingo. Nabídka je tvořena tlačítky, které každé představuje jednu komponentu. Kliknutím na dané

tlačítko se komponenta přidá do programu a nabídka se aktualizuje. Touto nabídkou je možné pohybovat se nahoru a dolů.

Komponenta, která zobrazuje možnosti jak dál pracovat s příkazem, je ukázána na obrázku 5.2. Na mobilním zařízení se komponenta objeví po podržení příkazu. Toto chování bylo inspirováno aplikací Messenger. Ve webové aplikaci se komponenta objeví po kliknutí pravým tlačítkem myši. Nabídka možností pro složené a jednotkové příkazy se mírně liší. Společné možnosti jsou: smazat blok, nahradit jiným a vložit pod. Složené příkazy pak mají navíc možnosti rozšířit na šířku obrazovky a uložit interakci. Tyto akce jsou dále rozepsány v kapitole 5.2.



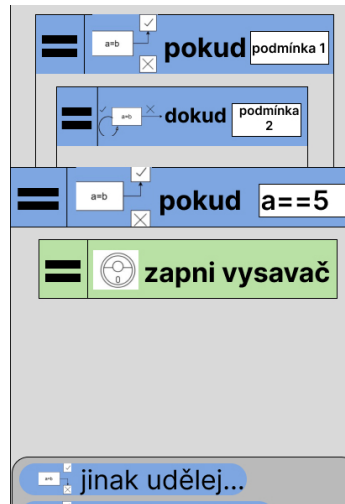
Obrázek 5.2: Na obrázku je znázorněn stav, kdy uživatel podržel příkaz. Po podržení se ve spodní části obrazovky zobrazí možnosti další práce s příkazem a samotný příkaz se zvýrazní.

Stejná komponenta, jako je použita u zobrazení možností pro příkazy, je použita i pro akce s podmínkami. Je zobrazena při podržení podmínky, nebo při kliknutí pravým tlačítkem myši. Komponenta nabízí možnosti editovat podmínku a nahradit jinou podmínkou.

5.2 Akce příkazů

S příkazy v programu je možné provádět různé akce, jak bylo zmíněno výše. Možnost zvětšení složeného příkazu na celou šířku obrazovky slouží pro přehlednější editaci příkazů, které jsou v programu vnořeny v několika jiných složených příkazech. Pokud je zvolena tato možnost, vybraný blok se rozšíří na šířku grafického editoru, jak je vidět na obrázku 5.3. Všechny nové příkazy jsou v tomto zobrazení vkládány na konec rozšířeného složeného příkazu. Při tomto zobrazení je zachována šířka ostatních příkazů mimo tělo rozšířeného. Tak aby uživatel neztratil přehled o tom, do jaké úrovně je zanořen. Rozšířené zobrazení lze ukončit pomocí tlačítka v pravé části hlavičky zvětšeného příkazu, nebo uzavřením rozšířeného bloku.

Pro přesun příkazu do jiné části programu je využívána metoda *drag and drop*. Ta je častá například v mobilních aplikacích pro přehrávání hudby. Podle studie prezentované v *Elderly User Evaluation of Mobile Touchscreen Interactions* [23] bylo pro účastníky studie



Obrázek 5.3: Obrázek ukazuje rozložení obrazovky po rozšíření složeného příkazu na plnou šířku obrazovky.

přetahování (*dragging*) preferovanější a snadnější než klikání na tlačítka. K provedení této techniky se v aplikaci využívá tlačítko se dvěma vodorovnými čarami, které je umístěno v levé části hlavičky každého příkazu. Po podržení tohoto tlačítka se v programu objeví barevně zvýrazněná místa, kam lze příkaz přesunout, aniž by vznikla syntaktická chyba. Rozložení obrazovky během této operace můžeme vidět na obrázku 5.4.

Po zvolení možnosti smazání je příkaz odstraněn z programu a jeho místo je zrušeno. Tuto možnost je možné volat pouze, pokud se nejedná o poslední příkaz ve složeném příkazu, jelikož by mohla vzniknout syntaktická chyba. V případě nahrazení příkazu je místo v programu necháno a v dalším kroku tvorby je nový příkaz vložen na toto místo. Pokud je vložen složený příkaz, další příkazy jsou vkládány do něj, dokud není uzavřen. Následně je v tvorbě kódu opět pokračováno na konci programu. Podobným způsobem je postupováno i u možnosti vložení pod vybraný příkaz. Zde se vždy vkládá na úroveň zanoření označeného příkazu.

Tato bakalářská práce je zaměřena na uživatelské rozhraní aplikace jako celku. Návrh obsahuje pouze základní náčrt, jak by práce s funkcemi v aplikaci mohla vypadat. Nicméně této problematice se dále nevěnuje, jelikož se jedná o komplexní problém, který není součástí této práce. Této problematice se věnuje práce pana Ivana Chodáka [8]. Do budoucna je plánováno obě práce propojit.

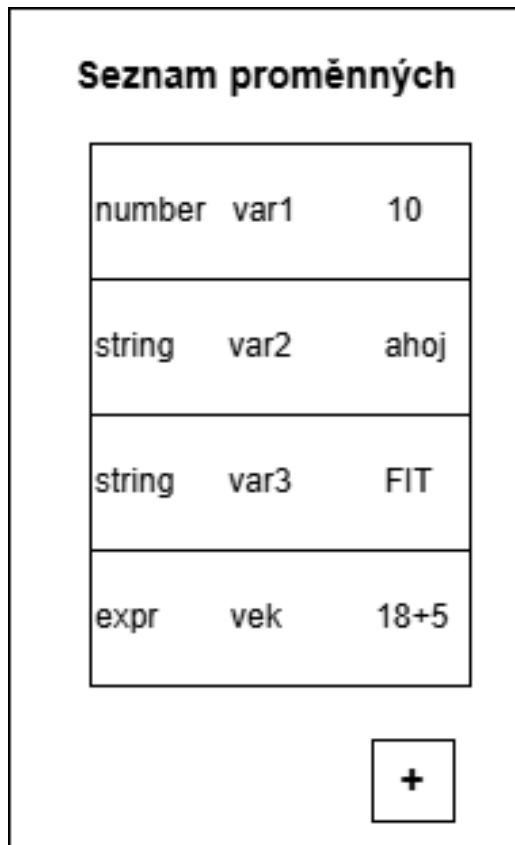
5.3 Proměnné

Proměnné budou v programu pouze globální. Uživatel je díky tomu bude mít přehledně na jednom místě. Pokud bude potřeba provést změnu proměnné, nebude nutné ji hledat v kódu. Zobrazení všech vytvořených proměnných bude možné na obrazovce, kterou můžeme vidět na obrázku 5.5. Tlačítko pro toto okno se bude nacházet v menu. Proměnné budou uživateli zobrazeny v tabulce, kde každý řádek reprezentuje jednu proměnnou. Tabulka bude obsahovat sloupce s typem, názvem a hodnotou proměnné. Při podržení řádku s proměnnou na mobilním zařízení (kliknutí pravým tlačítkem na desktopovém zařízení) se objeví možnosti editovat, nebo smazat. V případě kliknutí na smazání bude uživatel upozorněn,

že smazání proměnné může způsobit chybu v kódu. V pravém dolním rohu obrazovky bude umístěno tlačítko pro přidání nové proměnné. Po jeho stisknutí se objeví vyskakovací okno, které je znázorněno na obrázku 5.6. Stejně okno bude sloužit také k editaci již existujících proměnných.



Obrázek 5.4: Na obrázku je znázorněno, jak vypadá obrazovka aplikace poté, co uživatel podržel u komponenty zapni vysavač tlačítko s dvěma čárkami a snaží se tuto komponentu přesunout.



Obrázek 5.5: Obrazovka obsahující seznam všech proměnných v programu.

Vyskakovací okno pro vytvoření a editaci proměnných se skládá ze čtyř částí. V první části je z možností vybrán typ. V případě, že editujeme již existující proměnnou, její typ nelze měnit. Ve druhé a třetí části je nastaveno jméno a hodnota. Poslední část obsahuje tlačítka pro uložení, nebo zrušení změn. Proměnná musí mít při uložení název, který je unikátní, a hodnotu. Pokud některou z těchto podmínek nespĺňuje, uložení se neprovede. Problematické místo se označí červeně a je vypsána chybová hláška.

5.4 Nastavení parametrů příkazu

Parametry jednotlivých příkazů jsou vypsány v jejich hlavičce. Při kliknutí na ně se objeví vyskakovací okno, které vidíme na obrázku 5.7. Zde je možné změnit jejich hodnotu. Jako hodnoty je možné doplnit řetězec, booleovskou hodnotu, číslo, proměnnou a výraz. Do každého z nich lze doplnit pouze hodnota, nebo proměnná stejného typu, jako je parametr. Na konci okna se nachází tlačítka pro uložení a zrušení změn. Tlačítko pro uložení lze stisknout pouze v případě, že je doplněna do pole validní hodnota. V případě vybrání proměnné se hodnota ihned vloží a okno se uzavře.

Obrázek 5.6: Vyskakovací okno pro vytvoření a editaci proměnné.

Obrázek 5.7: Vyskakovací okno pro editaci parametru příkazu.

5.5 Textový editor programu

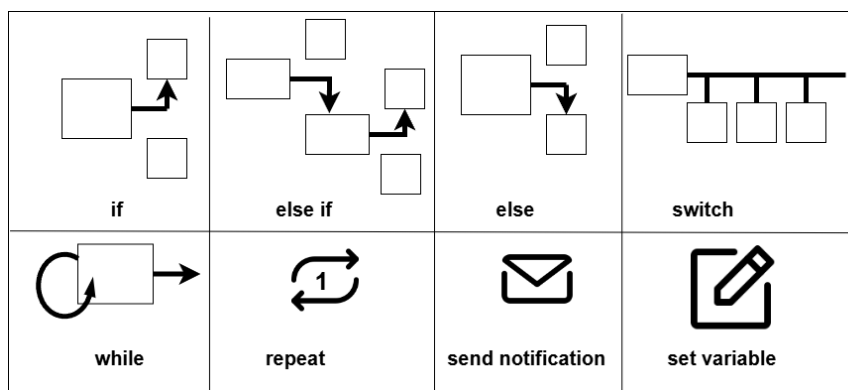
Textový editor slouží ke zobrazení a editaci serializované formy programu. Je zde zobrazena aktuální verze programu z grafického editoru ve formátu JSON. Změny z grafického editoru jsou přenášeny do textového editoru a naopak. Toto prostředí je určené primárně pro uživatele, kteří již mají s programováním nějaké zkušenosti a jsou schopni dělat úpravy kódu v jeho serializované podobě. Mezi grafickým a textovým editorem je možné přepínat pomocí ovládacích prvků.

