

# **Komponentovo orientované a udalost'ami riadené programovanie arduino zariadení**

SDI1b

Autor práce: **Patrik Pekarčík**

Vedúci práce: **František Galčík**

# Obsah

1	Úvod .....	4
1.1	Ciele práce.....	5
2	Arduino.....	6
2.1	Parametre dosky .....	7
2.2	Rôzne typy dosiek .....	9
2.3	Dostupné riešenia pre platformu Arduino .....	9
2.3.1	Arduino EventManager .....	10
2.3.2	Quantum Leaps Modeling Tool .....	10
2.3.3	Cayenne .....	11
3	Programovacia paradigma a návrhové vzory .....	12
3.1	Programovacia paradigma.....	12
3.1.1	Udalosťami orientované programovanie .....	12
3.1.2	Komponentovo orientované programovanie .....	13
3.2	Návrhové vzory .....	14
3.2.1	Singleton.....	14
3.2.2	Proxy .....	14
4	Architektúra riešenia .....	16
4.1	Princíp použitia .....	16
4.2	Generátor zdrojového kódu.....	17
5	Integrované používateľské prostredie.....	19
5.1	Abstraktný syntaktický strom .....	21
5.2	Automatické dokončovanie kódu.....	22
6	Analýza komponentov .....	23
6.1	Časovač (acp.common.timer).....	24
6.2	Prepínač (acp.common.switch) .....	24

6.3	Digitálne čítanie vstupu (acp.common.digital_input_pin).....	25
6.4	Analógové čítanie vstupu (acp.common.analog_input_pin).....	26
7	Záver.....	27
	Zoznam použitej literatúry .....	28
	Príloha č. 1 – Demo projekt .....	29

# 1 Úvod

IoT alebo „internet vecí“ je pojem resp. téma, o ktorej dnes počujeme zo všetkých médií. Vývoju IoT zariadení sa venujú známe spoločnosti, ako napríklad firma Philips, ktorá medzi inými ponúka aj inteligentné osvetlenie. Popularizácii IoT pomohol aj príchod zariadení Arduino. Arduino je open-source platforma s mikrokontrolerom ATmega, ktorá vývojárom neponúkla iba hardvér s procesorom, ale aj pomerne jednoduché vývojové prostredie Arduino IDE. Vývoj na platforme Arduino je založený na programovaní jazyku C++.

Program pre platformu Arduino sa skladá z dvoch základných funkcií:

- `setup()` – funkcia spustená iba raz po zapnutí zariadenia
- `loop()` – periodicky spúšťaná funkcia, pokiaľ je zariadenie zapnuté

Tento prístup nepodporuje efektívny multitasking, na aký sú programátori zvyknutí z programovania pre operačný systém. 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. Obmedzená veľkosť operačnej pamäte nám zatvára dvere pred použitím komplexnejšieho operačného systému, čo však nevylučuje vytvorenie minimalistického plánovača úloh pre toto zariadenie. Pri návrhu možného fungovania sa chceme inšpirovať komponentovo orientovaným prístupom, aký poznáme z vývoja používateľských aplikácií v operačných systémoch (napr. Swing v jazyku Java).

## **1.1 Ciele práce**

1. Preskúmať, analyzovať a porovnať existujúce prístupy, softvérové aplikácie a knižnice využívané pri programovaní Arduino zariadení.
2. Preskúmať a analyzovať možnosti komponentového a udalosťami riadeného programovania s ohľadom na hardvérové obmedzenia Arduino zariadení.
3. Vychádzajúc z existujúcich open-source projektov a knižníc navrhnuť a implementovať užívateľsky prívetivé riešenie na jednoduché komponentovo-orientované a udalosťami riadené programovanie Arduino zariadení.
4. Implementovať vzorové komponenty využiteľné pri návrhu a implementácii IoT riešení.

## 2 Arduino

Arduino je hardvér a softvér s otvoreným zdrojovým kódom, ktoré sú licencované na základe licencie GNPL Lesser General Public License (LGPL) alebo GNU General Public License (GPL), umožňujúce výrobu dosiek Arduino a distribúciu softvéru každým , Arduino dosky sú komerčne dostupné v predmontovanej podobe alebo v súpravách do-it-yourself.

Dosky typu Arduino používajú rôzne mikroprocesory a riadiace jednotky. Dosky sú vybavené súbormi digitálnych a analógových vstupno-výstupných (I/O) pinov, ktoré môžu byť prepojené s rôznymi rozširujúcimi doskami (shields) a inými obvodmi. Dosky obsahujú sériové komunikačné rozhrania, vrátane univerzálnej sériovej zbernice (USB) na niektorých modeloch, ktoré sa používajú aj na načítanie programov z osobných počítačov. Mikrokontroléry sú typicky naprogramované pomocou dialektu funkcií z programovacích jazykov C a C ++. Popri používaní tradičných kompilátorov poskytuje projekt Arduino integrované vývojové prostredie (IDE) založené na projekte Language Processing.



Obrázok: Doska Arduino UNO

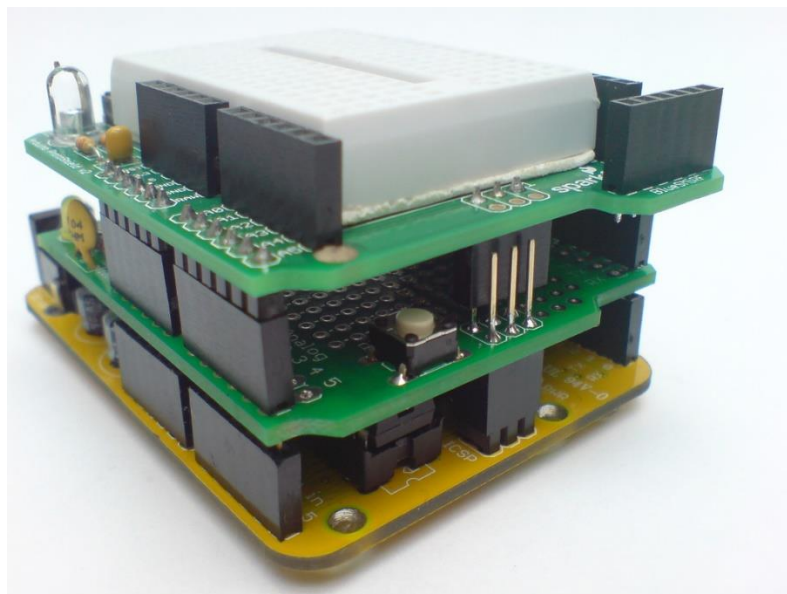
Projekt Arduino sa začal v roku 2003 ako program pre študentov v Interaction Design Institute Ivrea v Ivrea v Taliansku s cieľom poskytnúť novým a profesionálnym používateľom lacný a jednoduchý spôsob vytvárania zariadení, ktoré interagujú s prostredím pomocou senzorov a pohony. Bežné príklady takýchto zariadení určené pre začínajúcich fanatikov zahŕňajú jednoduché roboty, termostaty a detektory pohybu.

Návrhy hardvéru sú distribuované pod licenciou Creative Commons Attribution Share-Alike 2.5 a sú k dispozícii na webovej stránke spoločnosti Arduino. Sú tiež k dispozícii aj výrobné súbory pre niektoré verzie hardvéru. Zdrojový kód pre integrované vývojové prostredie (IDE) je uvoľnený pod licenciou GNU General Public License, verzia 2. Napriek tomu firmou Arduino nikdy nebol uvoľnený oficiálny zoznam materiálov Arduino.

