UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

KOMPONENTOVÉ A UDALOSŤAMI RIADENÉ PROGRAMOVANIE ZARIADENÍ NA PLATFORME ARDUINO

2018 **Bc. Patrik PEKARČÍK**

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

KOMPONENTOVÉ A UDALOSŤAMI RIADENÉ PROGRAMOVANIE ZARIADENÍ NA PLATFORME ARDUINO

DIPLOMOVÁ PRÁCA

Študijný program:	Informatika
Pracovisko (katedra/ústav):	Ústav Informatiky
Vedúci diplomovej práce:	RNDr. František Galčík, PhD.

Košice 2018

Bc. Patrik PEKARČÍK

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

Rád by som sa poďakoval vedúcemu diplomovej práce RNDr. Františkovi Galčíkovi, PhD. za námet, cenné rady a jeho ochotu kedykoľvek pomôcť.

Abstrakt v štátnom jazyku

Kľúčovým prvkom internetu vecí (IoT) je prepojenie fyzického a digitálneho sveta prostredníctvom senzorov a aktuátorov. Tieto senzory a aktuátory sú riadené mikrokontrolérmi. Najznámejšie mikrokontroléry určené primárne na prototypovanie sú mikrokontroléry Arduino.

V práci sa venujeme programovaniu mikrokontrolérov Arduino inovatívnym prístupom, ktorý je založený na myšlienke komponentového a udalosťami riadeného programovania. Teda na prístupe, ktorý poznáme zo sveta programovania desktopových a mobilných aplikácií. Priame aplikovanie tohto prístupu vo svete mikrokontrolérov nie je možné vzhľadom na ich obmedzené výpočtové zdroje (napr. RAM veľkosti 2048 bajtov).

Pre efektívne prototypovanie uvedeným prístupom je nevyhnutná softvérová podpora vo forme integrovaného vývojového prostredia (IDE). Hlavným cieľom práce je vytvorenie IDE, ktoré umožní intuitívne vytváranie projektov z komponentov, ich konfiguráciu a prepojenie so zdrojovým kódom používateľa na báze jeho syntaktickej analýzy. IDE umožňuje verifikáciu projektu či automatické naprogramovanie mikrokontroléra.

IDE založené na tomto inovatívnom prístupe má predpoklad zrýchliť a zjednodušiť prototypovanie aplikácií v oblasti IoT tak pre vývojárov, ako aj neprofesionálnych záujemcov o oblasť IoT.

Kľúčové slová: Arduino, Komponentové programovanie, Udalosťami orientované programovanie, Integrované vývojové prostredie, Abstraktná syntaktická analýza.

Abstrakt v cudzom jazyku

The key element of the Internet of Things is to connect physical and digital world through sensors and actuators. Sensors and actuators are controlled by microcontrollers. The most well-known microcontrollers, primarily designed for prototyping, are the Arduino microcontrollers.

We are working on the Arduino programming with an innovative approach based on component and event-driven programming. We know this approach from the world of programming desktop and mobile applications. Direct implementation of this approach on microcontroller is not possible due to its limited computing resources (eg. only 2048 bytes of RAM).

Effective prototyping with this approach requires software support in the form of the integrated development environment (IDE). The main goal of this thesis is to create an IDE that will allow intuitive component creation, configuration and linking to user's source code based on its syntactic analysis. The main features include project verification and automatic upload to microcontroller.

An IDE based on this innovative approach is supposed to accelerate and simplify prototyping of IoT devices for professional and non-professional developers.

Keywords: Arduino, Component-oriented programming, Event-driven programming, Integrated Development Environment, Abstract Syntax Analysis.

Obsah

Zoznam ilustrácií

Zoznam tabuliek

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
GUI	**G**raphical **U**ser **I**nterface
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
XML	E**x**tended **M**arkup **L**anguage

Úvod

Internet vecí (angl. Internet of Things, IoT) je oblasť, o ktorej dnes počujeme zo všetkých médií. Vývoju IoT zariadení sa venujú známe spoločnosti, ako Philips so svojím inteligentným osvetlením, Apple s ich technológiou HomeKit pre prepojenie inteligentných zariadení v domácnosti, spoločnosť Nest (patriaca Googlu) s ich inteligetným termostatom a mnoho ďalších. Internet vecí však nie je len o zariadeniach pre chytrú domácnosť, s produktmi internetu vecí sa stretávame v dopravnom priemysle (elektronické značky, radary), v modernom zdravotníctve a na mnohých ďalších miestach.

Najvýraznejšej popularizácii internetu vecí pomohol príchod zariadení **Arduino**. Arduino je open-source platforma s mikrokontrolérom ATmega, ktorá vývojárom neponúkla iba hardvér s mikrokontrolérom, ale aj pomerne jednoduché vývojové prostredie Arduino IDE a odľahčený jazyk c++ s pomerne jednoduchou podpornou knižnicou. K rozšíreniu týchto zariadení prispela aj ich nízka cena, ktorá sa pohybuje od niekoľkých dolárov. Cena zariadenia Arduino Nano sa pohybuje okolo 2 dolárov, za čo dostaneme úložný priestor FLASH 32 kB a operačnú pamäť SRAM 2048 B.

Vývoj na platforme Arduino je založený na odľahčenom programovacom jazyku C++. Pre program spustiteľný na platforme Arduino je nutné implementovať dve základné funkcie:

- `setup()` – funkcia spustená iba raz, pri inicializácii zariadenia,
- `loop()` – periodicky spúšťaná funkcia, pokiaľ je zariadenie zapnuté.

Tento prístup nepodporuje efektívny multitasking, na aký sú programátori zvyknutí z programovania pre operačný systém. Obmedzená veľkosť operačnej pamäte nám zatvára dvere pred použitím komplexnejšieho operačného systému, čo však nevylučuje vytvorenie plánovača úloh pre tieto zariadenia. Navrhli sme inovatívny spôsob programovania založený na myšlienke komponentového a udalosťami orientovaného programovania, ktorý poznáme zo sveta programovania desktopových a mobilných aplikácií. Tento spôsob sme implementovali v riešení nazvanom **ACProg**. Riešenie ACProg sme sa rozhodli doplniť integrovaným vývojovým prostredím. Programátor v prostredí bude môcť s projektom pracovať od vytvorenia až po kompiláciu a spustenie na Arduino zariadení. Integrované vývojové prostredie sme doplnili o verifikáciu

projektu pomocou abstraktnej syntaktickej analýzy a pravidiel aké musia komponenty aj zdrojový kód spĺňať.

Prácu sme rozdelili do siedmich kapitol. V prvej kapitole sa venujeme základnému pohľadu na internet vecí, ukážeme si schému IoT a priblížime o akom softvéri a hardvéri sa v IoT rozpráva. V nasledujúcej kapitole budeme rozprávať o projekte Arduino. Pozrieme sa na možnosti programovania Arduino dosiek. Analyzovali sme aj niekoľko dostupných riešení pre dosky Arduino, ktoré si v závere tejto kapitoly rozdelíme a štyri dostupné riešenia detailne opíšeme v samostatných sekciách.

V tretej kapitole sa venujeme komponentovému a udalosťami orientovanému programovaniu. Vysvetlíme čo rozumieme pod slovom komponent aj udalosť a uvedieme princíp fungovania programov implementujúcich tento návrhový vzor. Vo štvrtej kapitole vysvetlíme pojem integrovaného vývojového prostredia a poukážeme na jeho dôležitosť pri vývoji softvéru.

Piata kapitola obsahuje technické detaily nášho riešenia ACProg. V kapitole vysvetlíme princíp generovania zdrojového kódu a vysvetlíme ako vytvoriť projekt za použitia riešenia ACProg. V závere kapitoly vysvetlíme typy komponentov a ich uloženie v repozitári dostupných komponentov.

Šiesta kapitola sa venuje vytvoreniu integrovaného vývojového prostredia. Začneme navrhnutím používateľských požiadaviek pre prostredie. Pokračovať budeme technologickou analýzou pre naplnenie požiadaviek a prípravu implementácie. Kapitola pokračuje implementáciou rozdelenou po jednotlivých grafických komponentoch integrovaného vývojového prostredia. Zvlášť sekcia je venovaná verifikácii projektu pomocou abstraktnej syntaktickej analýzy.

