|  |  |
| --- | --- |
| Univerzita Pavla Jozefa Šafárika v Košiciach  Prírodovedecká fakultaNázov fakultyNázov vysokej školy | |
| Komponentové a udalosťami riadené programovanie zariadení na platforme Arduino | |
|  | |
|  | |
| 2018 | Bc. Patrik Pekarčík |

|  |  |
| --- | --- |
| Univerzita Pavla Jozefa Šafárika v Košiciach  Prírodovedecká fakultaPrírodovedecká fakulta | |
| Komponentové a udalosťami riadené programovanie zariadení na platforme ArduinoKomponentové a udalosťami riadené programovanie zariadení na platforme Arduino | |
| **Diplomová práca**Diplomová práca | |
| Študijný program: | InformatikaInformatika |
| Pracovisko (katedra/ústav): | Ústav InformatikyÚstav Informatiky |
| Vedúci diplomovej práce: | RNDr. František Galčík, PhD.RNDr. František Galčík, PhD. |
| Konzultant diplomovej práce: (nepovinný) | Titul Meno Priezvisko, HodnosťTitul Meno Priezvisko, Hodnosť |
|  |  |
| KošiceKošice 20182018 | Bc.Bc. PatrikPatrik PekarčíkPekarčík |

**Zadanie záverečnej práce**

Zadanie záverečnej práce (ďalej len „zadanie“) je dokument, ktorým vysoká škola stanoví študentovi študijné povinnosti v súvislosti s vypracovaním záverečnej práce. Zadanie spravidla obsahuje: typ záverečnej práce, názov záverečnej práce, meno, priezvisko a tituly študenta, meno, priezvisko a tituly školiteľa, v prípade externého školiteľa meno, priezvisko a tituly konzultanta, školiace pracovisko, meno, priezvisko a tituly vedúceho pracoviska, anotáciu záverečnej práce, jazyk, v ktorom sa práca vypracuje, dátum schválenia zadania.

**Poďakovanie (nepovinné)**

Na tomto mieste môže byť vyjadrenie poďakovania napr. vedúcemu práce resp. konzultantom za pripomienky a odbornú pomoc pri vypracovaní práce. Nie je zvykom ďakovať za rutinnú kontrolu, menšiu spoluprácu alebo všeobecné rady. Vyjadrenie poďakovania v prípade využitia inej práce sa uskutočňuje formou citácie na konci hlavného textu práce a odkazy na citáciu sa musia uviesť aj na zodpovedajúcich miestach v texte.

|  |
| --- |
| Abstrakt v štátnom jazyku |
| Abstrakt obsahuje informáciu o cieľoch práce, jej stručnom obsahu a v závere abstraktu sa charakterizuje splnenie cieľa, výsledky a význam celej práce. Súčasťou abstraktu je  3 - 5 kľúčových slov. Abstrakt sa píše súvisle ako jeden odsek a jeho rozsah je spravidla 100 až 500 slov. |

|  |
| --- |
| Abstrakt v cudzom jazyku |
| Text abstraktu v svetovom jazyku je potrebný pre integráciu do medzinárodných informačných systémov (napr. The Network Digital Library of Theses and Dissertations). Ak nie je možné jazykovú verziu umiestniť na jednej strane so slovenským abstraktom, je potrebné umiestniť ju na samostatnú stranu (cudzojazyčný abstrakt nemožno deliť a uvádzať na dvoch stranách). |

Obsah

[Obsah 5](#_Toc510380930)

[Zoznam ilustrácií 7](#_Toc510380931)

[Zoznam tabuliek 8](#_Toc510380932)

[Zoznam skratiek, značiek a termínov 9](#_Toc510380933)

[Úvod 10](#_Toc510380934)

[1 Internet vecí 11](#_Toc510380935)

[1.1 Hardvér 12](#_Toc510380936)

[1.2 Softvér 13](#_Toc510380937)

[1.3 Mikrokontrolér v porovnaní so single-board počítačom 13](#_Toc510380938)

[2 Platforma Arduino 14](#_Toc510380939)

[2.1 Parametre mikrokontroléra 15](#_Toc510380940)

[2.2 Programovanie pre mikrokontroléry 16](#_Toc510380941)

[2.3 Existujúce riešenia pre platformu Arduino 17](#_Toc510380942)

[2.3.1 Arduino EventManager 17](#_Toc510380943)

[2.3.2 Quantum Leaps Modeling Tool 18](#_Toc510380944)

[2.3.3 ARTe (Arduino Real-Time extension) 18](#_Toc510380945)

[2.3.4 Cayenne 18](#_Toc510380946)

[3 Komponentové a udalosťami orientované programovanie 20](#_Toc510380947)

[3.1 Spracovanie udalosti 21](#_Toc510380948)

[4 Integrované vývojové prostredie 23](#_Toc510380949)

[4.1 Rapid Application Development 23](#_Toc510380950)

[5 Architektúra komponentového a udalosťami orientovaného riešenia ACProg 24](#_Toc510380951)

[5.1 XML konfiguračný súbor 25](#_Toc510380952)

[5.2 Generátor knižnice pre Arduino 26](#_Toc510380953)

[5.3 Typy komponentov (moduly) 27](#_Toc510380954)

[6 IDE pre projekt ACProg 28](#_Toc510380955)

[6.1 Používateľské požiadavky 28](#_Toc510380956)

[6.1.1 Grafický návrh IDE 29](#_Toc510380957)

[6.2 Technologický návrh 31](#_Toc510380958)

[6.2.1 Grafický framework Java Swing 31](#_Toc510380959)

[6.2.2 Rozloženie aplikácie – Docking framework 32](#_Toc510380960)

[6.2.3 Code editor 33](#_Toc510380961)

[6.2.4 Ponuka komponentov (modulov) a inštancie komponentov. 34](#_Toc510380962)

[6.2.5 Vlastnosti inštancie komponentu (properties) 35](#_Toc510380963)

[6.3 Architektúra IDE 36](#_Toc510380964)

[6.4 Implementácia IDE 37](#_Toc510380965)

[6.4.1 Ponuka dostupných komponentov 37](#_Toc510380966)

[6.4.2 Skupinové zobrazenie inštancií komponentov 38](#_Toc510380967)

[6.4.3 Kompilácia a spustenie projektu 39](#_Toc510380968)

[6.4.4 Syntaktická analýza kódu 40](#_Toc510380969)

[6.4.5 Voliteľný UX pre programátora 42](#_Toc510380970)

[7 Vzorové komponenty pre ACProg 44](#_Toc510380971)

[7.1 Časovač 44](#_Toc510380972)

[7.2 Digitálny výstup 44](#_Toc510380973)

[7.3 Digitálny vstup 44](#_Toc510380974)

[7.4 Analógový vstup s threshold alert 44](#_Toc510380975)

[7.5 Komunikácia pomocou MQTT 44](#_Toc510380976)

[7.6 Rádiová komunikácia pomocou 433 MHz 44](#_Toc510380977)

[Záver 45](#_Toc510380978)

[Zoznam použitej literatúry 46](#_Toc510380979)

[Prílohy 48](#_Toc510380980)

Zoznam ilustrácií

Obr. 1 Schéma reprezentujúca IoT [2] 11

Obr. 2 Doska Arduino (vľavo) [3] a rozširujúca doska Shield (vpravo) [4] 14

Obr. 3 Arduino IDE s vyznačením kompilácie a spustenie. 15

Obr. 4 Rozdelenie zdrojového kódu Arduino príkladového programu 16

Obr. 5 Zaregistrovanie udalosti na analógovom pine 2 v setup() a spracovanie udalostí v loop(). 18

Obr. 6 Akcia zapnutia klimatizácie nastavená vo webovom prostredí Cayenne. 19

Obr. 7 Znázornenie inštancie komponentu typu tlačidlo s automatom pre prechody medzi jednotlivými stavmi. 21

Obr. 8 Práca plánovača úloh s registráciou a vyvolaním udalosti 22

Obr. 9 Priebeh spracovania projektu riešením ACProg 24

Obr. 10 Priebeh spracovania projektu, riešením ACProg 26

Obr. 11 Prototyp integrovaného vývojového prostredia pre ACProg 30

Obr. 12 DockingFrames ilustrácia ukladania záložiek v okne [12] 32

Obr. 13 Editor zdrojového kódu s ilustrovaním vyznačenia chyby. 34

Obr. 14 Ponuka komponentov (vľavo), inštancie komponentov (vpravo). 35

Obr. 15 Vizuál grafického komponentu JTree so zdrojovým kódom. 38

Obr. 16 Rozdelenie grafickej implementácie. 38

Obr. 17 Priebeh spracovania projektu riešením ACProg 40

Obr. 18 Ilustrácia chybnej konfigurácie inštancií komponentov. 41

Obr. 19 Ilustrácia AST (vpravo) zo zdrojového kódu (vľavo). 42

Obr. 20 Navrhnuté rozloženia integrovaného vývojového prostredia. 43

Zoznam tabuliek

Tab. 1 Porovnanie parametrov pre rôzne modely dosiek Arduino 16

Zoznam skratiek, značiek a termínov

.ino Prípona zdrojových súborov programovacieho jazyka Arduino

ACProg **A**rduino **C**omponent **Prog**ramming

AST **A**bstract **S**yntax **T**ree

Cloud Výkonné počítačové prostriedky umiestnené v sieti internet

CPU **C**entral **P**rocessing **U**nit – Centrálna výpočtová jednotka

EEPROM **E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory

Hz **H**ertz, jednotka frekvencie používaná na meranie rýchlosti CPU

IDE **I**ntegrated **D**evelopment **E**nvironment – Integrované vývojové prostredie

IoT **I**nternet **o**f **T**hings

I2C **I**nter-**I**ntegrated **C**ircuit − zbernica používaná pre nízkoúrovňové zariadenia

RAD **R**apid **A**pplication **D**evelopment

RAM **R**andom **A**ccess **M**emory

ROM **R**ead **O**nly **M**emory

SD **S**ecure **D**igital

USB **U**niversal **S**erial **B**us

UX **U**ser E**x**perience

XML E**x**tended **M**arkup **L**anguage

Úvod

Hlavnú textovú časť záverečnej práce tvorí: úvod, jadro, záver, resumé (povinné iba v prípade, ak je práca vypracovaná v inom ako štátnom jazyku), zoznam použitej literatúry.