Editor bude uživatele upozorňovat na syntaktické chyby, které vzniknou při úpravě textové formy programu. Textový editor by neměl primárně sloužit k psaní celého kódu, ale pouze k drobným úpravám. Pro psaní programu v textové podobě jsou lepší jiné programátorské nástroje. Z toho důvodu se bude správnost syntaxe vyhodnocovat po každé změně kódu. Pokud udělá uživatel při přepisování programu chybu, problematická část kódu se označí červeně. Na rozdíl od původního editoru bude barevně zvýrazněno pouze místo, kde k chybě přibližně došlo. Uživatel díky tomu bude vědět, kde narazil editor při kontrole na chybu, což mu může pomoci ji vyřešit.

5.6 Ikonky a jazyk aplikace

V průzkumu popsaném v kapitole 4.3 se často objevovala potřeba uživatelů, aby byly jednotlivé příkazy česky. Respondenti to považovali za přívětivější a domnívali se, že by jim to ulehčilo práci. Proto bude mít editor v nastavení možnost zvolit si jazyk, ve kterém bude editor i příkazy. Tato vlastnost se nebude týkat serializované formy programu v textovém editoru. Zde se předpokládá, že již uživatel má nějaké zkušenosti s programováním a rozumí anglickým názvům.

U mobilních zařízení je důležité využití ikonek. Umožňují uživatelům rychlejší orientaci. V aplikaci budou ikonky sloužit hlavně k odlišení jednotlivých typů složených i jednotkových příkazů. Jejich příklady můžeme vidět na obrázku 5.8. V úvodním nastavení budou jednotlivé příkazy obsahovat jak ikonky, tak stručný slovní popis (if, while, send notification). Uživatel může postupným užíváním editoru dojít do situace, kdy bude popisky příkazů považovat za zbytečné. V takovém případě bude mít možnost v nastavení vypnout textový popisek a zobrazovat pouze ikonky. Pokud je spuštěn tento mód aplikace, v pravém horním rohu obrazovky grafického editoru se objeví tlačítko s nápovědou. Ta bude obsahovat slovník ikon a k nim patřících příkazů.



Obrázek 5.8: Ikonky pro jednotlivé základní příkazy. První ikonka symbolizuje splnění zadané podmínky složeného příkazu (if). Další ukazuje složený příkaz, kdy neplatila první podmínka, ovšem druhá ano (elseif). Následující symbolizuje nesplnění zadané podmínky příkazu (else). Další vybraní jedné z existujících možností (switch). Pátá ikonka příkazů představuje určitý počet opakování (repeat). Další opakování těla příkazu, dokud platí podmínka (while). Předposlední ikonka symbolizuje posílání notifikace na zařízení uživatele (send notification). Poslední označuje zapsání hodnoty do proměnné (set variable).

Kapitola 6

Implementace

Mobilní editor určený pro vizuální programování IoT zařízení je implementován jako webová aplikace. Jedním z hlavních důvodů pro tento přístup byla možnost využívat stejnou aplikaci jak ve webovém prohlížeči, tak na mobilním zařízení. Oproti nativním mobilním aplikacím je možné ji využívat na mobilních zařízeních s různými operačními systémy (Android, iOS). Webová aplikace je zároveň příjemnější pro koncové uživatele, jelikož nejsou nuceni si ji instalovat do svého počítače a mohou ji využívat ve webovém rozhraní. Podle Kuhail a spol. je více než polovina nástrojů pro vizuální programování vyvíjena pro webové rozhraní [24]. Editor je implementován jako samostatná knihovna tak, aby mohly být jednotlivé komponenty využity v existujících webových aplikacích, jako je například systém RIoT, ACADA společnosti Logimic a další.

Pro reprezentaci programových bloků a argumentů jsou v knihovně vytvořeny struktury, které jsou definovány v souboru `interfaces.ts`. Tyto struktury jsou následně využívány v komponentách knihovny. Hlavní komponentou editoru je `main-element`, který ve svém těle obsahuje všechny další komponenty.

6.1 Integrace knihovny Lit

Aby výsledné řešení fungovalo dobře jako celistvá aplikace i jako knihovna webových komponent, byla při implementaci využita knihovna Lit. Webové komponenty vytvořené pomocí této knihovny jsou nativně podporovány prohlížeči a lze je používat v jakémkoliv prostředí HTML bez ohledu na použité webové technologie [28]. Pracuje se s nimi stejně jako s klasickými vestavěnými prvky HTML a není nutné znát jejich vnitřní strukturu. Jsou tak vhodné pro tvorbu samostatné knihovny. Lit komponenty jsou definovány vytvořením třídy rozšiřující `LitElement` a registrací této třídy v prohlížeči [27]. Skládají se z několika částí:

- vlastností, které mají výchozí hodnotu, nebo jsou součástí vstupu,
- stylů komponenty a prvků, které jsou v ní obsaženy,
- šablon, které popisují, jak by měla být komponenta vykreslena,
- metod komponenty.

Lit vychází ze standardů webových elementů a je rozšířen o reaktivitu a deklarativní šablony. Reaktivita komponent je v implementaci využita pro překreslení vzhledu na základě akcí uživatele. Umožňuje také vytvářet interaktivní aplikace z Lit komponent podobně, jako

je tomu u frameworků React nebo Vue. V neposlední řadě je výhodou malá velikost zdrojových kódů knihovny, které zabírají pouze kolem 5 KB [28]. Knihovna, vytvořená pomocí Lit, je díky tomu malá a rychle se načítá. Lit dobře pracuje s jazyky JavaScript a TypeScript. Tato aplikace byla napsána v jazyce Typescript, který obsahuje typový systém.

Předávání dat v komponentách

Jednotlivé komponenty editoru často pro své fungování potřebují vstupní data. Ta do nich mohou být předána dvěma základními způsoby. Prvním z nich je definování `@property` v komponentě. Ten je v implementaci využíván například pro uložení programu vytvořeného uživatelem. Každé `@property` je nutné přidělit výchozí hodnoty. Pokud je komponenta zanořena, její rodič ji při použití může tyto hodnoty definovat a následně jsou vstupní hodnoty použity namísto výchozích. V případě, že se hodnota změní v rodičovském elementu, je změna propsána i do potomka. Díky tomu se může hodnota uložená v hlavní komponentě editoru propisovat do všech vnořených komponent, kde je jí potřeba. Jakékoliv její změny způsobí automatické překreslení všech elementů, které ji dědili. Nejsou tak vždy překreslovány všechny elementy, což urychluje aktualizace. Pokud dojde ke změně v zanořené komponentě, proběhne její automatické překreslení. Do rodičovského elementu se ovšem tyto změny automaticky nepropíší. Pokud chceme změny přenést do rodičovské komponenty, je nutné v zanořeném prvku zavolat metodu `dispatchEvent` a následně v rodičovské komponentě definovat, jak s daty, která přijdou od potomka, dále pracovat. Tímto způsobem se například přenáší informace o změně programu z nabídky možností do hlavního elementu editoru.

Druhou možností je předání dat pomocí globálního kontextu. Ten je v tomto editoru využit u seznamů proměnných a podmínek, které si uživatel uložil v editoru. Výhodou této varianty je předání dat zanořenému elementu bez nutnosti posílat je přes všechny jeho rodičovské komponenty. V tomto případě jsou data globální a je možné k nim přistoupit odkudkoliv z programu. V komponentě, která tato globální data poskytuje, jsou data definována pomocí příkazu `@provide` a ta je následně označována jako *provider*. Komponenta, která tato data využívá, je označována jako *consumer*. V ní je nutné při definici dat pomocí `@property` přidat příkaz `@consume`. Ten zajistí uložení odkazu na globální data. Například seznamy proměnných jsou definovány v hlavním elementu editoru a poté využity v komponentě zanořené v editoru podmíněk.

6.2 Reprezentace programu v knihovně

Pro vnitřní reprezentaci programu v aplikaci bylo vytvořeno několik struktur. Samotný program je uložen jako pole objektů typu `ProgramBlock`. Ten reprezentuje jeden příkaz a obsahuje v sobě objekt typu `Block`, seznam argumentů příkazu a informaci, zda je blok v grafické podobě vykreslen včetně obsahu svého těla. Objekt typu `Block` v sobě obsahuje základní informace o daném bloku, které jsou pro něj neměnné v průběhu vytváření programu. `ProgramBlock` v sobě uchovává pole argumentů přiřazených k danému příkazu. Objekt typu `Argument` se skládá ze svého typu a buď hodnoty, nebo dalších argumentů. Tato struktura je vhodná pro uložení podmínek, kdy je jako typ uložen spojovací člen a jednotlivé argumenty podmínky jsou uloženy v poli argumentů.

V následujícím výpisu 6.2 můžeme vidět tři struktury určené k vnitřní reprezentaci programu v aplikaci. Struktura `Block` obsahuje neměnná data daného příkazu: zobrazený text (`name`), informaci, zda je příkaz jednotkový (`simple`), `id`, typ příkazu (`type`),

pole typů argumentů daného příkazu (`argTypes`), které jsou předdefinovány v rámci typu `TypeOption`. Struktura `ProgramBlock` se skládá z objektu typu `Block`, argumentů příkazu (`arguments`) a informace, zde je tělo bloku skryto (`hide`). Poslední struktura `Argument` je složena z typu argumentu (`type`), jeho hodnoty (`value`) a pole argumentů (`args`).