Aj keď sú hardvérové a softvérové vzory voľne k dispozícii na základe licencií copyleft, vývojári požiadali, aby bol názov Arduino výlučne k oficiálnemu produktu a nebol použitý na odvodené diela bez povolenia. V oficiálnom dokumente o používaní názvu Arduino sa zdôrazňuje, že projekt je otvorený pre začlenenie práce iných do oficiálneho produktu. Niekoľko komerčne uvoľnených výrobkov, ktoré sú kompatibilné s Arduino, sa vyhýbali názvu projektu použitím rôznych mien končiacich v -duino.

## **2.1 Parametre dosky**

Väčšina dosiek Arduino pozostáva z mikrokontrolera Atmel 8-bit AVR (ATmega8, ATmega168, ATmega328, ATmega1280, ATmega2560) s rôznym množstvom pamäte flash, čipov a inštrukcií. 32-bitový Arduino Due, založený na Atmel SAM3X8E bol predstavený v roku 2012. Dosky používajú jednočlenné alebo dvojradové piny alebo zásuvky, ktoré uľahčujú pripojenie na programovanie a začlenenie do iných obvodov. Môžu sa pripojiť s doplnkovými modulmi nazývanými shields. Viacnásobné a prípadne naskladané shields môžu byť individuálne adresovateľné prostredníctvom sériovej zbernice IIC. Väčšina dosiek obsahuje 5 V lineárny regulátor a 16 MHz kryštálový oscilátor alebo keramický rezonátor. Niektoré konštrukcie, ako napríklad LilyPad, pracujú na frekvencii 8 MHz a kvôli špecifickým obmedzeniam tvarových faktorov sa neuplatňujú s palubným regulátorom napätia.



Obrázok: Viacero shields napojených na dosku Arduino

Arduino mikrokontroléry sú vopred naprogramované pomocou zavádzača, ktorý zjednodušuje nahrávanie programov do flash pamäte na čipoch. Predvolený zavádzač Aduino UNO je bootloader pre optiboot. Dosky sú načítané programovým kódom cez sériové pripojenie k inému počítaču. Niektoré sériové dosky Arduino obsahujú obvod prevodníkov úrovne na konverziu medzi logickými úrovňami RS-232 a signály úrovne TTL. Súčasnú dosku Arduino sú naprogramované pomocou Universal Serial Bus (USB), ktoré sú implementované pomocou čipov typu USB-to-serial adaptérom, ako napríklad FTDI FT232. Niektoré dosky, ako napríklad neskoršie modely modelov Uno, nahradia čip FTDI samostatným čipom AVR obsahujúcim USB-to-serial firmware, ktorý je preprogramovateľný cez vlastnú hlavičku ICSP. Iné varianty, ako napríklad Arduino Mini a neoficiálny Boarduino, používajú odnímateľný USB-to-sériový adaptér alebo kábel, Bluetooth alebo iné metódy, keď sa používajú s tradičnými nástrojmi mikrokontrolérov namiesto Arduino IDE, štandardné programovanie v systéme AVR (ISP).

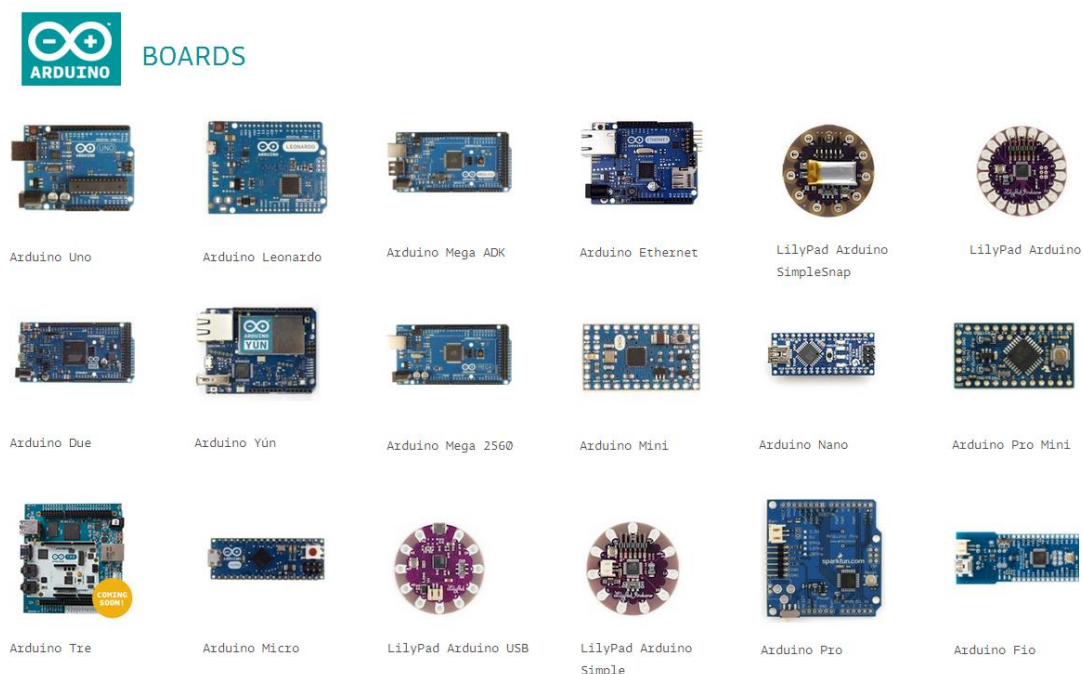
Doska Arduino vystavuje väčšinu I / O pinov mikrokontroléra pre použitie v iných obvodoch. Arduino Uno poskytuje 14 digitálnych I / O pinov, z ktorých šesť môže produkovať signály modulované šírkou pulzu a šesť analógových vstupov I / O pinov. Tieto piny sa nachádzajú na hornej časti dosky pomocou zásuviek s priemerom 0,1 palca (2,54 mm). Niektoré zásuvné aplikácie sú komerčne dostupné. Dosky Arduino Nano



môžu poskytovať zástrčky na spodnej strane dosky, ktoré sa môžu pripojiť do bezpájkových lôk.

## 2.2 Rôzne typy dosiek

Existuje mnoho dosiek odvodených od Arduino a Arduino. Niektoré sú funkčne ekvivalentné s Arduino a môžu byť použité zameniteľné. Mnohé zdokonaľujú základné Arduino tým, že pridávajú výstupné ovládače, často používané v školskom vzdelávaní, aby zjednodušili výrobu kočíkov a malých robotov. Ostatné sú elektricky ekvivalentné, ale menia tvarový faktor, niekedy si zachovávajú kompatibilitu so štítmi. Niektoré varianty používajú rôzne procesory s rôznou kompatibilitou.



Obrázok: Rôzne typy dosiek Arduino

## 2.3 Dostupné riešenia pre platformu Arduino

Stále rastúca komunita sa venuje zariadeniam Arduino. S tým je spojený aj vývoj rôznych frameworkov pre túto platformu, ktorých cieľom je zjednodušiť vývoj programátorom. Riešenia aké sme s pomocou Arduino komunity našli, rozdeľujeme do kategórií online a offline. Online sú riešenia, ktoré do zariadenia nainštalujú zavádzač preposielajúci a vykonávajúci akcie len od servera (logika systému je len na serveri).

Offline sú riešenia bežiacie priamo na Arduino zariadení a na vykonanie akcie sa rozhodujú bez servera. Nižšie si povieme niečo viac o jednom online a dvoch offline riešeniach. Všetky existujúce riešenia šli viac smerom automatov, alebo len udalostí.