V úvode autor stručne a výstižne charakterizuje stav poznania alebo praxe v oblasti, ktorá je predmetom záverečnej práce a oboznamuje čitateľa s významom, cieľmi a zámermi práce. Autor v úvode zdôrazňuje, prečo je práca dôležitá a prečo sa rozhodol spracovať danú tému.

# Internet vecí

Pojem internetu vecí (IoT) pokrýva viac ako len koncept alebo technológiu. Je to nový prístup, ktorý ovplyvňuje vývoj aktuálnych technológií, aplikácií a vízií.

Kvôli rozsiahlosti oblasti IoT zatiaľ neexistuje presná definícia tohto pojmu. Viaceré vedecké články sa zhodujú v tom, že kľúčovým prvkom IoT je prepojenie fyzického a digitálneho sveta pomocou senzorov a aktuátorov. Okrem toho je pre IoT zariadenia dôležité vedieť komunikovať s inými zariadeniami, počítačmi či ľuďmi pomocou počítačovej siete [1] pre následné vykonávanie akcií. Nasledujúca schéma reprezentuje IoT v komunikačnej postupnosti.



Obr.  Schéma reprezentujúca IoT [2]

Pri popise schémy je vhodné začať senzormi a následne pokračovať v smere toku dát. Senzory sú hardvérové zariadenia, ktoré zbierajú informácie z fyzického sveta a menia ich na digitálny signál. Signál smeruje do mikrokontroléra, ktorý vykonáva lokálne spracovanie. Lokálne spracovanie spočíva v prevode signálu na digitálne dáta. Okrem tohto prevodu môže mikrokontrolér vykonávať akcie a ihneď ovplyvňovať fyzický svet.

Po lokálnom spracovaní sa dáta uložia na lokálne úložisko s účelom ich budúcej analýzy. Keďže hovoríme o internete vecí, tak je na mieste uvažovať o online zdieľaní dát s ostatnými zariadeniami pomocou počítačovej siete. IoT server je zariadenie na sieti, ktoré prijíma dáta zo všetkých senzorov a vykonáva cloud spracovanie.

Pozícia cloud spracovania je veľmi podobná lokálnemu spracovaniu. Hlavným rozdielom je množstvo spracovávaných dát. Kým pre lokálne spracovanie máme k dispozícii nízke výkonnostné prostriedky, pri cloud spracovaní ide o gigabajty pamäte s výkonnými procesormi. Tie nám umožňujú vykonávať náročné spracovania prijatých dát. Všetky spracované dáta sa následne ukladajú na cloud úložisko.

Opísaná schéma predstavuje potenciál, ktorý IoT poskytuje. V mnohých IoT projektoch sa však stáva, že lokálne spracovanie je úplne vynechané a všetky nazbierané dáta zo senzorov sa priamo odosielajú na cloud spracovanie.

## Hardvér

Základným stavebným prvkom lokálneho spracovania v IoT je mikrokontrolér, ktorý riadi prácu zariadenia. Na mikrokontrolér sú pripojené senzory, aktuátory a obvody podporujúce komunikáciu s ďalšími zariadeniami. Mikrokontrolér je elektronický čip obsahujúci obvody pre zavedenie a beh softvéru. Menovite sú to RAM, ROM a CPU.

Avšak parametre týchto obvodov sú relatívne nízke oproti bežne dostupným počítačom. Napríklad mikrokontrolér ATMEGA328, ktorý používajú aj dosky Arduino, disponuje 32 kB ROM a 2 kB RAM so 16-bitovým 20 MHz CPU v cene približne dvoch dolárov.

**Senzory** sú elektronické súčiastky, pomocou ktorých mikrokontrolér získava informácie o fyzickom svete v digitálnej informácii, napr. odmeraním elektrického odporu. Komunikácia senzora s mikrokontrolérom spočíva v čítaní elektrického napätia, ktoré mikrokontrolér spracuje do digitálnej podoby a prevedie do meraných jednotiek. Napríklad pri meraní elektrického odporu by sme prijaté elektrické napätie prepočítali na digitálnu verziu v ohmoch.

**Aktuátory** sú elektronické súčiastky ovplyvňujúce fyzický svet. Zaraďujeme sem motory, svetlá, reproduktory a pod. Tieto súčiastky sú riadené informáciami prijatými od mikrokontrolérov.

## Softvér

Na riadenie mikrokontroléra je potrebný obslužný softvér napálený v jeho pamäti ROM. Tento softvér je zavedený pri privedení elektrického prúdu do mikrokontroléra. Pomocou neho mikrokontrolér číta hodnoty vstupných pinov (zo senzorov) a riadi výstupné piny (k aktuátorom). Najčastejším programovacím jazykom pre mikrokontroléry je upravená verzia jazyka C. Netreba zabúdať, že mikrokontroléry poskytujú len rádovo 2 kB operačnej pamäte pre beh vytvorených programov.

Okrem samotných programov pre mikrokontrolér je dôležitý aj podporný softvér. Podporný softvér slúži programátorom na uľahčenie vývoja. Základom je kompilátor, pomocou ktorého vieme previesť program. Integrované vývojové prostredia tiež radíme medzi podporné softvéry. Jeho hlavnou úlohou je odbremeniť programátorov od opakovaných krokov a poskytnúť jednoduchú konfiguráciu bez väčšej potreby znalosti celého systému. Integrované vývojové prostredia taktiež prispievajú k rýchlejšiemu prototypovaniu nových nápadov.

## Mikrokontrolér v porovnaní so single-board počítačom

Hlavnou návrhovou črtou mikrokontroléra je komunikácia so senzormi, aktuátormi a pod. Pre tento typ komunikácie je dôležité presné časovanie prenosu signálov, ktoré nám mikrokontroléry priamo poskytujú.

Pri single-board počítačoch je hlavnou črtou vyšší výpočtový výkon za nízku cenu. Výkon na týchto počítačoch sa pohybuje okolo 1 GB RAM s ARM 1,2 Ghz procesorom. Tieto parametre nám už dovoľujú spustiť na procesore odľahčený operačný systém.

Avšak nevýhodou operačných systémov je, že prenechávame časovanie a plánovanie na nich. Preto je priama stabilná komunikácia so senzormi a aktuátormi takmer nemožná. Single-board počítače často poskytujú rôzne mikrokontroléry implementujúce komunikačné protokoly používané v elektronike (napr. I2C). Vďaka nim vieme pomocou single-board počítačov komunikovať so senzormi. Avšak počet takých senzorov na trhu je veľmi nízky.

# Platforma Arduino

Projekt Arduino sa začal v roku 2003 ako program pre študentov v Interaction Design Institute Ivrea v talianskom Ivreu. Vznikol s cieľom poskytnúť lacný a jednoduchý spôsob vytvárania zariadení, ktoré interagujú s prostredím pomocou senzorov a aktuátorov. Bežné príklady takýchto zariadení, určených pre začiatočníkov, zahŕňajú jednoduché roboty, termostaty a detektory pohybu.

Arduino je hardvér a softvér s otvoreným zdrojovým kódom, chráneným licenciami GNPL Lesser General Public License (LGPL) a GNU General Public License (GPL). Licencie umožňujú výrobu dosiek Arduino a distribúciu softvéru komukoľvek.

Dosky Arduino sú špecifické vyvedenými pinmi na ich okraji, ktoré je kľúčové pre jednoduché prototypovanie embedovaných zariadení. Vďaka štandardizovanému tvaru a vyvedeniu pinov dosky Arduino je možné vytvárať rozširujúce dosky, tzv. Arduino Shield. Arduino Shield poskytuje rovnaký tvar a rozloženie pinov, ako má Arduino doska. Navyše dodáva rôzne funkcionality, či už sú to senzory, aktuátory, alebo kompletné riešenie pre embedovaný systém. Dosky Arduino majú vstavané sériové komunikačné rozhrania, vrátane univerzálnej sériovej zbernice (USB).