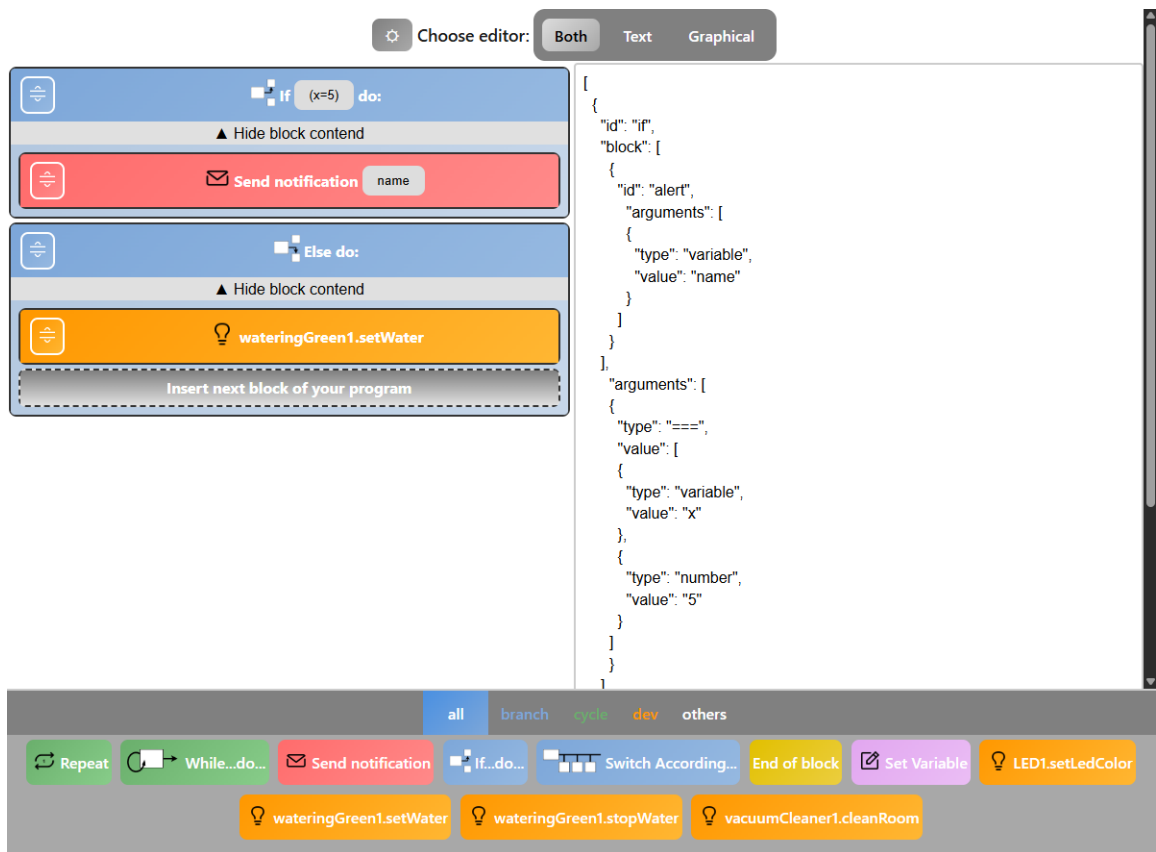
```
export interface Block {
  name: string;
  simple: boolean;
  id: string;
  type: BlockType;
  argTypes: TypeOption[];
}
export interface ProgramBlock {
  block: Block;
  arguments: Argument[];
  hide: boolean
}
export interface Argument {
  type: TypeOption;
  value: string;
  args: Argument[]
}
```

6.3 Komponenty editoru

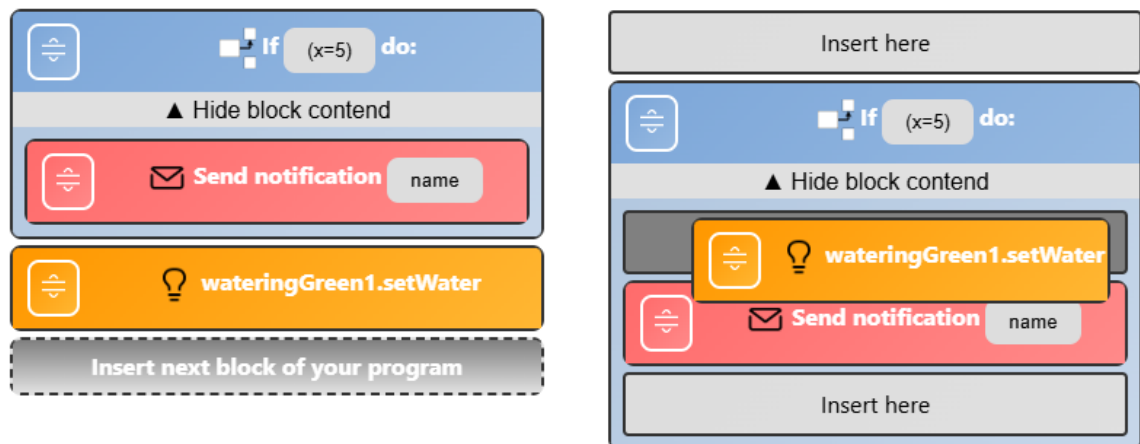
Na obrázku 6.1 můžeme vidět hlavní komponentu aplikace `main-element`. Ta se skládá ze tří hlavních částí. Nejvýznamnější je prostřední část, kde se nachází editor programu. Je možné využívat grafický, nebo textový editor. Uživatel mezi nimi může libovolně přepínat pomocí tlačítka, které se nachází v menu umístěném v horní části obrazovky. Tlačítko pro přepínání editorů obsahuje dvě, nebo tři možnosti. Pokud je aplikace spuštěna na dostatečně široké obrazovce, obsahuje tlačítko i možnost zobrazení obou editorů vedle sebe. Dále zahrnuje horní menu tlačítko pro zobrazení nastavení aplikace. Ve spodní části obrazovky se poté nachází nabídka elementů, které může uživatel doplnit do programu.

Komponenta `vp-editor-element` zapouzdřuje zobrazení programu v grafické podobě. Uvnitř něj se nachází elementy programových bloků `block-element` a grafický prvek nápovědy. Ten nás upozorňuje, kde se právě v programu nacházíme a kam se budou přidávat následující elementy. Zároveň může obsahovat doplňující informace, jako například typ argumentu, který bude přidán do příkazu. Uživatelé se pak ve svém programu snáze orientují. Bloky příkazů mohou být složené nebo jednotkové. Bloky složených příkazů obsahují navíc tělo, ve kterém se mohou nacházet další bloky, a tlačítko pro skrytí obsahu těla složeného příkazu. Každý blok programu má hlavičku, která se skládá z *drag and drop* tlačítka, ikony příkazu, názvu a argumentů daného příkazu. Při chycení tlačítka *drag and drop*, které slouží k přemísťování bloků, se v programu objeví místa, kam je možné příkaz včetně jeho těla přesunout bez vzniku syntaktické chyby, jak můžeme vidět na obrázku 6.2. Následně uživatel může blok příkazu včetně jeho těla přesunout. Pokud uživatel přesouvá příkaz *if*, *elseif*, nebo *else*, jsou přesunuty i bloky, které jsou s ním propojené. Na toto chování je uživatel upozorněn pomocí vyskakovacího okna.

Zobrazení programu v textové podobě zajišťuje komponenta `text-editor-element`, ve které je možné program upravovat. Při každé změně je provedena syntaktická kontrola programu. V případě, že je odhalena chyba, není změna programu propsána do ostatních



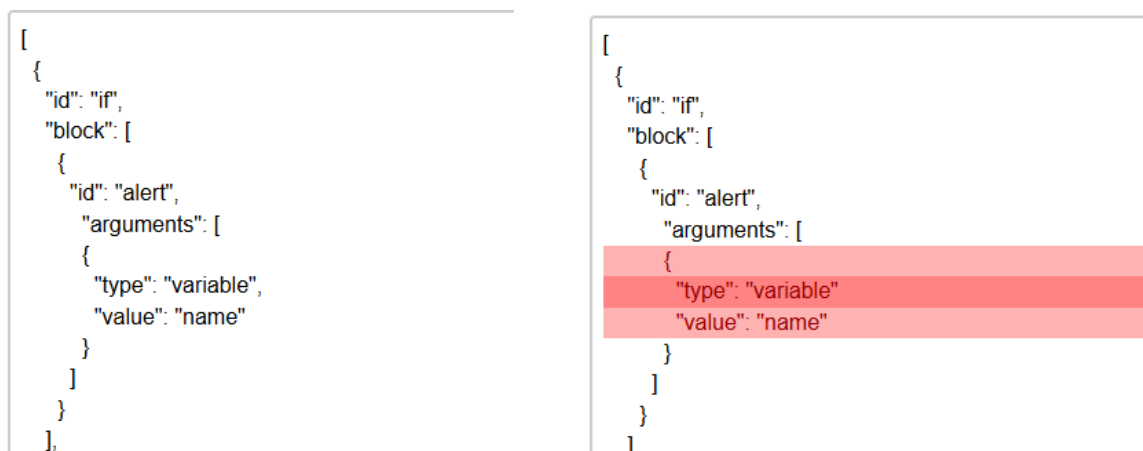
Obrázek 6.1: Hlavní obrazovka editoru spuštěná na počítači. Obrazovku lze rozdělit na tři části: menu, editor programu, nabídka možností. Jelikož má obrazovka dostatečnou šířku, můžeme vedle sebe vidět textový a grafický editor programu.



Obrázek 6.2: V levé části obrázku je znázorněn program před zahájením přetahování komponenty. V pravé části je ukázána situace těsně před umístěním bloku na začátek těla bloku `if`. Místo, kam bude blok vložen, je barevně odlišeno od ostatních.

komponent. Zároveň je číslo chybného řádku uloženo do pole `errorLines`. Text řádku s chybou spolu s okolními řádky se zbarví červeně. Zároveň se pozadí řádku s chybou

zbarví sytou červenou a okolní řádky světlejším odstínem této barvy, jak můžeme vidět na obrázku 6.3. Každý řádek je samostatná komponenta a o jejím zbarvení se rozhoduje podle pole chybových řádků `errorLines`.