### 2.3.1 Arduino EventManager

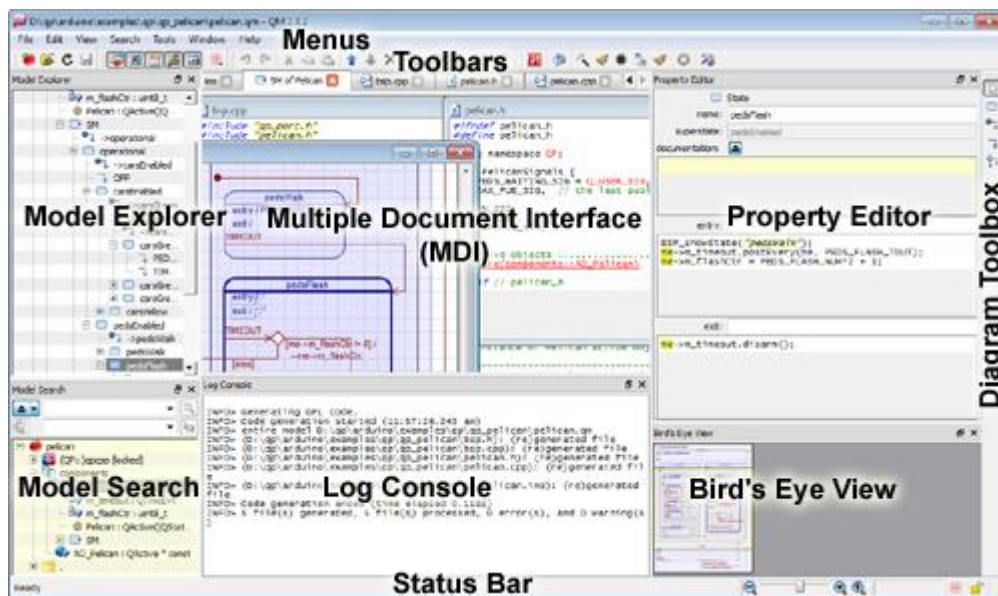
Je offline riešenie v ktorom neprogramujeme hlavné vlákno loop(). Programujeme však spracovanie udalostí. Na začiatku programu pre EventManager potrebujeme určiť, na zmeny ktorých pinov cheme vyvolať udalosť. Následne ak zmena nastane EventManager túto udalosť vykoná.

```
void myListener( int eventCode, int eventParam ) {  
    // Do something with the event  
}  
void setup() {  
    gMyEventManager.addListener( EventManager::kEventUser0, myListener );  
    // Do more set up  
}
```

Algoritmus: Príklad použitia EventManages

### 2.3.2 Quantum Leaps Modeling Tool

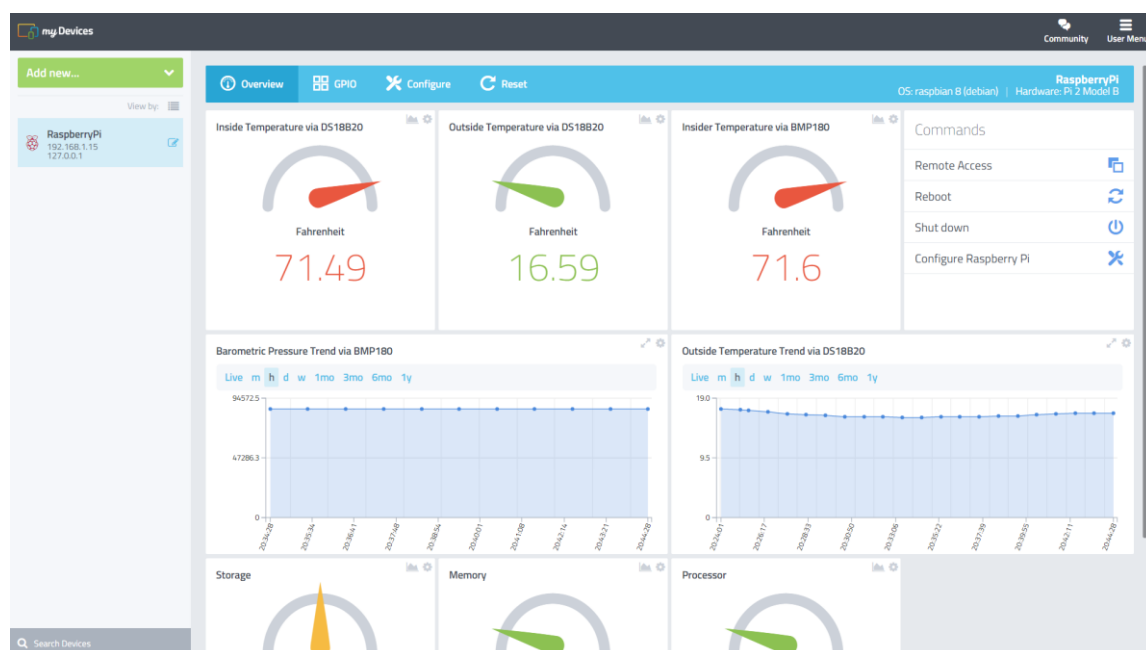
Je offline riešenie spolu s modelovacím nástrojom. Toto riešenie je špecifické tým, že pristupuje k programovaniu ako k stavovému automatu. Konečno stavový automat je matematický model výpočtu. Je to abstraktný stroj, ktorý môže byť v danom čase presne v jednom stave z konečného počtu stavov. Konečno stavový automat sa môže zmeniť z jedného stavu na druhý v reakcii na niektoré externé vstupy. Je definovaný zoznamom jeho stavov, ich počiatočným stavom a podmienkami pre každý prechod.



Obrázok: Rozhranie nástroja Quantum Leaps Modeling Tool

### 2.3.3 Cayenne

Je online riešenie podporujúce mnoho Arduino dosiek, kde každá doska posiela na server aktuálny stav pinov. Vyhodou tohto riešenia je priateľské používateľské prostredie s jednoduchým nastavením prahov pre jednotlivé piny Arduino dosiek, s priradením následnej akcie. Napr. Ak stúpne teplota (teplotný senzor je na pine 5) nad 30C, tak zapni klimatizáciu (zapni 5V na pine 4).



Obrázok: Rozhranie nástroja Cayenne

## 3 Programovacia paradigma a návrhové vzory

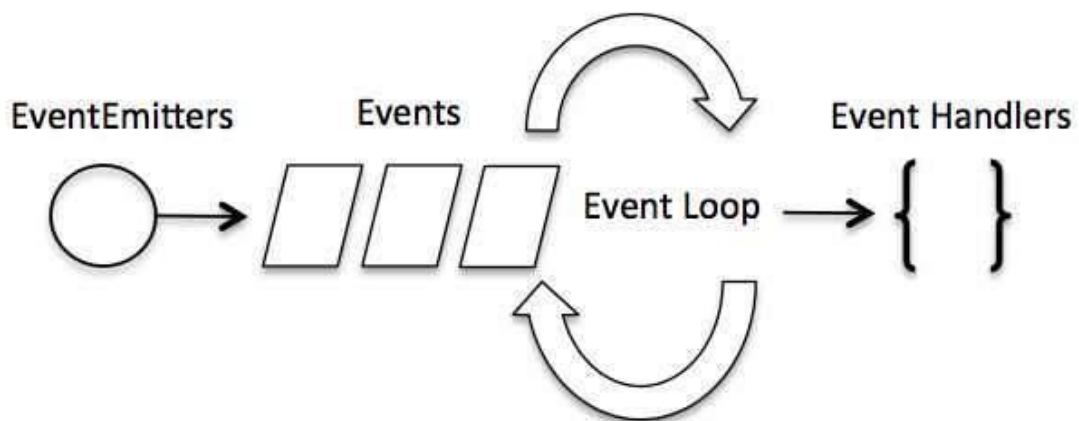
### 3.1 Programovacia paradigma

Programovacia paradigma je pomenovanie spôsobu myslenia pri programovaní. Nehovoríme tak programovacím jazykom, pretože na rôznych jazykoch vieme používať rôzne paradigmy. Existujú jazyky, ktoré vznikli na princípe niektorej z paradigiem. Napríklad jazyk Haskell je postavený na paradigme funkcionálneho programovania. Medzi paradigmy programovania patria:

- Imperative programming (slovenský výraz???),
- Funkcionálne programovanie,
- Deklaratívne programovanie,
- Objektovo orientované programovanie,
- Procedurálne programovanie,
- Logické programovanie,
- Symbolické programovanie.

#### 3.1.1 Udalosťami orientované programovanie

K programovacím paradigmám patrí aj udalosťami orientované programovanie. Priebeh programu v tejto paradigme je určený vznikajúcimi udalosťami. Udalosti vznikajú používateľskou akciou (stlačenie tlačidla), výstupom senzora alebo udalosťou vytvorenou v inej časti programu. S udalosťami orientovaným programovaním sa prevažne stretneme grafických používateľských rozhraniach (GUI), ktoré vykonávajú konkrétne akcie na základe používateľského vstupu.



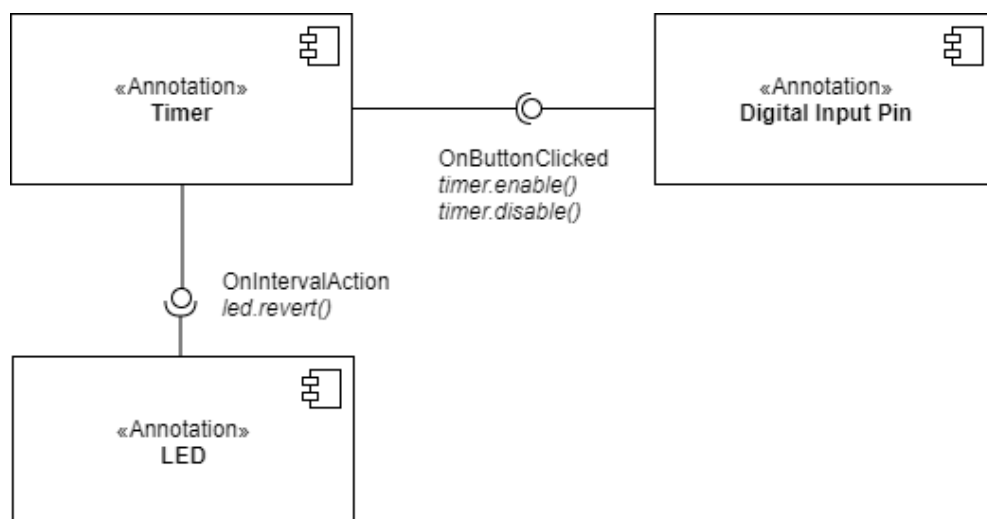
Obrázok: Schéma udalost'ami orientovaného programovania.

Hlavná slučka (metóda `look()`) v udalost'ami orientovaných aplikáciách čaká na spustenie udalosti. Po prijatí udalosti spustí zaregistrované callback funkcie. Callback je funkcia, ktoré bola v hlavnej slučke zaregistrovaná ako parameter, pre budúce spustenie kódu. V embedovaných systémoch je hlavná slučka riešená pomocou hardvérových prerušení namiesto aktívnej slučky.

### 3.1.2 Komponentovo orientované programovanie

Ďalšou paradigmou, ktorú sme zakomponovali do návrhu je komponentovo orientované programovanie. Táto paradigma je založená na skladaní voľne spojených nezávislých komponentov do výslednej aplikácie. Komponenty sú tvorené ako malé časti riešiace konkrétnu úlohu, poskytujú rozhrania na komunikáciu (analógia triedy a inštancie). Komponenty môžu vytvárať aj spracovávať udalosti, čím vieme prepojiť túto paradigmu s paradigmou udalost'ami orientovaného programovania.

**Definícia:** Softvérový komponent je softvérový modul, ktorý zapuzdruje množinu súvisiacich funkcií alebo údajov.



Obrázok: UML diagram komponentov pre demo projekt.

## 3.2 Návrhové vzory

Všeobecnou definíciou pre návrhový vzor je: Návrhový vzor je znovu použiteľné riešenie na bežne vyskytujúci sa problém v oblasti softvérového dizajnu. Nehovoríme o konkrétnom zdrojovom kóde, skôr o návode ako sa vysporiadať s konkrétnymi situáciami. Návrhové vzory delíme do troch hlavných kategórií:

- Creational patterns – zaoberajúce sa tvorbe inštancií tried.
- Structural patterns – zamerané na vzťahy medzi triedami.
- Behavioral patterns – zaoberajúce sa komunikáciou medzi objektmi.

Návrhových vzorov v jednotlivých kategóriách je veľa. V práci sme niektoré z nich aktívne využívali a preto si o nich povieme viac v nasledovných podkapitolách.

### 3.2.1 Singleton

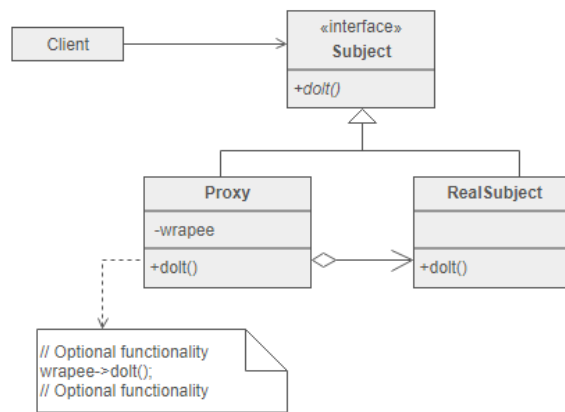
Singleton zaradzujeme do kategórie creational patterns. Definícia pre tento návrhový vzor znie: Je to trieda pre, ktorú existuje iba jedna inštancia. Príkladom použitia je trieda nesúca nastavenia programu, ktorú raz inicializujeme (načítame zo súboru) a rôzne komponenty projektu budú k tejto triede pristupovať.

```
public class Singleton {  
    private Singleton() {}  
  
    private static class SingletonHolder {  
        private static final Singleton INSTANCE = new Singleton();  
    }  
  
    public static Singleton getInstance() {  
        return SingletonHolder.INSTANCE;  
    }  
}
```

Zdrojový kód: Ukážka java kódu pre singleton.

### 3.2.2 Proxy

Návrhový vzor proxy zaradzujeme medzi štrukturálne. Použitie vzoru spočíva v obalení metód existujúcej triedy a jej rozširovanie. Vzor napomáha v rozdeľovaní aplikácií na menšie logické balíčky, ktoré sa medzi sebou tvária ako blackbox. Blackboxing znamená, že balíček môžeme použiť taký ako je, a nemôžeme priamo v ňom upravovať implementáciu.