Obr.  Doska Arduino (vľavo) [3] a rozširujúca doska Shield (vpravo) [4]

Softvérová časť projektu Arduino spočíva v integrovanom vývojovom prostredí (IDE) a programovacom jazyku postavenom na C++. Integrované vývojové prostredie bolo postavené na projekte Language Processing. Keďže väčšina dosiek Arduino má vstavané rozhranie USB, kompilácia s napálením projektu z IDE do zariadenia je záležitosťou jedného kliknutia.



Obr.  Arduino IDE s vyznačením kompilácie a spustenie.

## Parametre mikrokontroléra

Základným prvkom dosky Arduino je mikrokontrolér. Pri originálnych doskách Arduino sa používajú mikrokontroléry Atmel ATmega. Na čipe mikrokontroléra sa nachádzajú obvody procesora (CPU), operačnej pamäte (RAM), programovej pamäte (ROM) a trvalej pamäte (EEPROM).

Pre komunikáciu s ostatnými zariadeniami sú z mikrokontroléra vyvedené vstupno-výstupné piny. Výkonnostné hodnoty týchto parametrov sú pomerne nízke, a preto je potrebné na ne brať ohľad pri návrhu softvéru. Tabuľka č. 1 zobrazuje konkrétne parametre niekoľkých modelov dosiek Arduino pre predstavu nízkych parametrov. Na ilustráciu výkonnostného rozdielu medzi mikrokontrolérom a single-board počítačom sme v poslednom riadku tabuľky vypísali parametre single-board počítača Raspberry PI. Pri single-board počítači si môžeme všimnúť absenciu analógových vstupno-výstupných pinov.

Tab.  Porovnanie parametrov pre rôzne modely dosiek Arduino

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model Arduino** | **CPU Model** | **ROM (kB)** | **RAM (kB)** | **EEPROM (kB)** | **CPU frekvencia (MHz)** | **Analógové/digitálne piny** |
| UNO | ATmega323P | 32 | 2 | 1 | 16 | 6/14 |
| Mega 2560 | ATmega2560 | 256 | 8 | 4 | 16 | 16/54 |
| Nano | ATmega168 | 16 | 1 | 0,512 | 16 | 8/14 |
| 101 | Intel Curie | 196 | 24 | − | 32 | 6/14 |
| Gemma | ATtiny85 | 8 | 0,5 | 0,5 | 8 | 1/3 |
| Raspberry PI (single-board) | ARMv8 64 bit quad-core | SD slot | 512 MB | SD slot | 1200 | 0/40 |

## Programovanie pre mikrokontroléry

Programovacím jazykom pre mikrokontroléry je najčastejšie istá obdoba vyššieho programovacieho jazyka C++. Tvorcovia mikrokontrolérov vydávajú špecifikácie jazyka a syntaxe spolu s kompilátorom. Kompilátory prevádzajú napísaný program na assembler pracujúci s konkrétnymi registrami a inštrukciami, ktoré sú hardvérovo implementované na použitom mikrokontroléri.

Nasledujúci obrázok ukazuje syntax a usporiadanie zdrojového kódu na jednoduchom programe.



Obr.  Rozdelenie zdrojového kódu Arduino príkladového programu

V úvodnej sekcii je vypísaný zoznam knižníc a rozširujúcich súborov, ktoré program bude používať. V ďalšej sekcii sú definované premenné a konštanty ovplyvňujúce beh programu. Taktiež v tejto sekcii môžeme vytvoriť pomocné metódy pre prehľadnejší zdrojový kód.

Nasleduje sekcia zavedenia, v ktorej nastavujeme zariadenie na mód, v akom bude pracovať. Piny na mikrokontroléry sú vstupno-výstupné, avšak pre konkrétny program ich spravidla využívame na jeden mód (buď ako vstupné, alebo výstupné piny).

Pri zavedení môžeme nastaviť aj časovače zariadenia. Arduino UNO disponuje tromi časovačmi. Časovače v nastavenom intervale spustia zadanú sadu inštrukcií.

Poslednou sekciou je sekcia behu programu (ďalej loop sekcia). Loop sekcia je cyklicky spúšťaná a predstavuje správanie mikrokontroléra. V tejto sekcii mikrokontrolér vykonáva všetky čítania senzorov a základné lokálne spracovanie zozbieraných dát.

## Existujúce riešenia pre platformu Arduino

Zariadeniam Arduino sa venuje čoraz väčšia komunita. S tým je spojený aj vývoj rôznych frameworkov pre túto platformu, ktorých cieľom je zjednodušiť vývoj pre programátorov. Riešenia, aké sme našli vyhľadávaním na fórach Arduino komunity, sme rozdelili do dvoch kategórií: online a offline riešenia.

**Online** sú riešenia, v ktorých je do zariadenia nainštalovaný zavádzač. Úlohou zavádzača je preposielať všetky údaje zo senzorov na sieť, kde bude vykonané cloud spracovanie. Vo všetkých analyzovaných online riešeniach bola úplne vynechaná časť lokálneho spracovania a uloženia dát. **Offline** sú riešenia bežiace priamo na Arduino zariadení. Takéto riešenia využívajú najmä lokálne spracovanie s lokálnym uložením.

V nasledujúcich sekciách priblížime vybrané existujúce riešenia, ktoré sa svojou povahou najviac približujú nami navrhnutému riešeniu pre programovanie Arduino zariadení.

### Arduino EventManager

Knižnica Arduino EventManager [5] patrí do skupiny offline riešení. Pri využití tejto knižnice sa nevytvára periodicky vykonávaná funkcia loop(). Namiesto nej v inicializačnej funkcii setup() zaregistrujeme obslužné funkcie (angl. callback) a programujeme spracovanie udalostí. Pri registrovaní funkcie určíme konkrétny pin, pri ktorého zmene má byť udalosť vyvolaná.

void myListener( int eventCode, int eventParam ) {

// Do something with the event

}

void setup() {

gMyEventManager.addListener( EventManager::kEventAnalog2, myListener );

// Do more set up

}

void loop()

gMyEventManager.processEvent();

}

Obr.  Zaregistrovanie udalosti na analógovom pine 2 v setup() a spracovanie udalostí v loop().

### Quantum Leaps Modeling Tool

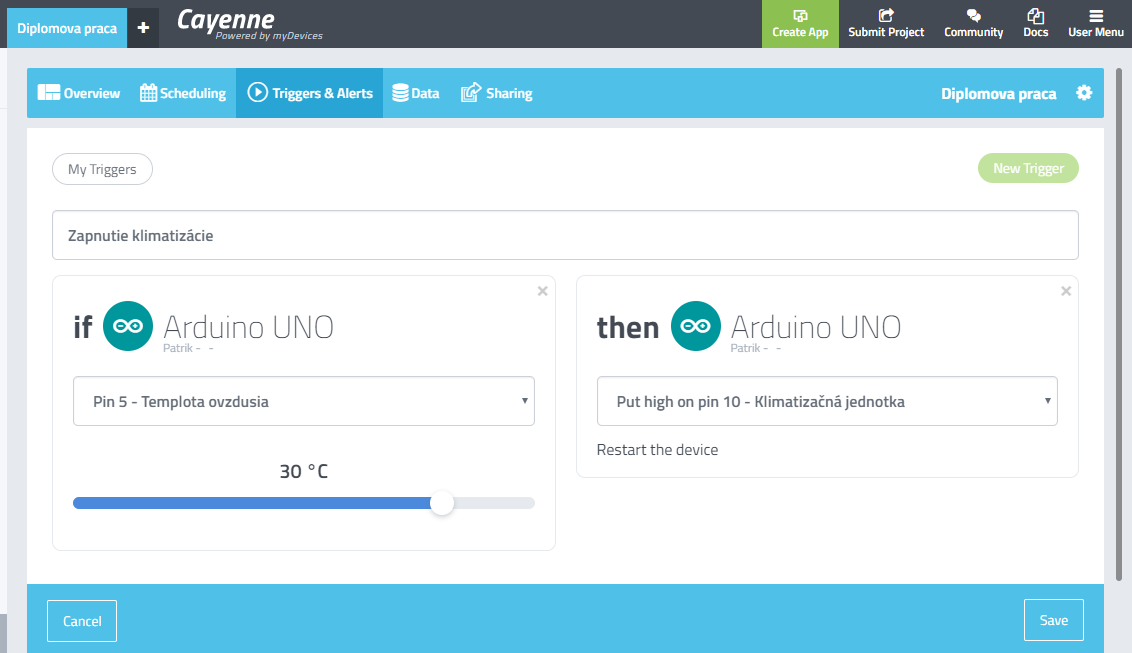
Modelovací nástroj Quantum Leaps Modeling Tool [6] zaradzujeme do skupiny offline riešení. Aplikácie v tomto nástroji modelujeme pomocou stavového automatu. Konečnostavový automat reprezentuje matematický model výpočtu. Ide o abstraktný stroj, ktorý môže byť v danom čase presne v jednom stave z konečného počtu stavov. Konečnostavový automat sa môže prepnúť z jedného stavu na druhý v reakcii na niektoré externé vstupy.

