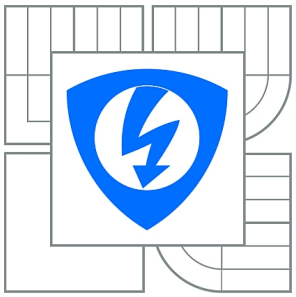
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ŘÍDICÍ JEDNOTKA VÝROBNÍ LINKY

PRODUCTION LINE CONTROL UNIT

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

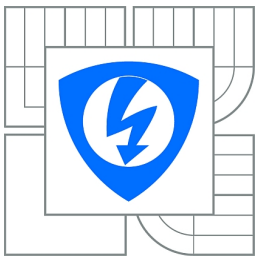
AUTOR PRÁCE — PAVOL VARGOVČÍK

AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR PETYOVSKÝ

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Pavol Vargovčík	***ID:***	158260
Ročník:	3	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Řídicí jednotka výrobní linky

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je nahradit původní řídicí jednotku výrobní linky realizovanou jako PLC pomocí modulu Raspberry Pi.

1. Seznamte se s požadavky na řídicí jednotku dle požadavků zadavatele. Definujte obecné požadavky výrobního procesu na základě několika typizovaných stanovišť konkrétních výrobních linek.
2. Seznamte se s HW možnostmi a programováním modulu Raspberry Pi.
3. Definujte ovládané parametry, měřené hodnoty a interakci s operátorem na výrobních stanovištích.
4. Navrhněte a realizujte HW řešení řídicí jednotky společně s modulem vstupů a výstupů.
5. Zvolte a implementujte vhodný skriptovací jazyk pro ovládání a sběr dat z jednotlivých výrobních stanovišť do modulu Raspberry Pi.
6. Seznamte se s požadavky kladenými na řídicí jednotku výrobní linky z nadřazených systémů.
7. Na straně řídicí jednotky implementujte SW rozhraní pro komunikaci s nadřazenými systémy.
8. Proveďte instalaci a testovací provoz nově navržené řídicí jednotky do výrobní linky zadavatele.
9. Zhodnoťte dosažené výsledky. Navrhněte další možná vylepšení.

DOPORUČENÁ LITERATURA:

[1] RICHARDSON, L., AMUNDSEN, M., RUBY, S.: RESTful Web APIs, O'Reilly Media, 2013, 408 s, ISBN 978-1-4493-5806-8.
[2] LUTZ, M.: Programming Python 4th ed., O'Reilly Media, 2006, 1628 s, ISBN 978-0596158101.

Termín zadání:	9.2.2015	***Termín odevzdání:***	25.5.2015

Vedoucí práce: Ing. Petr Petyovský
Konzultanti bakalářské práce: Ing. Josef Nevrlý

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

ABSTRAKT

V tejto práci som sa venoval vývoju riadiacej jednotky výrobnej linky. Pre jej implementáciu som zvolil vstavané zariadenie Raspberry Pi. Riadiaca jednotka komunikuje s centrálnym administrátorským systémom, ktorý prideľuje jednotlivým riadiacim jednotkám programy. Tieto programy popisujú interakciu s periférnymi zariadeniami, ako aj komunikáciu s databázovým systémom dohľadateľnosti (Traceability).

KLÍČOVÁ SLOVA

Kontrola výrobného procesu, Raspberry Pi, Linux, ARM, vstavaný systém, Blockly

ABSTRACT

This work is about development of a control unit for use in a production line. The control unit is implemented in an embedded device Raspberry Pi. It communicates with central administrating system, which assigns programs to particular control units. These programs describe interaction with peripherals of the control unit and with database system of traceability.

KEYWORDS

Process control, Raspberry Pi, Linux, ARM, embedded system, Blockly

VARGOVČÍK, Pavol *Řídicí jednotka výrobní linky*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřící techniky, 2015. s. Vedoucí práce byl Ing. Petr Petyovský

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Řídicí jednotka výrobní linky“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

Automatizácia výrobného procesu s využitím výpočtovej techniky je významnou súčasťou priemyslu dvadsiateho a dvadsiateho prvého storočia. Zvyšuje produktivitu a spoľahlivosť výroby, znižuje náklady. Prístroje na výrobnej linke sú riadené programovateľným počítačom—riadiacou jednotkou, účasť človeka ale nie je vylúčená.

Väčšinové zastúpenie medzi dnešnými riadiacimi jednotkami majú riadiace jednotky typu Programmable Logic Controller (PLC). Vlastnosti a možnosti PLC sú vhodné pre riadenie väčšiny automatizovaných výrobných liniek, nevýhodou však je vysoká cena a taktiež nutnosť použitia výrobcom pripravených nástrojov pre programovanie a konfiguráciu PLC, ktoré neposkytujú ani zďaleka takú flexibilitu, akú by poskytla riadiaca jednotka s operačným systémom s otvoreným zdrojovým kódom.

Témou tejto práce je realizácia práve takejto riadiacej jednotky. Na projekte pracujem od júna 2014, keď mi ho zadala firma ALPS Electric Czech s.r.o. Ako jadro riadiacej jednotky zadavateľ zvolil počítač Raspberry Pi s operačným systémom Linux. Ako hlavný programovací jazyk pre projekt bol zvolený jazyk *Python*. Riadiaca jednotka mala interagovať s výrobnou linkou pomocou digitálnych vstupov a výstupov, ktoré mali byť kompatibilné s riadiacimi jednotkami typu PLC. Stav vstupov a výstupov mal byť vizuálne indikovaný pre účely ladenia. Okrem toho mali byť k riadiacej jednotke pripojené ďalšie periférne zariadenia: grafický terminál s ovládacími tlačidlami a čítačka čiarových kódov.

Riadiaca jednotka mala taktiež komunikovať s nadriadenými systémami pomocou rozhrania Ethernet. Nadriadené systémy pozostávajú zo systému administrátorského a interného databázového systému firmy, s ktorým majú riadiace jednotky komunikovať počas vykonávania svojich programov.

Administrátorský nadriadený systém umožňuje vytvárať riadiace programy a priradzovať ich riadiacim jednotkám na základe pracoviska, na ktorom sa nachádzajú, a typu produktu vyrábaného na linke, ku ktorej pracovisko prislúcha. Jeho vedľajšou úlohou je zber údajov o chybách, ktoré sa objavili na riadiacich jednotkách počas výroby. Táto úloha má význam hlavne pre vývoj a servis. Ďalšou vedľajšou úlohou je synchronizácia času v riadiacich jednotkách s časom interného databázového systému firmy. Implementácia administrátorského systému a taktiež jazyka pre vytváranie riadiacich programov bola súčasťou zadania.

Cieľom tejto práce je teda hardvérová a softvérová implementácia riadiacej jednotky, návrh a realizácia programovacieho jazyka pre riadiacu jednotku, implementácia administrátorského servera a na záver testovanie celého systému.

1 DEFINÍCIA VŠEOBECNÝCH POŽIADAVIEK PRE RIADIACU JEDNOTKU

Od zadavateľa boli špecifikované následujúce všeobecné požiadavky, ktoré mali byť dodržané pri realizácii projektu:

Využitie techník zaužívaných vo firme
: Základom riadiacej jednotky má byť Raspberry Pi s operačným systémom Linux. Základnym programovacím jazykom má byť Python [4].

Nízka cena, Efektivita výroby
: Súčiastky zariadenia musia byť dostupné na trhu a zásahy pre ich úpravu musia byť minimálne. Z tejto požiadavky vyplýva obdĺžnikový tvar zariadenia s konektormi umiestnenými na dvoch protiľahlých stranách, aby pri použití krytov s odnímateľnými čelami stačilo vyfrézovať otvory do čiel. Taktiež z nej vyplýva to, že po nahratí obrazu do pamäťovej karty Raspberry Pi už netreba programovať ani konfigurovať žiadne iné zariadenia, pretože sa o to postará Raspberry Pi po prvom spustení.

Programovateľnosť
: Riadiaca jednotka nemôže mať pevný program, naopak, súčasťou inicializácie riadiacej jednotky po každom štarte musí byť vyžiadanie programu od centrálneho administrátorského systému. Uživateľ musí byť schopný manuálne prerušiť bežiaci program a vrátiť sa do inicializácie.

Jednoduché programovanie
: Programátor musí byť schopný poňať programovací jazyk bez väčšej teoretickej prípravy. Jazyk má byť veľmi jednoduchý, interaktívny a má predchádzať syntaktickým chybám.

Dohľadateľnosť
: Riadiaca jednotka musí vedieť komunikovať s databázovým serverom *dohľadateľnosti (Traceability)*[1]. Dohľadateľnosť musí byť implicitne zahrnutá v príkazoch programovacieho jazyka pre riadiacu jednotku.

Interakcia s hardvérom
: Riadiaca jednotka musí byť schopná obsluhovať tieto periférne zariadenia:
  - terminál pre uživateľské rozhranie s displejom a tlačidlami
  - čítačka čiarových kódov
  - rozhranie digitálnych vstupov a výstupov kompatibilných so vstupmi a výstupmi PLC (linky, ktoré budú riadené riadiacou jednotkou popísanou v tejto práci, sú momentálne riadené pomocou PLC)

Z požiadaviek vyplýva, že práca sa dotýka viacerých oblastí: návrh elektroniky, programovanie interakcie Raspberry Pi s ostatným hardvérom, nízkoúrovňové programovanie grafického uživateľského rozhrania, programovanie sieťovej komunikácie Raspberry Pi s nadriadenými systémami, návrh a implementácia administrátorského systému, návrh a implementácia špecializovaného programovacieho jazyka.

2 VÝVOJOVÝ MODUL RASPBERRY PI A JEHO HARDVÉROVÉ MOŽNOSTI

Raspberry Pi [9] je počítač o veľkosti kreditnej karty vyvinutý vzdelávacou charitou Raspberry Pi Foundation. Zariadenie má verejne dostupnú dokumentáciu k hardvéru a je preňho prispôsobených niekoľko operačných systémov (Linuxové distribúcie Debian, Fedora a Arch, RISC OS), ktoré majú taktiež otvorený zdrojový kód.

Aktuálna verzia zariadenia, Raspberry Pi 2 Model B, sa dá zakúpiť v cene od 850Kč. Podpora Linuxu výrazne rozširuje softvérové možnosti zariadenia. Linux poskytuje vývojárovi obrovské množstvo technológií a umožňuje mu vytvoriť z Raspberry Pi zariadenie podľa jeho predstáv. Hranice možností sú kladené jedine hardvérovými obmedzeniami.

2.1 Hardvérové možnosti

Srdcom Raspberry Pi B+ je mikrokontrolér BCM2835, ktorý sám pozostáva z dvoch jadier, kde jedno je hlavné aplikačné jadro ARM1176JZF-S, druhé je multimediálny koprocesor VideoCore IV®. Multimediálnemu koprocesoru nebudem v tejto práci venovať pozornosť, keďže multimédiá práca nevyužíva. Aplikačné jadro je postavené na architektúre ARM (to však pre projekt zatiaľ nebolo dôležité, keďže nízkoúrovňová interakcia s hardvérom je už vyriešená v jadre Linuxu), beží na 700MHz a disponuje operačnou pamäťou o veľkosti 512MiB.

Štandardné napájanie zariadenia je cez micro-USB konektor. Od verzie B+ je doska osadená 40-vývodovou kolíkovou lištou, na ktorej je 27 General Purpose Input/Output (GPIO) vývodov. Na ostatných vývodoch je vyvedené napájanie a vývody *ID_SD* a *ID_SC*, ktoré slúžia na automatické nahrávanie firmvéru do Raspberry Pi B+ z externej dosky. Niektoré GPIO sa dajú nastaviť ako vývody pre rozhrania Universal Asynchronous Receiver/Transmitter (UART), Inter-Integrated Circuit (I^2C) a Serial Peripheral Interface (SPI).

Ďalšími perifériami sú štyri Universal Serial Bus (USB) porty, rozhranie ethernet, rozhrania pre pripojenie kamery a displeja, High Definition Multimedia Interface (HDMI) a výstupný audio jack.