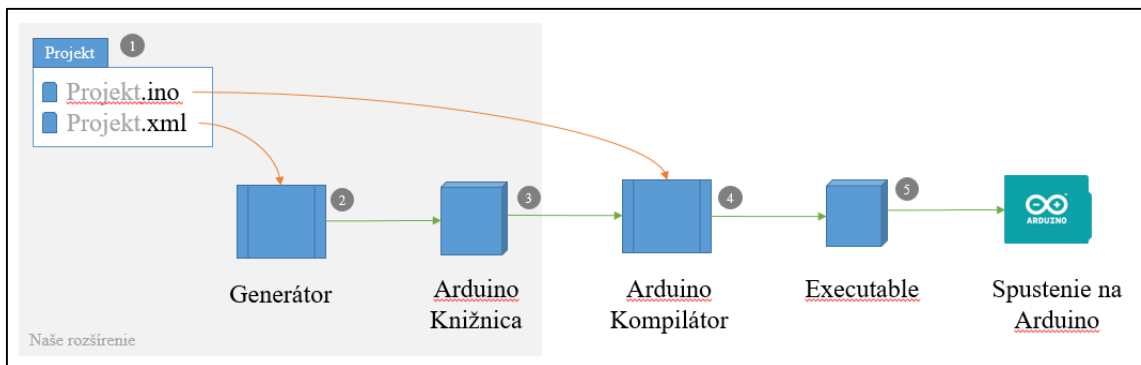
Obrázok: Schéma vzoru proxy. RealSubject je blackbox a v Proxy pridávame novú funkcionlitu.

## 4 Architektúra riešenia

V prvej časti tejto kapitoly si ukážeme základný princíp fungovania nášho riešenia. Definujeme pojmy s ktorými budeme ďalej pracovať a vysvetlíme schémy vyskytujúce sa v projekte. V ďalšej časti sa pozrieme na generátor zdrojového kódu, ktorý prevedie program z nášho riešenia do jazyka C podporovaného kompilátorom Arduino.

### 4.1 Princíp použitia

V našom riešení bude programátor okrem ino súboru vytvárať aj xml súbor s popisom komponentov a udalostí. Z xml súboru budeme generovať knižnicu, ktorá bude importovaná zdrojovým súborom. Vygenerovaná knižnica bude obsahovať triedy komponentov, s ich inštanciami. Okrem toho bude obsahovať aj hlavné funkcie `setup()` aj `loop()`, v ktorých bude za programátora riešiť plánovanie spustenia udalostí.



Obrázok č. – Pribeh spracovania projektu

Na obrázku vidíme priebeh spracovania zdrojových kódov až po spustenie na zariadení. Bod 1 znázorňuje súborovú štruktúru projektu (sketch-u). XML súbor obsahuje informácie o komponentoch a udalostiach. Tento XML súbor je vstupom pre generátor knižnice (bod 2). Generátor vytvorí knižnicu (Bod 3) s hlavičkovým súborom `Projekt.h`, ktorý bude povinne importovaný v INO súbore. INO súbor je bežným zdrojovým kódom pre arduino bez inicializačných funkcií `setup()` a `loop()` (tie obsahuje vygenerovaná knižnica). Vygenerovaná knižnica a INO súbor bude následne vstupom pre arduino kompilátor (Bod 4). Výstupom po kompilácii bude súbor inštrukcií, pre zariadenie. Posledným krokom je napálenie nových inštrukcií na zariadenie (Bod 5).



Priblížením schémy XML súboru si teraz vysvetlíme tvorbu komponentov. Hlavným tag-om je `project`. Jeho atribútom je `platform`, tým povieme generátoru aj kompilátoru, koľko má pamäte a iných systémových prostriedkov. Ďalej obsahuje tag `components`, ktorý je zoznamom tagov `component`. Component sa skladá z tagov `name`, `type`, `properties` a `events`.

- `Name` – obsahuje názov inštancie komponentu. Tento názov budeme používať ako remennú pri tvorbe zdrojového kódu.
- `Type` – obsahuje názov triedy komponentu.
- `Properties` – obsahuje tagy `property`. Ich atribút `name` určuje kľúč, ktorému patrí zadaná hodnota.
- `Events` – obsahuje tagy `event`. Atribút `name` určuje kľúč udalosti a hodnota názov metódy zo zdrojového kódu.

```
<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">onBlink</event>
  </events>
</component>
```

Zdrojový kód: Príklad XML konfigurácie komponentu.

## 4.2 Generátor zdrojového kódu

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, kde komponenty majú určený typ (modul). Generátor pridá do výslednej knižnice zdrojové kódy abstrakcie modulu ak tam ešte nie sú a v hlavnej časti behu programu vytvorí jeho inštanciu. Inštancie komponentov budú globálne a používateľ v nich bude môcť programátorsky meniť konfigurácie za behu. Okrem komponentov generátor vytvorí aj tzv. jadro nášho riešenia. Jadro spočíva v inicializácii komponentov a následnom plánovaní vykonávania

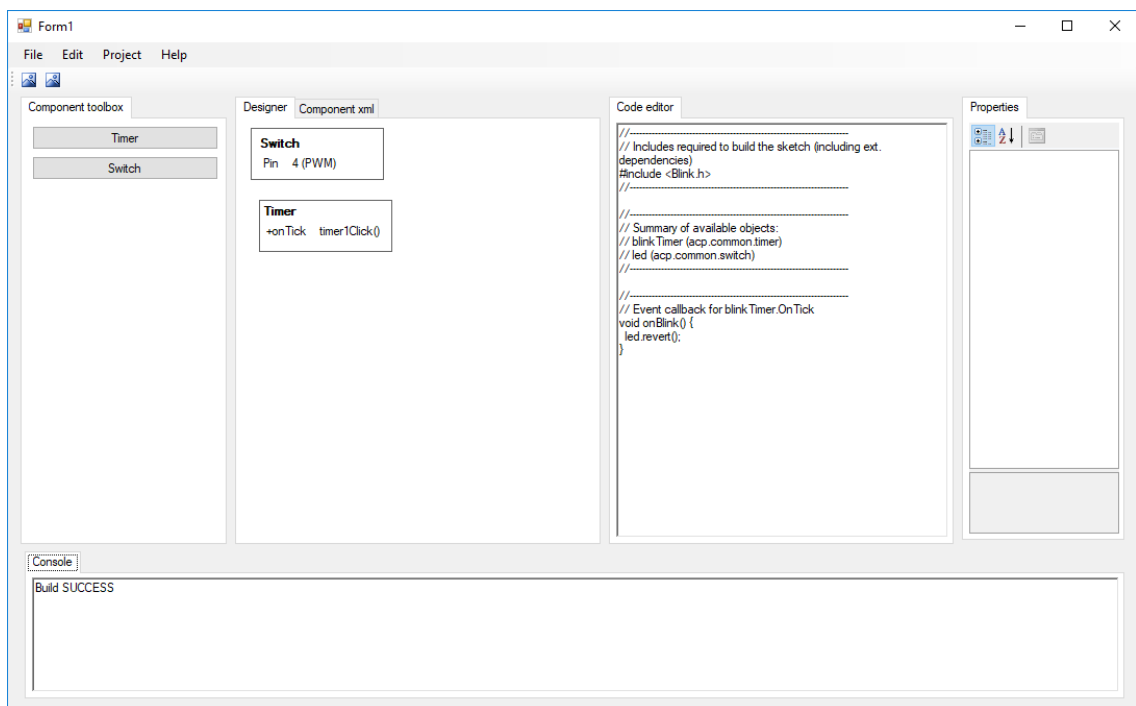
jednotlivých udalostí. Jadro poskytuje programátorovi aj aplikačné rozhranie na komunikáciu so stálou pamäťou EEPROM.