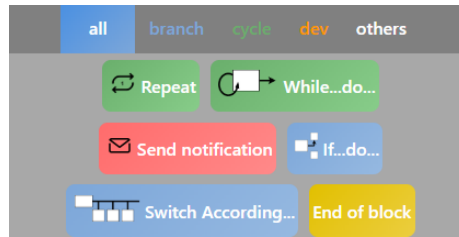
Obrázek 6.3: Obrázek znázorňuje změnu textového editoru v případě vytvoření chyby. V tomto případě jde o chybějící čárku.

Nabídka možností

Ve spodní části hlavní obrazovky se nachází nabídka možností, kterou můžeme vidět na obrázku 6.4. Tato komponenta je umístěna na podobném místě jako klávesnice na mobilním zařízení. Toto umístění je pro uživatele přehledné a intuitivní, protože jsou zvyklí v této oblasti zadávat informace. Pokud je do programu doplňován příkaz, v nabídce se nachází pouze možnosti, které lze použít. Úprava nabídky podle aktuálního stavu tvořeného programu snižuje možnost vzniku syntaktických chyb a nasměrovává uživatele k prvkům, které by měl v programu dále použít. Zároveň může uživatel příkazy filtrovat podle kategorií. Kromě příkazů je v nabídce i možnost ukončení bloku, která slouží k uzavření rozpracovaného složeného příkazu. Tato možnost se v nabídce objeví pouze v případě, že složený příkaz obsahuje ve svém těle alespoň jeden element. Pokud je do programu doplněn prvek, který obsahuje argumenty, v nabídce se objeví možnosti ze seznamů proměnných a podmínek, které jsou správného typu a lze je doplnit. Zároveň se zde objeví i varianta doplnění vlastní hodnoty. Prvky, které jsou doplňovány do programu, se tak vždy nacházejí na stejném místě, aby byla aplikace co nejvíce přívětivá. Oproti původnímu návrhu nebyla implementována možnost vypnout text u příkazů. Této možnosti by šlo využít pouze u několika příkazů, které nejsou importovány z IoT zařízení a nedošlo by tak k velké úspoře místa.

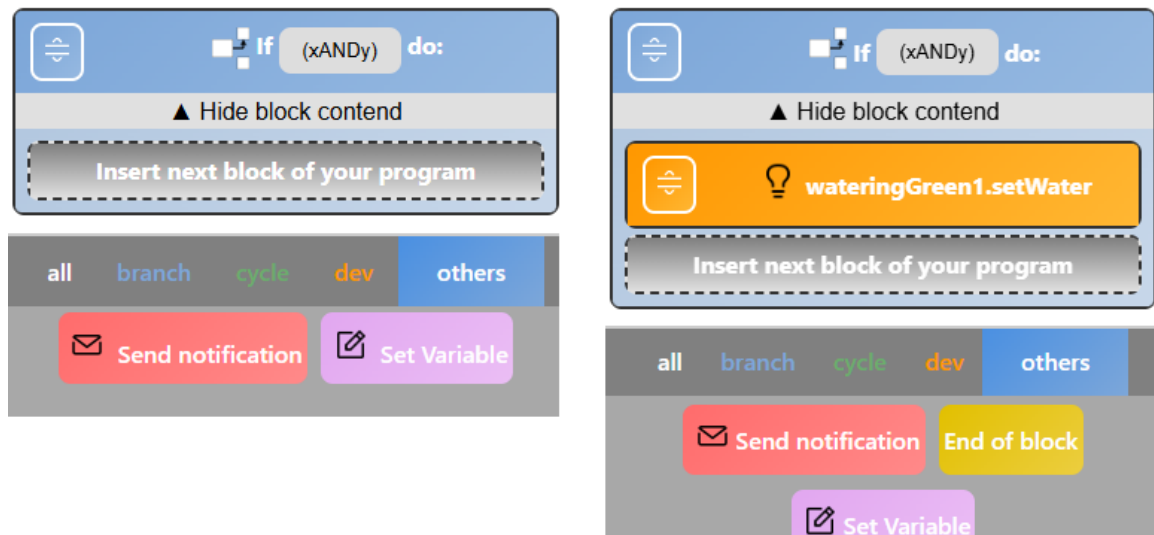
Fungování editoru

Program, který vytváří uživatel, je uložen pomocí pole objektů typu `ProgramBlock`, jak je popsáno výše v této kapitole. Toto pole je uloženo v kořenovém elementu a pracují s ním textový, grafický editor a nabídka možností. Jelikož všechny komponenty vychází z jedné reprezentace programu, nemůže v aplikaci dojít k nekonzistenci. Při jakékoliv změně programu dojde k překreslení obsahu všech těchto komponent, jak můžeme vidět na obrázku 6.5.



Obrázek 6.4: Nabídka bloků, které je možné doplnit do programu. Je možné ji filtrovat podle typu příkazu pomocí menu, které se nachází v její horní části. Jednotlivé bloky obsahují ikonky a jsou barevně odlišeny podle svého typu.

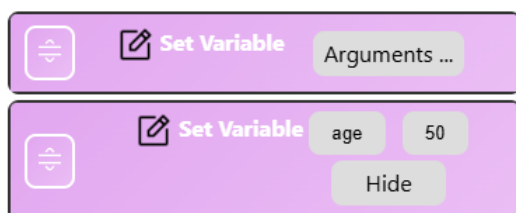
Všechny tyto elementy umožňují změnu programu. Nabídka možností zajišťuje přidávání prvků do programu a bloky v grafickém editoru obsahují akce pro jejich mazání, změnu pořadí, nebo úpravu argumentů.



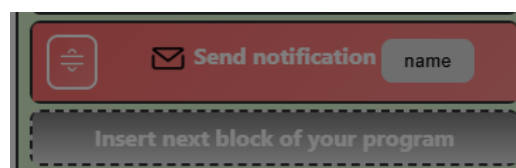
Obrázek 6.5: Změna vykreslení editoru pro přidání akce zařízení do programu.

Argumenty každého příkazu jsou součástí jeho hlavičky. Podmínka u složeného příkazu je zobrazena jako jeho argument. V případě, že má příkaz více než jeden argument, je v hlavičce místo nich zobrazeno tlačítko s názvem `Arguments...`, jak můžeme vidět na obrázku 6.6 tak, aby příkaz nezabíral zbytečně moc prostoru. Při kliknutí na něj se rozbalí všechny argumenty a tlačítko `Hide` pro navrácení do původního stavu. Každý argument je možné editovat. Editační okno, které můžeme vidět na obrázku 6.7, se spustí při kliknutí na daný prvek v hlavičce příkazu. Pro jednodušší orientaci je umístěno ve spodní části obrazovky stejně jako nabídka možností. Má ovšem odlišnou barvu pozadí, tak aby uživatel zaznamenal změny na obrazovce. Při kliknutí na podmínku se otevře okno pro její editaci, které je rozšířeno o možnost vložit na dané místo některou z již existujících podmínek. Zde došlo ke změně oproti původnímu návrhu, kde se počítalo se zobrazením možností akcí s danou podmínkou po podržení tlačítka. K této změně došlo kvůli sjednocení fungování stejně vypadajících tlačítek v hlavičce příkazu.

U každého bloku příkazu si lze otevřít nabídku možností dalších akcí, kterou můžeme vidět na obrázku 6.8. Nabídku si uživatel může zobrazit pomocí podržení hlavičky příkazu.



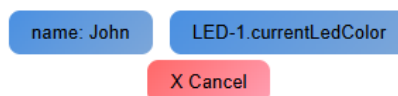
Obrázek 6.6: Zobrazení příkazu s více argumenty. Horní příkaz ukazuje zobrazení v běžném režimu a dolní příkaz s rozbalením všech argumentů.



Change value:



Use variable

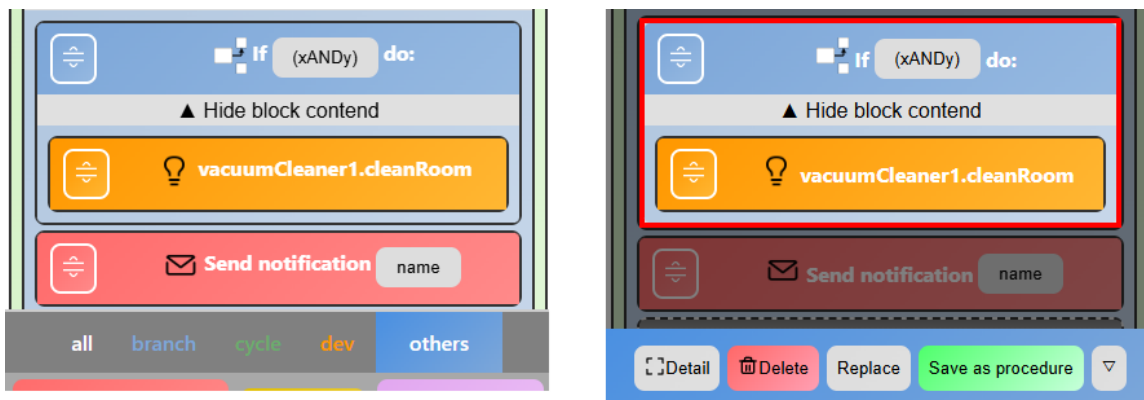


Obrázek 6.7: Zobrazení okna pro změnu hodnoty parametru příkazu. V horní části se nachází pole pro doplnění hodnoty. Ve spodní části jsou vidět tlačítka pro doplnění některé z existujících proměnných.