### ARTe (Arduino Real-Time extension)

Rozšírenie Arduino kompilátora ARTe [7] zaradzujeme do skupiny offline riešení. Rozšírenie umožňuje vytvorenie viacerých loop funkcií, ktoré sú spustené pseudoparalelne. Vývojári ARTe dosiahli pseudoparalelizmus prepísaním implementácie čakacích (delay[[1]](#footnote-1)) metód z aktívneho čakania na pasívne. Program si zapamätá, kde bol delay vyvolaný a spustí ďalšiu nasledujúcu loop funciu. Ďalšia loop funkcia je spustená od zapamätanej pozícii vyvolania delay. Ak funkcia nemala vyvolaný delay, tak bude spustená od začiatku.

### Cayenne

Aplikácia Cayenne [8] patrí do kategórie online riešení. Na dosku Arduino je nahraná špeciálna aplikácia, ktorá automatizovane odosiela na servery Cayenne aktuálny stav svojich vstupných pinov. Doska taktiež čaká na inštrukcie zo serverov Cayenne pre nastavenie výstupných pinov. Vo webovom nástroji Cayenne vidíme aktuálny stav vstupných pinov a dokážeme nastaviť rôzne akcie pre zapnutie výstupných pinov. Akcie nastavujeme ako podmieňovacie vety. Nasledujúci príklad zobrazuje zapojený senzor (teplomer) s aktuátorom (klimatizácia), ktorý čaká na zmenu stavu popísanú vetou: ak stúpne teplota nad 30 °C, tak zapni klimatizáciu.



Obr.  Akcia zapnutia klimatizácie nastavená vo webovom prostredí Cayenne.

# Komponentové a udalosťami orientované programovanie

S komponentovým a udalosťami orientovaným programovaním sa stretávame najmä pri používateľských aplikáciách na počítačoch, prípadne mobilných zariadeniach. Komponenty v týchto aplikáciách predstavujú jednotlivé grafické prvky zobrazené na obrazovke.

V objektovo orientovanom programovaní je **komponent** považovaný za znovu použiteľný blok programu, ktorý môže byť kombinovaný s inými komponentmi za účelom vytvorenia aplikácie. Príkladmi komponentov sú tlačidlo, scrollbar, ale aj tabuľka v grafických používateľských prostrediach. Zdrojový kód komponentu, inak nazývaný ako model komponentu, obvykle poskytuje nasledujúce hlavné typy služieb:

* Vlastnosti komponentu − vďaka vlastnostiam komponentov je inštancia komponentu rozlíšiteľná v aplikácii od ostatných komponentov rovnakého typu.
* Spracovanie udalostí − inštancia komponentu má definované vlastné spracovanie udalostí. Napríklad tlačidlo A po udalosti stlačenia uloží obsah pamäte na disk. Iné tlačidlo B po rovnakej udalosti stlačenia spustí zvonenie na komponente reproduktora.

Význam slova v  jazyku je „výskyt niečoho v určitom čase“. To isté platí aj pre udalosti v udalosťami orientovanom programovaní [9]. V programovaní sú s udalosťami spojené tri typy objektov:

* **Zdroj udalosti** − komponent, ktorého zmena vlastností vyvolá udalosť.
* **Udalosť** − pomenovanie konkrétnej zmeny vlastnosti komponentu.
* **Spracovanie udalosti** − obslužný kód, ktorý je spustený po vytvorení udalosti.

Typy komponentov majú spravidla definované mená udalostí, ktoré môžu nastať. Pre vyvolanie udalosti komponent sleduje zmeny svojich vlastností. Nasledujúci obrázok znázorňuje prechody medzi jednotlivými stavmi formou automatu. Na šípkach sa nachádza pomenovanie udalosti, ktorá nastáva pri prechode medzi vyznačenými stavmi. Hrubým kruhom je vyznačený aktuálny stav tlačidla.

obrazok

Obr.  Znázornenie inštancie komponentu typu tlačidlo s automatom pre prechody medzi jednotlivými stavmi.

## Spracovanie udalosti

Aby program mohol spracovávať udalosti, potrebuje niekoľko samostatných modulov [10]. Prvým je samotný komponent, ktorý udalosť vyvolá pri zmene svojho stavu. Druhým je metóda alebo objekt, ktorý čaká na vznik udalosti, a následne ju spracuje.

Posledným a zároveň najdôležitejším modulom, ktorý prepája prvé dva moduly, je plánovač udalostí. Plánovač udalostí je program, ktorý riadi beh programu prijímaním a  sekvenčným vykonávaním prijatých udalostí. V nasledujúcom obrázku je uvedený nákres práce plánovača úloh v čase.

obrazok

Obr.  Práca plánovača úloh s registráciou a vyvolaním udalosti

Na začiatku objekt SpracujUdalost vykoná registráciu na udalosť s názvom stlacenieTlacidla (komponentu Tlacidlo). Registráciu vykoná na plánovači úloh. Ten si do svojej internej pamäte zapíše túto registráciu. Následne, keď používateľ stlačí komponent Tlacidlo, tak komponent zistí zmenu stavu a vytvorí udalosť. Tá je odoslaná na plánovač úloh, ktorý ju zaradí do radu spracovávania úloh. Keď príde na rad, tak plánovač úloh skontroluje zoznam registrácií na udalosti a vyvolá príslušnú udalosť. V ukázanom prípade sa spustí spracovanie udalosti z objektu SpracujUdalost.

# Integrované vývojové prostredie

Integrované vývojové prostredia (IDE) vznikli, aby zjednodušili programovanie aplikácií a informačných systémov. IDE sú navrhnuté tak, aby zahŕňali všetky úlohy spojené s vývojom softvéru v jednej aplikácii. Medzi ne patrí hlavne:

* editor zdrojového kódu,
* kompilátor,
* nástroje automatizácie.

Editor zdrojového kódu je grafický komponent, navrhnutý tak, aby zjednodušil písanie programov. Jeho hlavnou črtou je grafické vyznačovanie kľúčových slov v zdrojovom kóde. V lepších IDE je editor doplnený aj o automatické dokončovanie. Automatické dokončovanie podľa znalosti programovacieho jazyka, ale aj už vytvoreného zdrojového kódu, programátorovi ponúka existujúce metódy či premenné. Úloha integrovaného kompilátora je kľúčová pre IDE, pretože testovanie/spúšťanie programu je vďaka tomu na jedno kliknutie. Medzi nástroje automatizácie radíme rôzne ďalšie rozšírenia, ktoré sú spúšťané napríklad pred kompiláciou programu. Medzi tieto nástroje môžeme zaradiť aj generátor zdrojového kódu, ktorý prevedie podporné súbory IDE do jazyka, v ktorom bude kompilovaný.

## Rapid Application Development

Metodológia stojaca bok po boku integrovaných vývojových prostredí je Rapid application development (RAD). RAD bol navrhnutý ako prístup adaptívneho softvérového vývoja. Odporúča sa, aby sa kládol menší dôraz na plánovanie a väčší na adaptívne zmeny v softvéri. Taktiež sa odporúča zvýšené používanie prototypov, pretože tie skôr poskytnú spätnú väzbu o výslednom softvéri.

# Architektúra komponentového a udalosťami orientovaného riešenia ACProg

Navrhnuté komponentovo a udalosťami orientované riešenie pre Arduino zariadenia nazývame ACProg. Toto riešenie, podľa kapitoly 2.3, zaradzujeme do kategórie offline riešení. Od bežného programovania Arduino zariadení sa líši hlavne tým, že programátor nepotrebuje programátorsky inicializovať zariadenie. Túto inicializáciu za neho vykoná ACProg generátor pomocou priloženého konfiguračného XML súboru.

Konfiguračný XML súbor obsahuje nastavenia modelu Arduina, definovanie projektu ACProg a zoznam komponentov s ich inicializačnými nastaveniami. Na nasledujúcom obrázku ilustrujeme schému použitia ACProg z pohľadu programátora.



Obr.  Priebeh spracovania projektu riešením ACProg

Súborová štruktúra v bode 1 zobrazuje dva hlavné projektové súbory. Projekt.xml je konfiguračný súbor projektu, ktorému sa budeme detailnejšie venovať v ďalšej podkapitole. Projekt.ino je zdrojový kód projektu. XML súbor je vstupom do generátora zdrojového kódu.

Generátor (bod 2) pomocou konfiguračného súboru vytvorí zdrojový kód komponentov a implementuje základné metódy Arduino programovania setup() a loop(). Vygenerovaný kód je dostupný ako Arduino knižnica (bod 3), ktorá musí byť načítaná v zdrojovom kóde Projekt.ino. Prvý riadok zdrojového kódu bude #include “Project.h“. Vygenerovaná knižnica a súbor zdrojového kódu Projekt.ino je následne vstupom pre Arduino kompilátor (bod 4). Kompilátor vytvorí súbor inštrukcií pre mikrokontrolér na zvolenom modeli Arduino dosky. Tieto inštrukcie môžeme nahrať na mikrokontrolér a spustiť vytvorený program.