Iným teoretickým riešením by bolo vytvoriť vlastnú Arduino knižnicu (Framework), ktorá by komponenty interpretovala priamo na zariadení. Arduino má ale obmedzenú pamäť a s pribúdajúcimi typmi komponentov (modulov) by neostal dostatok miesta pre vlastné aplikácie. Arduino má len 32kB pamäte pre zdrojové kódy. Pri našom riešení generovania nevkladáme moduly, ktoré programátor vo svojom projekte nepoužil.

## 5 Integrované používateľské prostredie

Integrované vývojové prostredie (IDE) je softvérová aplikácia, ktorá poskytuje komplexné vybavenie pre programátorov pre vývoj softvéru. IDE sa zvyčajne skladá z editora zdrojového kódu, nástrojov na tvorbu automatizácie a nástroja na ladenie. Väčšina moderných IDE má inteligentné dokončenie kódu. Niektoré IDE, ako napríklad NetBeans a Eclipse, obsahujú kompilátor, interpret alebo oboje; Iné, napríklad SharpDevelop a Lazarus, nie. Hranica medzi integrovaným vývojovým prostredím a inými časťami širšieho prostredia vývoja softvéru nie je dobre definovaná. Niekedy je integrovaný systém riadenia verzií alebo rôzne nástroje na zjednodušenie konštrukcie Grafického používateľského rozhrania (GUI). Mnoho moderných IDE má tiež triedny prehliadač, prehliadač objektov a hierarchický diagram triedy pre použitie v objektovo-orientovanom vývoji softvéru.

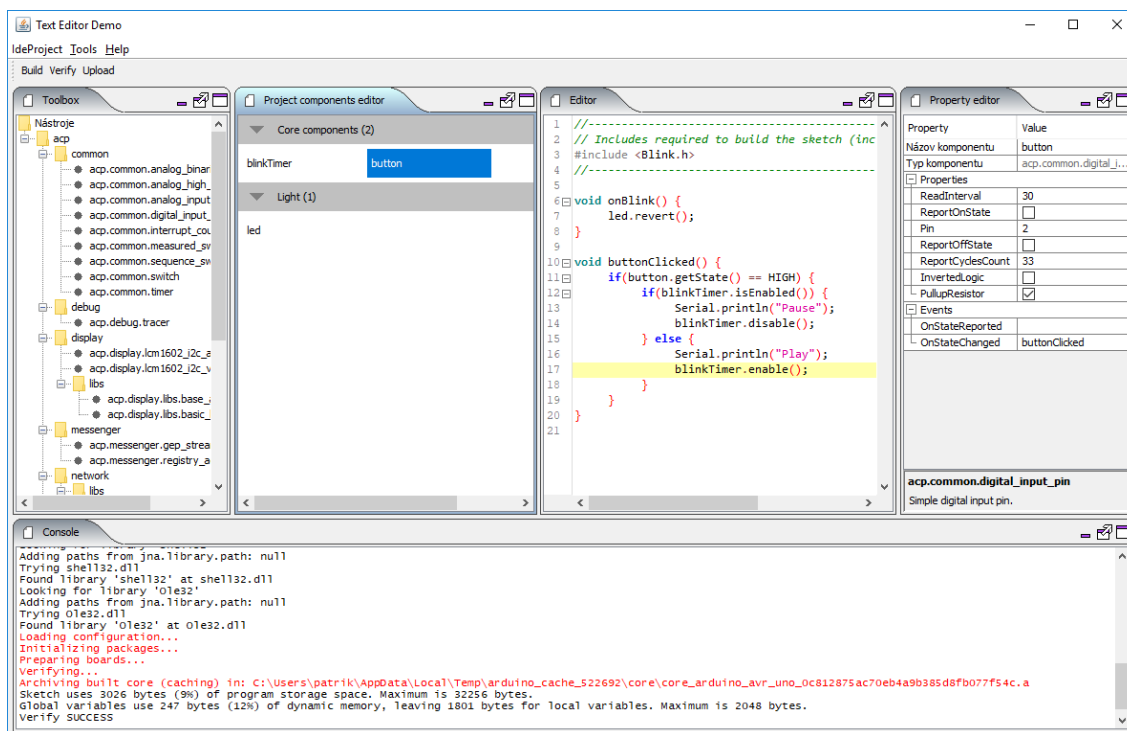
Pre doplnenie a jednoduchšie presadenie nášho riešenia sme sa rozhodli navrhnúť a implementovať vlastné integrované vývojové prostredie. Implementáciu sprevádzal návrh prostredia, ten bol inšpirovaný prostredím Visual Studio. Pre otestovanie sme vyrobili prototyp prostredia pomocou Windows Form.



Obrázok: Prototyp integrovaného vývojového prostredia.

Vývoj prostredia nasledoval prieskum dostupných riešení pre tvorbu používateľských aplikácií. Veľa komponentov dodáva priamo prostredia SWING, v ktorom budeme vyvíjať naše prostredie. Oblasti prototypu a potrebné komponenty:

- Component toolbox – chceme dostupné moduly zobrazit' v stromovej alebo skupinovej štruktúre. (použitý komponent TreeView)
- Designer – plachta pre ukladanie komponentov pre náš projekt a následne zoskupovanie. (vlastná implementácia plachty)
- Code editor – textový editor s vyznačováním zdrojového kódu a podporou syntaktickej analýzy. (použitý RSyntaxTextArea s vlastnou implementáciou analýzy zdrojového kódu)
- Properties – tabuľkový editor vlastností. (použitý komponent vedúceho práce PropertyEditor)
- Console – plocha na farebný výpis textu. (použitý komponent TextView)
- Dokovací framowork – jednou z nami určených požiadaviek bola prispôsobivosť prostredia. Dokovací framework je spôsob prispôsobenia na, ktorý je dnes už väčšina programátorov zvyknutých. (použitý komponent DockingFrames)

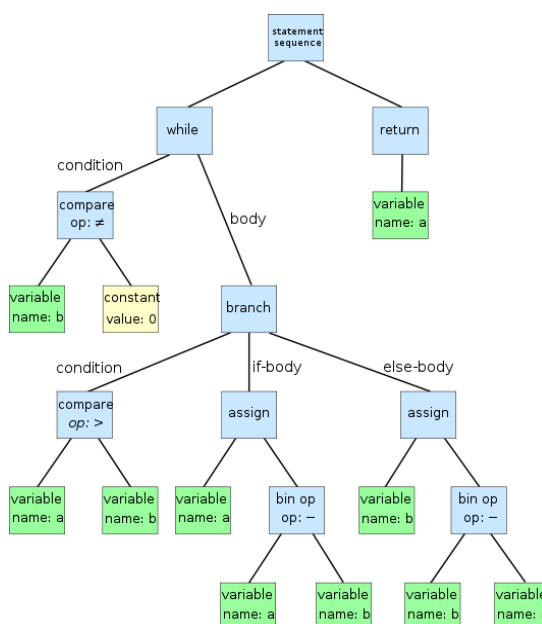


Obrázok: Nami implementované prostredie.