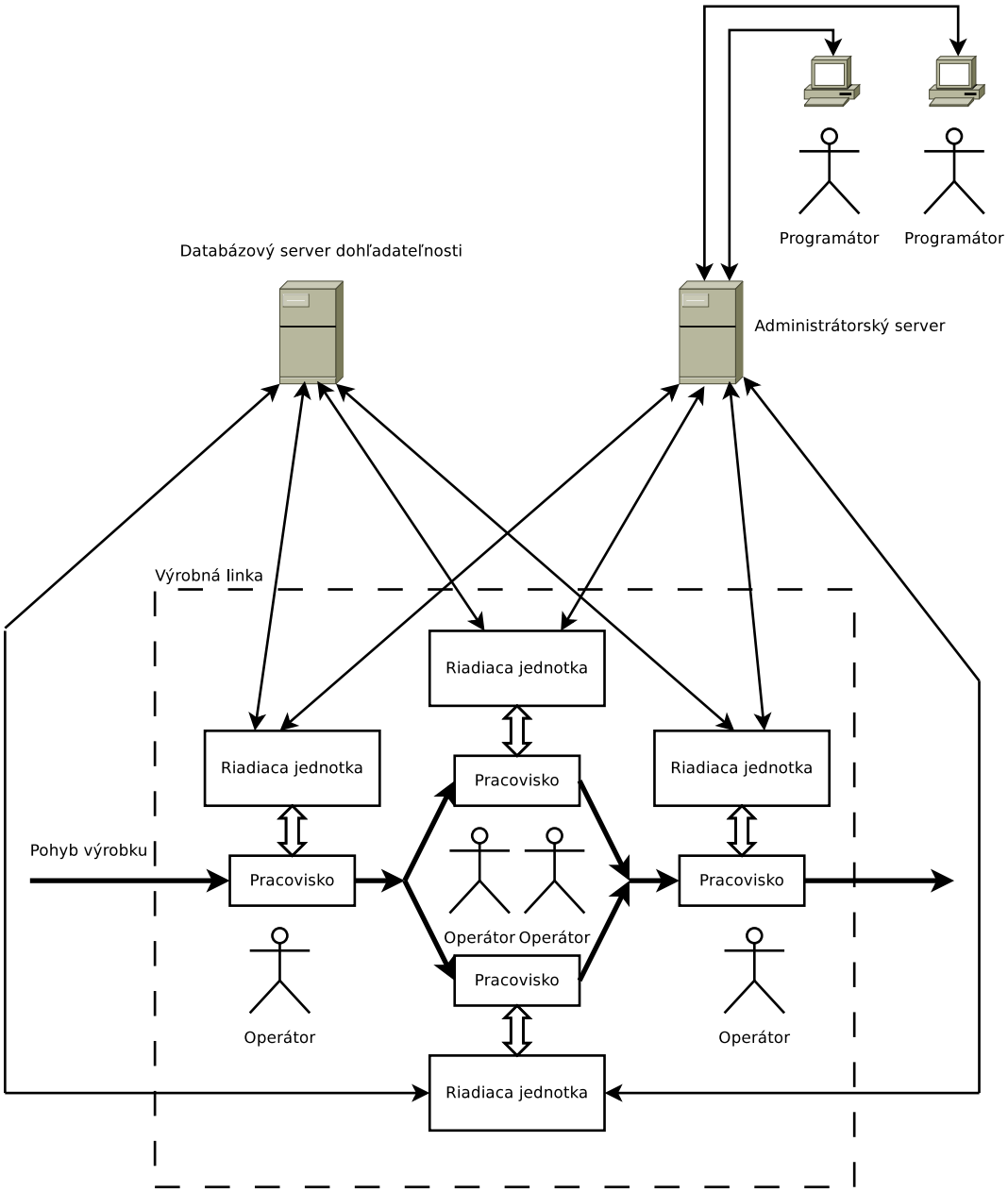
3 ČINNOSŤ RIADIACEJ JEDNOTKY

Riadiaca jednotka riadi jedno pracovisko výrobnej linky. Po zapnutí riadiaca jednotka vyžiada od operátora aby načítal čiarový kód pracoviska, na ktorom sa jednotka nachádza. Následne sa na displeji zobrazí výberové menu, kde si operátor volí z možnosti načítať čiarový kód produktu, ktorý sa ide vyrábať, alebo použiť posledný čiarový kód, ktorý už načítal iný operátor na tej istej výrobnej linke. Pri voľbe prvej možnosti sa čiarový kód produktu odošle na administrátorský server a vyžiada sa od servera na základe čiarového kódu pracoviska a čiarového kódu produktu program (tzv. Workflow (WF)), ktorý má riadiaca jednotka vykonávať. Pri voľbe druhej možnosti sa prvý krok vynechá a rovno sa vyžiada WF.

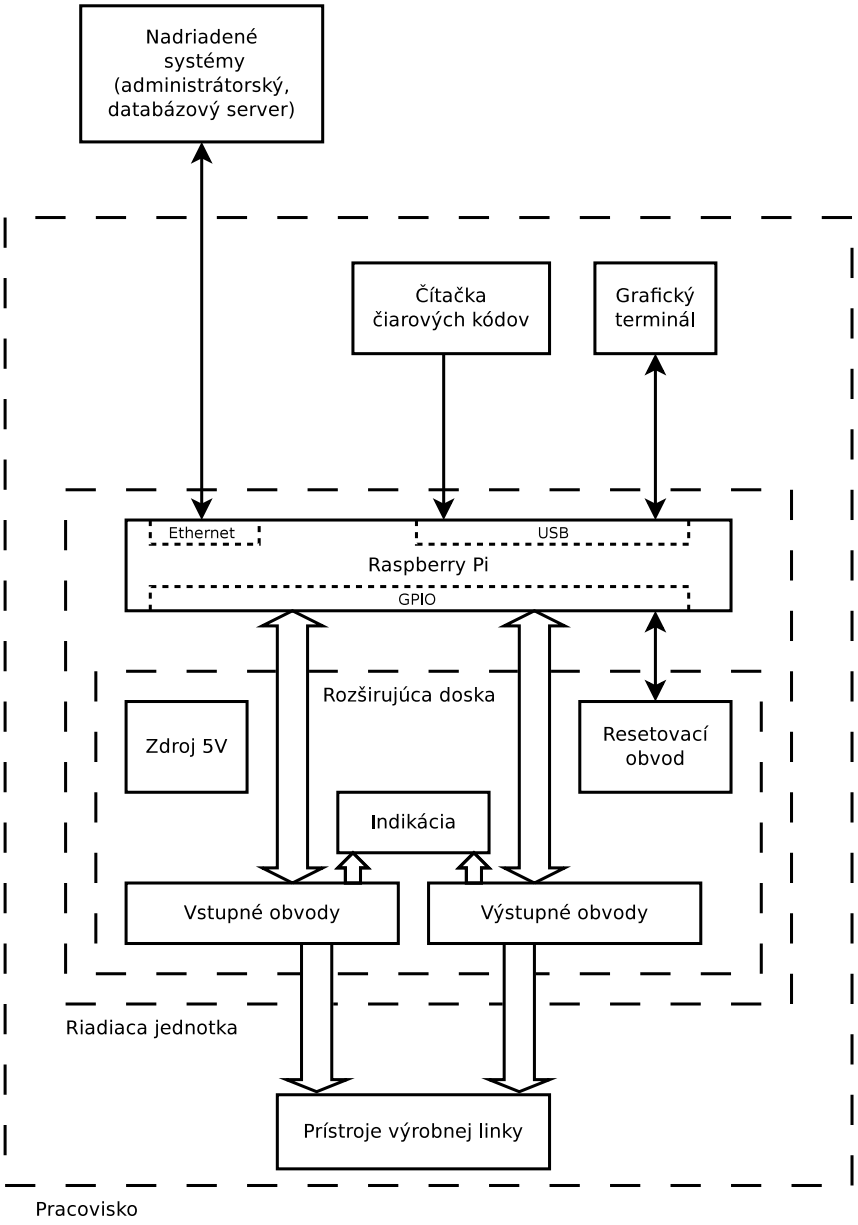
S databázovým serverom dohľadateľnosti komunikuje riadiaca linka pri plnení jednotlivých príkazov WF. Zasiela mu údaje o činnostiach vykonaných na výrobku a overuje, či sú v poriadku všetky súčiastky a nástroje, ktoré sa pri výrobnom procese používajú. Nástroj (alebo súčiastka) je v poriadku, ak je evidovaný v databáze a databáza neobsahuje záznam o jeho neopravenej poruche.

Interakcia s databázovým serverom zatiaľ nie je implementovaná, pretože požiadavky na ňu ešte nie sú úplne jasné, špecifikácie sa stále navyšujú. Pravdepodobne pôjde o nízkoúrovňovú komunikáciu na úrovni TCP protokolu s niekoľkými databázovými servermi naraz.

Obr. 3.1: Bloková schéma zariadení vo výrobnom procese



Obr. 3.2: Bloková schéma riadiacej jednotky a jej periférií



Vykonávanie WF môže byť prerušené dlhým podržaním stredného tlačidla na termináli. Vtedy sa na displeji zobrazí prerušovacie menu, kde si môže operátor vybrať z výnimočných úkonov, ako napríklad zmena operátora, výmena nástroja, zrušenie Work Order (WO).

Obr. 3.3: Prerušovacie menu

3.1 Program vykonávaný riadiacou jednotkou — Workflow

Workflow (WF) je program, ktorý môžu technici spravovať (vytvárať, upravovať, mazať) po pripojení na administrátorský server. WF je uložený v textovej forme v databáze servera. Implementácia WF je popísaná v sekcii Programovací jazyk pre riadiacu jednotku, v tejto podsekcii sa pokúsim slovne popísať činnosť riadiacej jednotky pri vykonávaní vzorového WF. Pre lepšiu predstavu vložím k činnostiam, ktoré sa zobrazujú na displeji fotografie displeja (vždy prvý krát keď sa vykonáva činnosť daného typu).

3.1.1 init

Procedúra **init** sa vykoná na začiatku programu.

1. Operátor zadá aktuálnu hodnotu atmosférického tlaku.

Obr. 3.4: Voľba číselnej hodnoty

2. Riadiaca jednotka pošle zadaný parameter databázovému serveru.
3. Operátor načíta čiarový kód prípravku.

Obr. 3.5: Jednoduchá výzva

4. Riadiaca jednotka overí s databázovým serverom, či sa môže nástroj použiť (tzn. nachádza sa v databáze a všetky akcie na ňom vykonané dopadli výsledkom OK).
5. Operátor načíta svoj čiarový kód.
6. Riadiaca jednotka pošle zadaný parameter databázovému serveru.
7. Operátor potvrdí pripravenosť pracoviska.

Obr. 3.6: Výzva k výberu možnosti OK alebo NG

3.1.2 run

Procedúra **run** sa vykonáva cyklicky, pokiaľ ju operátor manuálne nepreruší.

1. Riadiaca jednotka počká na uvoľnenie montážneho prípravku. Koncový spínač je pripojený na jednom zo vstupných vývodov riadiacej jednotky.
2. Operátor načíta čiarový kód dielu.
3. Riadiaca jednotka overí s databázovým serverom, či sa diel môže použiť a či bol daný diel spracovaný na predchádzajúcej stanici. V linke môže byť viacero paralelných staníc, takže sa niekedy musí overovať, či bol diel spracovaný na aspoň jednej z daných staníc.
4. Riadiaca jednotka počká, kým operátor nevloží diel do prípravku.
5. Operátor načíta postupne čiarové kódy ďalších dielov.
6. Riadiaca jednotka overí s databázovým serverom, či sa môžu diely použiť.
7. Riadiaca jednotka počká, kým operátor nezavrie prípravok. Uzavretie prípravku je signalizované na jednom zo vstupných vývodov riadiacej jednotky.
8. Riadiaca jednotka uzamkne prípravok. Akčný člen zámku je pripojený na jeden z výstupných vývodov jednotky.
9. Riadiaca jednotka počká, kým nie sú konektory všetkych dielov dobre zastrčené. Detektory chybného zastrčenia konektorov sú pripojené na vstupných vývodoch riadiacej jednotky.
10. Riadiaca jednotka odomkne prípravok.
11. Riadiaca jednotka počká, kým operátor neotvorí prípravok.
12. Operátor potvrdí, že produkt je v poriadku.
13. Riadiaca jednotka uloží potvrdenie do databázy.

3.1.3 error

Procedúra **error** sa vykoná, ak v procedúre **run** nastala chyba. Chybou je napríklad, ak operátor pri výzve pre potvrdenie stavu súčasti alebo pracoviska zvolí možnosť No Good (NG), čo je opak k výrazu OK, alebo ak databázový server vráti odpoveď NG, čo znamená, že niečo s danou súčasťou nie je v poriadku. Ak nastane chyba aj v procedúre **error**, procedúra sa jednoducho zopakuje.

1. Riadiaca jednotka odomkne prípravok.
2. Riadiaca jednotka pošle databázovému serveru identifikátory operátora, nástroja a produktu, pri ktorom chyba nastala.
3. Operátor vloží diel do krabice NG výrobkov.

3.1.4 at box change

Ak programátor definuje túto procedúru, pre procedúru **run** sa vytvorí počítadlo pre vyrobené kusy (predpokladá sa, že procedúra **run** produkuje jeden kus výrobku). Ak sa počítadlo (kusovník) naplní, zavolá sa procedúra **at box change**. Táto procedúra zatiaľ nie je implementovaná, pretože ešte bude treba doplniť jej špecifikáciu o to, kedy a kde sa má kusovník zobrazovať. Pripravil som však už dopredu v knižnici grafického terminálu metódy pre zobrazenie kusovníka.

Obr. 3.7: Kusovník

4 NÁVRH A REALIZÁCIA HARDVÉRU

Aby bola nová riadiaca jednotka aspoň čiastočne hardvérovo kompatibilná s PLC, bolo potrebné k Raspberry Pi navrhnúť rozširujúcu dosku plošných spojov (DPS). Okrem prispôsobenia, indikácie a rozšírenia vstupov a výstupov sa na doske nachádza aj resetovací obvod a step-down napäťový menič, ktorý prevádza vstupné 24V napájanie (potrebné pre vstupno-výstupné obvody) na 5V napájanie pre Raspberry Pi. Resetovací obvod obstaráva odpojenie výstupov, keď je Raspberry Pi v stave vypnutia alebo počas zavádzania a inicializácie operačného systému pred spustením štartovacieho skriptu[1] a taktiež obsluhuje resetovacie tlačidlo.

4.1 Spínanie výstupov

Akčné členy spínané logickými výstupmi riadiacej jednotky sú napájané napätím 24V. Pre spínanie logických výstupov som použil obvod ULN2803, čo je sieť ôsmych spínačov riešených ako tranzistory v Darlingtonovom zapojení. Pre ochranu tranzistorov pri indukčných záťažiach sú od výstupov vedené ochranné diódy, ktorých katódy sú prepojené a vyvedené na jednom spoločnom vývode, ktorý je pre správnu funkciu potrebné pripojiť na napätie, ktorým je napájaná záťaž, čiže na spomínaných 24V.