V poslednej kapitole vysvetlíme ako vytvárať nové typy komponentov do projektu ACProg. Ukážeme si na základnom komponente digitálneho vstupu všetky oblasti aké môže typ komponentu obsahovať. Na základe vysvetlenia vytvoríme nový komunikačný typ pre rádiovú komunikáciu 433MHz.

1 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 ustálená 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. 1 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 (angl. Local Processing) spočíva v prevode signálu na digitálne dáta a vykonaní analýzy zozbieraných dát. Lokálne spracovanie môže rozhodovať o vykonaní akcie a ihneď ovplyvňovať fyzický svet.

Po lokálnom spracovaní sa dáta môžu uložiť 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 (angl. network,

internet). Ostatnými zariadeniami rozumieme rovnocenné IoT mikrokontroléry alebo IoT server. IoT server je zariadenie na počítačovej sieti, ktoré prijíma dáta zo všetkých senzorov po ich lokálnom spracovaní. Spracovaniu dát na serveri hovoríme cloud spracovanie (angl. Cloud Processing). IoT server po spracovaní dát môže vyvolať vykonanie akcie aktuátormi na mikrokontroléroch v sieti a ovplyvňovať tým fyzický svet.

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.

V mnohých IoT projektoch sa však stretávame s tým, že lokálne spracovanie je úplne vynechané a všetky nazbierané dáta zo senzorov sa priamo odosielajú na cloud spracovanie.

1.1 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 typické 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 namerané elektrické napätie prepočítali na digitálnu informáciu v ohmoch.

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

1.2 Softvér

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

Výrobcovia mikrokontrolérov dodávajú k mikrokontrolérom aj kompilátor zdrojového kódu. Úlohou kompilátora je previesť program napísaný v upravenej verzii jazyka C do assembleru pre konkrétny model mikrokontroléra. Okrem kompilátorov výrobcovia dodávajú aj integrované vývojové prostredie. Ich hlavnou úlohou je odbremeniť programátorov od opakovaných krokov a poskytnúť jednoduchú konfiguráciu bez väčšej potreby znalosti celého systému kompilátora. Integrované vývojové prostredia výrazne prispievajú k rýchlejšiemu prototypovaniu nových nápadov.

1.3 Mikrokontrolér v porovnaní so single-board počítačom

Hlavnou funkciou mikrokontroléra je komunikácia so senzormi, aktuátormia 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 týchto počítačov sa pohybuje okolo 1 GB RAM s ARM 1,2 Ghz procesorom. Tieto parametre nám už dovoľujú spustiť na procesore operačný systém.

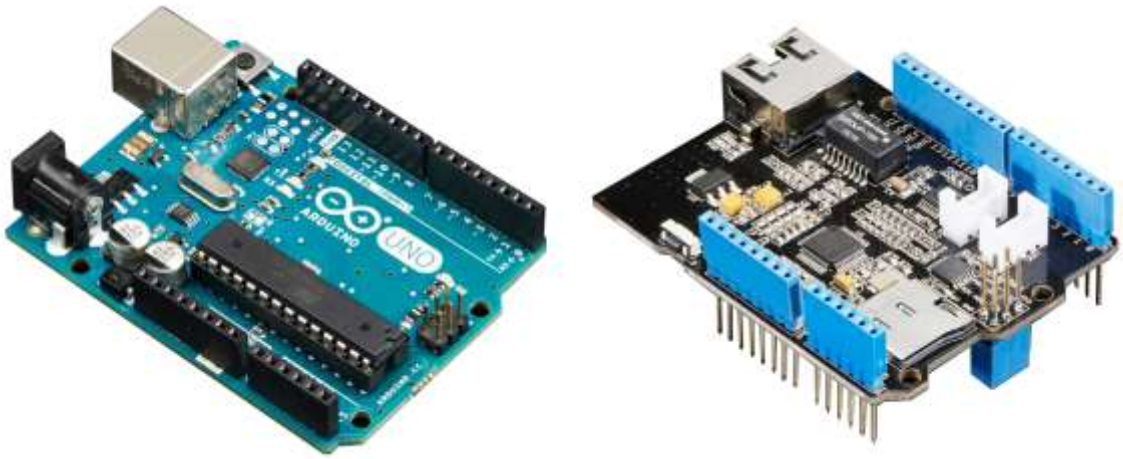
Avšak nevýhodou operačných systémov je, že prenechávame časovanie a plánovanie na nich. Preto je priama stabilná komunikácia so senzormi a aktuátormi takmer nemožná. Riešením problému časovania by mohli byť digitálne senzory, ktoré obsahujú vlastný jednoúčelový mikrokontrolér s komunikačným protokolom pre komunikáciu so single-board počítačom. Single-board počítače často poskytujú mikrokontroléry implementujúce tieto komunikačné protokoly (napr. I2C). Vďaka nim vieme pomocou single-board počítačov komunikovať s digitálnymi senzormi.

2 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 voľnú výrobu dosiek Arduino a distribúciu softvéru.

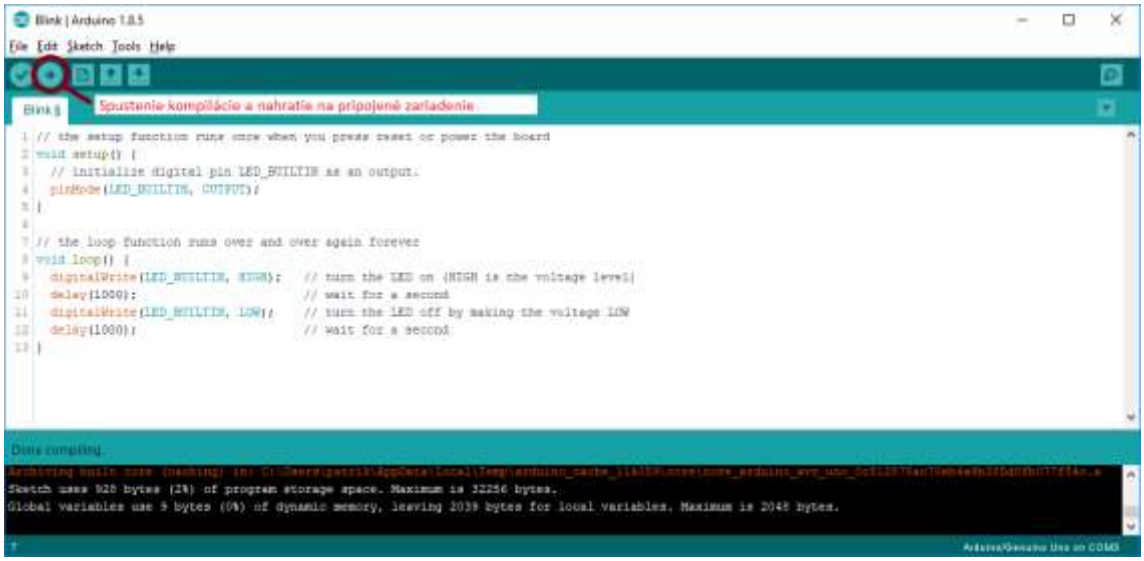
Dosky Arduino sú špecifické vyvedenými pinmi na ich okraji. Vyvedenie pinov je kľúčové pre jednoduché prototypovanie embedovaných zariadení. Vďaka štandardizovanému tvaru a vyvedeniu pinov dosky Arduino je možné vytvárať rozširujúce dosky, tzv. Arduino Shield. Arduino Shield poskytuje rovnaký tvar a rozloženie pinov, ako má Arduino doska. Preto môžeme skladať rôzne Arduino Shield na seba. Arduino Shield dodáva rôzne nové 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. 2 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. 3 Arduino IDE s vyznačením kompilácie a spustenie.