## XML konfiguračný súbor

V tejto podkapitole vysvetlíme, ako správne vytvoriť konfiguračný súbor pre generátor projektu ACProg. Na začiatok ukážeme vo vnorenom zozname všetky elementy, s vysvetlením v nasledujúcich odstavcoch. V poslednej sekcii tejto podkapitoly je nami vytvorená ukážková XML konfigurácia pre lepšiu vizuálnu predstavu.

* <Project>
  + <Program>
    - <Events> – pole elementov
      * <Event>
  + <Components> – pole elementov
    - <Component>
      * <Name>
      * <Type>
      * <Properties> – pole elementov
        + <Property>
      * <Events> – pole elementov
        + <Event>

Hlavným koreňovým elementom XML konfiguračného súboru je project. Povinným atribútom je platform, ktorým určíme model Arduino dosky, pre ktorú budeme vytvárať projekt. Tento atribút slúži na kontroly použitých hardvérových prvkov v nasledujúcich prvkoch konfigurácie a tiež na nastavenie kompilátora pri kompilácii.

Nasledujúci element program obsahuje číselný atribút watchdog-level, na základe ktorého generátor nastaví hardvérový modul watchdogu na mikrokontroléri. Watchdog je modul, kontrolujúci, či sa program nezasekol v mŕtvej slučke. Ak by detekoval podobnú udalosť, zariadenie samostatne reštartuje.

Element program obsahuje pole elementov events, uchovávajúce registrácie na udalosti. Programátor sa tak môže pripojiť na metódy setup() a loop() z bežného programovania Arduino zariadení.

Nasledujúcim elementom konfiguračného súboru je pole inicializácií komponentov components. Pole obsahuje elementy component, tie obsahujú 2 povinné elementy name a type. Element type určuje typ komponentu, aký budeme inicializovať. Element name určuje názov inštancie komponentu, pod ktorým k nemu budeme pristupovať v zdrojovom kóde.

Element komponentu ďalej obsahuje pole vlastností properties, pomocou ktorých sa nastavujú počiatočné vlastnosti podľa zvoleného typu komponentu. Posledným elementom komponentu je pole udalostí events, v ktorom registrujeme udalosti, aké komponent dokáže vytvoriť na naše obslužné funkcie.

<?xml version=“1.0“ encoding=“UTF-8“ standalone=“no“?>

<**project** platform=“ArduinoUno“>

<**program** watchdog-level=“5“>

<**events**>

<**event** name=“OnLoop“>onLoop</event>

<**event** name=“OnStart“>onStart</event>

</events>

</program>

<**components**>

<**component**>

<**name**>blinkTimer</name>

<**type**>acp.common.timer</type>

<**properties**>

<property name=“Enabled“>true</property>

<property name=“Interval“>1000</property>

</properties>

<**events**>

<**event** name=“OnTick“>mojeTajneKliknutie</event>

</events>

</component>

</components>

</project>

Obr.  Priebeh spracovania projektu, riešením ACProg

## Generátor knižnice pre Arduino

Pri snahe zachovať syntax a princípy programovania Arduino zariadení sme sa rozhodli nevytvoriť úplne nový programovací jazyk. Vybrali sme sa smerom vytvorenia Arduino knižnice podľa popisu komponentov. Toto vytvorenie sa nazýva generovanie zdrojového kódu.

Generátor dostáva na vstupe zoznam komponentov, ktoré majú určený typ (modul). Generátor pridá do výslednej knižnice zdrojové kódy komponentu, ak tam ešte nie sú, a pri zavedení vytvorí kód vytvorenia inštancie. Inštancie komponentov sú globálne a používateľ v nich bude môcť priebežne programátorsky meniť konfigurácie. Okrem komponentov generátor vytvorí aj jadro nášho riešenia. Jadro spočíva v inicializácii komponentov a následnom plánovaní vykonávania jednotlivých udalostí. Taktiež poskytuje programátorovi aj aplikačné rozhranie na komunikáciu so stálou pamäťou EEPROM.

## Typy komponentov (moduly)

Predpisy typov komponentov sú uložené v repozitári generátora. Repozitár je priečinok uložený na disku. Názov typu komponentu využíva generátor na nájdenie definičného súboru komponentu v súborovej štruktúre repozitára. Názov je delený bodkami, čo pre repozitár znamená delenie po priečinkoch.

Každý typ komponentu potrebuje definičný XML súbor pre generátor. Definičný súbor okrem rozšírenej poznámky o komponente obsahuje aj informácie o súboroch s triedou komponentu a so zoznamom parametrov konštruktora. Ďalej sú v tomto súbore definované názvy všetkých vlastností komponentov a udalostí, ktoré definovaný komponent dokáže vyvolať.

# IDE pre projekt ACProg

Po vzore úspechu projektu Arduino je dôležité vytvoriť k novému riešeniu aj používateľsky prívetivé prostredie. Prostredie, v ktorom programátor nie je nútený poznať celé pozadie projektu ACProg a bude mu stačiť krátky koncepčný popis tohto riešenia.

V nasledujúcich podkapitolách priblížime používateľské podmienky na nové integrované vývojové prostredie a predstavíme technológie navrhované pre vytvorenie prostredia a architektúru výsledného projektu. V ďalšej podkapitole sa budeme venovať implementačným krokom, ktoré viedli k naplneniu používateľských požiadaviek.

## Používateľské požiadavky

Pred začatím prác na akomkoľvek informačnom systéme (alebo softvéri) je dobré definovať, čo od neho budeme ako používatelia očakávať. Používateľské požiadavky sme tvorili so znalosťou dnes dostupných integrovaných vývojových prostredí, ako sú Eclipse, NetBeans, ale aj samotné Arduino IDE. Naše používateľské požiadavky na systém sme spísali do nasledujúceho use-case zoznamu:

1. Kompilácia a spustenie projektu.
2. Manažment komponentov.
3. Ponuka dostupných komponentov.
4. Voliteľný UX pre programátora.
5. Syntaktická analýza zdrojového kódu.

Kľúčovou úlohou týchto softvérov bolo zjednodušiť programátorom spracovanie ich napísaného zdrojového kódu až po jeho spustenie. Preto jednou z používateľských požiadaviek je automatizovaná kompilácia a spustenie projektu na doske Arduino (1. use case). Konkrétne splnenie tejto požiadavky stojí za pomerne veľkým úspechom Arduino dosiek vo svete.

Všetky ďalšie požiadavky sú priamo spojené s riešením ACProg uvedeným v 5. kapitole. Z XML konfiguračného súboru už poznáme štruktúru komponentov, a preto ako ďalšiu požiadavku uvádzame používateľsky prívetivý editor pre túto štruktúru (2. use case). Editor, v ktorom po výbere komponentu nám budú ponúknuté len tie vlastnosti a udalosti, aké môže komponent nadobúdať podľa definičného súboru jeho typu. Keď hovoríme o typoch komponentov, tak potrebujeme používateľovi ukázať aj ponuku všetkých dostupných komponentov z ACProg repozitára (3. use case).

Neodmysliteľnou súčasťou IDE je editor zdrojového kódu projektu a možnosť usporiadania prostredia podľa vlastných predstáv (4. use case). Dôležitou požiadavkou na editor je, aby bol programátorovi nápomocný. Tým sa myslí, že by mal dokázať analyzovať vytvorenú konfiguráciu komponentov so zdrojovým kódom a interaktívne napomáhať programátorovi (5. use case). Na začiatok pomoc predstavuje automatizované vytvorenie obslužných funkcií k udalostiam na správnom mieste v zdrojovom kóde alebo ponuka existujúcich funkcií v zdrojovom kóde.

Okrem toho sa od editora očakáva aj to, že bude vykonávať i rozsiahlejšiu analýzu projektu, počnúc kontrolou použitých prostriedkov zvolenej dosky Arduino cez kontrolu potenciálne duplicitného využívania vstupno-výstupných pinov až po kontrolu chýbajúcich povinných konfigurácií projektu. Všetky tieto požiadavky by mali programátorovi pomôcť k rýchlejšiemu prototypovaniu nových zariadení, ale aj k vytvoreniu produkčných zariadení.

### Grafický návrh IDE

Po spísaní používateľských požiadaviek sme navrhli prototyp grafického rozhrania. Prototyp bol určený na otestovanie rozloženia obrazovky tak, aby bolo k dispozícii všetko potrebné práve tam, kde by to programátor hľadal. Inšpiráciu sme čerpali z existujúcich používateľských rozhraní pre tvorbu počítačových Windows Forms aplikácií, konkrétne z programov Microsoft Visual Studio a NetBeans.



Obr.  Prototyp integrovaného vývojového prostredia pre ACProg

Na tvorbu prototypu sme vyskúšali rôzne online nástroje (mock-up generátory), avšak ako najlepší nástroj na vytvorenie verného vzhľadu počítačovej formulárovej aplikácie sa osvedčilo Microsoft Visual Studio. Na predchádzajúcom obrázku je nami vytvorený prototyp integrovaného vývojového prostredia.