Aby bolo naše prostredie dobré potrebujeme minimalizovať, nadbytočné kroky používateľa k dosiahnutiu svojho cieľa. Preto sme sa potrebovali zaoberať kompiláciou priamo v našom prostredí. To sme docielili vďaka dostupného konzolového interface (CLI) od Arduino kompilátora. Textový výstup počas kompilácie sme nasmerovali do objektu Console aby bol programátor informovaný o stave kompilácie. Takto sme dokázali zachytiť celý priebeh tvorby projektu v našom prostredí od vytvorenia až po upload na zariadenie Arduino.

## 5.1 Abstraktný syntaktický strom

K vývojovým prostrediam neodmysliteľne patrí aj online analýza napísaného zdrojového kódu. Je to spôsob šetrenia času hľadania chýb z nepozornosti a zároveň príležitosť odporučiť programátorovi dostupné metódy práve keď ich potrebuje. Aby sme mohli tieto veci robiť jednoducho, potrebujeme porozumieť zdrojovému kódu. Na to slúžia abstraktné syntaktické stromy. Abstraktný syntaktický strom je stromová reprezentácia abstraktnej syntaktickej štruktúry zdrojového kódu v programovacom jazyku. Každý uzol stromu predstavuje konštrukt vyskytujúceho sa kódu. Abstraktná je pretože nepopisuje, každý detail. Nasledujúci obrázok predstavuje vizualizáciu abstraktného syntaktického stromu pre tento zdrojový kód.



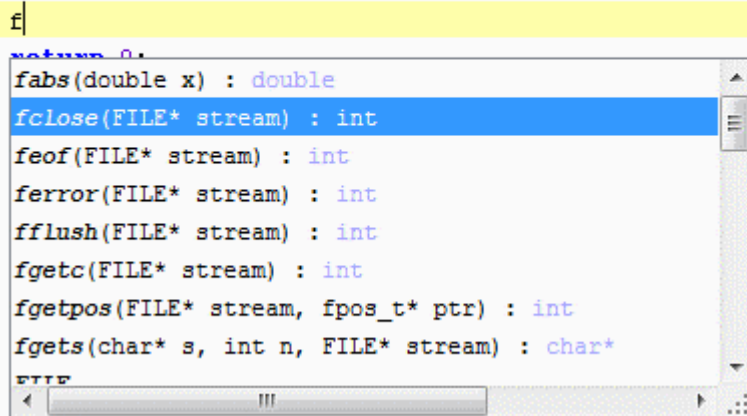
Obrázok: Vizualizácia abstraktného syntaktického stromu.

```
while b ≠ 0
  if a > b
    a := a - b
  else
    b := b - a
return a
```

Zdrojový kód: Algoritmus zobrazený v syntaktickom strome.

## 5.2 Automatické dokončovanie kódu

```
printf("Opening file: %s...\n", argv[0]);
FILE *f = fopen(argv[0], "r");
process(f);
```



Obrázok: Príklad automatického dokončovania kódu.

## 5.3 Statická analýza komponentov

V integrovanom prostredí chceme používateľa upozorňovať na bežné chyby. Jednou z chýb môže byť simultánne použitie toho istého pinu pre viac komponentov. Problém by nastal aj sa jeden komponent snaží čítať hodnotu pinu a iný nastavuje pin pre účely zapnutia len. V takých prípadoch potrebujeme upozorniť programátora, na pravdepodobne chybné nastavenie vlastností.

## 6 Analýza komponentov

Existuje množstvo komponentov, s ktorými sa stretávame každodenne. Či už s tlačidlom na ovládači, LED diódov alebo potenciometrom pri úprave hlasitosti v aute, a pod. Všetky tieto komponenty majú popis fungovania. Preto sa v práci pokúsime pripraviť balík základných komponentov. Na obrázku si môžeme pozrieť ako také komponenty vyzerajú.



Obrázok: Príklady rôznych elektronických komponentov

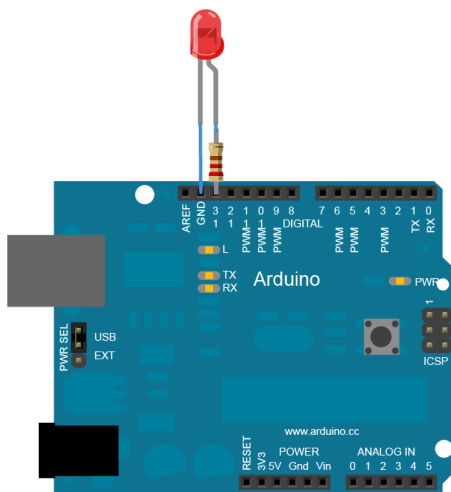
Typy komponentov (moduly) v našom riešení môžeme rozdeliť na dve kategórie: hardvérové a softvérové. Do prvej kategórie patria typy, ktoré bezprostredne vychádzajú z dostupného hardvéru. Príkladom pre tieto komponenty sú rôzne senzory, zobrazovacie displeje a pod. Medzi softvérové komponenty budeme zaradzovať tie, ktoré sa vykonávajú iba na procesore mikrokontroléra a pracujú iba so svojimi vlastnosťami.

## 6.1 Časovač (acp.common.timer)

Softvérový komponent časovač slúži na periodické spúšťanie vygenerovanie udalosti. Komponent je spúšťaný v intervale zadanom vo vlastnosti Interval a prvé vygenerovanie vieme odsunúť vlastnosťou InitialDelay. Je dôležité vedieť časovač ovládať, preto sme pre neho navrhli aplikačné rozhranie. Pomocou rozhrania ho vieme vypnúť (disable) a zapnúť (enable). Neodmysliteľnou súčasťou rozhrania sú getery a setery všetkých dostupných vlastností.

Vlastnosti	Udalosti	API
<ul style="list-style-type: none"><li>Interval</li><li>InitialDelay</li><li>Enabled</li></ul>	<ul style="list-style-type: none"><li>OnTick</li></ul>	<ul style="list-style-type: none"><li>.isEnabled()</li><li>.enable()</li><li>.disable()</li><li>.setInterval(int)</li><li>.getInterval()</li></ul>

## 6.2 Prepínač (acp.common.switch)



Obrázok: Príklad zapojenia LED použitého ako prepínač.

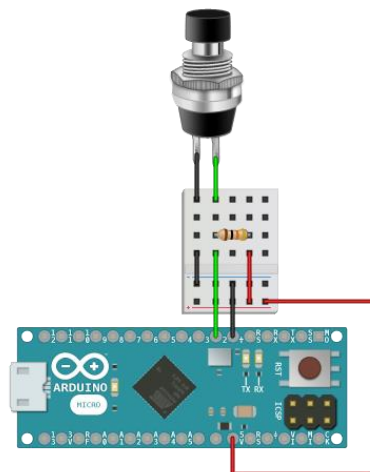
Prepínač je hardvérový komponent. Hlavnou úlohou tohto komponentu je pracovať s booleovskou funkciou, ktoré výstupný pin zapne alebo vypne. Pomocou tohto komponentu nevieme generovať žiadne udalosti, pretože nespracovávame žiaden vstupný signál. Hovoríme tiež pasívny typ komponentu. Preto je dôležité práve aplikačné rozhranie. Aplikačným rozhraním môžeme meniť aktuálny stav funkcie a tým ovládať



digitálny výstup zo zariadenia. Okrem rozhraní zapnutia (turnOn) a vypnutia (turnOff) sme sa rozhodli implementovať aj rozhranie zmeny vynútenej zmeny stavu (revert). Toto nám výstup vypne ako bol zapnutý, a naopak.