2.1 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. 1 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	512 000	SD slot	1200	0/40

2.2 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.

```
#include "Blink.h"

// Setup funkcia spustená raz pri zapnutí zariadenia.
void setup() {
    // Inicializácia vsupno výstupných pinov
}

// Loop funkcia je opakovaná do nekonečna
void loop() {
  // Programovanie správania sa arduino zariadenia v čase
}
```

```
Príklad z C++
int main() {
    setup();
    while(true) {
        loop();
    }
}
```

Obr. 4 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.

2.3 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.

2.3.1 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. 5 Zaregistrovanie udalosti na analógovom pine 2 v `setup()` a spracovanie udalostí v `loop()`.

2.3.2 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.

2.3.3 ARTe (Arduino Real-Time extension)

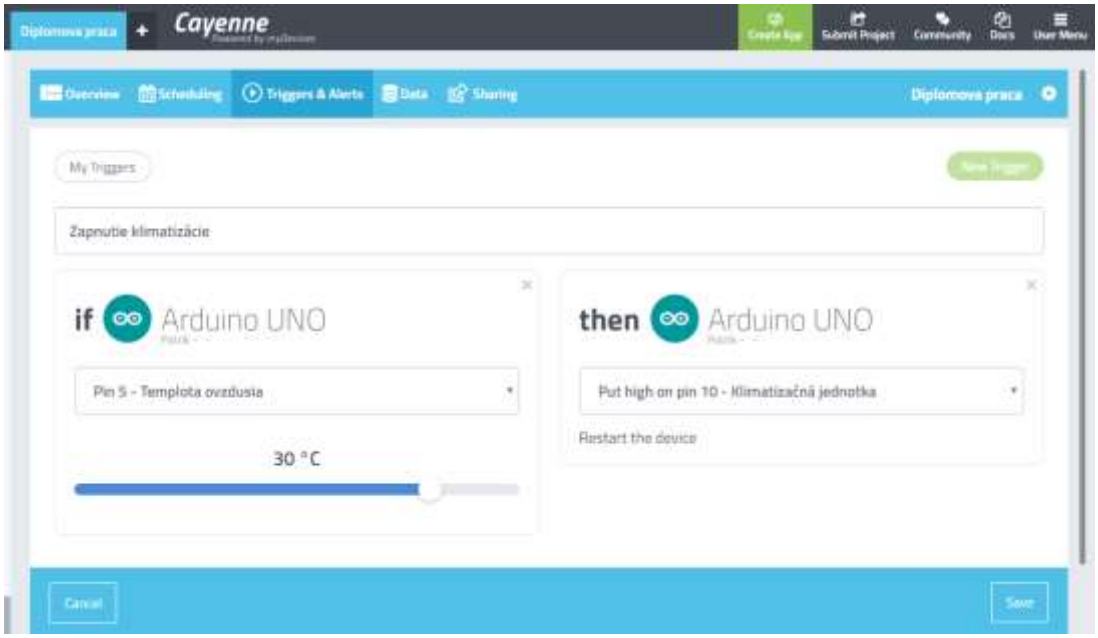
Rozšírenie Arduino kompilátora ARTe [7] zaradzujeme do skupiny offline riešení. Rozšírenie umožňuje vytvorenie viacerých `loop` funkcií, ktoré sú spustené pseudoparalelne. Vývojári ARTe dosiahli pseudoparalelizmus prepísaním implementácie čakacích (`delay`[1]) metód z aktívneho čakania na pasívne. Program 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.

2.3.4 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

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

(teplomer) s aktuátorom (klimatizácia), ktorý realizuje nasledujúcu akciu: ak stúpne teplota nad 30 °C, tak zapni klimatizáciu.



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

3 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.

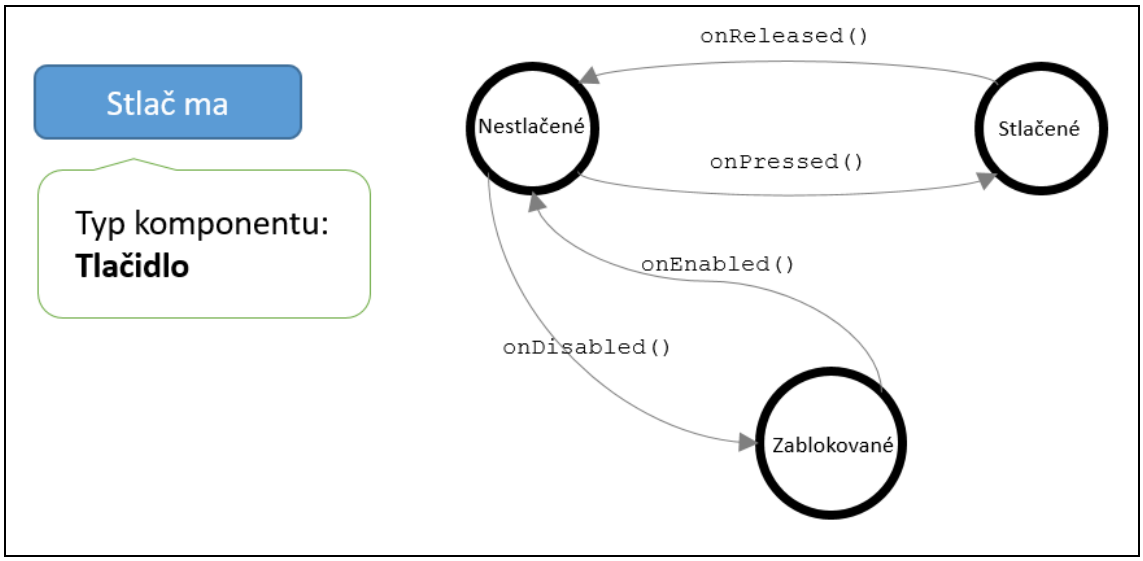
V objektovo orientovanom programovaní je **komponent** považovaný za znovu použiteľný blok programu, ktorý môže byť kombinovaný s inými komponentmi za účelom vytvorenia aplikácie. Príkladmi komponentov sú tlačidlo, scrollbar, ale aj tabuľka v grafických používateľských prostrediach. Zdrojový kód komponentu, 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.

V udalosťami orientovanom programovaní sú s udalosťou spojené tri typy objektov [9]:

- **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.

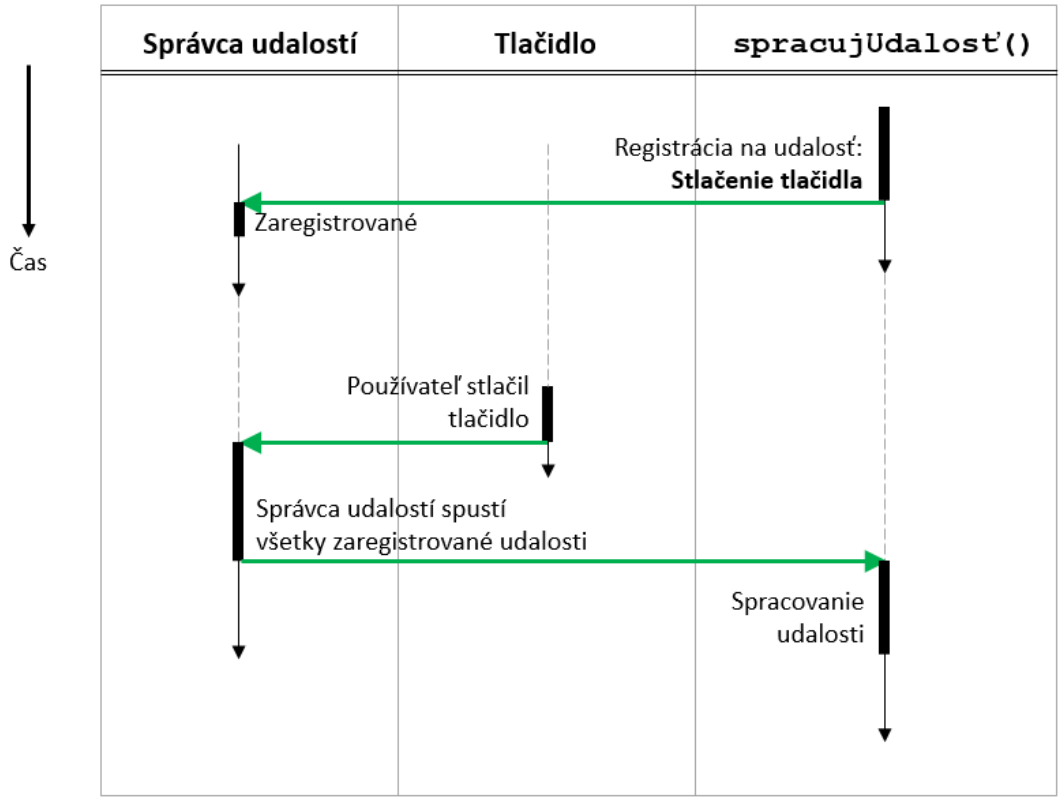


Obr. 7 Znázornenie inštancie komponentu typu tlačidlo s automatom pre prechody medzi jednotlivými stavmi.