Základným prvkom prostredia sú karty, pomocou ktorých naplníme používateľskú požiadavku voliteľného UX. Každá karta obsahuje grafický používateľský komponent. Komponenty v grafickom prostredí sú:

* **Component toolbox** – komponent s ponukou dostupných ACProg komponentov, ktorý naplní 3. use case.
* **Designer** – komponent s prehľadom inštancií ACProg komponentov použitých v projekte, ktorý čiastočne naplní 2. use case.
* **Code editor** – komponent určený na editovanie zdrojového kódu. Tento komponent bude vyvolávať syntaktickú analýzu a prípadné chyby v zdrojovom kóde vyznačí na príslušných riadkoch. Tým komponent naplní 5. use case.
* **Properties** – komponent pre nastavovanie inštančných vlastností ACProg komponentu, zvoleného v grafickom komponente Designer. Grafický komponent Properties, v spolupráci s komponentom Designer, naplní 2. use case.
* **Horná lišta programu** – komponent, s akým sa stretávame v každej aplikácii, bude okrem bežných položiek, ako sú pomoc, súbor či nastavenia, obsahovať správu projektu (uloženie, vytvorenie), ale aj spustenie generovania kódu s kompiláciou až po nahratie na dosku Arduino.

## Technologický návrh

Vzhľad a funkčnosť integrovaného vývojového prostredia boli predstavené v predchádzajúcej podkapitole. V tejto podkapitole budeme analyzovať technologické požiadavky prototypu a vyberieme technológie potrebné na implementáciu komponentov z prototypu.

Prvou technológiou na zváženie je programovací jazyk. Keďže chceme, aby bolo výsledné prostredie multiplatformové, tak dobrým riešením je jazyk Java. Ďalším argumentom k vybranému jazyku je aj kompatibilita s existujúcim Arduino IDE. S výberom programovacieho jazyka sme potrebovali aj balíčkovací systém pre automatizované sťahovanie rôznych knižníc a zostavenie projektu. V našom projekte sme použili balíčkovací nástroj Maven.

### Grafický framework Java Swing

Java ponúka dva grafické formulárové frameworky, JavaFX a Swing. JavaFX je pomerne mladý framework oproti Swingu a v budúcnosti by ho potenciálne mal nahradiť [11]. Avšak v súčasnosti je novosť pre framework JavaFX skôr nevýhodou. Dôvodom je zatiaľ nevybudovaná rozsiahlejšia komunita programátorov. S tým je spojená aj slabšia podpora rôznych štandardných formulárových komponentov.

Z toho dôvodu sme sa rozhodli prikloniť sa k frameworku Swing, ktorý má za sebou od roku 1997 veľkú komunitu a rozsiahlu ponuku rozširujúcich modulov. V nasledujúcich sekciách sa pozrieme na ponuku modulov s cieľom výberu vhodných modulov pre rozloženie aplikácie a grafické komponenty z prototypu:

* Code editor,
* Properties,
* Component toolbox a Designer.

### Rozloženie aplikácie – Docking framework

V prototype aplikácie sme navrhli, aby sa jednotlivé záložky mohli striedať medzi sebou, meniť svoje rozmery, dokonca aj skryť, ak ich programátor práve nepotrebuje. Toto správanie však Java Swing vo vlastnej ponuke nemá. Preto sme sa pokúsili implementovať rozloženie pomocou bežne dostupných panelov zo Swingu.

Hlavným problémom, ktorý nás obmedzoval, bolo rozloženie na 4 stĺpce s možnosťou zmeny šírky jednotlivých záložiek. Najdôležitejším dôvodom bola fixná implementácia layoutov vo Swingu na 3 panely (BorderLayout). Pri iných layoutoch vo Swingu boli problémy s chýbajúcou responzívnosťou na základe veľkosti okna spustenej aplikácie. Preto sme túto implementáciu pod čistým Swingom zavrhli a pozreli sa na ponuku dostupných modulov.

Medzi nájdené moduly pre rozloženie aplikácie patria Eclipse Rich Client Platform (RCP), JIDE Docking Framework a DockingFrames.

Eclipse RCP je grafická nadstavba, na ktorej je postavené prostredie Eclipse. Pri širšej analýze tohto modulu sme zistili, že jeho použitie by pridalo aplikácii zbytočné preťaženie nepotrebnou funkcionalitou.

Modul JIDE je lepšie navrhnutý a viac by pasoval k našim podmienkam. No keďže naším cieľom je poskytovať integrované vývojové prostredie ako open source systém, tento platený modul by sme nemohli použiť.

Posledným nájdeným riešením bol DockingFrames. Podobne ako JIDE, aj DockingFrames vyhovoval stanoveným grafickým požiadavkám voliteľného UX. DockingFrames je tiež open source projektom, čiže ho môžeme využiť v našom integrovanom vývojovom prostredí.



Obr.  DockingFrames ilustrácia ukladania záložiek v okne [12]

### Code editor

V integrovanom vývojovom prostredí programátor vytvára zdrojový kód pre spracovanie udalosti. Preto je na mieste potreba editora zdrojového kódu s vyznačovaním syntaxe. Vhodným rozšírením editora by bolo aj automatické dokončovanie s ponukou vhodných existujúcich metód, funkcií a premenných. Preto sme sa pozreli na editory, ktoré sú využívané v iných open source IDE, postavených na jazyku Java. Zistili sme, že používajú open source riešenie RSyntaxTextArea [13] alebo majú vlastné uzatvorené riešenie.

RSyntaxTextArea je nastaviteľný textový editor s vyznačovaním syntaxe pre Java Swing aplikácie. Editor je rozširovateľný pomocou pluginov. Medzi bežne používané pluginy patria:

* code folding – zoskupovanie zdrojového kódu podľa odsadenia,
* search and replace – vyhľadanie a nahradenie textu v kóde,
* code completion – automatické ponúkanie známych metód počas programovania,
* spell checking – kontrola použitých slov podľa jazykového slovníka.

Pomocou vlastného pluginu pre RSyntaxTextArea sme doplnili ďalšie funkcionality, potrebné k naplneniu stanovených používateľských požiadaviek. Hlavnou úlohou pluginu je analyzovať zdrojový kód a získavať z neho jeho štruktúru. Zjednodušene, pod analýzou rozumieme vyňatie inicializovaných premenných, funkcií a tried, aké programátor v projekte vytvoril.

Počas analýzy plugin staticky skontroluje rovnosti dátových typov v zdrojovom kóde. Taktiež skontroluje, či programátor neodstránil povinný import pre fungovanie riešenia ACProg. Všetky zaznamenané nedostatky zdrojového kódu sú vhodnou formou prezentované programátorovi. Nasledujúci obrázok ilustruje zobrazenie nedostatkov zdrojového kódu programátorovi, ktorý omylom odstránil import knižnice.



Obr.  Editor zdrojového kódu s ilustrovaním vyznačenia chyby.

### Ponuka komponentov (modulov) a inštancie komponentov.

V podkapitole 5.3 sme hovorili o typoch komponentov, ktoré generátor uchováva vo svojom repozitári. Tieto komponenty potrebujeme používateľovi vizualizovať, aby ich mohol jednoducho nájsť a použiť vo svojom projekte.

Grafický komponent **Ponuka komponentov** zobrazuje všetky podporované typy komponentov, aké má generátor k dispozícii vo svojom repozitári. Keďže repozitár ukladá moduly do samostatných priečinkov, tak najvernejším grafickým komponentom, v akom je dobré uchovávať ponuku, je stromový zoznam. Požadovaný komponent je dostupný priamo v Java Swing frameworku a preto využijeme túto štandardnú implementáciu.

Grafický komponent **Inštancie komponentov** zobrazuje už vybrané typy ACProg komponentov v projekte, na akom programátor pracuje. V tomto grafickom komponente sa typy môžu opakovať, pretože tu vizualizujeme inštancie typov komponentov. Aby grafický komponent nebol zmätočný a obsahoval iba zoznam komponentov, budeme v ňom rozdeľovať inštancie komponentov na skupiny. Inštancie komponentov budeme môcť medzi skupinami presúvať. Programátor si tak zoskupí inštancie podľa toho, ako spolu súvisia.

Z open source ponuky komponentov pre Java Swing sme nenašli grafický komponent, ktorý by napĺňal naše požiadavky. Preto sme grafický komponent navrhli a implementovali vo vlastnej réžii. Prihliadali sme na design pattern vyskytujúci sa v bežných Java Swing grafických komponentoch, zobrazujúcich formulárové dáta (napr. JTree). Na nasledujúcom obrázku ilustrujeme dva vyššie predstavené grafické komponenty.



Obr.  Ponuka komponentov (vľavo), inštancie komponentov (vpravo).

### Vlastnosti inštancie komponentu (properties)

K inštanciám komponentov je dôležité mať grafický prvok pre nastavenie vlastností inštancie komponentu. Grafický komponent je nazvaný **Vlastnosti inštancie komponentu**. Jeho hlavnou úlohou bude po vybraní inštancie komponentu načítať z definičného súboru vlastnosti aj udalosti a zobraziť ich programátorovi. Udalosť budeme uvažovať ako vlastnosť typu event. Vlastnosti komponentu majú jeden z nasledujúcich dátových typov:

* int – číselný typ,
* string – textový typ,
* pin – číselný typ označujúci hardvérový prostriedok dosky Arduino,
* boolean – dátový typ uchovávajúci 1 bit (true/false),
* event – textový typ ukazujúci na názov funkcie.