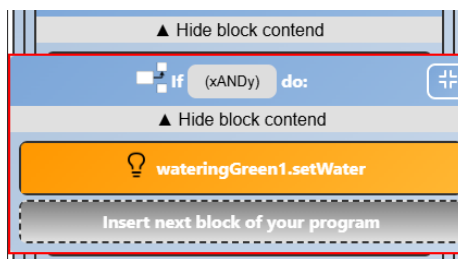
Možnost podržení zde byla zvolena, aby nedocházelo ke kolizi s klikáním na argumenty. Nabídka umožňuje u všech příkazů posun o jednu pozici v programu nahoru/dolů, nahrazení příkazu jiným a smazání příkazu. Díky těmto akcím je možné provádět změny v kterékoliv části programu. V případě, že se uživatel rozhodne nahradit původní blok složeným příkazem, pokračuje se v doplňování složeného příkazu, dokud není uzavřen. Oproti návrhu je možné smazat i příkaz, který je poslední v těle složeného příkazu. Uživatel díky tomu může vytvořit prázdné tělo složeného příkazu a následně do něj příkaz přesunout například pomocí *drag and drop*. Zároveň byla oproti návrhu vynechána možnost vložit pod, jelikož stejné operace lze dosáhnout s použitím jiných dostupných tlačítek. U složeného příkazu jsou navíc dostupné možnosti detail bloku a uložit jako proceduru. Možnost detail bloku můžeme vidět na obrázku 6.9. Při využití této možnosti se daný blok rozšíří na plnou šířku obrazovky a následně se pokračuje s tvorbou programu v jeho těle. Tento režim je ukončen uzavřením daného bloku, nebo pomocí tlačítka pro zmenšení, které se nachází v pravé části hlavičky rozšířeného příkazu.

Nastavení editoru

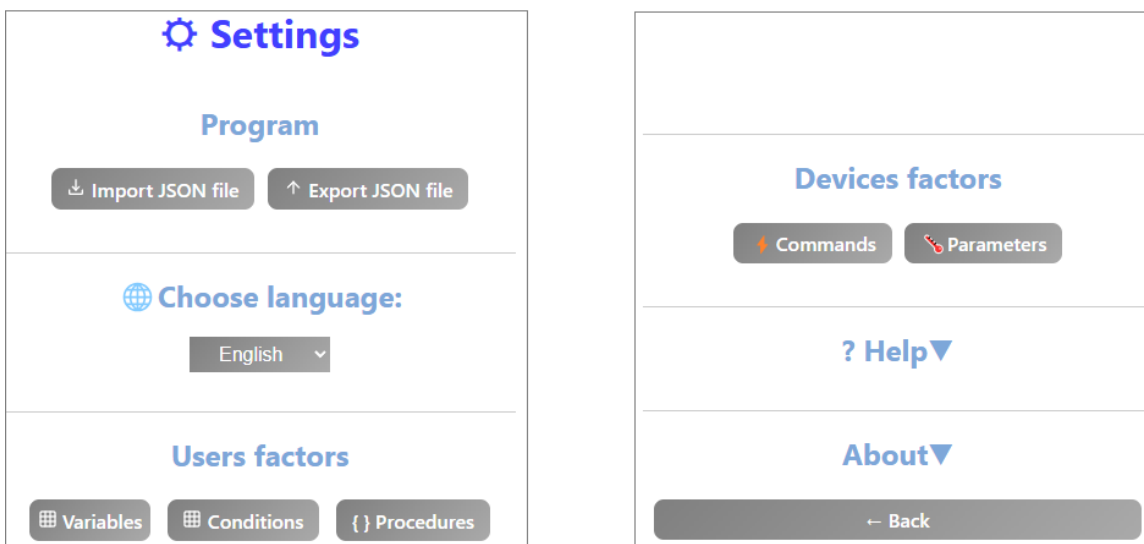
Na obrazovku nastavení, kterou můžeme vidět na obrázku 6.10, se lze dostat z menu hlavní obrazovky aplikace. Tlačítka, která se na ní nachází, je možné rozdělit do několika skupin. První z nich jsou tlačítka pro import a export programu. V další části je k dispozici volba jazyka aplikace, což zvyšuje její přívětivost a srozumitelnost pro uživatele. Zároveň je možné aplikaci v budoucnu rozšiřovat o další jazyky. Následující segment obsahuje tlačítka pro přístup k seznamu uživatelských proměnných, předdefinovaných podmínek a uživatelem vytvořených procedur. V další části se nachází tlačítka pro zobrazení příkazů připojených zařízení internetu věcí a parametry získané z IoT zařízení. Poslední část zahrnuje stručné informace o aplikaci a nápovědu.



Obrázek 6.8: Zvýraznění vybraného příkazu a zobrazení nabídky. Vybraný příkaz je zvýrazněn červeným rámečkem, zatímco zbytek editoru je upozaděn. Zároveň se objeví nabídka s možnými akcemi pro daný příkaz, která je filtrována podle dostupných možností.



Obrázek 6.9: Rozšíření příkazu v detail módu editoru.



Obrázek 6.10: Nastavení editoru. V levé části obrázku se nachází horní polovina obrazovky pro nastavení editoru. V pravé části se nachází spodní část obrazovky, která má spíše informační funkci.

Proměnné a podmínky

Seznamy proměnných a podmínek je možné otevřít z obrazovky nastavení. Obrazovku se seznamem proměnných můžeme vidět na obrázku 6.11. Obrazovka se seznamem podmínek

je téměř identická. Pokud chce uživatel proměnnou či podmínku editovat, nebo vymazat, musí na ni nejdříve kliknout. Následně si vybere mezi možnostmi *Edit* a *Delete*. Zde došlo ke změně oproti původnímu návrhu, kde se počítalo s otevřením této nabídky po podržení dané proměnné, nebo podmínky. Podržení bylo nahrazeno kliknutím na komponentu, jelikož se jedná o jednodušší akci. Zároveň zde nehrozí možnost nepřesného kliknutí, kvůli které bylo podržení využito u hlavičky bloku příkazu. Další změnou oproti návrhu je samotný koncept vytváření podmínek pro jednorázové použití a pro uložení v aplikaci. Zde došlo k oddělení těchto činností. Uložit si podmínku je možné pouze na obrazovce se seznamem podmínek a vytvoření jednorázové podmínky probíhá jen během samotné tvorby programu. Tato změna by měla zjednodušit orientaci v aplikaci.

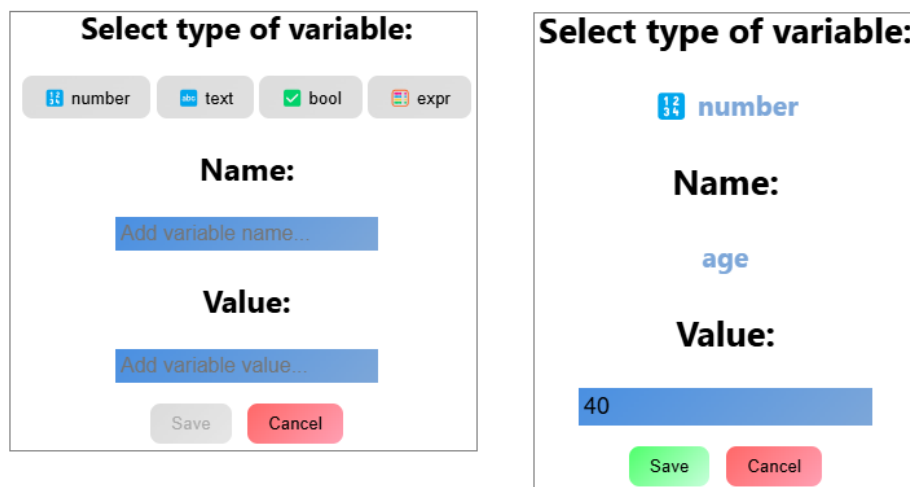
List of variables

Type	Name	Value
abc text	name	John
123 number	age	40
✓ bool	isAdmin	true
📄 expr	fee	((a+b)=6)

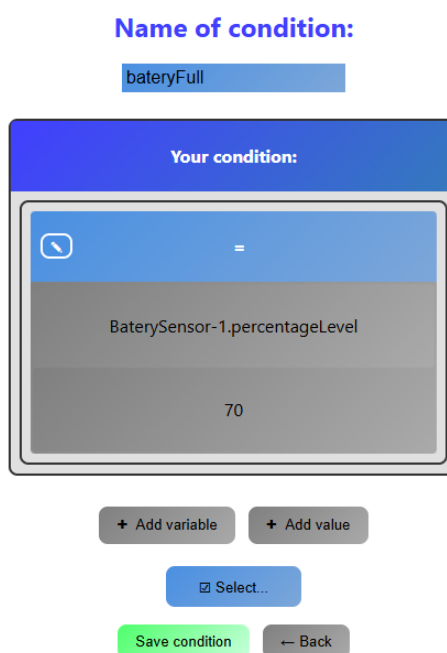
← Back
+ New variable

Obrázek 6.11: Tabulka se seznamem dostupných uživatelských proměnných. Ke každé proměnné je uvedeno její jméno, typ a hodnota. Ve spodní části se nachází tlačítka pro opuštění obrazovky a vytvoření nové proměnné.

Obrazovka pro změny proměnné má dvě varianty: vytváření a úprava proměnné. Obě tyto varianty můžeme vidět na obrázku 6.12. Novou proměnnou, nebo změny v proměnné lze uložit pouze v případě, že byly správně vyplněny jméno, typ a hodnota. Pokud tomu tak není, je tlačítko pro uložení zašedlé a nelze na něj kliknout. Toto řešení bylo zvoleno jako uživatelsky přívětivější, než zabarvování chybných polí, které bylo prezentováno v návrhu. Stejně jako editor proměnných i editor podmínek má variantu pro vytvoření nové podmínky, kterou můžeme vidět na obrázku 6.13, a úpravu existující podmínky. Základní koncept editoru podmínek byl převzat z práce pana Lukáše Podvojského [36].



Obrázek 6.12: V levé části obrázku se nachází editor proměnné při definování nové hodnoty. Jelikož nejsou vyplněna všechna pole, není možné proměnnou uložit. V pravé části je znázorněna úprava již existující proměnné, u které nelze měnit jméno a typ proměnné, tak aby nevznikaly syntaktické chyby v kódu.



Obrázek 6.13: Editor podmínek, který se skládá ze tří hlavních částí. První je prostor pro zadání jména (tato část není při editaci existující podmínky zobrazena). Druhá část obsahuje grafické zobrazení podmínky a poslední tlačítka pro tvorbu podmínky a její uložení.