3.1 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.



Obr. 8 Práca plánovača úloh s registráciou a vyvolaním udalosti

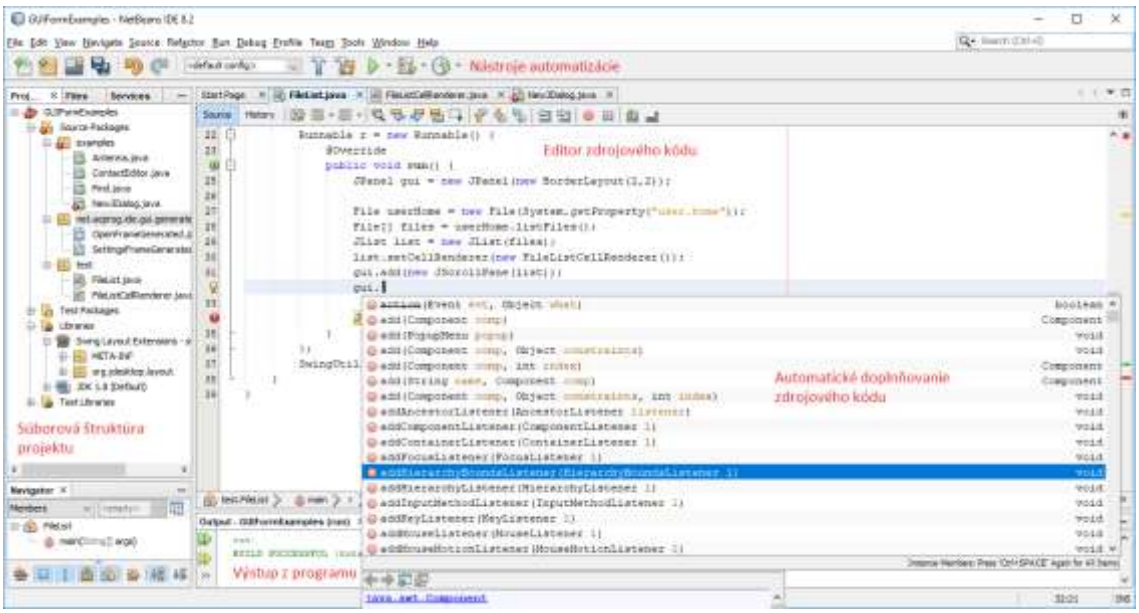
Na začiatku objekt s funkciou `spracujUdalosť()` vykoná registráciu na udalosť s názvom `Stlačenie tlačidla` (komponentu `Tlačidlo`). Registráciu vykoná odoslaním funkcie na správcu úloh. Ten si do svojej internej pamäte zapíše túto registráciu. Následne, keď používateľ stlačí komponent `Tlačidlo`, tak komponent zistí zmenu stavu a vytvorí udalosť. Udalosť je odoslaná na správcu úloh, ktorý skontroluje zoznam registrácií na prijatú udalosť a vyvolá príslušné zaregistrované funkcie. V ukázanom príklade sa spustí funkcia `spracujUdalosť()`.

4 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.

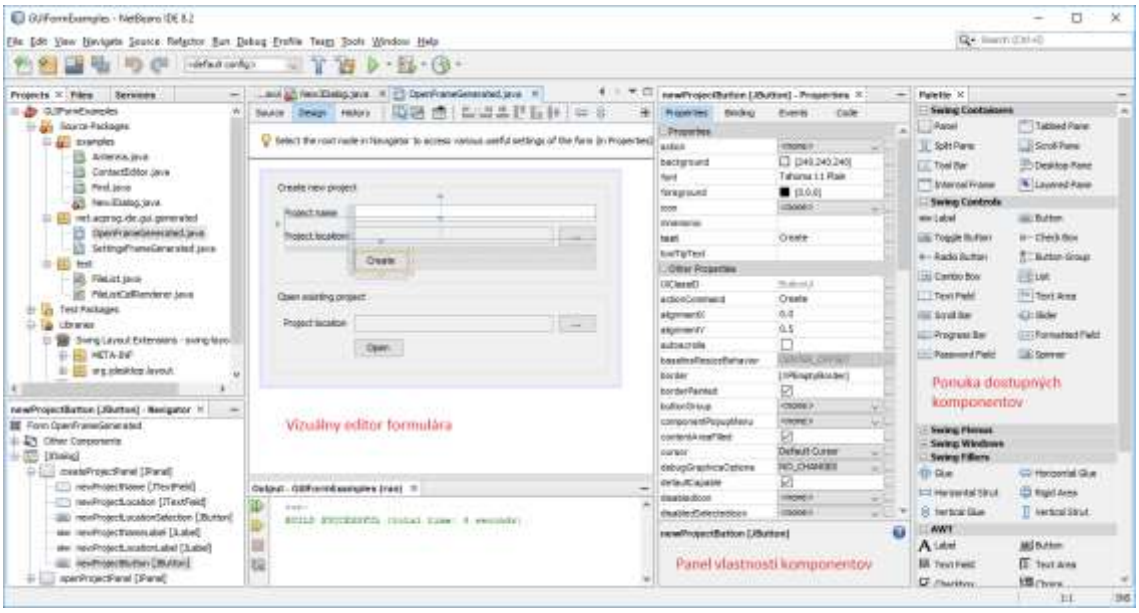


Obr. 9 Integrované vývojové prostredie NetBeans pre jazyk Java s vizualizáciou automatického dokončovania

4.1 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.

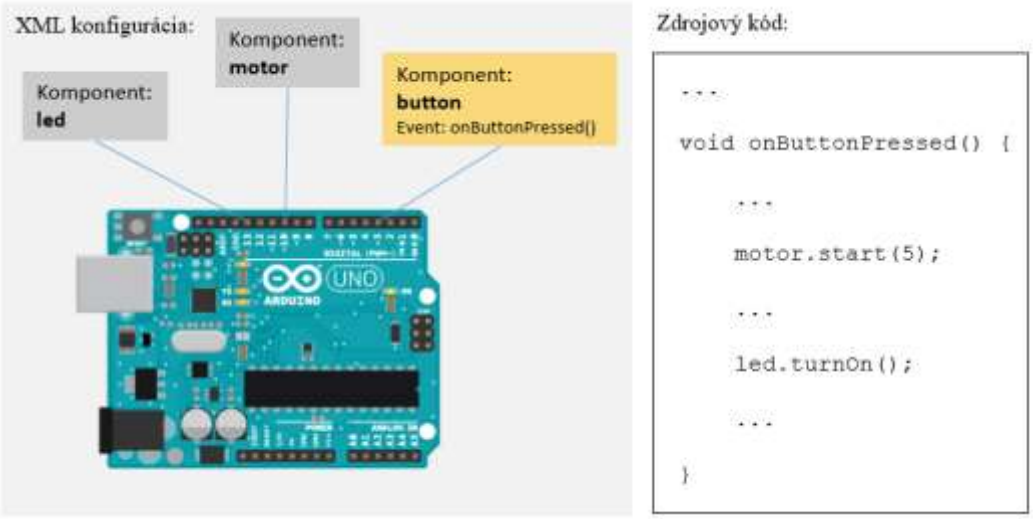
Príkladom integrovaných vývojových prostredí postavených na metodológii RAD sú vizuálne editory formulárov (GUI). Na nasledujúcom obrázku je zobrazený vizuálny editor formulárov z vývojového prostredia programu NetBeans. Hlavným prvkom je Vizuálny editor formulára, v ktorom pomocou drag'n'drop vieme rozmiestniť komponenty ako potrebujeme. Po kliknutí na komponent je zobrazený grafický komponent vlastností komponentov, kde vieme nastaviť všetky inštančné premenné komponentu a priradiť metódy pre spracovanie udalostí. IDE poskytuje aj ponuku všetkých dostupných komponentov, ktoré môžeme vložiť do vytváraného formulára. Použitím tohto vizuálneho editora programátor prenecháva generovanie zdrojového kódu formulára na IDE a môže sa plne sústrediť na správne rozloženie všetkých komponentov.