Vstupy ULN2803 môžu byť ovládané napäťovými úrovňami 0V/3.3V, ale Raspberry Pi nemá dostatočný počet GPIO pre pripojnie všetkych vstupov a výstupov riadiacej jednotky, preto sú výstupy rozšírené pomocou posuvných registrov 74HC595. Je to výstupný posuvný register ovládaný tromi signálmi *DATA*, *SCLK* a *RCLK*. Viac informácií o tomto integrovanom obvode je uvedených v literatúre [2]. Pri jeho voľbe som považoval za dôležité tri veci:

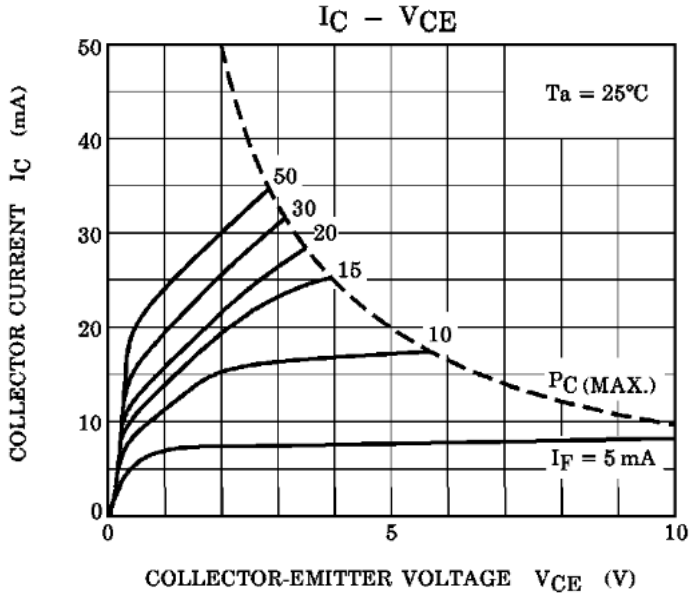
1. Hradlá je možné reťaziť a tým pádom za tri GPIO zabrané z Raspberry Pi sa dá získať v podstate ľubovoľný počet výstupov.
2. Ak prepojíme *DATA* so signálom *MOSI*, *SCLK* so signálom *SPI_SCK* a signál *RCLK* použijeme ako $\overline{CS}$, môžeme pre ovládanie hradiel použiť SPI.
3. Maximálny výstupný prúd je ±35mA, čo postačuje s veľkou rezervou pre napájanie indikačnej Light-emitting Diode (LED) a ovládanie vstupu ULN2803.

[1] štartovací skript je súbor `/etc/rc.local` a má za úlohu spustiť hlavný program riadiacej jednotky

4.2 Ochrana vstupov

Vstupy riadiacej jednotky majú indikovať zopnutie na zem. Sú oddelené od Raspberry Pi jednoduchými optočlenmi s jednou LED a jedným tranzistorom. Použil som optočlenové siete TLP281-4. Svietivým diódam v optočlenoch som predradil rezistory o hodnote 2.2kΩ, takže pri napájaciom napätí 24V tečie diódou prúd približne 10mA, čo by malo pri výstupnom prúde 10mA (prúd tečúci indikačnou LED) vytvoriť na výstupe tranzistora úbytok napätia dostatočne malý na to, aby ho vstupné obvody Raspberry Pi rozpoznali ako nízku úroveň. Pri rozopnutom stave svietivou diódou optočlena netečie prúd, tranzistor je teda uzavretý a na vstupe Raspberry Pi je vysoká úroveň napätia 3.3V.

Obr. 4.1: Prevodová charakteristika TLP281-4 [3]



Pre vstupy bolo na Raspberry Pi dostatok vývodov, preto nie sú rozšírené posuvnými registrami tak, ako je to pri výstupoch. Keby však bolo treba viac vstupov, dali by sa pripojiť rozšírujúce posuvné registre 74HC165 na tie isté signály, kam sú pripojené rozšírujúce posuvné registre výstupov.

4.3 Indikácia digitálnych úrovní

Pre účely ladenia je vhodné opatriť digitálne vstupy a výstupy riadiacej jednotky indikačnými LED. Môžu pomôcť programátorovi určiť, či sa chyba nachádza v hardvéri alebo v softvéri.

Pre indikáciu som zvolil LED s úzkym vyžarovacím uhlom (15°) AVAGO TECHNOLOGIES ASMT-BG20-AS000. Empiricky som zistil, že pre dostatočné rozsvietenie stačí 10mA a keďže sú napájané napätím 3.3V a je na nich úbytok 2V, predradil som im rezistory o hodnote 130Ω.

LED som umiestnil do dvoch radov — jeden pre vstupy, druhy pre výstupy. Rozostupy medzi nimi sú 4mm.

4.4 Napájanie

Kvôli napájaniu menej náročných akčných členov priamo z konektora riadiacej jednotky a kvôli ochranným diódam v obvodoch ULN2803 (viď. Spínanie výstupov) bolo treba napájanie 24V. Raspberry Pi však vyžaduje napájanie 5V. Taktiež je potrebné 3.3V napájanie, pre GPIO Raspberry Pi, túto napäťovú úroveň však poskytuje samotné Raspberry Pi na jednom z vývodov GPIO konektora. Pre vytvorenie 5V napäťovej vetvy som použil hotové riešenie, step-down napäťový menič DSN2596.

4.5 Resetovací obvod

Po odskúšaní prvej verzie hardvéru 0.1 sme zistili, že je nevyhnutné pridať na dosku resetovací obvod, ktorého základnou úlohou by bolo odpojiť výstupy keď je Raspberry Pi vo vypnutom stave (shutdown) alebo v stave, kým Raspberry Pi nespustí program riadiacej jednotky, a obstarávať resetovacie tlačidlo, ktorým by sa dalo Raspberry Pi vypínať, zapínať a resetovať. Menej dôležitou funkciou resetovacieho obvodu je šetriť energiu odpojením LED v optočlenoch pri vypnutom stave.

Hneď od začiatku nám napadla myšlienka použiť pre riadenie resetovacieho obvodu ATtiny (malý osemvývodový mikroprocesor s architektúrou *Atmel AVR*). Problémom tohto nápadu bolo, že by pri sériovej výrobe riadiacej jednotky pribudol ďalší výrobný krok — programovanie mikroprocesora. Taktiež bolo otázne, či by nebola cena nižšia, keby bola požadovaná jednoduchá činnosť resetovacieho obvodu implementovaná diskrétnymi súčiastkami a logickými hradlami.

Navrhol som teda jednoduchý riadiaci obvod. Po privedení napájania a po stlačení resetovacieho tlačidla sa obvod uviedol do stavu čakania na spustenie programu riadiacej jednotky, tento stav bol monostabilný a bol riešený časovacím obvodom NE555. Po monostabilnom stave sa obvod dostal do stavu zapnutia a pred vypnutím dalo Raspberry Pi resetovaciemu obvodu signál, ktorý ho dostal do stavu vypnutia. Pre udržanie stavu vypnutia a zapnutia resetovací obvod obsahoval RS preklápací obvod vystavaný pomocou hradiel NAND. Schéma tejto verzie resetovacieho obvodu je uvedená v prílohe .

Obvod bol funkčný, ale vyšlo najavo, že ATtiny by bola lacnejšia a flexibilnejšia. Jedinou nevýhodou by bolo jej programovanie. Zistili sme však, že kontribútori projektu *kcuzner/avrdude* nám už pripravili hotové riešenie pre programovanie mikroprocesorov cez In-System Programming (ISP) priamo z Raspberry Pi. Rozhodli sme sa teda použiť ATtiny13 ako resetovací obvod.

Keď sme sa už rozhodli pre ATtiny, naskytla sa možnosť väčšej flexibility pomocou obojstrannej komunikácie s Raspberry Pi. ATtiny má v ISP móde päť využiteľných GPIO[2]. Jeden výstup bolo treba pre odpájanie vstupov a výstupov riadiacej jednotky, ďalší výstup pre resetovanie Raspberry Pi, jeden vstup bolo treba pre resetovacie tlačidlo. Ostali dva GPIO, ktoré sme využili pre obojsmernú bit-banging komunikáciu s Raspberry Pi.

Aktuálna verzia programu pre AtTiny je zdokumentovaná v prílohe 3.

4.6 Realizácia hardvéru

Doska plošných spojov je navrhnutá rozmerovo pre krabičku HAMMOND 1455L1202. Konektory sú rozmiestnené tak, aby sa Raspberry Pi dalo zasunúť priamo do dosky.

Hardvér prešiel od začiatku vývoja niekoľkými verziami. Verzia 0.1 mala nie veľmi prehľadné rozmiestnenie LED a ich zapojenie bolo iné (boli napájané z 24V). Taktiež nemala resetovací obvod a bola navrhnutá do väčšej krabičky, keďže museli byť rezistory predradené LED väčších rozmerov kvôli 24V napájaniu.

Verzia 0.2 bola doplnená o resetovací obvod, LED boli napájané 3.3V a boli zoradené do jednej prehľadnej línie. Do krabičky boli vyvrtané diery, cez ktoré mali LED svietiť, ale boli príliš hlboko (asi pol centimetra), takže ich bolo dobre vidno iba pri kolmom pohľade na krabičku. Diery boli zalepené difúznou páskou, aby nebolo treba kolmý pohľad, LED však boli príliš blízko pri sebe a mali príliš veľký vyžarovací uhol, takže boli presluchy medzi nimi priveľké na to, aby sa dalo s riadiacou jednotkou pohodlne pracovať. Jednou z možností bolo použitie svetelných vodičov, tie sú však príliš drahé a ich zasúvanie do krabičky by spomaľovalo výrobu riadiacej jednotky. Preto bolo treba použiť LED z menším vyžarovacím uhlom a navrhnúť novú dosku, kde by boli LED ďalej od seba.

Verzie 0.1 a 0.2 som navrhoval v softvéri *CadSoft Eagle*, verziu 0.3 som sa rozhodol navrhnúť v softvéri *KiCad EDA*, čím som celý projekt postavil na softvéri s otvoreným zdrojovým kódom. Pre návrh dosky pre verziu 0.3 som použil *autorouter FreeRouting* [17]. Plánovaná verzia 0.4 už bude navrhnutá do prispôsobenej krabičky, napäťový menič (viď 4.4 Napájanie) nebude použitý externý ale bude osadený

[2] *PB5* sa používa ako *RESET* pri ISP programovaní

priamo na doske a bude naviac možné k riadiacej jednotke pripojiť expanzný modul pre ďalšie vstupy a výstupy.

Aktuálna verzia schémy aj návrhu dosky je uvedená v prílohe 1. Dosky plošných spojov boli vyrobené externou firmou.

4.6.1 Postup oživenia modulu Raspberry Pi

V tomto oddieli popíšem, ako som postupoval s Raspberry Pi, aby som ho nakonfiguroval pre potreby projektu a spustil na ňom jednoduchý program. Oddiel sa dá použiť aj ako návod pre prácu s Raspberry Pi, preto budem postup písať podrobnejšie, nebudem v ňom však riešiť žiadne alternatívne situácie. U čitateľa sa taktiež predpokladajú najzákladnejšie vedomosti o jazyku *Python* [4] a práci v Linuxe [11].

Nahranie obrazu operačného systému

Prvou úlohou bolo nahrať operačný systém. Zvolil som *Raspbian*, čo je Debian upravený pre Raspberry Pi. Obraz systému je dostupný na hlavnej stránke Raspberry Pi[3]. Na SD kartu, ktorá bola reprezentovaná súborom `/dev/mmcblk0` som nahral obraz pomocou programu `dd`.

```
# dd if=/path/to/raspbian.img of=/dev/mmcblk0
```

Po nahratí obrazu som ju vložil do slotu a priviedol som napájanie. LED slúžiaca pre signalizáciu čítania z SD karty sa rozblikala, čo značilo, že zariadenie žije. Pripojil som k nemu monitor (cez HDMI) a USB klávesnicu a videl som, že sa systém dostal do stavu úvodnej konfigurácie. Ponechal som východzie nastavenia, prihlásil som sa pod východzím uživateľom [4] a príkazom `passwd` som nastavil nové heslo.

Vzdialený prístup

Linux už bežal, ale dalo sa k nemu pristupovať len pomocou klávesnice a monitora. Aby som sa k nemu mohol pripojiť cez sieť a ovládať ho zo svojho počítača, musel som na Raspberry Pi založiť Secure Shell (SSH) server.

Najprv som nastavil zariadeniu statickú sieťovú adresu na rozhraní ethernet.

`file: /etc/network/interfaces`

```
iface eth0 inet static
address 192.168.0.10
netmask 255.255.255.0
gateway 192.168.0.1 [5]
```

[3] `http://www.raspberrypi.org/downloads/`

[4] username: pi, password: raspberry

[5] nastavenie východzej brány bude potrebné až v podsekcii *Inštalácia softvéru*

Reštartoval som zariadenie a prepojil som ho cez ethernet s mojím počítačom. Na svojom počítači som taktiež nastavil rozhranie ethernet.

```
# ip link set eno1 up
# ip addr add 192.168.0.11/24 dev eno1
```

Skúsil som ping z môjho počítača.

```
$ ping 192.168.0.10
PING 192.168.0.10 (192.168.0.10) 56(84) bytes of data.
64 bytes from 192.168.0.10: icmp_seq=1 ttl=64 time=0.880 ms
64 bytes from 192.168.0.10: icmp_seq=2 ttl=64 time=0.557 ms
64 bytes from 192.168.0.10: icmp_seq=3 ttl=64 time=0.548 ms
64 bytes from 192.168.0.10: icmp_seq=4 ttl=64 time=0.475 ms
^C
--- 192.168.0.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.475/0.615/0.880/0.156 ms
```

Raspberry Pi odpovedalo. Stačilo už len spustiť SSH server a nastaviť systém tak, aby ho spustil po každom nabehnutí. To už vyriešili vývojári Raspberry Pi a stačilo ho povoliť v nastaveniach.

```
# raspi-config
```

V menu na to bola položka `Enable or disable ssh server`. Teraz som sa už mohol z môjho počítača bezpečne prihlásiť do Raspberry Pi.

```
$ ssh pi@192.168.0.10
```

Napísal som heslo, prihlásenie dopadlo úspešne.