Podľa dátového typu zobrazíme programátorovi príslušný grafický formulárový komponent. V prípade int a string je to textové pole na vypísanie hodnoty, ktorá bude validovaná po vložení programátorom. Ak máme dátový typ int, tak používateľ môže do textového poľa vložiť iba číselné znaky. Pri type boolean používateľovi zobrazíme checkbox a v prípade typov pin a event zobrazíme select box obsahujúci zoznam dostupných hodnôt.

## Architektúra IDE

V tejto podkapitole si predstavíme diagram tried integrovaného vývojového prostredia. Diagram je predstavený v poradí, v akom sa používateľ stretáva s jednotlivými súčasťami prostredia. Základná štruktúra tried je nasledovná:

* App – spúšťacia trieda obsahujúca otváranie všetkých okien aplikácie a inicializáciu nastavení platformy operačného systému, na ktorom je integrované vývojové prostredie spustené.
* GUI – package obsahujúci triedy spojené s vykresľovaním používateľských okien a komponentov. Hlavnou častou package-u sú tieto triedy okien:
  + OpenFrame – používateľské okno, v ktorom programátor vytvorí nový projekt, resp. otvorí existujúci projekt z disku.
  + SettingsFrame – používateľské okno slúžiace na základné nastavenia programu. Nastavovať sa môže cesta k Arduino kompilátoru a cesta k umiestneniu ACProg modulov.
  + EditorFrame – hlavné používateľské okno integrovaného vývojového prostredia. Okno implementuje len jeden swing komponent frameworku DockingFrames. Framework sa ďalej postará o zobrazenie a dotiahnutie grafických komponentov záložiek načrtnutých v predchádzajúcej podkapitole.
* Platform – package obsahujúci špecifické triedy pre automatickú konfiguráciu programu na rôznych operačných systémoch.
* Lang – package obsahujúci triedy spracovania a syntaktického vyznačovania zdrojového kódu.
* Utils – package obsahujúci pomocné všeobecné triedy.

EditorFrame integrovaného vývojového prostredia, ktorý priamo obsahuje len jeden komponent frameworku DockingFrames, pri inicializácii vytvorí inštancie pre všetky grafické komponenty záložiek. Záložky sú samostatné grafické komponenty, ktoré od EditorFrame získajú dáta, aké majú zobrazovať programátorovi. Dostupné záložky sú:

* ponuka komponentov (ToolboxIdeComponent),
* inštancie komponentov v projekte (VisualGroupEditorIdeComponent),
* editor zdrojového kódu (EditorIdeComponent),
* editor vlastností (PropertiesIdeComponent),
* konzola s výpisom o aktivite na pozadí integrovaného vývojového prostredia (ConsoleIdeComponent).

Pre editor je dôležitá možnosť komunikácie medzi jednotlivými grafickými komponentmi. Jednou možnosťou komunikácie by bola distribúcia inštancie editora (EditorFrame) do všetkých komponentov obojsmerne. Avšak táto možnosť by zhoršovala čitateľnosť a nútila nás mať všetky referencie vždy k dispozícii.

Preto sme sa rozhodli využiť udalosťami orientované programovanie aj pre túto komunikáciu. Navrhli sme princíp registrovania na udalosti pomocou singleton triedy EventManager. K nej je umožnený prístup z celého projektu. V prípade, že niektorý grafický komponent bude chcieť komunikovať s iným (prípadne viacerými naraz), tak odošle spustenie udalosti s potrebnými dátami do singleton triedy EventManager. EventManager sa následne postará o spustenie všetkých zaregistrovaných udalostí. Spúšťanie udalosti je vo vlastnom vlákne na pozadí, aby v prípade dlhotrvajúceho spracovania (napríklad kompilácie) neblokoval používanie integrovaného vývojového prostredia.

## Implementácia IDE

V tejto podkapitole sa budeme venovať implementačným problémom, aké sme riešili pri tvorbe integrovaného vývojového prostredia. Problémy vysvetlíme v spojení s príslušnými use case, ktoré sme navrhli v podkapitole 6.1.

### Ponuka dostupných komponentov

Úlohou grafického komponentu ponuky dostupných komponentov je načítať všetky dostupné komponenty a vhodnou formou ich zobraziť používateľovi. Zobrazenie bolo navrhnuté formou stromu, pre ktorý využijeme vstavaný komponent frameworku Swing JTree.

Pre zobrazenie komponentov potrebujeme vložiť do JTree koreňový element ponuky komponentov. Dostupné komponenty načítame z priečinku zadaného v konfigurácii projektu. Načítané komponenty následne vložíme do stromovej štruktúry začínajúcej v koreňovom elemente nazvanom Nástroje. Na nasledujúcom obrázku je zobrazený výsledný vzhľad ponuky komponentov. Zdrojový kód ilustruje vytvorenie modelu.



Obr.  Vizuál grafického komponentu JTree so zdrojovým kódom.

### Skupinové zobrazenie inštancií komponentov

V sekcii 6.2.4 sme načrtli grafický komponent z pohľadu používateľa. V tejto sekcii rozoberieme implementáciu tohto grafického komponentu. Pri implementácii sme sa inšpirovali návrhom podobných komponentov vo frameworku Swing, a preto samotný obsah grafického komponentu je ovládaný modelom.

Model je údajová štruktúra uchovávajúca konfiguráciu komponentov a ich príslušnosti k skupinám. Zmeny v údajoch modelu sú automaticky prevedené aj do grafického komponentu. Model je vytvorený z údajovej štruktúry mapa. Typom kľúča je názov skupiny komponentov a hodnoty v mape tvorí pole inštancií komponentov. Grafickú implementáciu skupinového rozdelenia sme rozdelili na menšie stavebné komponenty. Ich rozdelenie ilustrujeme na nasledujúcej schéme.



Obr.  Rozdelenie grafickej implementácie.

GroupSection je trieda, ktorej úlohou je vykresliť skupinu spolu s komponentmi. Vykresľovanie rozdeľujeme na dve časti. Prvou je vykreslenie hlavičky skupiny, o ktoré sa postará trieda GroupTitle. Druhou časťou je vykreslenie komponentov, ktoré patria do skupiny. Pre toto vykreslenie sme použili grafický komponent JList z frameworku Swing.

Dôležitou vlastnosťou pre grafický komponent je aj jednoduché presúvanie inštancií komponentov medzi skupinami. Na naplnenie tejto požiadavky potrebujeme spracovanie udalosti kliknutia na konkrétnu inštanciu komponentu a taktiež prostriedok z Java API podporujúci drag’n’drop.

Pri presúvaní grafických komponentov v Java API je potrebné zdieľať konkrétny presúvaný komponent s ostatnými komponentmi, na ktoré ho budeme chcieť pustiť. V komponentoch frameworku Java Swing stačí povoliť vlastnosť drag. Ďalším krokom je spracovanie pustenia komponentu. Na to potrebujeme na cieľovom grafickom komponente zaregistrovať udalosť transferHandler. Tá v prípade pustenia komponentu dostane presúvaný komponent na vstup a spracuje presun medzi skupinami inštancií komponentov.

### Kompilácia a spustenie projektu

Pre integrované vývojové prostredie je dôležité zjednodušiť prácu programátorovi. Preto potrebujeme, aby bola kompilácia jednoducho prístupná. Keďže ACProg projekt je modulárne rozdelený na generátor a integrované vývojové prostredie, tak k spracovaniu XML súboru máme k dispozícii aplikačné rozhranie generátora. Pomocou tohto rozhrania spustíme generovanie zdrojového kódu.

Kompilácia je už komplikovanejšia, pretože pre ňu nemáme k dispozícii Java aplikačné rozhranie. Kompiláciu môžeme urobiť konzolovou verziou programu Arduino IDE. Java API podporuje spúšťanie externých skriptov, slúži na to trieda Runtime z package-u java.lang. Pomocou tejto triedy spustíme program Arduino kompilátora s potrebnými parametrami:

* zdrojový kód,
* vygenerovaná knižnica,
* sériový port pripojenej dosky (len pre nahratie na dosku).

Nasledujúci obrázok zobrazuje externé skripty s parametrami potrebnými pre spustenie kompilácie a nahratie na dosku Arduino.

*// Spustenie kompilácie*  
arduino.exe—verify—board arduino:avr:uno—pref build.path=*{cesta k vygenerovanej kniznici}* Source.ino  
  
*// Nahratie na dosku Arduino*  
arduino.exe—upload—board arduino:avr:uno—port COM3 --pref build.path=*{cesta k vygenerovanej kniznici}* Source.ino

Obr.  Priebeh spracovania projektu riešením ACProg

Pre výstup z kompilácie a generovania kódu sme vytvorili grafický komponent neinteraktívnej konzoly. Komponent má vytvorený interface podobný konzolovému výpisu z Java API.

### Syntaktická analýza kódu

Neodmysliteľnou súčasťou integrovaných vývojových prostredí je analýza zdrojového kódu ešte pred tým, ako bude kompilovaný. Pre programátora to znamená rýchlu spätnú väzbu, či je jeho zdrojový kód správny. Analýzu kódu pre riešenie ACProg robíme v dvoch krokoch:

1. analýza inštancií komponentov,
2. analýza zdrojového kódu s prehľadaním vytvorených funkcií a premenných.