Obr. 10 Vizuálny editor integrovaného vývojového prostredia NetBeans

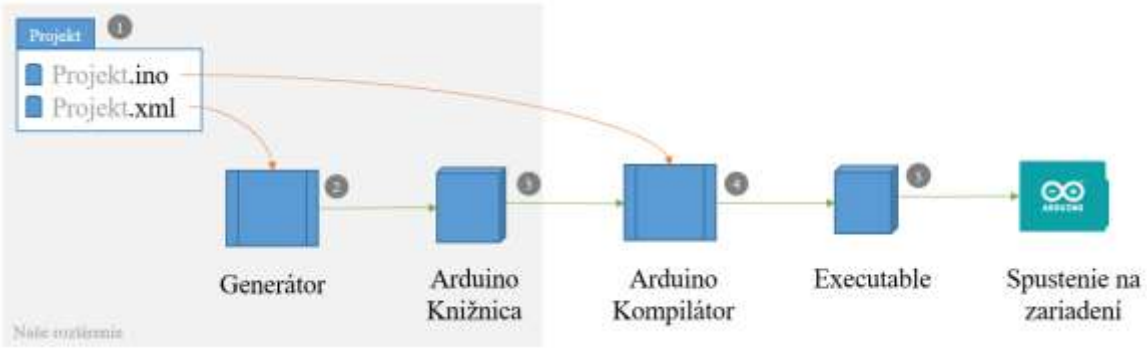
5 Architektúra komponentového a udalosťami orientovaného riešenia ACProg

Navrhnuté komponentovo a udalosťami orientované riešenie pre Arduino zariadenia sme nazvali 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. Komponentami v projekte ACProg primárne rozumieme senzory a aktuátory pripojené k Arduino doske. Na obrázku je príklad zapojenia komponentov spolu s útržkom kódu pre priblíženie dostupnej práce s komponentami pomocou zdrojového kódu.



Obr. 11 Schéma generovania a kompilácie projektu ACProg.

V nasledujúcej časti tejto kapitoly predstavíme ako naše komponentové a udalosťami orientované riešenie ACProg pracuje a rozširuje dostupný kompilátor Arduino. Na nasledujúcom obrázku je schéma označená krokmi, ktoré budú vysvetlené v nasledujúcich odstavcoch. Kroky schémy sú v poradí akom sa s nimi pracuje.



Obr. 12 Schéma generovania a kompilácie projektu ACProg.

Súborová štruktúra v bode 1 zobrazuje dva hlavné projektové súbory. `Projekt.xml` je konfiguračný súbor projektu, ktorému sa budeme detailnejšie venovať v podkapitole 5.1. `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 je `#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.

5.1 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 sa nachádza nami vytvorená ukážková XML konfigurácia pre lepšiu vizuálnu predstavu. XML súbor obsahuje tieto elementy:

- `<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 kontrolu 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`, obsahujúce elementy `component`. Element `component` obsahuje dva 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 `component` 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. 13 Príklad XML súboru projektu ACProg.

5.2 Generátor knižnice pre Arduino

Naše riešenie ACProg nedefinuje nový programovací jazyk. ACProg ide cestou Arduino knižnice vytvorenej podľa popisu komponentov v XML súbore. 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 ich môže upravovať počas behu programu. 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.

5.3 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.

6 IDE pre projekt ACProg

Projekt Arduino bol úspešný aj vďaka svojmu používateľsky prívetivému prostrediu. Preto sme sa rozhodli nasledovať jeho príklad a vytvoriť 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 podkapitole 6.4 sa budeme venovať implementačným krokom, ktoré viedli k naplneniu používateľských požiadaviek.

6.1 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. Voliteľné prostredie 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

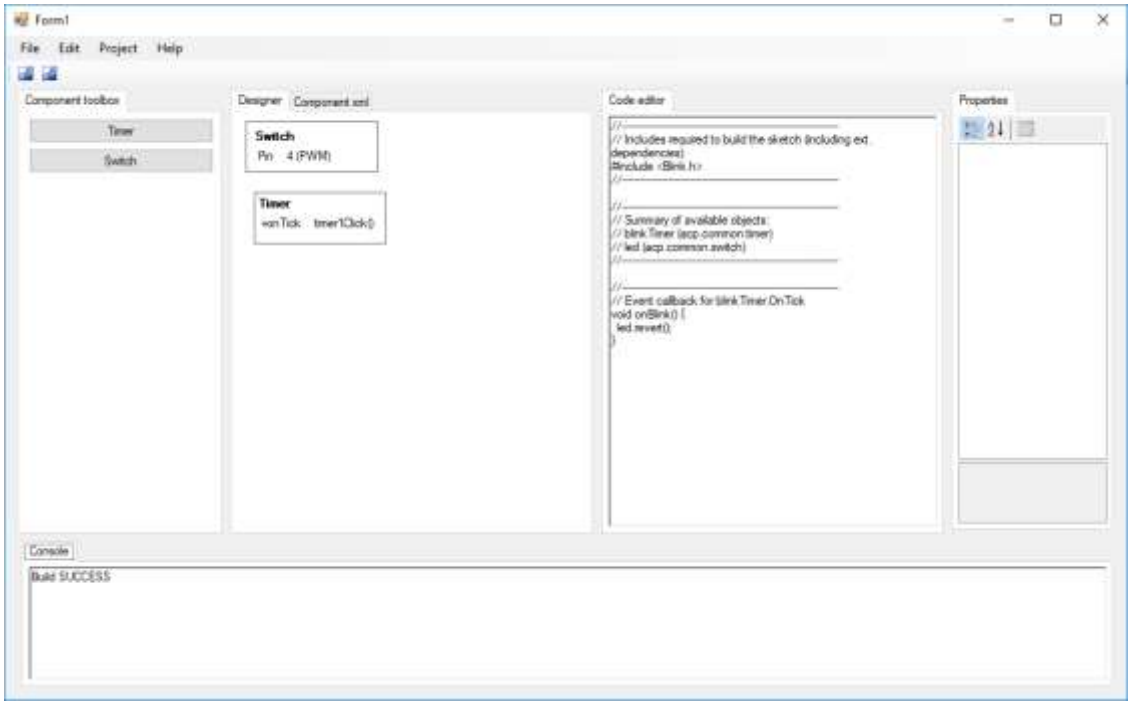
hovoríme o typoch komponentov, tak potrebujeme používateľovi ukázať aj ponuku všetkých dostupných komponentov z ACProg repozitára (3. use case).

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 jednoduchšiemu vytvoreniu produkčných zariadení.

6.1.1 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. 14 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 obrázku číslo 14 je nami vytvorený prototyp integrovaného vývojového prostredia.

Základným prvkom prostredia sú karty, pomocou ktorých naplníme používateľskú požiadavku voliteľného prostredia. 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.

6.2 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. K výberu programovacieho jazyka potrebujeme aj správcu knižníc (angl. package manager) pre automatizované sťahovanie rôznych knižníc a zostavenie projektu. V našom projekte sme použili nástroj Maven.

6.2.1 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:

- Editor zdrojového kódu,
- Vlastnosti inštancie komponentu (angl. properties),
- Ponuka komponentov (modulov) a Inštancie komponentov.

6.2.2 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.