Kapitola 7

Testování

Po implementaci knihovny proběhlo uživatelské testování. Jeho účelem bylo ověření splnění požadavků na řešení a získání užitečných postřehů od potenciálních uživatelů. Testování se odehrávalo vždy prezenčně za mé účasti po celou jeho dobu. Díky tomu jsem mohl sledovat chování testovaných osob, jejich přístup k aplikaci a ptát se jich na doplňující otázky. Testování proběhlo se třemi účastníky, z nichž se někteří účastnili i průzkumu uživatelských požadavků. Byli záměrně vybráni tak, aby co nejlépe reprezentovali široké spektrum potenciálních uživatelů:

- dva muži a jedna žena,
- dva lidé mezi 18 – 25 roky, jeden člověk mezi 55 – 60 roky,
- dva lidé s vysokoškolským vzděláním, jeden se středoškolským vzděláním,
- dva respondenti měli technické vzdělání, jeden přírodovědné,
- dva respondenti měli základní znalosti programování.

Žádný z účastníků nebyl programátor, jelikož pro tuto skupinu lidí by bylo pravděpodobně jednodušší si program napsat v některém z běžně využívaných programovacích editorů. Zároveň byli vybráni respondenti, kteří by do budoucna mohli mít zájem podobný nástroj využívat.

Na začátku byly všichni zúčastnění dotázáni na základní informace, které jsou uvedeny výše. Dále následovala otázka, zda znají pojmy internet věcí a vizuální programování. Dva z účastníků znali pojem internet věcí. Nikdo z nich neznal termín vizuální programování, nicméně jeden se s nástrojem pro vizuální programování již dříve setkal. Poté byli respondencům nové pojmy vysvětleny.

Před začátkem samotného testování nebyli respondenti s aplikací seznámeni a nebylo jim popsáno jak pracuje. Zvolil jsem tento přístup, abych mohl sledovat postupné seznamování se uživatelů s prostředím a abych ověřil, zda budou schopni se v aplikaci sami zorientovat. Tento přístup v některých případech vedl k situaci, kdy uživatel v aplikaci provedl operaci, kterou provést nechtěl a následně byl nucen svůj omyl napravit. Díky tomu byly otestovány i neočekávané scénáře a robustnost samotné aplikace. Jelikož aplikace by měla být využívána především na mobilním zařízení, testování probíhalo v simulátoru mobilního zařízení v developerském nástroji webového prohlížeče Chrome. Samotné testování se skládalo ze dvou částí. V rámci první z nich dostala testovaná osoba následující úkol.

Úkol první části testování: Představte si, že máme doma chytrý vysavač, retenční nádrž, zavlažování a několik čidel. Zadejte chytré domácnosti následující program. Pokud

je hodnota `BatterySensor-1.percentageLevel` alespoň 70 procent, tak se provede následující. Zjistí se, jestli je `DistanceSensor-1.waterLevel` přes 50 procent a `TemperatureDevice-1.temperatureLevel` pod 20. Pokud ano, spustí se zavlažování a odešle se zpráva, že je zalito na telefon. Jinak se spustí vysavač (střídavě vysává a vytírá). Akce spuštění vysavače se provede dvakrát.

V rámci druhé části pak měly testované osoby zkoušet provádět různé změny programu a další funkce editoru:

- Zkuste si stáhnout vytvořený program.
- Zprávu pro telefon budete chtít použít na více místech, zkuste si ji v editoru uložit.
- Pokuste se přepnout si editor do textového módu.
- Zkuste změnit počet zapnutí vysavače.
- Akce zapnutí vysavače se vám nelíbí a chcete ji vymazat, nebo nahradit jinou akcí. Zkuste provést tuto změnu programu.
- Zkuste posláni notifikace přesunout mimo *if* blok.
- Doplňující otázka: Preferujete přesun pomocí tlačítek, nebo *drag and drop* metodou?

V poslední části testování jsem se účastníků ptal, co se jim na aplikaci líbilo/nelíbilo, co by změnil a jak se jim s ní pracovalo. Zároveň byly v této části testování položeny otázky, které vyplynuly z jeho průběhu. Nakonec dostali účastníci možnost zeptat se mě na otázky okolo aplikace a svobodně si ji proklikat.

Výsledky testování

U prvního úkolu se všem účastníkům podařilo vytvořit funkční program podle zadání. Jednomu z nich bylo nutné napovědět. Náповěda proběhla na začátku, kdy se uživatel seznamoval s aplikací. Účastníci využívali různé přístupy pro splnění úkolu. Jeden z nich si nejdříve uložil všechny podmínky, které byly v zadání a teprve poté začal sestavovat samotný program. V dalším případě osoba zkoušela postupně klikat na všechna tlačítka v aplikaci a sledovala, jak se chovají a následně začala sestavovat program. Také došlo k situaci, kdy se uživatel snažil vyhnout využití programovacích konstrukcí, kterým nerozuměl a vytvořil složitější program, který ovšem splňoval zadání úkolu.

Ve druhé části účastníci plnili různé úkoly, které testovaly práci s dalšími funkcemi editoru, které nebyly ověřené v rámci první části. Všechny úkoly byly všemi účastníky splněny. V některých případech testované osoby musely chvíli přemýšlet a zkoušet různé možnosti. Ovšem vždy došly ke správnému výsledku a nebylo třeba jim radit. Zároveň čím déle účastníci pracovali s aplikací, tím rychleji a lépe byli schopni plnit nově zadané úkoly. V rámci otázky na využití tlačítka *drag and drop* jeden člověk odpověděl, že tuto techniku znal již dříve. Všichni po ukázání fungování tohoto tlačítka souhlasili, že se jim tato varianta líbí více než posouvání pomocí šipek.

V rámci testování byly odhaleny následující nedostatky:

- špatně rozpoznatelné *check box* v editoru podmínek,
- nedostatečně odlišen blok nápovědy od bloků programu,
- snadno zaměnitelná tlačítka příkazů *else* a *elseIf*,

- nejasnost, co je typ *string* a jeho nahrazení pojmem *text*.

Všechny tyto problémy byly následně opraveny v poslední verzi řešení. Účastníci testování si pochvalovali orientaci v programu a barevné odlišení různých typů příkazů. Dva z nich hodnotili práci s editorem jako příjemnou a chtěli si vyzkoušet i další možnosti tohoto nástroje. Jeden účastník řekl, že by ocenil vysvětlení fungování na začátku používání. Uživatelské testování potvrdilo splnění požadavků na řešení definovaných v kapitole 4.4. S aplikací se účastníkům dobře pracovalo v mobilním rozhraní, byla hodnocena jako uživatelsky přívětivá a pomáhala účastníkům s řešením problémů.

Budoucí rozšíření

Výsledný editor je pouze část celkového řešení programování zařízení internetu věcí. Jeho výstupem je vygenerovaný serializovaný program v jazyce JSON. Ten by si následně mělo převzít aplikační rozhraní (API), které zajistí interpretaci programu, zaslání příkazů na koncová zařízení a fungování systému jako celku. Z tohoto důvodu by měl být při dalším vývoji editor propojen s API v rámci projektu Pocketix.

Budoucí rozšiřování by také mělo zahrnovat propojení editoru s externími knihovnami, které rozšíří jeho funkcionalitu. Řešení by mělo umožnit napojení knihovny pro tvorbu uživatelských procedur a kontrolu syntaxe a sémantiky textové verze uživatelského programu. Výsledné řešení je dále možné rozšířit o některé nové užitečné funkce. Mezi ně patří například zrušení/vrácení poslední změny, nastavení barev a ikon u vizuálního programovacího jazyka nebo analýza funkčnosti uživatelem vytvořeného programu. V neposlední řadě by mohlo být upraven celkový design aplikace.

Kapitola 8

Závěr

Tato práce se věnovala tvorbě editoru pro vizuální programování IoT zařízení. Cílem bylo umožnit koncovým uživatelům snadno a bez znalostí programování definovat chování zařízení internetu věcí. Z tohoto důvodu bylo výsledné řešení postaveno na konceptu vizuálního programování. Za cílovou platformu byla vybrána mobilní zařízení, jelikož jsou běžnými uživateli čím dál více využívána a řada existujících nástrojů pro ně není přizpůsobena. Řešení mělo zároveň umožnit práci s odlišnými druhy IoT zařízení od různých výrobců. V neposlední řadě byl výsledný editor implementován jako knihovna, aby bylo možné jednotlivé komponenty využít v rámci jiných webových aplikací.

Nejdříve bylo nutné se blíže seznámit s tématem internetu věcí. Následně bylo potřeba prostudovat oblast vizuálního programování a současné nástroje, které jej využívají. Poté byla provedena analýza populárních mobilních aplikací. Ta měla za cíl lépe pochopit práci uživatelů s mobilními aplikacemi a seznámit se s technikami, které se zde běžně používají a na které jsou uživatelé zvyklí. Následoval průzkum mezi potenciálními uživateli, který mi pomohl lépe pochopit jejich požadavky a vybrat vhodné základní principy pro své řešení. Ze všech těchto získaných informací vzešly požadavky na návrh aplikace. Následně bylo navrženo rozhraní aplikace, které mělo splňovat požadavky na řešení této práce. Poté byl návrh implementován jako samostatná knihovna v jazyce TypeScript s využitím knihovny Lit.

V průběhu implementace byly doladovány a upravovány detaily návrhu tak, aby bylo výsledné řešení co nejvíce uživatelsky přívětivé. Oproti implementacím, ze kterých návrh vycházel, byly přidány nové funkce a interakce. Mezi ně patří například nový způsob přidávání prvků do programu, možnost rozšířit blok v grafickém zobrazení programu na plnou šířku obrazovky, vybrání si jazyka aplikace, možnosti ukládat si podmínky a využití techniky *drag and drop* k úpravám programu. Nakonec bylo provedeno uživatelské testování. Jeho výsledkem bylo potvrzení splnění požadavků na řešení a nalezení nedostatků editoru. Tyto nedostatky byly následně opraveny ve finální verzi editoru. Součástí dalšího vývoje by mělo být propojení editoru s API. Zároveň by v budoucnu mělo dojít k napojení nástroje pro práci s uživatelskými procedurami a knihovnou pro provádění syntaktických kontrol kódu v textové podobě.