Inštalácia softvéru

Pre ďalší vývoj som sa potreboval dostať z Raspberry Pi k balíkovým repozitárom Debianu, čiže na internet. Východziu bránu som už nastavil predom v pododstavci *Vzdialený prístup*. Mal som prístup napr. na adresu 147.229.2.90, ale nemal som prístup na `vutbr.cz`. Bolo treba ešte nakonfigurovať statický DNS server.

`file: /etc/resolv.conf`

```
nameserver 192.168.0.1
```

Po tomto kroku som sa vedel dostať na internet a nainštalovať potrebný softvér.

```
pi # apt-get install vim tmux git python-pip
```

Vim je profesionálny Command Line Interface (CLI) textový editor (existuje aj Graphical User Interface (GUI) verzia). *tmux* je terminálový multiplexor. Dokáže

vytvárať okná a záložky v termináli a prepínať medzi nimi. Má mnoho ďalších funkcií, ale pre prácu s Raspberry Pi nie sú až také podstatné. Zvlášť je vhodný pri práci cez SSH. Alternatívou je známejší program *screen*. *Git* je známy distribuovaný systém riadenia revízií. Obsiahlejší popis programu *git* je nad rámec tejto práce. *pip* slúži na inštaláciu softvérových balíkov (napr. knižníc) pre jazyk *Python*.

Jednoduchý test konfigurácie

V tomto pododdieli konkrétne popíšem odoslanie programu, ktorý bude periodicky prepínať jeden výstupný GPIO pin.

Na svojom počítači som vytvoril následujúci súbor:

file: /blink.py

```
import time
import RPi.GPIO as GPIO

GPIO.setup(17, GPIO.OUT) # Set pin 17 as output.

while True:
  GPIO.output(17, 1)     # Set HIGH voltage level on the pin 17.
  time.sleep(2)          # Wait 2 seconds.
  GPIO.output(17, 0)     # Set LOW voltage level on the pin 17.
  time.sleep(2)          # Wait 2 seconds.
```

Súbor som poslal cez protokol SSH do Raspberry Pi, pripojil som sa k Raspberry Pi cez SSH a program som spustil.

```
$ scp blink.py pi@192.168.0.10:/home/pi
pi@192.168.14.203's password:
$ ssh pi@192.168.0.10
pi@192.168.14.203's password:
pi $ python blink.py
```

Pripojil som voltmeter medzi zem a pin 17 a skontroloval som, že sa napätie mení každé 2 sekundy medzi úrovňami 3.3V a 0V.

Ukončil som program, vypol som Raspberry Pi, vložil som pamäťovú kartu do svojho počítača a zálohoval som si obraz.

```
# dd if=/dev/mmcblk0 of=/path/to/backup/image.img
```

5 INTERAKCIA RASPBERRY PI S HARDVÉROM

V následujúcom texte popíšem spôsob komunikácie Raspberry Pi s periférami.

5.1 Komunikácia s resetovacím obvodom

GPIO, ktorými je ATtiny v resetovaciom obvode prepojená s Raspberry Pi nemajú hardvérovo žiadnu špeciálnu komunikačnú funkciu, preto som musel použiť tzv. bit-banging (softvérovo riadenú) komunikáciu, čo znamená, že som musel ručne naprogramovať, ako zariadenie preloží vysielaný bajt do signálu a ako preloží prijatý signál do bajtu. Mohol som naprogramovať jeden z používaných protokolov druhej vrstvy, z dvojlinkových protokolov mne známych ma Ethernet a I^2C odradili svojou zložitosťou a UART ma odradil slabým prístupom k synchronizácii (synchronizuje sa až po celom bajte, čo sa mi nezdalo veľmi bezpečné pre bit-banging). Preto som si vymyslel vlastný jednoduchý protokol inšpirovaný protokolom používaným istým ďiaľkovým ovládačom, ktorý som v minulosti využíval pre iný projekt.

Komunikačné slovo obsahuje jeden bajt údajov. Začína štartovacou značkou, za ňou nasleduje osem značiek pre jednotlivé datové bity a potom ukončovacia značka. Každá značka sa odosiela ako sekvencia vysokej a nízkej úrovne, význam značky je určený časom týchto úrovní.

Tab. 5.1: Značky komunikačného slova

	Čas vysokej úrovne [ms]		Čas nízkej úrovne [ms]	
Značka	min.	max.	min.	max.
START	40	70	40	70
STOP	100	130	-	-
BIT_1	70	100	10	40
BIT_0	10	40	70	100

Prijímacia časť programu komunikujúcich zariadení (pri Raspberry Pi je to vlákno čakajúce na udalosť zmeny úrovne na vývode v spojení s časovačom Linuxu, pri ATtiny je to externé prerušenie súbežné s prerušením od časovača) sa synchronizuje po každej zmene úrovne na prijímacom vývode—dvakrát za značku.

5.2 Komunikácia s čítačkou čiarových kódov

Čítačka čiarových kódov sa správa ako USB klávesnica, ktorá za každým prečítaným čiarovým kódom odošle znak konca riadku (End of Line (EOL) alebo *newline*). To sa hodí, pretože si môžem programy, ktoré by bežali na Raspberry Pi testovať na svojom počítači (stačí odstrániť interakciu s GPIO) a namiesto čítačky čiarových kódov môžem použiť klávesnicu.

Komunikácia s čítačkou teda na uživateľskej úrovni prebieha ako jednoduché čítanie zo štandardného vstupu.

5.3 Komunikácia s grafickým terminálom

Grafický terminál sa pripája k Raspberry Pi cez USB rozhranie, využíva FTDI prevodník sériovej linky na USB. Ovládač pre FTDI je súčasťou Linuxového jadra. Po pripojení grafického terminálu ho Linux automaticky rozpozná a vytvorí preňho súbor znakového zariadenia (napr. `/dev/ttyUSB0`). Vyššiu vrstvu nad systémovými volaniami pre tento súbor tvorí napríklad knižnica pre *Python—pySerial*. Nad touto knižnicou som vytvoril vyššiu vrstvu pre riadenie grafického terminálu, viď. sekcia Knižnica pre riadenie grafického terminálu.

6 KNIŽNICA PRE RIADENIE GRAFICKÉHO TERMINÁLU

Pre jednoduchšiu prácu s terminálom, poriadok v kóde a pre využitie terminálu v ďalších projektoch som zostavil dve vyššie vrstvy nad knižnicou *pySerial*, pomocou ktorej je možné komunikovať s terminálom cez sériovú linku pomocou protokolu popísaného v dokumentácii. Nižšia vrstva *terminal.matrix_orbital* definuje triedu pre použitý terminál Matrix Orbital EGLK19264 a terminály s ním kompatibilné, zatiaľ čo vyššia vrstva *terminal* vytvára abstraktnejšiu triedu pre grafický terminál a sú pre ňu definované metódy pre celistvé akcie, ako napríklad zobrazenie a obslúženie menu, výberu čísla, kusovníka, atď.

Viac informácií poskytuje príloha 2. Jednoduché demonštračné programy sa nachádzajú v súboroch `FW/bin/terminal_demo` a `FW/bin/interpreter_demo`.

7 PROGRAMOVACÍ JAZYK PRE RIADIACU JEDNOTKU

Zo začiatku bola jedinou požiadavkou pre programovací jazyk možnosť napísať v ňom program popísaný v sekcii Činnosť riadiacej jednotky.

Pre tieto požiadavky som vyskúšal dve možnosti:

1. Napísať jednoduchý interprét inštrukcií vo formáte *JSON*, čo poskytovalo väčšiu kontrolu nad tým, čo programátor môže a čo nemôže naprogramovať.
2. Použiť samotný jazyk *Python* ako programovací jazyk pre riadiacu jednotku s tým, že by sa programátorovi poskytla knižnica s hotovými funkciami vysokoúrovňovými aj nízkoúrovňovými. Toto riešenie poskytovalu programátorovi naopak väčšiu flexibilitu.

Viac sa mi pozdávala druhá možnosť, prišli však ďalšie požiadavky.

Ďalšími požiadavkami bolo, že jazyk by mal byť jednoduchý, interaktívny a aby zabraňoval tvoreniu syntaktických chýb. Dosť zvláštnou požiadavkou bolo, že sa nemá podobať na programovací jazyk, tzn. nemajú tam byť premenné, vetvenia, ani cykly, taktiež neprešli ani iné spôsoby riadenia toku programu, ako napríklad reaktívne programovanie. Od zakázania premenných sa nakoniec upustilo, ale ako hlavný spôsob, ako ukladať data, sa zvolil akýsi zoznam výsledkov, ktorý sa indexuje od posledne pridaného prvku. Implicitný je index jedna (posledne pridaný prvok), a keďže sa v drvivej väčšine prípadov data spracujú hneď po ich obdržaní, indexácia zoznamu výsledkov alebo premenné sa v programe spomínajú zriedkavo.

Prvotné plány použiť *Python* ako programovací jazyk pre riadiacu jednotku s týmito požiadavkami padli, ale stále sme chceli, aby kvôli väčšej flexibilite ostala možnosť programovať riadiacu jednotku v jazyku *Python*.

Po zhodnotení týchto požiadaviek sme sa nakoniec rozhodli vytvoriť vlastný jazyk v knižnici *Blockly*.

7.1 Blockly

Blockly [5] [6] je knižnica s otvoreným kódom od firmy *Google* pre *JavaScript* [7], ktorá umožňuje definovať si vlastný grafický programovací jazyk. Je pomocou nej možné definovať vzhľad a správanie grafických blokov, a spôsob, akým sa majú preložiť do textovej podoby (v mojom prípade jazyk *Python*). Keď si užívateľ v internetovom prehliadači otvorí stránku, ktorá má v sebe *JavaScript*, ktorý spúšťa prostredie *Blockly* so spomínanými definíciami, môže si tieto bloky skladať a vytvoriť z nich program. Program je potom možné pomocou funkcií poskytovaných *Blockly*

preložiť na textový kód, *JavaScript* potom vytvorí XMLHttpRequest (XHR), pomocou ktorého odošle preložený skript na administrátorský server (inak sa všetko deje na klientskej strane). Odtiaľ si môže Raspberry Pi program stiahnúť.

Namiesto jazyka *JavaScript* som použil jazyk *CoffeeScript* [8], ktorý je vlastne prekladaný priamo do jazyka *JavaScript*, má však prehľadnejšiu a stručnejšiu syntax. Taktiež niektoré programové konštrukcie, ktoré by sa v jazyku *JavaScript* písali zložito, napriek tomu, že ich funkcia je jednoduchá, získali jednoduchšiu podobu inšpirovanú modernými jazykmi.

7.2 Popis blokov úrovne workflow

Na jednotlivé bloky boli kladené požiadavky popísané týmito vlastnosťami:

1. vzhľad bloku
2. správanie sa bloku počas programovania
3. činnosť riadiacej jednotky, ktorú má blok reprezentovať

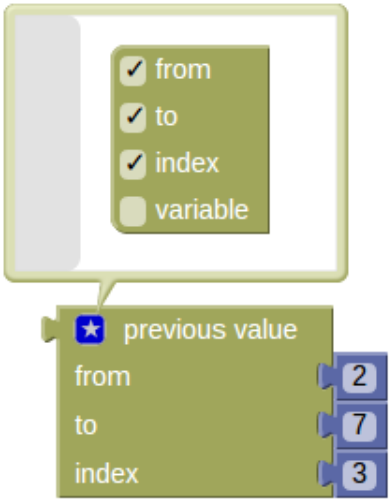
V tejto podsekcii popíšem požadované vlastnosti niektorých zaujímavých blokov.

Blok *fill* naplní premennú s daným názvom posledným prvkom v zozname výsledkov.

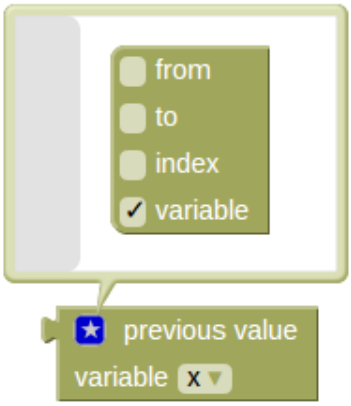
Obr. 7.1: Blok *fill*

Blok *previous value* vracia posledne pridaný výsledok. Pomocou parametrov `from` a `to` sa dá výsledok orezať (výsledky sú vždy reťazce). Pomocou parametrov `index` a `variable` sa dá vybrať aj skôr pridaný výsledok pomocou indexácie zoznamu (index jedna znamená posledne pridaný výsledok) alebo s využitím predtým naplnenej premennej.

Obr. 7.2: Blok *previous value* s indexáciou zoznamu výsledkov



Obr. 7.3: Blok *previous value* s využitím premennej



Blok *procedure* definuje procedúru, ktorá sa potom dá v programe zavolať pomocou bloku *call*. Parameter `continuation` určuje povahu procedúry v prerušovacom menu:

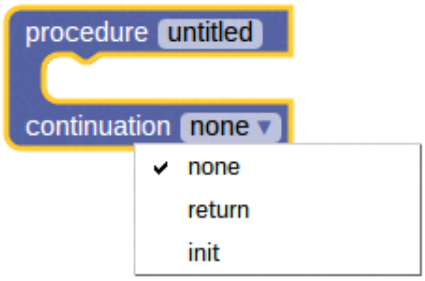
none
: Procedúra nie je zobrazená v prerušovacom menu.

return
: Procedúra je zobrazená v prerušovacom menu. Ak ju užívateľ v prerušovacom menu vyberie, procedúra prebehne a program sa vráti tam, kde bol prerušený.

to_init
: Procedúra je zobrazená v prerušovacom menu. Ak ju užívateľ v prerušova-

com menu vyberie, procedúra prebehne a program sa vráti do procedúry **init**.