Vlastnosti	Udalosti	API
<ul style="list-style-type: none"> <li>Pin</li> <li>InitialState</li> <li>InvertedLogic</li> </ul>	Žiadne udalosti	<ul style="list-style-type: none"> <li>.isOn()</li> <li>.turnOn()</li> <li>.turnOff()</li> <li>.setState(bool)</li> <li>.confirmState()</li> <li>.revert()</li> </ul>

### 6.3 Digitálne čítanie vstupu (acp.common.digital\_input\_pin)

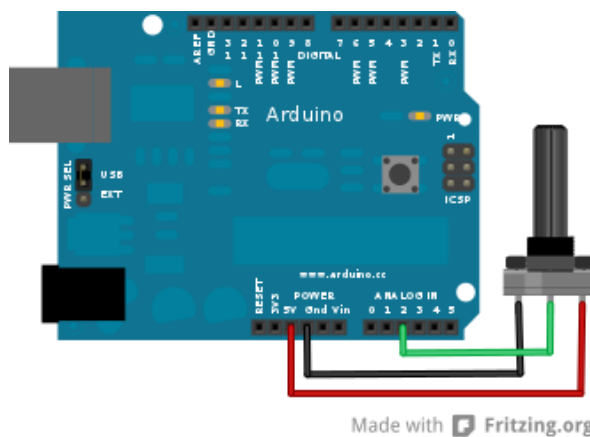


Obrázok: Príklad zapojenia tlačidla k zariadeniu.

Ďalším hardvérovým typom komponentu poskytuje digitálne čítanie vstupu. Tento komponent je opakom ku komponentu prepínača. Komponent v zadanom krátkom intervale (vlastnosť ReadInterval) načíta aktuálnu hodnotu a prevedie ju do aktuálneho stavu v booleovskej funkcii. Ak sa stav zmenil tak sa vygeneruje udalosť OnStateChanged. Aplikačné rozhranie obsahuje len získanie stavu aby sme mohli spracovať zmenu stavu na konkrétnu hodnotu.

Vlastnosti	Udalosti	API
<ul style="list-style-type: none"> <li>Pin</li> <li>InvertedLogic</li> <li>PullupResistor</li> <li>ReportOnState</li> <li>ReportOffState</li> <li>ReportCyclesCount</li> <li>ReadInterval</li> </ul>	<ul style="list-style-type: none"> <li>OnStateChanged</li> <li>OnStateReported</li> </ul>	<ul style="list-style-type: none"> <li>.getState()</li> </ul>

#### 6.4 Analógové čítanie vstupu (acp.common.analog\_input\_pin)



Obrázok: Príklad zapojenia analógového komponentu.

Arduino zariadenia nám okrem digitálneho vstupu poskytujú aj analógové čítanie. Analógové vstupu sú dobré pre presné merania hodnoty. Príkladom je potenciometer, ktorým vieme meniť napätie otočením. Tento komponent funguje rovnakým princípom ako predchádzajúci a v zadanom intervale sleduje aktuálnu hodnotu. Po zmene hodnoty vygeneruje udalosť `OnValueChanged`. Aktuálnu hodnotu získame pomocou aplikačného rozhrania metódou `getValue`.

Vlastnosti	Udalosti	API
<ul style="list-style-type: none"> <li>Pin</li> <li>ReadInterval</li> </ul>	<ul style="list-style-type: none"> <li>OnValueChanged</li> </ul>	<ul style="list-style-type: none"> <li>.getValue()</li> </ul>

## 7 Záver

V práci sme analyzovali sme existujúce riešenia a knižnice pre Arduino zariadenia. Väčšina dostupných riešení však bola riešená online a je náchylná na výpadky internetového spojenia. Dostupné offline riešenia často riešili len jednu úlohu, poskladanie jednotlivých riešení môže byť pre programátora náročné, pretože nemali jednotnú štruktúru. Preto sme navrhli vytvoriť riešenie, ktoré skombinuje komponentové programovanie s udalosťami riadeným programovaním. Po analýze hardvérových prostriedkov zariadení Arduino a plánovanému rozsahu nášho riešenia, sme boli obmedzovaný veľkosťou ROM na zariadení. Preto sme sa rozhodli využiť princíp generovania kódu, vďaka ktorému do konkrétnej aplikácie príde len malá použitá časť celého dostupného riešenia.

Arduino sa stalo populárne hlavne pre svoje integrované vývojové prostredie. To výrazne zjednodušilo prácu s elektronikou. Preto sme sa v ďalšej časti práci venovali vytvoreniu integrovaného vývojového prostredia práve pre naše riešenie. Vychádzali sme z otvoreného zdrojového kódu pre Arduino IDE, a niekoľkých ďalších open-source projektov. Výsledkom je prostredie, v ktorom potrebuje programátor minimálne zručnosti s technológiou XML a dokáže vytvoriť aplikáciu. Pri návrhu IDE sme sa inšpirovali dostupnými riešeniami pre javu a c#.

V poslednej časti sme sa venovali implementácii vzorových komponentov. Vytvorili sme 10+ typov komponentov, ktoré môžu programátori začať používať. Vzorovými komponentami sme sa snažili pokryť bežne používané softvérové aj hardvérové postupy.

## **Zoznam použitej literatúry**

- [1] Doukas, C. Building internet of things with the arduino. ISBN: 978-14-700-23430.
- [2] Schwartz, M. Internet of things with arduino cookbook. ISBN: 978-17-852-86582.
- [3] Waher, P. Learning internet of things. ISBN: 978-17-835-5353-2.
- [4] The Arduino projects book, Second reprint, May 2013
- [5] Shvets, A. Design Patterns Explained Simply, ISBN:
- [6] Maurizio Gabbrielli, Simone Martini Programming Languages: Principles and Paradigms, ISBN: 9781848829145

## Príloha č. 1 – Demo projekt

Motivačným príkladom tejto práce je zariadenie zložená z tlačidla a led diódy. Led dióda má 1 sekundu svietiť a následne na 1 sekundu zhasnúť. Tlačidlo simuluje play/pause tlačidlo, na pozastavenie blikania led diódy. Nasledujúce súbory ukazujú implementáciu tohto príkladu v riešení navrhnutom touto prácou.

Nech priečinok Blink obsahuje nasledujúce súbory Blink.xml a Blink.ino.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project platform="ArduinoUno">
  <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">onBlink</event>
      </events>
    </component>
    <component>
      <name>led</name>
      <type>acp.common.switch</type>
      <properties>
        <property name="Pin">13</property>
        <property name="InitialState">true</property>
      </properties>
    </component>
    <component>
      <name>button</name>
      <type>acp.common.digital_input_pin</type>
      <properties>
        <property name="ReadInterval">30</property>
        <property name="ReportOnState">false</property>
        <property name="Pin">2</property>
        <property name="ReportCyclesCount">33</property>
        <property name="PullupResistor">true</property>
      </properties>
      <events>
        <event name="OnStateChanged">buttonClicked</event>
      </events>
    </component>
  </components>
</project>
```

Zdrojový kód – Blink.xml

```
//-----  
// Includes required to build the sketch (including ext. dependencies)  
#include <Blink.h>  
//-----  
  
void onBlink() {  
    led.revert();  
}  
  
void buttonClicked() {  
    if(button.getState() == HIGH) {  
        if(blinkTimer.isEnabled()) {  
            // Pause  
            blinkTimer.disable();  
        } else {  
            // Play  
            blinkTimer.enable();  
        }  
    }  
}
```

Zdrojový kód – Blink.ino