V prvom kroku analyzujeme nakonfigurované inštancie ACProg komponentov. Základnou analýzou je overenie správnosti dátových typov. Základné dátové typy, ako int a string, je jednoduché overiť.

Avšak pri dátových typoch ako pin potrebujeme overiť, či je zadaný pin dostupný na zvolenej doske Arduino, aj s požadovanými vlastnosťami. Okrem kontroly dátových typov potrebujeme overiť, či sa rôzne inštancie komponentov nepokúšajú pristupovať k rovnakým hardvérovým prostriedkom. Na nasledujúcom obrázku môžeme vidieť chybnú konfiguráciu spolu s upozorneniami, ktoré dáme programátorovi po analýze inštancií komponentov.



Obr.  Ilustrácia chybnej konfigurácie inštancií komponentov.

V druhom kroku analyzujeme zdrojový kód. Na analýzu zdrojového kódu nestačí jednoduché vyhľadávanie vzoriek v texte. Napísaný text potrebujeme komplexne analyzovať ako syntax C++ programu.

Na takú analýzu je vhodné vytvoriť abstraktný syntaktický strom (AST). AST je stromová reprezentácia abstraktnej syntaktickej štruktúry zdrojového kódu. Uzly v strome zobrazujú konštrukt, aký programátor použil. Hovoríme mu abstraktný, pretože v strome nepotrebujeme všetky detailné informácie. Pre vytvorenie AST potrebujeme lexikálny analyzátor (angl. lexer), pomocou ktorého skenujeme zdrojový kód a označujeme v ňom symboly, ak splnia podmienky napísané regulárnymi výrazmi.

Druhou zložkou vytvorenia AST je parser, ktorý pomocou gramatiky zdrojového kódu v symboloch odhalí všetky konštrukty vyskytujúce sa v napísanom zdrojovom kóde. Nasledujúci obrázok ilustruje AST vytvorený z jednoduchého zdrojového kódu.



Obr.  Ilustrácia AST (vpravo) zo zdrojového kódu (vľavo).

Vytvorili sme lexer, ktorý pre znaky nájdené v zdrojovom kóde vytvorí pomenovaný symbol triedy sym. Podľa dostupnej špecifikácie syntaxe pre C++ sme vytvorili gramatiku, ktorou analyzujeme prečítané symboly a prejdeme celý súbor zdrojového kódu. Parser pri prechode zdrojovým kódom zaznamenáva nájdené konštrukty a uchováva ich vo svojej reprezentácii AST.

Pri používaní už existujúcich funkcií a tried parser skontroluje typové priradenia parametrov a v prípade nezrovnalostí upozorní programátora. Po prechode zdrojovým kódom máme k dispozícii zoznam všetkých premenných, funkcií, tried a importov použitých v programe.

Pomocou takto získaných informácií overíme priradenie udalostí v inštanciách komponentov. Kontrolujeme, či priradené funkcie existujú v zdrojovom kóde. Okrem toho overíme existenciu API volaní na inštanciách komponentov a existenciu potrebného importu generovanej knižnice.

### Voliteľný UX pre programátora

Na grafickú vizualizáciu integrovaného vývojového prostredia sme využili knižnicu DockingFrames. Táto knižnica pri inicializácii potrebuje grafické komponenty (záložky) a ich rozloženie pri spustení. Knižnica počas spustenia podporuje úpravu rozloženia záložiek programátorom.

Aby sme dosiahli cieľ voliteľného UX pre programátora, potrebujeme zabezpečiť, aby pri opakovanom spustení mal programátor rovnaké rozloženie, ako naposledy nastavil. To zabezpečíme uložením rozloženia a načítaním pri opätovnom spustení.

Uloženie rozloženia bude spracované pri zatváraní integrovaného vývojového prostredia. Rozloženie uložené v XML súbore obsahuje informácie o záložkách a ich presnom umiestnení, ktoré sme získali pomocou aplikačného rozhrania knižnice DockingFrames, konkrétne inštanciou triedy CControl.

Pre integrované vývojové prostredie sme pripravili dve rozloženia záložiek, ktoré bude mať používateľ k dispozícii cez menu. Používateľ sa tak môže kedykoľvek vrátiť k prednastaveným rozloženiam integrovaného vývojového prostredia. Na nasledujúcom obrázku sú načrtnuté nami navrhnuté rozloženia.



Obr.  Navrhnuté rozloženia integrovaného vývojového prostredia.

# Vzorové komponenty pre ACProg

V 5. kapitole sme vysvetlili fungovanie ACProg riešenia a priblížili sme aj typy komponentov. Táto kapitola sa bude venovať vytvoreniu niekoľkých vzorových typov komponentov, pomocou ktorých je možne vytvoriť rozličné IoT projekty.

Prvý typ komponentu sa zaoberá vykonávaním udalosti v zadanom intervale, tri ďalšie typy sa zaoberajú vstupno-výstupnými pinmi dosky Arduino s cieľom čítania informácií, resp. ich zapisovania. Nasledovať budú dva typy komponentov určené pre komunikáciu s ostatnými IoT zariadeniami. Na konci kapitoly si ukážeme, ako efektívne debugovať ACProg aplikácie za pomoci špeciálneho komponentu.

## Časovač

## Digitálny výstup

## Digitálny vstup

## Analógový vstup s threshold upozornením

## Komunikácia pomocou MQTT

## Rádiová komunikácia pomocou 433 MHz

## Komponent pre zjednodušenie debugovania

Záver

V závere je potrebné v stručnosti zhrnúť dosiahnuté výsledky vo vzťahu k stanoveným cieľom.

Zoznam použitej literatúry

|  |  |
| --- | --- |
| [1] | Giusto, D.; Iera, A.; Morabito, G.; Atzori, L.;, The Internet of Things, 20th Tyrrhenian Workshop on Digital Communications, Springer, New York, NY, 2010. |
| [2] | Radovici, A.; Culic, I.; Vaduva, A., „Lecture 1: Introduction to IoT,“ [Online]. Available: https://ocw.cs.pub.ro/courses/iot/courses/01. |
| [3] | Adafruit, „Arduino Uno R3 (Atmega328 - assembled),“ [Online]. Available: https://www.adafruit.com/product/50. |
| [4] | Adafruit, „Ethernet Shield for Arduino - W5500 Chipset,“ [Online]. Available: https://www.adafruit.com/product/2971. |
| [5] | I. Mikolic-Torreira, „Arduino EventManager,“ 2016. [Online]. Available: https://github.com/igormiktor/arduino-EventManager/blob/master/README.md. |
| [6] | „About QM,“ 2018. [Online]. Available: http://www.state-machine.com/qm/. |
| [7] | ReTis Lab - Institute of Communication, „Arduino Real-Time extension,“ 2018. [Online]. Available: http://retis.sssup.it/?q=arte. |
| [8] | „Dokumentácia Cayenne,“ 2017. [Online]. Available: https://mydevices.com/cayenne/docs/intro/. |
| [9] | Sharan, K., Beginning Java 8 APIs, Extensions and Libraries, Berkeley, CA: Apress, 2014. |
| [10] | Tilmanis, P.; White, G., Further Programming in Java, Chapter 7: Event-Driven Programming, Melbourne: RMIT University, 2006. |
| [11] | Oracle, „JavaFX Frequently Asked Questions,“ [Online]. Available: http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html#6. |
| [12] | Sigg, B., „DockingFrames 1.1.1 - Common,“ 2012. [Online]. Available: http://www.docking-frames.org/dockingFrames\_v1.1.1/common.pdf. |
| [13] | bobbylight, „RSyntaxTextArea,“ 2017. [Online]. Available: https://github.com/bobbylight/RSyntaxTextArea. |

Prílohy

1. CD médium – diplomová práca v elektronickej podobe, prílohy v elektronickej podobe.
2. Používateľská príručka
3. Systémová príručka

Táto časť diplomovej práce je povinná a obsahuje zoznam všetkých príloh vrátané elektronických nosičov. Názvy príloh v zozname musia byt’ zhodné s názvami uvedenými na príslušných prílohách. Tlačené prílohy majú na prvej strane identifikačné údaje – informácie zhodné s titulnou stranou diplomovej práce doplnené o názov príslušnej prílohy (Systémová príručka, Používateľská príručka). Identifikačné údaje sú aj na priložených diskoch alebo disketách. Ak je médií viac, sú označené aj číselne v tvare I/N, kde I je poradové číslo a N je celkový počet daných médií.

Každá príloha začína na novej strane a je označená samostatným písmenom alebo číslom (Príloha A, Príloha B, ... alebo Príloha 1, Príloha 2, ...). Číslovanie strán príloh nadväzuje na číslovanie strán v hlavnom texte.

1. Funkcia delay(int microseconds) zastaví vykonávanie programu na zadaný počet mikrosekúnd. Tradičný kompilátor Arduino implementuje zastavenie aktívnym čakaním. [↑](#footnote-ref-1)