Obr. 7.4: Blok *procedure*



Blok *retry*

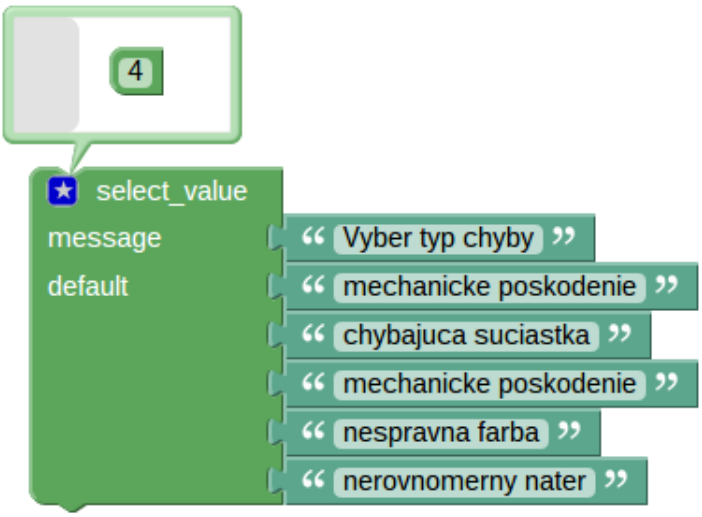
Ak nastane NG chyba v tele tohto bloku, chyba sa odstráni a opakuje sa telo bloku *retry*.

Obr. 7.5: Blok *retry*

Blok *select value*

Týmto blokom sa na grafickom termináli zobrazí žiadosť pre voľbu jednej z hodnôt predaných ako parametre. Možnosť, ktorú uživateľ zvolil, sa pridá do zoznamu výsledkov.

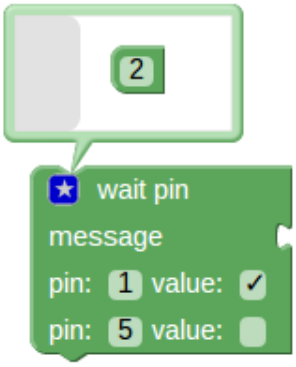
Obr. 7.6: Blok *select value*



Blok *wait pin*

Na grafickom termináli sa zobrazí zadaná správa a riadiaca jednotka čaká, pokiaľ na všetkych jej vstupoch určených parametrami tohto bloku nebudú naraz požadované hodnoty.

Obr. 7.7: Blok *wait pin*



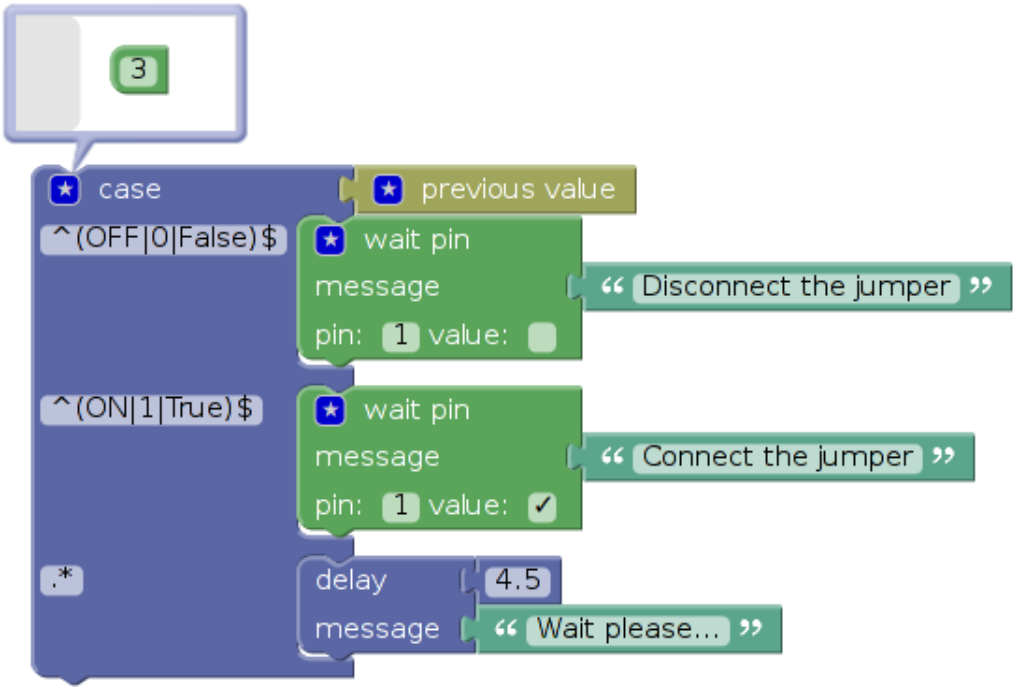
V tomto prípade riadiaca jednotka čaká, kým neplatí, že vstup č. 1 je zopnutý a zároveň je vstup č. 5 vypnutý.

Blok *case*

Vstupom do tohto bloku je textová hodnota, ktorá sa porovnáva s regulárnymi výrazmi na ktoré sú mapované sekvencie príkazov. Pri prvej zhode regulárneho výrazu so vstupom sa spustí príslušná sekvencia príkazov. Po ukončení tejto

sekvencie program vyskočí z bloku a pokračuje ďalším blokom.

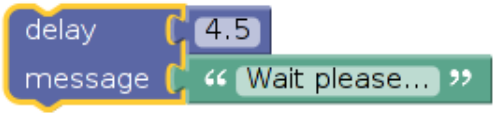
Obr. 7.8: Blok *case*



Blok *delay*

Na grafickom termináli sa zobrazí zadaná správa a riadiaca jednotka čaká zadaný čas (v sekundách).

Obr. 7.9: Blok *delay*



Blok *goto_ng*

Keď sa proces riadiacej jednotky dostane k tomuto bloku, vyvolá sa výnimka, ako keby databázový server vrátil hodnotu NG. Pomocou tohto bloku, bloku *retry* a bloku *case* je možné vytvoriť cyklus. Ak by sa však ukázalo, že cykly sú naozaj potrebné, určite pre nich vytvorím samostatné bloky.

Obr. 7.10: Blok *goto_ng*

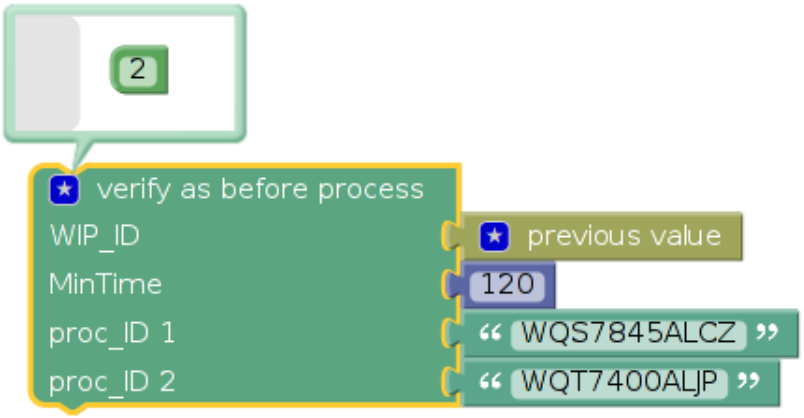
Blok *save_process_result*

Jednoduchý príkaz pre jednosmernú komunikáciu s databázovým serverom. Pošle sa pomocou neho požiadavka o uloženie výsledku akcie popísanej daným identifikátorom do databázy. To, či bola akcia úspešná alebo nie určuje zaškrtávacie pole.

Obr. 7.11: Blok *save_process_result*

Blok *verify_as_before_process* Overí sa, či súčiastka s čiarovým kódom `WIP_ID` bola spracovaná na aspoň jednom z pracovísk identifikovaných parametrami *proc_ID*. Parameter `MinTime` určuje minimálny čas, ktorý by mal prejsť od spracovania súčiastky na danom pracovisku. Väčšinou bude tento parameter nulový, ale v niektorých prípadoch (napr. lakovanie, farbenie) je nutné aby súčiastka vyschla, kým sa pôjde spracovávať.

Obr. 7.12: Blok *verify_as_before_process*



Blok *shared_procedure_call*

Blok pre volanie zdieľanej procedúry. Pri rozbalení výberového menu sa odošle na administrátorský server požiadavka o asociatívny zoznam s ID a názvami zdieľaných procedúr. Do mutácie bloku sa uloží iba ID procedúry, a názov sa zisťuje vždy dynamicky (požiadavkou odoslanou na administrátorský server). Ak sa teda zmení názov zdieľanej procedúry, nie je ho treba meniť vo WF.

Obr. 7.13: Blok *shared_procedure_call*

8 ADMINISTRÁTORSKÝ SERVER

Úlohou administrátorského servera je poskytovať rozhranie pre programovanie WF a pre ich priradenie do výrobného procesu. Toto rozhranie je hierarchicky rozdelené na niekoľko úrovní:

1. **Koreňová úroveň** môže obsahovať dva typy blokov:

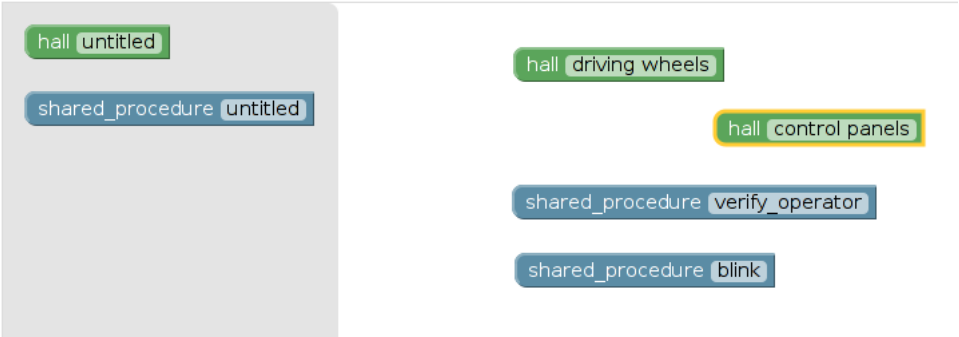
   hall

   Blok zatiaľ slúži len pre rozdelenie úrovní haly na menšie celky, aby sa v nich dalo ľahšie orientovať. V rozbaľovaciom menu bloku (zobrazí sa po kliknutí pravým tlačidlom myši) je položka *enter*, ktorá uživateľovi otvorí v novej záložke prehliadača obsah haly.

   shared_procedure

   Tento blok definuje procedúru, ktorá sa dá volať z ktoréhokoľvek WF. V rozbaľovacom menu bloku je položka *enter*, ktorá uživateľovi otvorí v novej záložke prehliadača telo zdieľanej procedúry. Ďalšou položkou v rozbaľovacom menu je položka *dependents*, ktorá zobrazí zoznam všetkých WF a zdieľaných procedúr, kde sa procedúra používa. Názov zdieľanej procedúry musí byť unikátny, o čo sa stará aplikačné rozhranie.

Obr. 8.1: Koreňová úroveň administrátorského rozhrania



2. **Úroveň haly** priradzuje WF jednotlivým riadiacim jednotkám, ktoré sú organizované do výrobných liniek. Úroveň haly obsahuje následujúce bloky:

   production_line

   Výrobná linka má k sebe priradený zoznam riadiacich jednotiek, ktoré sa budú vždy podieľať na výrobe toho istého produktu.

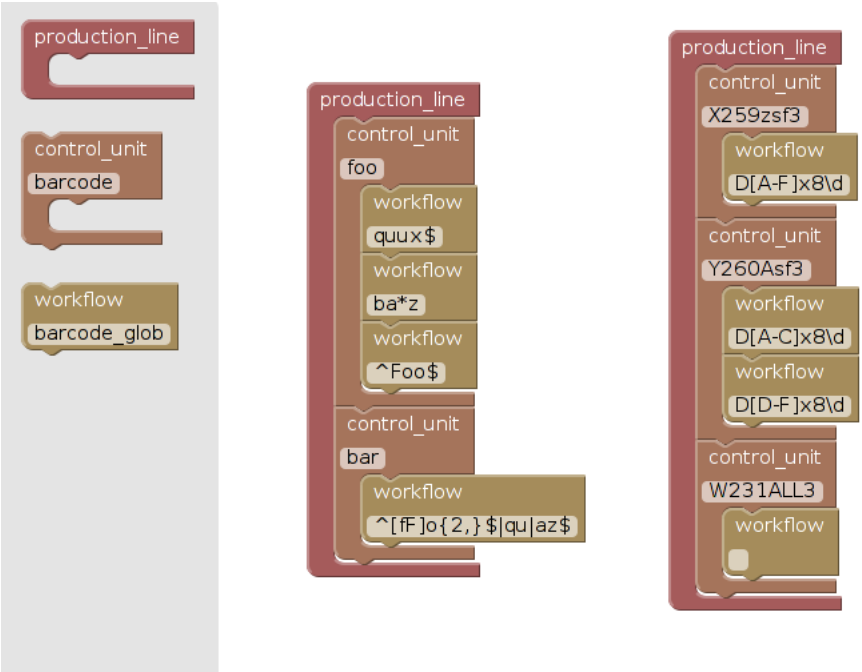
   control_unit

   Riadiaca jednotka obsahuje asociatívny zoznam WF mapovaných na regulárne výrazy popisujúce čiarový kód produktu, pre ktorý riadiaca jednotka vykonáva daný WF. Riadiace jednotky sú identifikované unikátnym čiarovým kódom, ktorý si načítajú po zapnutí. Bližší popis procesu bude popísaný v podsekcii Sťahovanie WF z administrátorského servera