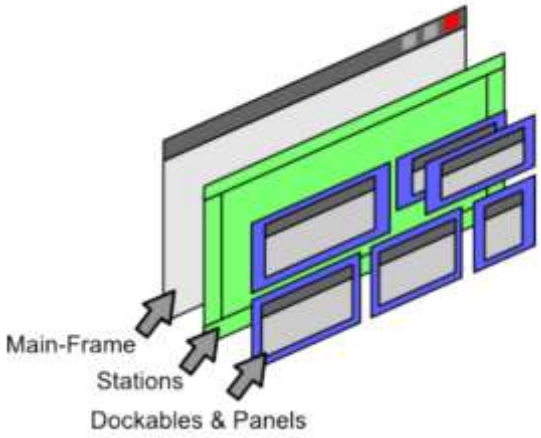
Počas implementácie záložiek pomocou frameworku Swing sme narazili na niekoľko problémov. Prvým problémom bola často obmedzená implementácia layout manažérov na 3 stĺpce (`BorderLayout`). Keďže sme v návrhu uvažovali 4 stĺpce tak sme sa pokúsili splniť cieľ vnáraním layout manažérov, čo spôsobovalo neintuitívne správanie záložiek. Pri iných layout manažéroch vo frameworku 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 bolo poskytovať integrované vývojové prostredie ako open source systém, tento platený modul sme nemohli použiť.

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



Obr. 15 DockingFrames ilustrácia ukladania záložiek v okne [12]

6.2.3 Editor zdrojového kódu

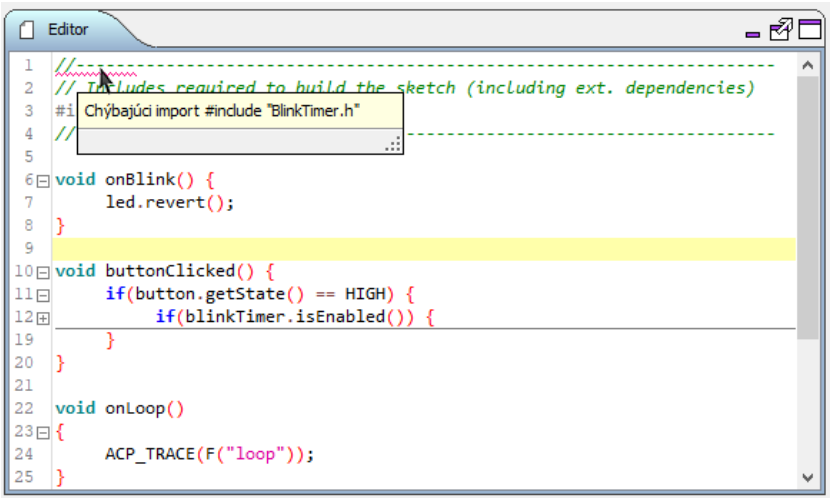
V integrovanom vývojovom prostredí programátor vytvára zdrojový kód pre spracovanie udalosti. Preto je na mieste potreba editora zdrojového kódu s vyznačovaním syntaxe. Vhodným rozšírením editora by bolo aj automatické dokončovanie s ponukou vhodných existujúcich metód, funkcií a premenných. Preto sme sa pozreli na editory, ktoré sú využívané v iných open source IDE a sú implementované v 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 bolo 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. 16 Editor zdrojového kódu s ilustrovaním vyznačenia chyby.

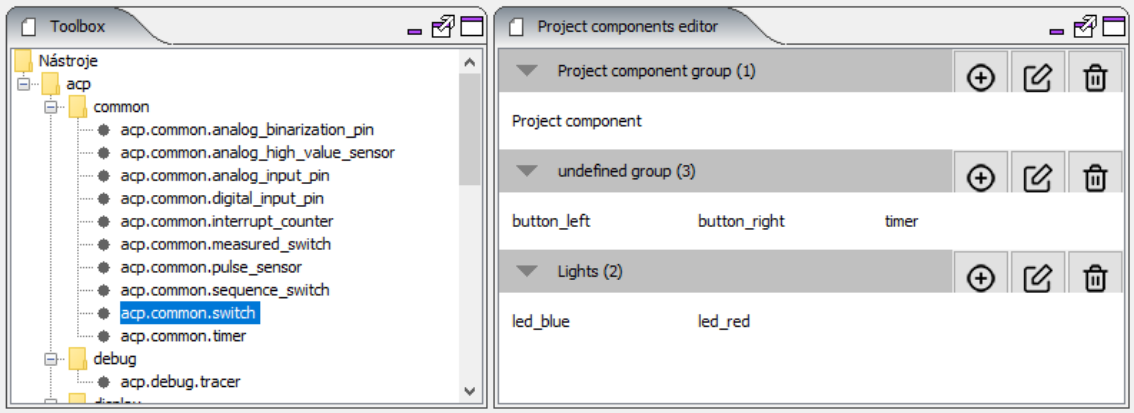
6.2.4 Ponuka komponentov (modulov) a inštancie komponentov

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, rozdeľujeme inštancie komponentov na skupiny. Inštancie komponentov je možné presúvať medzi skupinami. 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 návrhový vzor vyskytujúci sa v bežných Java Swing grafických komponentoch, zobrazujúcich formulárové dáta (napr. JTree). Na nasledujúcom obrázku ilustrujeme dva vyššie predstavené grafické komponenty.



Obr. 17 Ponuka komponentov (vľavo), inštancie komponentov (vpravo).

6.2.5 Vlastnosti inštancie komponentu (angl. properties)

K inštanciám komponentov je dôležité mať grafický prvok pre nastavenie vlastností inštancie komponentu. Grafický komponent je nazvaný **Vlastnosti inštancie komponentu**. Jeho hlavnou úlohou je po vybraní inštancie komponentu načítať z definičného súboru vlastnosti aj udalosti a zobraziť ich programátorovi. Udalosť uvažujeme ako vlastnosť typu `event`. Vlastnosti komponentu majú jeden z nasledujúcich dátových typov (uvedené sú najbežnejšie používané dátové typy):

- `int` – číselný typ,
- `string` – textový typ,
- `pin` – číselný typ označujúci hardvérový prostriedok dosky Arduino,
- `analog-pin` – číselný typ označujúci analógový 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.

6.3 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. K dispozícii sú tieto záložky:

- 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.

6.4 Implementácia IDE

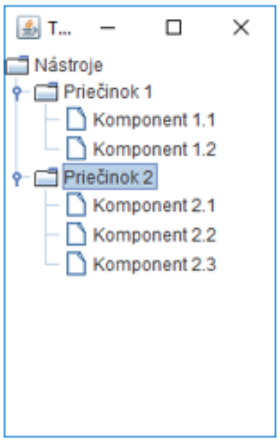
V tejto podkapitole sa budeme venovať implementačným problémom, aké sme riešili pri tvorbe integrovaného vývojového prostredia. Problémy vysvetlíme v spojení s príslušnými use case, ktoré sme navrhli v podkapitole 6.1.

6.4.1 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 komponent JTree frameworku Swing.

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.



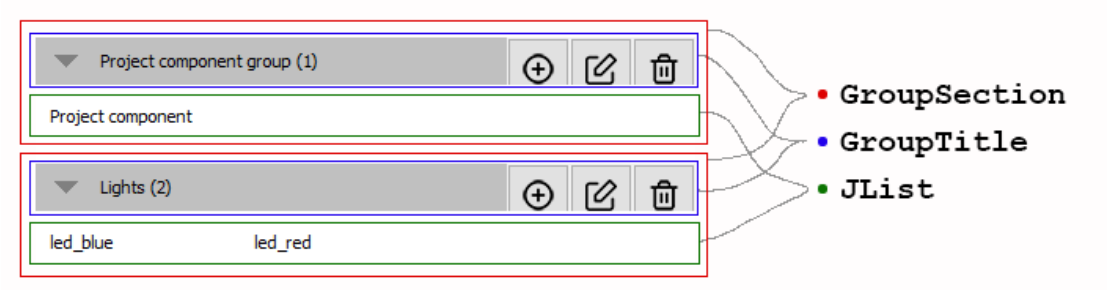

```
DefaultMutableTreeNode rootTreeNode, treeNode;
// Koreňový element stromu
rootTreeNode = new DefaultMutableTreeNode("Nástroje", true);
// Pridanie stromu elementov
treeNode = new DefaultMutableTreeNode("Priečinok 1");
treeNode.add(new DefaultMutableTreeNode("Komponent 1.1"));
treeNode.add(new DefaultMutableTreeNode("Komponent 1.2"));
rootTreeNode.add(treeNode);
treeNode = new DefaultMutableTreeNode("Priečinok 2");
treeNode.add(new DefaultMutableTreeNode("Komponent 2.1"));
treeNode.add(new DefaultMutableTreeNode("Komponent 2.2"));
treeNode.add(new DefaultMutableTreeNode("Komponent 2.3"));
rootTreeNode.add(treeNode);
// Vytvorenie modelu
TreeModel treeModel = new DefaultTreeModel(rootTreeNode);
// Grafický element stromu a vloženie do okna swing
JTree jTree = new JTree(treeModel);
add(jTree, BorderLayout.CENTER);
```

Obr. 18 Vizuál grafického komponentu JTree so zdrojovým kódom.