Výsledný editor splnil požadavky definované na základě analýzy problematiky a cíle této práce. Editor je určen především pro mobilní zařízení, nicméně je možné s ním pracovat i ve webovém prohlížeči. Je možné jej využít jako celek, nebo pouze některé jeho komponenty v rámci jiných aplikací.

Literatura

- [1] AKPAKWU, G. A.; SILVA, B. J.; HANCKE, G. P. a ABU MAHFOUZ, A. M. A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges. *IEEE Access*, 2018, sv. 6, s. 3619–3647. ISSN 2169-3536. Dostupné z: <http://dx.doi.org/10.1109/ACCESS.2017.2779844>.
- [2] AL QASEEMI, S. A.; ALMULHIM, H. A.; ALMULHIM, M. F. a CHAUDHRY, S. R. IoT architecture challenges and issues: Lack of standardization. In: IEEE. *2016 Future technologies conference (FTC)*. 2016, s. 731–738. Dostupné z: <https://doi.org/10.1109/FTC.2016.7821686>.
- [3] ASHTON, K. et al. That ‘internet of things’ thing. *RFID journal*. Hauppauge, New York, 2009, sv. 22, č. 7, s. 97–114.
- [4] BADI, C.; BELLINI, P.; DIFINO, A.; NESI, P.; PANTALEO, G. et al. Microservices suite for smart city applications. *Sensors*. MDPI, 2019, sv. 19, č. 21, s. 4798. ISSN 1424-8220. Dostupné z: <https://www.mdpi.com/1424-8220/19/21/4798>.
- [5] BAK, N.; CHANG, B.-M. a CHOI, K. Smart Block: A visual block language and its programming environment for IoT. *Journal of Computer Languages*. Elsevier, 2020, sv. 60, s. 100999. ISSN 2590-1184. Dostupné z: <https://doi.org/10.1016/j.co1a.2020.100999>.
- [6] BELFIORE, A.; CUCCURULLO, C. a ARIA, M. IoT in healthcare: A scientometric analysis. *Technological Forecasting and Social Change*, Listopad 2022, sv. 184, s. 122001. ISSN 0040-1625. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0040162522005224>.
- [7] BOOTH, T. a STUMPF, S. End-user experiences of visual and textual programming environments for Arduino. In: Springer. *End-User Development: 4th International Symposium, IS-EUD 2013, Copenhagen, Denmark, June 10-13, 2013. Proceedings 4*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, s. 25–39. ISBN 978-3-642-38706-7. Dostupné z: https://doi.org/10.1007/978-3-642-38706-7_4.
- [8] CHODÁK, I. *Support for User-Defined Functions in a Visual Programming Language*. Brno, 2025. Bachelor’s thesis. Brno University of Technology, Faculty of Information Technology. Vedoucí práce JOHN, I. P.
- [9] CHOURABI, H.; NAM, T.; WALKER, S.; GIL GARCIA, J. R.; MELLOULI, S. et al. Understanding smart cities: An integrative framework. In: IEEE. *2012 45th Hawaii international conference on system sciences*. 2012, s. 2289–2297. ISSN 1530-1605. Dostupné z: <https://doi.org/10.1109/HICSS.2012.615>.

- [10] DEOGIRIKAR, J. a VIDHATE, A. Security attacks in IoT: A survey. In: *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. říjen 2017, s. 32–37. ISBN 978-1-5090-3243-3. Dostupné z: <https://doi.org/10.1109/I-SMAC.2017.8058363>.
- [11] DEVALAL, S. a KARTHIKEYAN, A. LoRa technology-an overview. In: IEEE. *2018 second international conference on electronics, communication and aerospace technology (ICECA)*. IEEE, 2018, s. 284–290. ISBN 978-1-5386-0965-1. Dostupné z: <https://doi.org/10.1109/ICECA.2018.8474715>.
- [12] EUROPEAN PARLIAMENT. *The Internet of Things Briefing*. European Parliamentary Research Service, 2015. Dostupné z: [https://www.europarl.europa.eu/RegData/etudes/BRIE/2015/557012/EPRS_BRI\(2015\)557012_EN.pdf](https://www.europarl.europa.eu/RegData/etudes/BRIE/2015/557012/EPRS_BRI(2015)557012_EN.pdf). [cit. 2025-01-18].
- [13] FORD, R.; PRITONI, M.; SANGUINETTI, A. a KARLIN, B. Categories and functionality of smart home technology for energy management. *Building and Environment*, 2017, sv. 123, s. 543–554. ISSN 0360-1323. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0360132317303062>.
- [14] FRANCESE, R. a RISI, M. Supporting elderly people by ad hoc generated mobile applications based on vocal interaction. *Future Internet*, 2016, sv. 8, č. 3, s. 42. ISSN 1999-5903. Dostupné z: <https://www.mdpi.com/1999-5903/8/3/42>.
- [15] GOKHALE, P.; BHAT, O. a BHAT, S. Introduction to IOT. *International Advanced Research Journal in Science, Engineering and Technology*, Leden 2018, sv. 5, č. 1, s. 41–44. ISSN 2393-8021. Dostupné z: <https://doi.org/10.17148/IARJSET.2018.517>.
- [16] GREENGARD, S. *The Internet of Things*. MIT Press, 2015. The MIT Press Essential Knowledge series. ISBN 9780262527736.
- [17] GUPTA, B. B. a QUAMARA, M. An overview of Internet of Things (IoT): Architectural aspects, challenges, and protocols. *Concurrency and Computation: Practice and Experience*. Wiley Online Library, 2020, sv. 32, č. 21, s. e4946. Dostupné z: <https://doi.org/10.1002/cpe.4946>.
- [18] HARING, O.; AZUMAH, S. a ELSAYED, N. A Review of Network Evolution towards a Smart Connected World. *International Journal of Computer Applications*. Foundation of Computer Science, Květen 2021, sv. 183, č. 5, s. 1–8. ISSN 0975-8887. Dostupné z: <http://dx.doi.org/10.5120/ijca2021921311>.
- [19] HOZDIĆ, E. Smart factory for industry 4.0: A review. *International Journal of Modern Manufacturing Technologies*, Červen 2015, sv. 7, č. 1, s. 28–35. ISSN 2067–3604.
- [20] HYNEK, J.; JOHN, P.; FORMÁNKOVÁ, K. a VALNÝ, M. *Služby pro systém řízení a monitoringu vody v retenčních nádržích*. Výzkumná zpráva. Vysoké učení technické v Brně, 2023. [cit. 2024-12-27].
- [21] JOHN, P. *Optimising Processes in IoT*. Brno, 2024. Pojednání k tématu dizertační práce. Brno University of Technology, Faculty of Information Technology. Vedoucí práce ING. TOMÁŠ HRUŠKA, C. prof.

- [22] JOST, B.; KETTERL, M.; BUDDE, R. a LEIMBACH, T. Graphical programming environments for educational robots: Open roberta-yet another one? In: *IEEE. 2014 IEEE International Symposium on Multimedia*. IEEE, Prosinec 2014, s. 381–386. ISBN 978-1-4799-4311-1. Dostupné z: <https://doi.org/10.1109/ISM.2014.24>.
- [23] KOBAYASHI, M.; HIYAMA, A.; MIURA, T.; ASAKAWA, C.; HIROSE, M. et al. Elderly User Evaluation of Mobile Touchscreen Interactions. In: *Human-Computer Interaction – INTERACT 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 83–99. ISBN 978-3-642-23774-4. Dostupné z: https://doi.org/10.1007/978-3-642-23774-4_9.
- [24] KUHAİL, M. A.; FAROOQ, S.; HAMMAD, R. a BAHJA, M. Characterizing visual programming approaches for end-user developers: A systematic review. *IEEE Access*. IEEE, 2021, sv. 9, s. 14181–14202. ISSN 2169-3536. Dostupné z: <https://doi.org/10.1109/ACCESS.2021.3051043>.
- [25] LAUFS, J.; BORRION, H. a BRADFORD, B. Security and the smart city: A systematic review. *Sustainable cities and society*. Elsevier, Duben 2020, sv. 55, s. 102023. ISSN 2210-6707. Dostupné z: <https://doi.org/10.1016/j.scs.2020.102023>.
- [26] LIEBERMAN, H.; PATERNÒ, F.; KLANN, M. a WULF, V. End-user development: An emerging paradigm. In: *End user development*. Dordrecht: Springer, 2006, s. 1–8. ISBN 978-1-4020-5386-3. Dostupné z: https://doi.org/10.1007/1-4020-5386-X_1.
- [27] LIT. *Definining a component* [online]. Dostupné z: <https://lit.dev/docs/components/definining/>. [cit. 2025-04-26].
- [28] LIT. *What is Lit?* [online]. Dostupné z: <https://lit.dev/docs/#what-can-i-build-with-lit>. [cit. 2025-04-26].
- [29] LOGIMIC. *ACADA IoT platforma* [online]. 2014–2021. Dostupné z: <https://www.logimic.com/cs/platforma/>. [cit. 2025-02-05].
- [30] LOXONE. *Loxone Config* [online]. 2025. Dostupné z: <https://www.loxone.com/cscz/kb-cat/loxone-config/>. [cit. 2025-01-26].
- [31] MALAN, D. J. a LEITNER, H. H. Scratch for budding computer scientists. *ACM Sigcse Bulletin*. ACM New York, NY, USA, Březen 2007, sv. 39, č. 1, s. 223–227. Dostupné z: <https://doi.org/10.1145/1227504.1227388>.
- [32] MINERVA, R.; LEE, G. M. a CRESPI, N. Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models. *Proceedings of the IEEE*, Říjen 2020, sv. 108, č. 10, s. 1785–1824. ISSN 1558-2256. Dostupné z: <https://doi.org/10.1109/JPROC.2020.2998530>.
- [33] MOHAMED, K. S. *The Era of Internet of Things: Towards a Smart World*. 1. vyd. Cham: Springer International Publishing, květen 2019. 118 s. ISBN 978-3-030-18133-8. Dostupné z: <http://link.springer.com/10.1007/978-3-030-18133-8>.
- [34] NODE RED. *Node-RED* [online]. Dostupné z: <https://nodered.org/>. [cit. 2025-02-03].