workflow

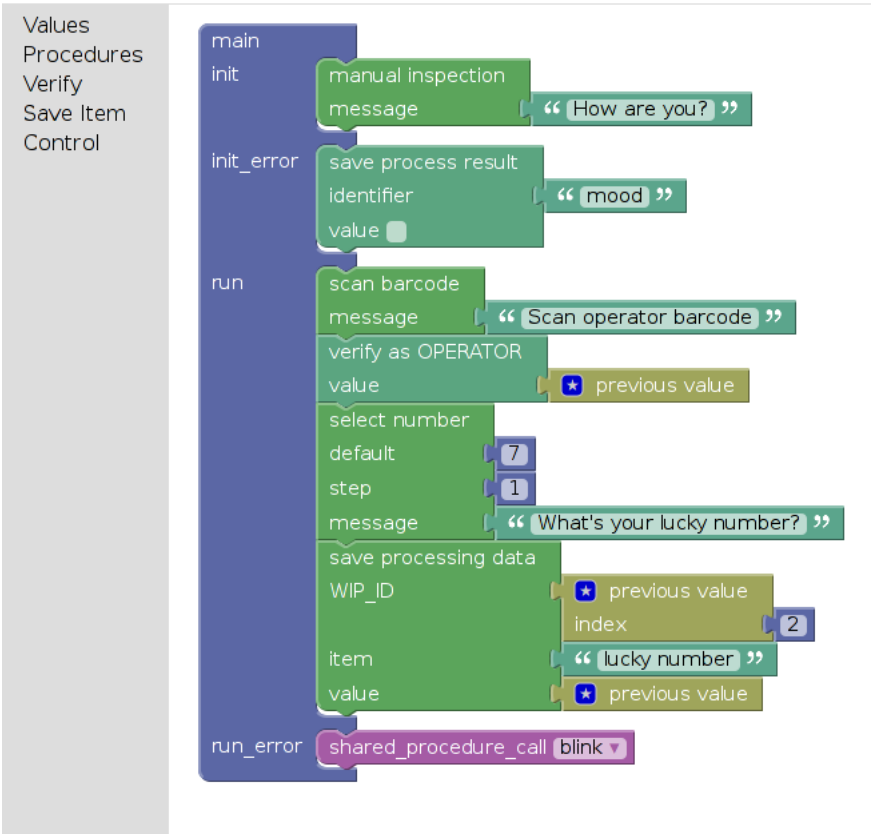
Blok obsahuje regulárny výraz, popisujúci čiarový kód produktu, pre ktorý sa má daný WF vykonať. V rozbaľovacom menu bloku je položka *enter*, ktorá uživateľovi otvorí v novej záložke prehliadača telo WF.

Obr. 8.2: Úroveň haly v administrátorskom rozhraní

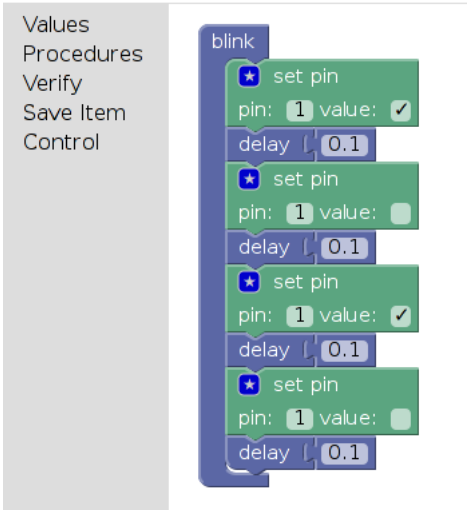


3. **Úroveň workflow** obsahuje samotný program činnosti riadiacej jednotky. Môže mať podobu tela zdieľanej procedúry alebo tela WF. Tieto dve podoby sa líšia iba základným vrchným blokom obaľujúcim program (tento blok je uzamknutý - nedá sa s ním hýbať, nedá sa vymazať). Bloky tejto úrovne sú popísané v podsekcii 7.2 Popis blokov úrovne workflow.

Obr. 8.3: Úroveň workflow - *workflow*



Obr. 8.4: Úroveň workflow - *shared_procedure*



8.1 Databáza administrátorského servera

Administrátorský server ukladá údaje do MongoDB [12] databázy (s využitím knižnice *pymongo*). MongoDB je NoSQL databázový stroj s dynamickým databázovým modelom. Údaje MongoDB sú fyzicky reprezentované formátom BSON—špecializovaný binárny JSON rozšírený o indexačné údaje a ďalšie datové typy (regulárny výraz, *timestamp*, *Object ID*). Jeho údaje sú organizované do kolekcií, základným prvkom kolekcie je dokument. Výhodou MongoDB oproti klasickým SQL databázovým strojom je práve dynamickosť modelu ako napríklad polia, objekty, regulárne výrazy, ktoré rozširujú možnosti návrhu databázového modelu. Okrem klasickej reprezentácie väzieb pomocou referencií na rodičovský prvok popisuje dokumentácia ďalšie vzory, ktoré môže programátor rôzne kombinovať aby dosiahol čo najväčšiu optimalizáciu prístupu pre danú aplikáciu. Administrátorský server využíva vzor poľa referencií na dcérske prvky v prvku rodičovskom a taktiež vnorené dokumenty.

Kolekcia `top_xml` obsahuje jediný dokument, v ktorom je uložené *Blockly* XML pre koreňovú úroveň administrátorského rozhrania (obr. 8.1).

Kolekcia `halls` reprezentuje úroveň haly (obr. 8.2). Okrem *Blockly* XML sa tu nachádza názov a identifikátor haly, referencie na výrobné linky, ktoré sa v hale nachádzajú, a referencie na WF, ktoré sú síce vložené na *Blockly* pracovnú plochu danej haly ale nie sú v nej priradené k žiadnej riadiacej jednotke a výrobnej linke. Programátor môže upravovať kód v týchto nepriradených WF a priradiť ich k linkám neskôr.

Kolekcia `plines` reprezentuje výrobné linky. Dokumenty obsahujú číslo naposledy vyábaného produktu `part_number`, podľa neho sa riadiacim jednotkám priradzuje WF. Taktiež obsahuje pole objektov typu riadiaca jednotka. Objekt typu riadiaca jednotka obsahuje čiarový kód pracoviska a pole objektov typu WF. Objekt typu WF obsahuje regulárny výraz, ktorý sa pri vyžiadaní WF riadiacou jednotkou porovnáva s položkou `part_number` príslušnej výrobnej linky, a ak dojde k zhode, WF s daným `_id` sa odošle riadiacej jednotke.

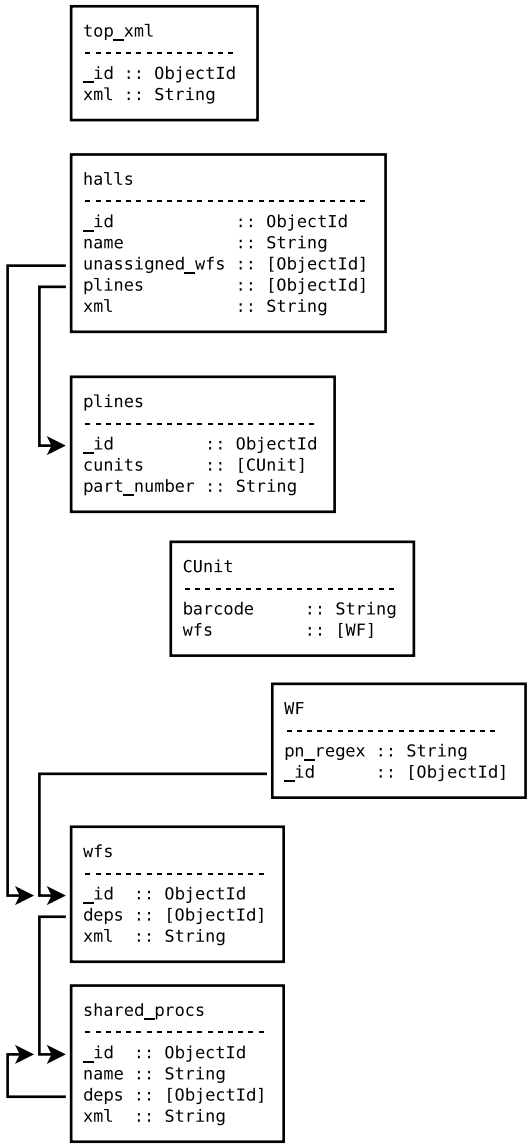
Dokumenty kolekcie `wfs` obsahujú *Blockly* XML pre daný WF (obr. 8.3) a referencie na zdieľané procedúry, ktoré sa vo WF využívajú.

Dokumenty kolekcie `shared_procs` obsahujú názov zdieľanej procedúry, *Blockly* XML procedúry (obr. 8.4) a taktiež zoznam zdieľaných procedúr, ktoré sa v nej využívajú.

Niektoré prvky (`hall`, `pline`, `shared_proc`, `wf`) sa vytvárajú hneď po ich uložení na pracovnú plochu *Blockly* (na server sa pošle XHR požiadavka o vytvorenie prvku, server vráti odpoveď s ID vytvoreného prvku a toto ID sa uloží ako mutácia [13] bloku *Blockly*). Ostatné údaje sa ukladajú až po odoslaní XML gene-

rovaného z pracovnej plochy *Blockly* po stlačení tlačidla *store*. Na strane servera sa XML rozloží pomocou knižnice *xml.etree.ElementTree*, získajú sa z neho potrebné údaje a uložia sa do databázy. Jedine údaj `plines.part_number` sa nezískava z aplikačného programovacieho rozhrania ale posielajú ho administrátorskému serveru riadiace jednotky.

Obr. 8.5: Štruktúra databázy administrátorského servera



8.2 **Sťahovanie** WF z administrátorského servera

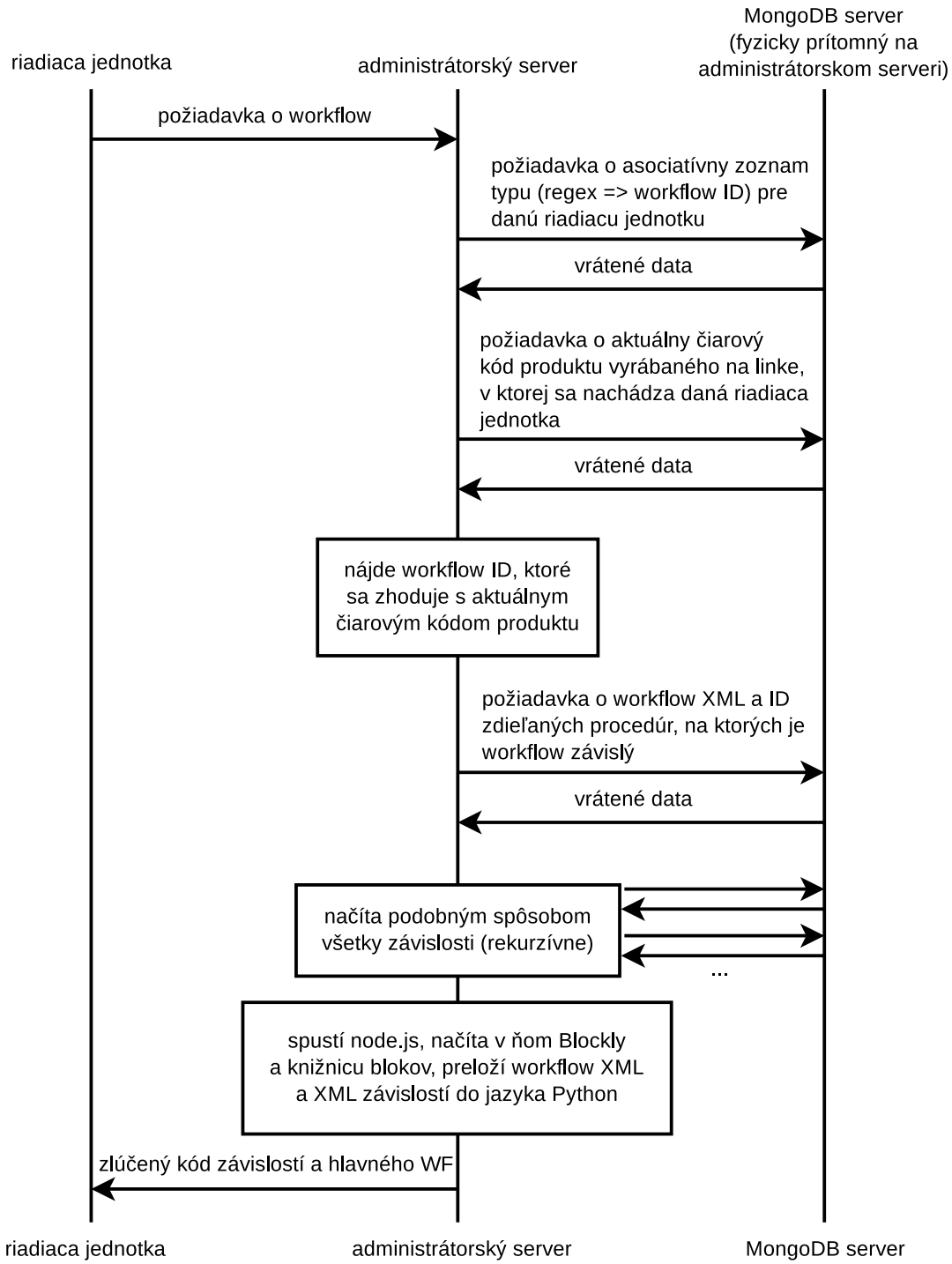
Riadiace jednotky po každom zapnutí vyžiadajú operátora k načítaniu čiarového kódú riadiacej jednotky, ktorý sa nachádza na pracovisku. Pomocou tohto kódu sa potom identifikujú pri komunikácii s administrátorským serverom.