6.4.2 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. 19 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 grafického komponentu je aj jeho jednoduché presúvanie inštancií medzi skupinami. Na naplnenie tejto požiadavky potrebujeme spracovanie udalosti kliknutia na konkrétnu inštanciu 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.

6.4.3 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, ktorými sú:

- 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. 20 Konzolové príkazy poskytované v Arduino IDE.

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

6.4.4 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

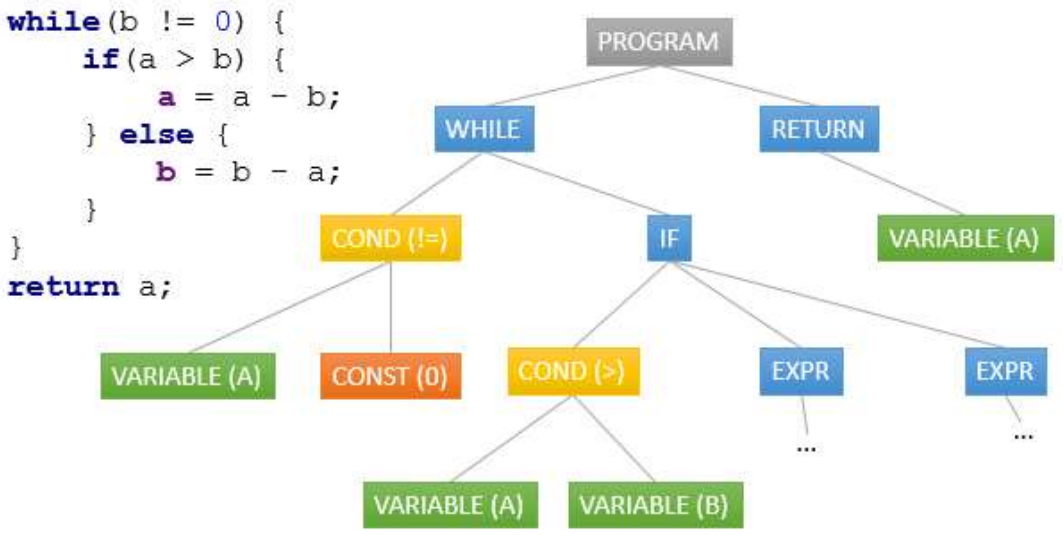
vidieť chybnú konfiguráciu spolu s upozorneniami, ktoré dáme programátorovi po analýze inštancií komponentov.

Obr. 21 Ilustrácia chybnej konfigurácie inštancií komponentov.

V druhom kroku analyzujeme samotný 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 konštrukciu, akú programátor použil. Stromu hovoríme abstraktný, pretože v ňom nepotrebujeme poznať 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 konštrukcie vyskytujúce sa v napísanom zdrojovom kóde. Nasledujúci obrázok ilustruje AST vytvorený z jednoduchého zdrojového kódu.



Obr. 22 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.

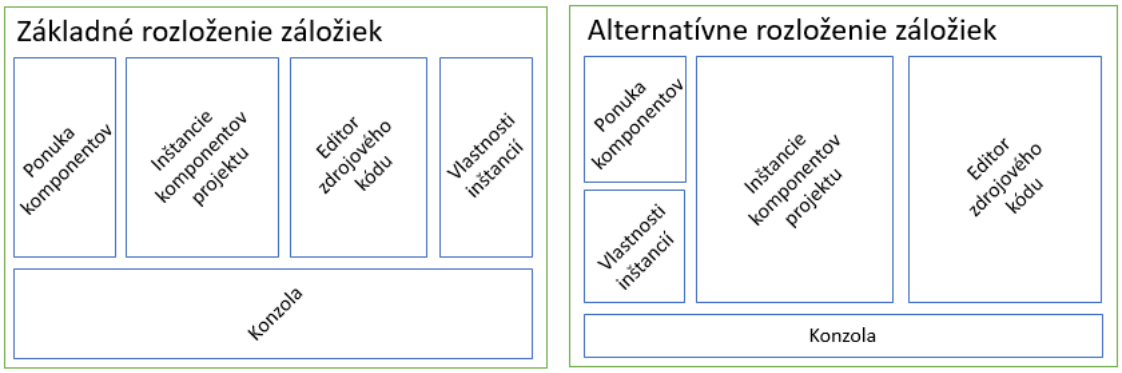
6.4.5 Voliteľné prostredie pre programátora

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.

Za účelom dosiahnutia voliteľného prostredia pre programátora sme potrebovali zabezpečiť, aby sa načítalo rovnaké rozloženie, ako programátor 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. 23 Navrhnuté rozloženia integrovaného vývojového prostredia.

7 Komponenty pre ACProg

V 5. kapitole sme vysvetlili fungovanie ACProg riešenia a v jej podkapitole 5.3 sme priblížili typy komponentov. Táto kapitola sa bude venovať vytváraniu nových typov komponentov. V úvodnej časti kapitoly priblížime spôsob tvorby nových typov komponentov. V nasledujúcej časti sa budeme venovať novému komunikačnému typu komponentu pre komunikáciu cez 433 MHz.

Pre vytvorenie nového typu komponentu potrebujeme vytvoriť definičný XML súbor (definition.xml). Definičný súbor obsahuje nutné popisné informácie o novom type komponentu počnúc jeho názvom a poznámky, až po implementačné informácie pre generátor zdrojového kódu. Pre tvorbu typov komponentov sme sa rozhodli využiť návrhový vzor View-Controller, ktorý prehľadne a efektívne rozdeľuje zdrojový kód. Zdrojový kód pre každý typ komponentu bude vo vlastnom `namespace`, ktorého názov kopíruje uloženie komponentu v repozitári. Každý typ komponentu povinne obsahuje `View` triedu. Programátor má inštanciu tejto triedy k dispozícii pre jeho projekt. Pomocou aplikačného rozhrania môže meniť stavy komponentu alebo čítať aktuálnu hodnotu stavu. Trieda `Controller` slúži pre komplikovanejšiu logickú vrstvu typu komponentu. Riešenie ACProg pre interné plánovanie úloh cyklicky spúšťa obslužné metódy komponentov, ak sú povolené. Tieto cyklicky spustiteľné metódy nazývame `looper` a ich nastavenie je potrebne v definičnom súbore typu komponentu. Metóda `looper` je spravidla implementovaná v triede `Controller`. Ak by bola trieda `looper`-a vo `View` tak by k nej mohol programátor pristupovať a negatívne tak ovplyvňovať beh projektu. Typy komponentov obsahujúce udalosti potrebujú v zdrojovom kóde triedy Controller definovať udalosti ako verejné inštančné premenné typu `ACPEventHandler`. Do inštančných premenných generátor vloží názvy obslužných tried pri generovaní. Pre vyvolanie udalosti stačí ak zavoláme triedu typu `ACPEventHandler`. V definičnom súbore potrebujeme zadefinovať udalosti tiež. Nasledujúci zdrojový kód predstavuje definičný XML súbor pre typ komponentu digitálneho vstupu.

```
<component-type name="acp.common.digital_input_pin">
     <description>Simple digital input pin.</description>
     <view> <!-- Trieda View -->
          <includes>
               <include>DigitalInputPin.h</include>
          </includes>
          <class>acp_common_dip::TDigitalInputPin</class>
```

```
          <constructor-args>
               <arg type="autogenerated">controller</arg>
          </constructor-args>
     </view>
     <controller> <!-- Trieda Controller -->
          <includes>
               <include>DigitalInputPin.h</include>
          </includes>

     <class>acp_common_dip::DigitalInputPinController</clas
s>
          <constructor-args>
               <arg type="property">Pin</arg>
               <arg type="property">InvertedLogic</arg>
          </constructor-args>
     </controller>
     <loopers> <!-- Metódy looper v triede Controller -->
          <looper>
               <method>readLooper</method>
               <interval>ReadInterval</interval>
          </looper>
     </loopers>
     <properties>
          <property>
               <name>Pin</name>
               <type>pin</type>
               <description>The pin to which the button is
connected.</description>
          </property>
          <property>
               <name>InvertedLogic</name>
               <type>bool</type>
               <value type="default">false</value>
               <description>True, if the button state is
inverted comparing to a value that is read at the pin,
false otherwise.</description>
          </property>
     </properties>
     <events>
          <event>
               <name>OnStateChanged</name>
               <binding
type="attribute">stateChangedEvent</binding>
               <description>When the logical state of pin
changed.</description>
          </event>
     </events>
</component-type>
```