- [35] PATERNÒ, F. End user development: Survey of an emerging field for empowering people. *International Scholarly Research Notices*. Wiley Online Library, Červen 2013, sv. 2013, č. 1, s. 532659. ISSN 2356-7872. Dostupné z: <https://doi.org/10.1155/2013/532659>.
- [36] PODVOJSKÝ, L. *Vizuální programování IoT zařízení*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: <https://hdl.handle.net/11012/246907>. [cit. 2024-11-25].
- [37] PURANIK, A. a. K. A. Automation in Agriculture and IoT. In: *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*. Duben 2019, s. 1–6. ISBN 978-1-7281-1253-4. Dostupné z: <https://doi.org/10.1109/IoT-SIU.2019.8777619>.
- [38] QAZI, S.; KHAWAJA, B. A. a FAROOQ, Q. U. IoT-Equipped and AI-Enabled Next Generation Smart Agriculture: A Critical Review, Current Challenges and Future Trends. *IEEE Access*, 2022, sv. 10, s. 21219–21235. ISSN 2169-3536. Dostupné z: <http://dx.doi.org/10.1109/ACCESS.2022.3152544>.
- [39] RABBY, M. K. M.; ISLAM, M. M. a IMON, S. M. A Review of IoT Application in a Smart Traffic Management System. In: *2019 5th International Conference on Advances in Electrical Engineering (ICAEE)*. IEEE, 2019, s. 280–285. ISBN 978-1-7281-4934-9. Dostupné z: <https://doi.org/10.1109/ICAEE48663.2019.8975582>.
- [40] RANDAZZO, V.; FERRETTI, J. a PASERO, E. ECG WATCH: a real time wireless wearable ECG. In: *2019 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*. IEEE, Srpen 2019, s. 1–6. ISBN 978-1-5386-8428-3. Dostupné z: <https://doi.org/10.1109/MeMeA.2019.8802210>.
- [41] RAY, P. P. A survey on visual programming languages in internet of things. *Scientific Programming*. Wiley Online Library, 2017, sv. 2017, č. 1, s. 1231430. ISSN 1058-9244. Dostupné z: <https://doi.org/10.1155/2017/1231430>.
- [42] RAYHANA, R.; XIAO, G. a LIU, Z. Internet of Things Empowered Smart Greenhouse Farming. *IEEE Journal of Radio Frequency Identification*. IEEE, Zář 2020, sv. 4, č. 3, s. 195–211. ISSN 2469-7281. Dostupné z: <http://dx.doi.org/10.1109/JRFID.2020.2984391>.
- [43] REPENNING, A. Moving Beyond Syntax: Lessons from 20 Years of Blocks Programming in AgentSheets. *J. Vis. Lang. Sentient Syst.*, Červenec 2017, sv. 3, č. 1, s. 68–91. Dostupné z: <https://doi.org/10.18293/VLSS2017-010>.
- [44] SCRATCH. *O Scratchi* [online]. Dostupné z: <https://scratch.mit.edu/about>. [cit. 2025-02-02].
- [45] SINGH, M.; FUENMAYOR, E.; HINCHY, E. P.; QIAO, Y.; MURRAY, N. et al. Digital twin: Origin to future. *Applied System Innovation*. MDPI, 2021, sv. 4, č. 2, s. 36. ISSN 2571-5577. Dostupné z: <https://www.mdpi.com/2571-5577/4/2/36>.
- [46] SNAP4CITY. *Snap4City* [online]. 2022. Dostupné z: <https://www.snap4solutions.org/snap4city/>. [cit. 2025-02-04].

- [47] SOBIN, C. C. A survey on architecture, protocols and challenges in IoT. *Wireless Personal Communications*. Springer, Červen 2020, sv. 112, č. 3, s. 1383–1429. ISSN 1572-834X. Dostupné z: <https://doi.org/10.1007/s11277-020-07108-5>.
- [48] SONI, D. a MAKWANA, A. A survey on mqtt: a protocol of internet of things (iot). In: *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*. Duben 2017, sv. 20, s. 173–177.
- [49] UR, B.; MCMANUS, E.; PAK YONG HO, M. a LITTMAN, M. L. Practical trigger-action programming in the smart home. In: *Proceedings of the SIGCHI conference on human factors in computing systems*. Duben 2014, s. 803–812. Dostupné z: <https://doi.org/10.1145/2556288.2557420>.
- [50] VALDERAS, P.; TORRES, V. a PELOCHANO, V. Supporting a hybrid composition of microservices. The eucaliptool platform. *Journal of Software Engineering Research and Development*, Únor 2020, sv. 8, s. 1:1 – 1:14. ISSN 2195-1721. Dostupné z: <https://doi.org/10.5753/jserd.2020.457>.
- [51] VON HALDENWANG, C. Electronic government (e-government) and development. *The European journal of development research*. Springer, 2004, sv. 16, č. 2, s. 417–432. ISSN 0957-8811. Dostupné z: <https://doi.org/10.1080/0957881042000220886>.
- [52] WASHBURN, D.; SINDHU, U.; BALAOURAS, S.; DINES, R. A.; HAYES, N. et al. Helping CIOs understand “smart city” initiatives. *Growth*. Cambridge, MA: [b.n.], 2010, sv. 17, č. 2, s. 1–17.
- [53] WIKIPEDIE. *Wi-Fi* [online]. 21. listopadu 2025. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=Wi-Fi&oldid=23562095>. [cit. 2025-01-18].
- [54] YIN, C.; XIONG, Z.; CHEN, H.; WANG, J.; COOPER, D. et al. A literature survey on smart cities. *Science China. Information Sciences*. Springer Nature BV, Srpen 2015, sv. 58, č. 10, s. 1–18. Dostupné z: <https://doi.org/10.1007/s11432-015-5397-4>.
- [55] ZHANG, K. *Visual languages and applications*. 1. vyd. Springer Science & Business Media, březem 2007. ISBN 978-0-387-68257-0.

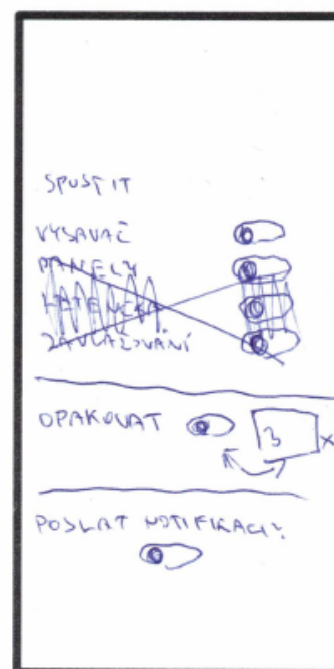
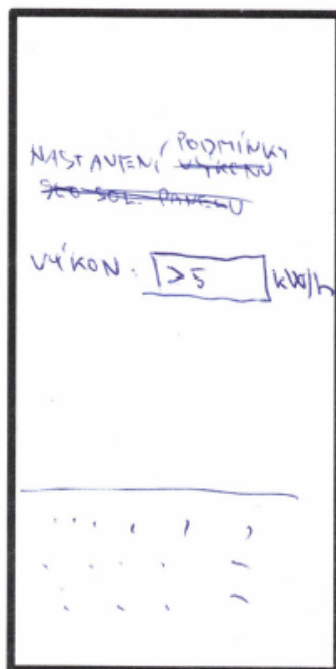
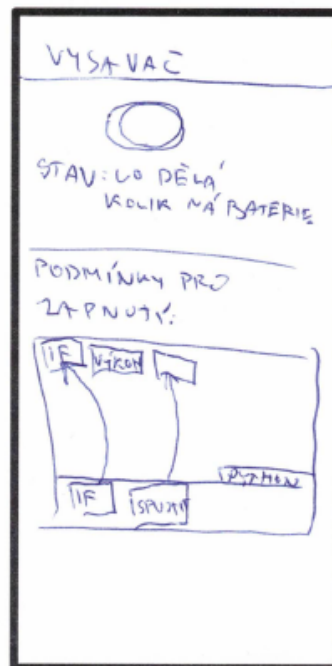
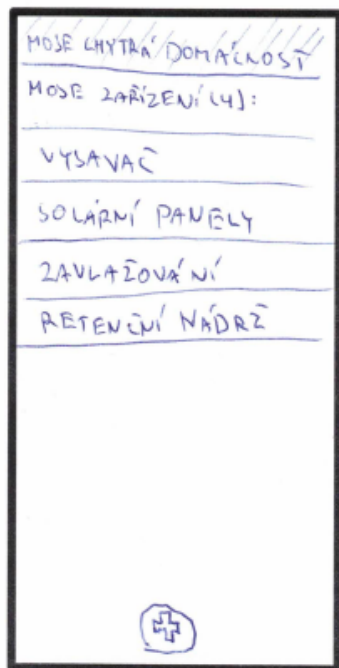
Příloha A

Slovní popis ilustračního programu pro automatizaci chytré domácnosti

Úkol: Představte si, že máme doma solární panely, chytrý vysavač, retenční nádrž, zavlažování a robotickou sekačku. Jak byste chytré domácnosti zadali následující úkol. Pokud mají solární panely aktuální výkon pět kilowatt hodin, tak pustíme třikrát vysavač. Pokud vysavač právě nejede a zároveň platí, že výkon panelů je tři kilowatt hodiny nebo je hladina vody v nádrži nad 70 procent. Spustí zavlažování, schová robotickou sekačku do garáže a pošle mi notifikaci na mobil.

Příloha B

Náčrt aplikace jednoho z respondentů



Obrázek B.1: Jeden z náčrtů vytvořených během výzkumu.

Příloha C

Obsah externí přílohy

- Tento dokument ve formátu pdf.
- Zdrojové kódy tohoto dokumentu.
- `iot-vpl-editor/README.md` popis repozitáře.
- `iot-vpl-editor/src` - zdrojové kódy knihovny.
 - `condition` - komponenty sloužící pro práci s podmínkami.
 - `convert` - funkce pro konverzi programu do textu a VPL.
 - `editor` - komponenty textového a grafického editoru.
 - `general` - globálně používané typy, proměnné a funkce.
 - `icons` - obrázky ikon a komponenty, které je využívají.
 - `variable` - komponenty pro práci s proměnnými.