Následne sa zobrazí menu, kde operátor zvolí z dvoch možností:

1. **`use most recent barcode`** - riadiaca jednotka si vyžiada od administrátorského servera svoj WF pre naposledy načítaný Part Number (PN) na príslušnej výrobnej linke.
2. **`set new barcode`** - načítanie čiarového kódu produktu (PN), ktorý sa ide vyrábať. Túto voľbu zvolí vedúci pracovník linky. Načítaný čiarový kód sa odošle administrátorskému serveru a ten ho uloží do databázy pre linku, ku ktorej je riadiaca jednotka priradená. Následne riadiaca jednotka vyžiada príslušný WF a začne ho vykonávať.

Proces sťahovania WF z administrátorského servera znázorňuje obrázok 8.6.

Obr. 8.6: Sťahovanie WF



8.3 Implementácia administrátorského servera

Administrátorský server beží na virtuálnom stroji s linuxovou distribúciou *Turnkey* [14]. *Turnkey Linux* je založený na distribúcii *Debian Wheezy*, naviac obsahuje softvér pre automatické zálohovanie databázových a konfiguračných súborov na šif-

rované úložisko a ich automatické obnovenie v prípade poruchy. Taktiež obsahuje softvér pre každodenné sťahovanie bezpečnostných aktualizácii.

Pre spustenie a pohodlnú správu administrátorského servera bolo treba doinštalovať ďalší softvér: *Git*, *tmux*, *MongoDB*, *Python 3.4*, *python3-pip*, *npm*. *Debian Wheezy* má vo svojich repozitároch len stabilné overené vyzrelé balíky, ktoré sú bohužiaľ staré. Najviac to vadilo pri balíku MongoDB, pretože verzia 2.0.6 nepodporuje ani príkaz *aggregate*, ktorý sa používa v programe administrátorského servera hojne a bez ktorého by bolo treba v niektorých prípadoch neelegantne vykonávať aj štvornásobok databázových dopytov. Našťastie je na stránke MongoDB uvedený návod pre inštaláciu najnovšej verzie 3.0.2 na *Debian* [15]. Ďalším problémom bol balík *Python 3.4*. *Debian Wheezy* má vo svojich repozitároch najnovšiu verziu *Python 3.2* a keďže knižnica *Flask*, slúžiaca ako back end (back end) pre HTTP server, podporuje *Python 3* až od verzie 3.4, naskytli sa dve možnosti: použitie starej verzie *Python 2.7* alebo prechod na nestabilnú distribúciu *Debian Sid*. Keďže zmeny v *Python 3* boli podľa môjho názoru dobrým krokom k modernizácii jazyka, volil som radšej dočasný prechod na *Debian Sid* a tým vyriešenie mnohých ďalších problémov so zastaralým softvérom distribúcie *Debian Wheezy*.

Ako back end pre HTTP server bola donedávna použitá veľmi jednoduchá knižnica *Bottle*. Jej výhodou bola hlavne jednoduchosť, mala však niekoľko nevýhod. Prvý problém sa objavil pri preklade *Blockly* programov do jazyka *Python*.

Ako bolo spomenuté v podsekcii **??**, požiadavka GET `/program/default/<cunit_barcode>` spúšťa *node.js* interprét, do ktorého sa načíta knižnica *Blockly*, vytvorí sa pomocou nej pracovná plocha, do ktorej sa načíta XML programu pre danú riadiacu jednotku. Toto XML sa načítava pomocou požiadavky GET `/wf/xml/<id>`. Tu sa problém objaví prvý krát. Server obstaráva jednu požiadavku a počas nej príde druhá. Keďže server *Bottle* nepoužíva vlákna, túto druhú požiadavku neobstará a ostane zaseknutý. Dalo by sa to vyriešiť tak, že by sa pri preklade na serveri poskytnú interprétu všetky data tak, aby nemusel posielať ďalšie požiadavky na server. Možno sa to neskôr kvôli rýchlosti takto vyrieší, ale riešenie nie je také jednoduché, ako by mohlo na prvý pohľad vyzerať. Vyžiadanie XML nie je jedinou požiadavkou, ktorú interprét pri preklade vykonáva. Niektoré bloky (napr. *shared_procedure_call*) pri svojej inicializácii taktiež vytvárajú HTTP požiadavky.

Riešenie tohto problému bolo jednoduché. Knižnica *Bottle* umožňuje zmeniť svoj back end z bezvláknového jednoduchého *wsgiref* na niektorý z ďalších podporovaných [16]. Zvolil som viacvláknový back end *cherrypy*.

Ďalším problémom ktorý s knižnicou *Bottle* nastal bola slabá podpora logovania. Keď sa začali riadiace jednotky testovať na reálnych pracoviskách vo výrobnej hale, objavila sa zvýšená potreba logovať všetky chyby, ktoré nastali do súboru.

Bottle však loguje iba na štandardný chybový výstup, ktorý sa dá síce presmerovať do súboru, ale pri zabití servera ukončovacím signálom (čo je vlastne jediný spôsob jeho vypnutia) sa neuzavrú korektne súbory, do ktorých bol výstup presmerovaný a stratia sa niektoŕé logované údaje. Všetky pokusy o odstránenie tohto problému (väčšinou týkajúce sa vypnutia vyrovnávacích pamätí pri presmerovaní výstupu) skončili neúspešne. Knižnica pre logovanie požiadaviek serverov založených na knižnici *WSGI* — *wsgi-request-logger* síce logovala požiadavky, pri chybe však nedokázala logovať text výnimky, ale iba správu o tom, že server vrátil odpoveď s kódom 500 (interná chyba servera).

Lepšia podpora logovania bude pri nasadení do výroby nutnosťou, napríklad pre logovanie kritických chýb na e-mail alebo SMS, aby sa mohli čím skôr opraviť. Preto som sa rozhodol zameniť knižnicu *Bottle* za knižnicu *Flask*. Knižnice sú si veľmi blízke a potrebné zmeny v kóde administrátorského servera pri tomto prechode boli minimálne. Ako back end servera som ponechal *cherrypy*.

9 KOMUNIKÁCIA RIADIACEJ JEDNOTKY A ADMINISTRÁTORSKÉHO SERVERA

Pre komunikáciu s administrátorským serverom je použitý HTTP protokol. Požiadavky sú identifikované pomocou Uniform Resource Identifier (URI). Pre lepšiu predstavu o činnosti administrátorského servera uvediem popis všetkych požiadaviek, na ktoré administrátorský server odpovedá:

GET `/static/<path>`

Požiadavka o statický súbor, ktorý sa nachádza v priečinku `static`. Medzi tieto súbory patrí knižnica *Blockly* a knižnice blokov preložené z jazyka *CoffeeScript* do jazyka *JavaScript*.

GET `/`

Odpoveďou pre túto požiadavku je HTML stránka obsahujúca pracovnú plochu *Blockly* pre koreňovú úroveň programátorského rozhrania (obr. 8.1). Na stránke sa nachádza *JavaScript*, ktorý opätovne vytvorí ďalšiu požiadavku GET `/xml` a stiahne pomocou nej obsah koreňovej úrovne.

GET `/xml`

Požiadavka o XML koreňovej úrovne programátorského rozhrania.

POST `/xml`

Požiadavka o uloženie koreňovej úrovne do databázy. Telo požiadavky obsahuje XML koreňovej úrovne vygenerované knižnicou *Blockly*. Na strane administrátorského servera sa okrem aktualizácie záznamu v kolekcii `top_xml` (kolekcia obsahuje jediný záznam s jedinou položkou `xml`) analyzuje XML. Pre novovzniknuté haly a zdieľané procedúry sa v databáze aktualizujú ich mená. Haly, ktoré boli z pracovnej plochy odstránené, sa odstránia aj z databázy spolu so všetkými výrobnými linkami a WF, ktoré hala obsahovala.

GET `/hall/new`

Požiadavka o vytvorenie novej haly. Server vráti klientovi odpoveď s ID novej haly. Túto požiadavku vytvára klient po vtiahnutí bloku *hall* na pracovnú plochu koreňovej úrovne programátorského rozhrania. Všetky prvky, ktoré sú v databáze administrátorského servera reprezentované ako dokument, sa vytvárajú hneď po vložení na pracovnú plochu a tlačidlom *store* sa k ním len priradia údaje ako napríklad názov haly. Je to takto, pretože *Blockly* volá v niektorých prípadoch funkciu pre preklad do XML aj pre svoje interné účely a v XML už musí byť uložená mutácia s ID dokumentu reprezentovaného daným blokom.

GET `/hall/<id>`

Odpoveďou pre túto požiadavku je HTML stránka obsahujúca pracovnú plochu

Blockly pre úroveň haly programátorského rozhrania (obr. 8.2). Na stránke sa nachádza *JavaScript*, ktorý stiahne a nainicializuje obsah pracovnej plochy pre halu s daným ID pomocou požiadavky GET `/hall/xml/<id>`.

GET `/hall/xml/<id>`

Požiadavka o XML úrovne haly programátorského rozhrania pre halu s daným ID.

POST `/hall/xml/<id>`

Požiadavka o uloženie XML pre halu s daným ID do databázy. Na serverovej strane dochádza okrem uloženia XML k jeho analýze a aktualizácii kolekcií `halls`, `plines` a `workflows` na základe tejto analýzy. V hale môže dôjsť k vymazaniu, presunu blokov alebo k zmene ich atribútov (čiarový kód riadiacej jednotky, regulárny výraz pre WF). Všetky tieto zmeny musia byť premietnuté do databázy.

GET `/pline/new`

Požiadavka o vytvorenie novej výrobnej linky (podobne ako GET `/hall/new`).

GET `/wf/new`

Požiadavka o vytvorenie nového WF (podobne ako GET `/hall/new`).

GET `/wf/<id>`

Požiadavka o pracovnú plochu pre daný WF (podobne ako GET `/hall/<id>`)

GET `/wf/xml/<id>`

Požiadavka o XML pre daný WF (podobne ako GET `/hall/xml/<id>`).

POST `/wf/xml/<id>`

Požiadavka o uloženie XML pre WF s daným ID do databázy. Na serverovej strane dochádza okrem uloženia XML k jeho čiastočnej analýze — nájdenie blokov *shared procedure call* a uloženie týchto závislostí na zdieľaných procedúrach do položky `deps` v databázovom dokumente pre daný WF.

GET `/shared_proc/new`

Požiadavka o vytvorenie novej zdieľanej procedúry (podobne ako GET `/hall/new`).

GET `/shared_proc/<id>`

Požiadavka o pracovnú plochu pre danú zdieľanú procedúru (podobne ako GET `/hall/<id>`)

GET `/shared_proc/xml/<id>`

Požiadavka o XML pre danú zdieľanú procedúru (podobne ako GET `/hall/xml/<id>`).

GET `/shared_proc/list`

Požiadavka o asociatívny zoznam názvov a ID všetkých zdieľaných procedúr.

Táto požiadavka sa vytvára pri zobrazení rozbaľovacieho menu pre blok *shared procedure call*.

GET `/shared_proc/dependents/<id>`
Odpoveďou na túto požiadavku je HTML stránka obsahujúca zoznam odkazov na všetky WF a zdieľané procedúry, ktoré sú závislé na zdieľanej procedúre s daným ID.

POST `/part_number/<cunit_barcode>` Požiadavka o uloženie čiarového kódu aktuálne vyrábaného produktu (nachádza sa v tele požiadavky) pre linku, v ktorej sa nachádza riadiaca jednotka s daným čiarovým kódom (cunit_barcode). Táto požiadavka je vytvorená riadiacou jednotkou, ktorú obsluhuje vedúci pracovník výrobnej linky, na začiatku výroby daného produktu.

GET `/program/default/<cunit_barcode>` Požiadavka o WF pre danú riadiacu jednotku a čiarový kód aktuálne vyrábaného produktu v jazyku *Python*. Na strane administrátorského servera sa pri tejto požiadavke spustí *node.js* server, do ktorého sa načítajú skripty v jazyku *JavaScript* v tomto poradí:

1. Knižnica, ktorá definuje objekt `window` a objekt `document`. Tieto objekty sú definované v interpréte jazyka *JavaScript* v internetových prehliadačoch, ale nie sú definované v *node.js*. Keďže knižnica *Blockly* a knižnice blokov pre WF sú určené pre internetové prehliadače, je pre ich funkčnosť treba definovať tieto objekty explicitne.
2. Ďalšie zložitejšie knižnice s funkciami využívanými v knižnici *Blockly* a v knižniciach blokov pre WF, ktoré pre *node.js* nie sú definované. Tieto knižnice sú klonované z repozitárov `https://github.com/driverdan/node-XMLHttpRequest` a `https://github.com/jindw/xmldom`.
3. Knižnica *Blockly*
4. Knižnice blokov používaných na úrovni *workflow* programátorského rozhrania
5. Skript ktorý inicializuje pracovnú plochu *Blockly*, načíta do nej prekladané XML pomocou požiadavky GET `/wf/xml/<id>` alebo GET `/shared_proc/xml/<id>`, preloží obsah pracovnej plochy do jazyka *Python* a vypíše preložený kód na štandardný výstup.

Obsluha požiadavky prečíta štandarný výstup *node.js*, zlúči programy pre zdieľané procedúry a hlavný WF, na začiatok pridá hlavičku s importom modulov potrebných pre správny beh programu a definíciou globálnej premennej reprezentujúcej zoznam výsledkov (viď Programovací jazyk pre riadiacu jednotku) a odošle výsledný program ako odpoveď na požiadavku.