Obr. 24 Konzolové príkazy poskytované v Arduino IDE.

7.1 Digitálny vstup

Na zdrojovom kóde č. 24 je zobrazený definičný XML súbor typu komponentu digitálneho vstupu. Úloha typu komponentu je načítavať hodnotu na digitálnom pine (zadaný vlastnostou Pin). Načítanie sa vykonáva pomocou `looper`-a, ktorý okrem samotného čítania informácie na pine vykonáva aj overenie zmeny novo načítanej hodnoty. Ak nastane zmena načítanej hodnoty, tak bude vyvolaná udalosť `onValueChanged`.

Tento typ komponentu zaradzujeme medzi hlavné typy komponentov riešenia ACProg. Okrem neho k hlavným typom komopnentov patria aj:

- Analógový vstup – typ komponentu je podobný digitálnemu vstupu. Hlavným rozdielom je načítanie viacerých hodnôt čo sa odzrkadluje v rozšírenej definícii komponentu o možnosť nastavenia hranice po ktorej prekročení bude vyvolaná udalosť.
- Digitálny výstup – typ komponentu slúžiaci na ovplyvňovanie fyzického sveta, obsahujúci iba aplikačné rozhranie. Udalosti v tomto type nie sú. Základným príkladom tohto typu komponentu je ovládanie LED diódy.

V prílohe A sú vypísané dokumentačné tabuľky pre vyššie spomenuté typy komponentov.

7.2 Rádiová komunikácia pomocou 433 MHz

Jedným z komunikačných typov komponentov, ktorý sme implementovali v riešení ACProg je komponent rádiovej komunikácie pomocou frekvencie 433MHz. Hardvérové zariadenie spočíva v dvoch integrovaných doskách. Jedna pre prijímanie a druhá pre vysielanie signálu, z čoho vyplýva, že táto komunikácia je jednosmerná. Preto sme vytvorili 2 typy komponentov (prijímač, vysielač). Ako vzor sme vytvorenie komponentu sme použili Arduino knižnicu RCSwitch [14], pretože efektívne implementuje komunikáciu s týmito integrovanými doskami.

Prvým krokom návrhu typu komponentu je analýza vlastností komponentov a možných udalostí. Nasledovať bude návrh potrebných metód aplikačného rozhrania. Hlavnou vlastnosťou je pin pripojenia komponentu. Medzi ďalšie vlastnosti potrebné pre komunikáciu patria tieto:

- Protokol – určuje spôsob a poradie odosielania bitov.

- Dĺžka pulzu – určuje časovanie pre odosielanie bloku bitov.
- Počet opakovaní odosielania – opakované odoslania.

Spomenuté vlastnosti platia rovnako pre prijímač aj pre vysielač. Udalosti aj aplikačné rozhranie sa už odlišujú pre oba typy komponentov. Typ komponentu vysielač bude obsahovať iba aplikačné rozhranie spočívajúce jedinej metóde `send`. Úlohou metódy bude odoslanie zadaného prúdu dát. Typ komponentu prijímač neobsahuje aplikačné rozhranie, avšak obsahuje `looper` pre príjem informácií. Po prijatí informácií na prijímač ich `looper` zaznamená a vyvolá udalosť `onMessageReceived`. Nasledujúce tabuľky zhrňujú navrhnuté vlastnosti, udalosti a aplikačné rozhranie typov komponentov.

Záver

Arduino svojím príchodom na IoT scénu sprístupnilo drobnú elektroniku a jej prototypovanie širšej verejnosti. Výpočtové parametre týchto zariadení nepodporujú spustenie operačných systémov a s pravidla sú limitované jedným výpočtovým vláknom. Toto obmedzenie môže byť problémom pre programátorov, ktorí prišli k drobnej elektroniky zo sveta programovania desktopových a mobilných aplikácií. V práci sme ukázali naše riešenie ACProg, ktoré prináša komponentovo orientované a udalosťami riadené programovanie Arduino zariadení. Dôležitou súčasťou nového riešenia ACProg je integrované vývojové prostredie. Pre programátorov je toto riešenie rovnako jednoduché ako tvorba desktopových alebo mobilných aplikácií.

Integrované vývojové prostredie sme obohatili abstraktnou syntaktickou analýzou c++ zdrojového kódu, vďaka čomu dokážeme zo zdrojového kódu vyňať funkcie, premenné a pod. Pomocou týchto informácií vieme kde v kóde sa funkcie nachádzajú a programátorovi ich vieme ponúknuť pri nastavovaní udalostí. Pomocou takto zozbieraných informácií vykonávame verifikáciu projektu a na prípadné chyby upozorníme programátora ešte skôr ako spustí projekt na svojom Arduino zariadení.

Poslednou no dôležitou úlohou bolo vytvoriť základné typy komponentov, s akými sa programátori bežne stretnú pri prototypovaní ich zariadení. Vytvorili sme základne typy komponentov pre digitálny vstup, digitálny výstup, analógový vstup. Okrem týchto sme sa venovali aj rádiovej komunikácii 433 MHz a vytvorili sme taktiež typ komponentu aký môžu programátori použiť bez potreby poznať komunikačný protokol s integrovanými obvodmi pre túto rádiovú komunikáciu.

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.

[14] Sui77. [Online]. Available: https://github.com/sui77/rc-switch.

Prílohy

Príloha A: Dokumentačné tabuľky k typom komponentov.

Príloha B: CD médium obsahujúce vytvorené riešenie s integrovaným vývojovým prostredím.

Príloha A

Tab. 2 Dokumentačná tabuľka komponentu digitálny vstup

Vlastnosti	Udalosti	Aplikačné rozhranie
Pin InvertedLogic	OnStateChanged()	getState()

Tab. 3 Dokumentačná tabuľka komponentu analógový vstup

Vlastnosti	Udalosti	Aplikačné rozhranie
Pin ReadInterval	OnValueChanged()	getValue()

Tab. 4 Dokumentačná tabuľka komponentu digitálny výstup

Vlastnosti	Udalosti	Aplikačné rozhranie
Pin InitialState InvertedLogic	*bez udalostí*	isOn() / isOff() turnOn() / turnOff() setState() revert()

Tab. 5 Dokumentačná tabuľka komponentu RC 433 vysielač

Vlastnosti	Udalosti	Aplikačné rozhranie
Pin Protocol PulseLength RepeatTransmit	*bez udalostí*	send()

Tab. 6 Dokumentačná tabuľka komponentu RC 433 prijímač

Vlastnosti	Udalosti	Aplikačné rozhranie
Pin Protocol PulseLength RepeatTransmit	OnMessageReceived()	*bez aplikačného rozhrania*