10 TESTOVANIE NA PRACOVISKU VÝROBNEJ LINKY

Riadiaca jednotka bola odskúšaná s reálnou úlohou na reálnom pracovisku, aby sa zistilo, či je návrh programovacieho jazyka dostatočný a aby sa odhalilo čo najviac chýb, či už na strane administrátorského servera alebo riadiacej jednotky.

Ešte pri písaní WF sa zistilo, že budú potrebné dva nové bloky:

- *assert_pin* vyhodi NG výnimku, ak digitálne hodnoty na vstupoch neodpovedajú požadovaným hodnotám.
- *blink_pin* na digitálnych výstupoch zapne obdĺžnikový signál o danej frekvencii, tento signál sa vypne pomocou bloku *set_pin*. Signály na rôznych výstupoch s rovnakou frekvenciou zapnuté jednym príkazom *blink_pin* sú synchronizované.

Po doplnení blokov na strane administrátorského servera a riadiacej jednotky som sa cez prehliadač pripojil k administrátorskému serveru a zostavil som jednoduchý WF pre obsluhu pracoviska.

Následne bolo treba umožniť nasadenie riadiacej jednotky do výrobnej linky. Vyrobil sa potrebný prepojovací plochý kábel a IT technici firmy umožnili riadiacej jednotke sieťové pripojenie na pracovisku. K riadiacej jednotke som pripojil všetky periférie a spustil ju.

Riadiaca jednotka ma vyzvala k načítaniu čiarového kódu pracoviska, pomocou aplikácie do mobilného telefónu pre generovanie čiarových kódov som vytvoril čiarový kód korešpondujúci s čiarovým kódom definovaným v rozhraní haly (obr. 8.2) administrátorského servera a načítal som ho. Na termináli sa mi zobrazilo výberové menu s možnosťami **`get last workflow`** a **`scan new part number`**. Keďže som priradil vytvorený WF k ľubovoľnému čiarovému kódu produktu (pomocou regulárného výrazu `/.*/`), nemusel som načítavať čiarový kód produktu (implicitne je to prázdny reťazec) a zvolil som možnosť **`get last workflow`**. Riadiaca jednotka poslala na administrátorský server žiadosť o WF.

V dôsledku nedávnych zmien v softvéri administrátorského servera a riadiacej jednotky, ktoré som ešte nesthol odladiť, som musel vyriešiť niekoľko chýb (týkajúcich sa zväčša odsadzovania pri preklade *Blockly* do jazyka *Python*), kým sa riadiacej jednotke úspešne podarilo stiahnúť si WF a spustiť ho.

Následne sa ukázalo, že nefungujú výstupy. Indikačné LED svietili ale výrobná linka nereagovala. Po dlhom hľadaní chyby v zapojení a na rozširujúcej doske riadiacej jednotky som prišiel na to, že je chyba v softvéri, a to v nesprávnom číslovaní výstupov. Výstupy teda boli funkčné, spínali sa však nesprávne vývody. Opravil som v programe riadiacej jednotky poradie výstupov a systém bol funkčný.

Testovanie splnilo svoj účel — odhalili sa nedostatky programovacieho jazyka aj

chyby v softvéri. Nedostatky jazyka sa doplnili, chyby sa opravili. V najbližšom čase bude prebiehať dlhé a detailné testovanie robustnosti systému na rôznych pracoviskách s rôznymi WF, musí sa ale dokončiť komunikácia s databázovým serverom, či je hlavný problém, ktorý sa momentálne rieši.

Obr. 10.1: Riadiaca jednotka nasadená v testovacej prevádzke

11 ZÁVER

Zoznámil som sa s požiadavkami na riadiacu jednotku a s možnosťami vstavaného zariadenia Raspberry Pi. Taktiež som sa zoznámil s výrobnymi linkami, do ktorých budú riadiace jednotky zavedené, aby som mal predstavu o konkrétnych parametroch a o interakcii s operátorom. Týchto informácií som sa držal pri vývoji riadiacej jednotky.

Podarilo sa mi navrhnúť, osadiť a oživiť tri verzie dosky plošných spojov pre riadiacu jednotku.

Ku grafickému terminálu som napísal funkčnú knižnicu vo dvoch vrstvách. Programátor, ktorému stačia funkcie, ktoré som do knižnice implementoval, sa nemusí nižšou vrstvou vôbec zaoberať. Komunikácia s grafickým displejom pomocou príkazov stanovených výrobcom je od neho úplne abstrahovaná. Ak by však niekto v budúcnosti potreboval knižnicu rozšíriť o nové interaktívne prvky, bude sa mu nižšia vrstva hodiť.

Vyvinul som taktiež grafický interaktívny programovací jazyk pre riadiacu jednotku. Jazyk je jednoduché sa naučiť aj pre laikov v oblasti programovania. Prekladá sa do jazyka *Python*, takže v prípade nutnosti väčšej flexibility sa dá použiť priamo *Python*. Programátorské rozhranie beží v internetovom prehliadači, takže je dobre prenositeľné.

Zoznámil som sa s požiadavkami na administrátorský server, server som taktiež implementoval. Server obsluhuje požiadavky pre *Blockly* aplikačné programátorské rozhranie, ukladá programy pre riadiace jednotky a k ním rôzne potrebné metadata do MongoDB databázy, prekladá programy do jazyku *Python* a odosiela ich riadiacim jednotkám. Komunikácia s databázovými servermi dohľadateľnosti zatiaľ nie je plne špecifikovaná.

V riadiacej jednotke som implementoval sťahovanie programov preložených do jazyka *Python* z administrátorského servera podľa čiarového kódu pracoviska a vyrábaného produktu. Program riadiacej jednotky je pripravený na doplnenie komunikácie s databázovými servermi, keď sa urovnajú požiadavky a implementujú sa servery.

Otestoval som základnú funkčnosť riadiacej jednotky na pracovisku výrobnej linky.

V najbližšom čase je na pláne implementovať komunikáciu s databázovými servermi, vyrobiť väčší počet riadiacich jednotiek a uskutočniť s nimi robustné testovanie celého systému. Taktiež sa bude navrhovať nový hardvér s niekoľkými vylepšeniami, ako napríklad možnosť rozšírenia vstupov a výstupov expanznými modulmi. V dohľadnej dobe sa riadiace jednotky nasadia do riadnej výroby. Najprv sa nasadia v českej pobočke firmy, a potom, ako sa overí plná integrovateľnosť do výrobného

systému, sa môžu šíriť aj do zahraničia.

Táto bakalárska práca ma naučila pracovať s mnohými technológiami v oblasti vstavaných zariadení, návrhu hardvéru a hlavne v softvérovej oblasti. Získal som mnohé ďalšie vedomosti o operačnom systéme Linux, navrhol som a naprogramoval komplexný systém, pričom som sa naučil pracovať s MongoDB databázovým strojom, využívať knižnicu *Blockly* pre vytváranie grafických programovacích a konfiguračných rozhraní zobrazovaných v internetových prehliadačoch, taktiež knižnice *Bottle* a *Flask* pre programovanie HTTP serverov v jazyku *Python*, knižnicu *pySerial* pre sériovú komunikáciu a mnoho ďalších menších modulov štandardnej knižnice jazyka *Python*. Získal som taktiež mnoho skúseností z práce vo vývojovom oddelení firmy — konzultoval som so zadavateľom jeho požiadavky na vyvíjaný systém, zapájal som sa návrhmi a pripomienkami do diskusií o detailoch projektu, dokumentoval a organizoval som svoju prácu tak, aby bola udržiavateľná inými programátormi, s ktorými v budúcnosti budem na projekte spolupracovať.

LITERATÚRA

[1] Traceability. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-01-28]. Dostupné z: http://en.wikipedia.org/wiki/Traceability

[2] SN54HC595, SN74HC595: 8-BIT SHIFT REGISTERS WITH 3-STATE OUTPUT REGISTERS. In: *SparkFun Electronics (US)* [online]. Dallas, Texas: Texas Instruments, 2004. Dostupné z: https://www.sparkfun.com/datasheets/IC/SN74HC595.pdf

[3] TLP281,TLP281-4: TOSHIBA PHOTOCOUPLER GaAs I RED & PHOTOTRANSISTOR. In: *Digikey* [online]. TOSHIBA, 2007-10-01. Dostupné z: http://media.digikey.com/pdf/Data%20Sheets/Toshiba%20PDFs/TLP281%28-4%29.pdf

[4] PYTHON SOFTWARE FOUNDATION (US). *Python.org: Our Documentation* [online]. [cit. 2015-01-28]. Dostupné z: https://www.python.org/doc

[5] GOOGLE. *Blockly Block Wiki* [online]. Github wiki. [cit. 2015-01-28]. Dostupné z: https://github.com/google/blockly/wiki

[6] GOOGLE. *Blockly: Language Design Philosophy* [online]. [cit. 2015-01-28]. Dostupné z: https://developers.google.com/blockly/about/language

[7] MOZILLA CORPORATION (US). *About Javscript* [online]. [cit. 2015-01-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[8] *CoffeeScript* [online]. [cit. 2015-01-28]. Dostupné z: http://coffeescript.org/

[9] RASPBERRY PI FOUNDATION. *RASPBERRY PI DOCUMENTATION* [online]. [cit. 2015-01-28]. Dostupné z: http://www.raspberrypi.org/documentation/

[10] Programmable logic controller. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-01-28]. Dostupné z: http://en.wikipedia.org/wiki/Programmable_logic_controller

[11] SHOTTS, JR., William E. LINUXCOMMAND.ORG. *The Linux Command Line* [online]. 2012 [cit. 2015-01-28]. ISBN 978-1-59327-389-7. Dostupné z: http://it-ebooks.info/book/2012/

[12] MONGODB.ORG *The MongoDB Manual* [online]. [cit. 2015-04-18]. Dostupné z: `http://docs.mongodb.org/manual/`

[13] GOOGLE. *Blockly Mutators* [online]. [cit. 2015-04-18]. Dostupné z: `https://developers.google.com/blockly/custom-blocks/mutators`

[14] TURNKEYLINUX.ORG. *About Turnkey GNU/Linux* [online]. [cit. 2015-04-29]. Dostupné z: `http://www.turnkeylinux.org/about`

[15] MONGODB.ORG. *Install MongoDB on Debian* [online]. [cit. 2015-04-29]. Dostupné z: `http://docs.mongodb.org/manual/tutorial/install-mongodb-on-debian/`

[16] BOTTLEPY.ORG. *Bottle deployment* [online]. [cit. 2015-04-30]. Dostupné z: `http://bottlepy.org/docs/dev/deployment.html`

[17] NIKROPHT. *FreeRouting* [online]. [cit. 2015-04-30]. Dostupné z: https://github.com/nikropht/FreeRouting/

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

GPIO	General Purpose Input/Output označuje vstupno-výstupný vývod mikroprocesora, ktorý je možné nastaviť a ovládať počas behu programu.
UART	Universal Asynchronous Receiver/Transmitter je jednoduchý komunikačný protokol pre komunikáciu medzi dvoma zariadeniami.
SPI	Serial Peripheral Interface je jednoduchý, rýchly komunikačný protokol pre *master-slave* komunikáciu. *Slave* zariadenia sú adresované signálom *chip select* ($\overline{CS}$).
I^2C	Inter-Integrated Circuit je zložitejší komunikačný protokol s adresáciou pomocou adresovacieho bajtu v hlavičke rámca.
HDMI	High Definition Multimedia Interface
USB	Universal Serial Bus
LED	Light-emitting Diode označuje svietivú diódu.
SSH	Secure Shell je zabezpečený protokol pre vzdialený prístup k príkazovému riadku počítača.
CLI	Command Line Interface označuje rozhranie príkazového riadku.
GUI	Graphical User Interface je zaužívaný pojem pre interakciu s uživateľom pomocou grafických prvkov.
WO	Work Order označuje pracovnú objednávku.
WF	Workflow označuje činnosť, ktorú má riadiaca jednotka vykonávať.
NG	No Good je výraz používaný predovšetkým v Japonsku (*ALPS* je japonská firma) ako opak k výrazu OK.
DPS	Doska plošných spojov
ISP	In-System Programming označuje programovanie mikroprocesora bez toho, aby musel byť vyňatý z DPS zariadenia.
EOL	End of Line je znak ASCII tabuľky označujúci koniec riadku, tiež sa používajú pojmy *line feed* (LF) alebo *newline*.

step-down step-down je napäťový menič, ktorý vytvára na výstupe nižšiu hodnotu napätia ako na vstupe.

shutdown shutdown je stav, keď je operačný systém vypnutý.

bit-banging bit-banging komunikačný protokol nie je spravovaný hardvérovo, ale softvérovo.

XHR XMLHttpRequest je aplikačné programátorské rozhranie pre skriptovacie jazyky internetových prehliadačov (napr. Javascript). Používa sa na odoslanie HTTP alebo HTTPS požiadaviek na server a na načítanie údajov z odpovede servera.

PLC Programmable Logic Controller je digitálny počítač používaný spravidla pre automatizáciu elektromechanických procesov vo výrobe.

PN Part Number označuje identifikátor výrobku.

URI Uniform Resource Identifier je reťazec znakov identifikujúci zdroj, ktorý je cieľom sieťovej požiadavky.

back end back end — Medzi hardvérom a koncovým uživateľom zvykne byť niekoľko vrstiev softvéru. Vyššia vrstva (bližšie ku koncovému uživateľovi) získava údaje z nižšej (bližšie k hardvéru) a prezentuje ich špecifickým spôsobom, ktorý je vhodný pre vrstvu nachádzajúcu sa nad ňou. Pre danú vrstvu sa vrstva, ktorá je pod ňou označuje ako back end, naopak vrstva, ktorá je nad ňou, sa označuje ako front end.

jig montážny prípravok

ZOZNAM PRÍLOH