

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

**Kalibrační a ovládací software sítě částicových pixelových
detektorů umístěných uvnitř experimentu ATLAS
na LHC v CERN**

Jakub Begera

Vedoucí práce: Ing. Štěpán Polanský

Studijní program: Otevřená informatika, Bakalářský

Obor: Softwarové systémy

17. května 2016

zadani

Poděkování

Zde můžete napsat své poděkování, pokud chcete a máte komu děkovat.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. 5. 2016

.....

Abstract

Translation of Czech abstract into English.

Abstrakt

Abstrakt práce by měl velmi stručně vystihovat její obsah. Tedy čím se práce zabývá a co je jejím výsledkem/přínosem.

Očekávají se cca 1 – 2 odstavce, maximálně půl stránky.

Obsah

1	Úvod	1
1.1	Motivace	1
2	Detektory ionizujícího záření	3
2.1	Klasifikace detektorů a jejich parametry	3
2.2	Geiger-Müllerův počítač	3
2.3	Sintilační detektory	3
2.4	Polovodičové detektory	3
2.4.1	Princip	3
2.4.2	Módy	3
2.4.3	Medipix	3
2.4.4	Timepix	3
2.4.5	Cluster analýza	3
2.4.6	FitPix	3
2.4.7	Pixelman	3
2.4.8	Aplikace	3
3	Energetická kalibrace	5
4	Atlas TPX	7
4.1	Atlas MPX	8
4.1.1	Hardwarová a softwarová architektura sítě Atlas MPX	9
4.2	Hardwarová architektura	10
4.3	Softwarová architektura	12
4.4	Řídící software a jeho implementace	14
4.4.1	Řízení detektorů	14
4.4.1.1	Komunikační protokol	14
4.4.1.2	Popis příkazů komunikačního protokolu	15
4.4.1.3	Synchronní a asynchronní příkazy komunikačního protokolu	19
4.4.1.4	Implementace modulu pro řízení detektorů na straně serveru	21
4.4.1.5	Emulátor detektoru	23
4.4.2	REST API server	25
4.4.2.1	Konfigurace a spuštění serveru	25
4.4.2.2	Metody poskytované serverem	26
4.4.2.3	Výkonost	27

4.4.3	Zpracování a ukládání dat	28
4.4.3.1	Data server	28
5	Závěr	29
A	Konfigurační soubor Atlas TPX serveru	31
B	Obsah přiloženého CD	35

Seznam obrázků

4.1	Atlas TPX detektor - vrstvy a rozmístění konvertorů	7
4.2	Atlas TPX - přehledem rozmístění detektorů	8
4.3	Snímek z Atlas MPX detektoru s výřezem zachycených částic (převzato z [?])	9
4.4	Fotografie znázorňující Medipix2 detektor s neutronovými konvertory (převzato z [?])	9
4.5	Atlas MPX - řídicí aplikace (převzato z [?])	10
4.6	Atlas TPX - diagram hw komponent	11
4.7	Atlas TPX - fotografie hw komponent	11
4.8	Atlas TPX - diagram softwarových komponent	12
4.9	Cluster analýza - 6 základních typů clusterů (převzato z [?])	14
4.10	Příklad použití komunikačního protokolu	20
4.11	Diagram tříd modulu pro ovládání detektorů	22
4.12	Emulátor detektoru - sekvenční diagram připojení klienta a pořízení snímku	24
B.1	Seznam přiloženého CD — příklad	35

Seznam tabulek

4.1	Komunikační protokol - struktura paketů z pohledu serveru	15
4.2	Komunikační protokol - přehled příkazů	15

Kapitola 1

Úvod

Tato bakalářská práce se zabývá návrhem a implementací software pro ovládání a kalibraci sítě hybridních částicových pixelových detektorů umístěných uvnitř experimentu Atlas na LHC v CERN - projekt AtlasTPX. Jelikož proces kalibrace je zcela nezávislý na následném řízení činnosti těchto detektorů, je tento software členěn na dvě nezávislé části - energetickou kalibraci částicových pixelových detektorů (viz kapitola 3) a řízení sítě těchto detektorů - AtlasTPX (viz kapitola 4).

TODO

1.1 Motivace

Ionizující záření je spjato s naším světem už od začátku jeho existence. Jeho studium započalo koncem 19. století a pomáhá nám pochopit podstatu hmoty, její interakce s prostředím a další vlastnosti. Tyto poznatky našli své uplatnění v mnoha oborech, jako například v defektoskopii, zdravotnictví, energetice a v mnoha dalších. Spolu s rostoucími poznatky o ionizujících záření a s technickým postupem se rozvíjela i detekční technika, která za poslední století prodělala veliký posun. Od prvních bublinových komor, až po nejmodernější polovodičové pixelové detektory, kterými se tato práce zabývá.

Kapitola 2

Detektory ionizujícího záření

2.1 Klasifikace detektorů a jejich parametry

2.2 Geiger-Müllerův počítač

2.3 Sintilační detektory

2.4 Polovodičové detektory

2.4.1 Princip

2.4.2 Módy

2.4.3 Medipix

2.4.4 Timepix

2.4.5 Cluster analýza

2.4.6 FitPix

2.4.7 Pixelman

2.4.8 Aplikace

Kapitola 3

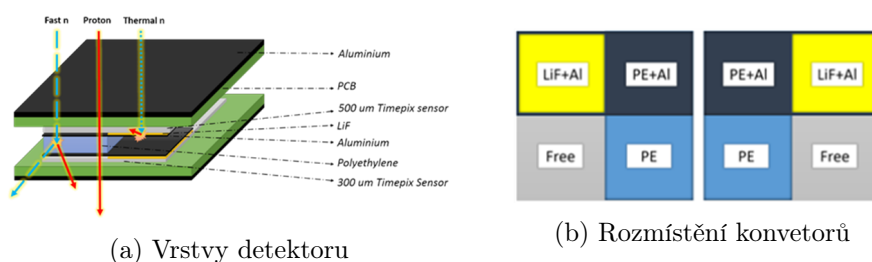
Energetická kalibrace

Kapitola 4

Atlas TPX

Atlas TPX, síť 16¹ hybridních částicových pixelových detektorů typu Timepix 2.4.4, nainstalovaných na různé pozice experimentu Atlas na LHC² v CERN během LS2³ (leden 2013 až březen 2015) je následníkem svého předchůdce - sítě Atlas MPX (viz 4.1). Hlavní motivační výměny této sítě bylo využití nových technologií, především pak nového detekčního čipu Timepix. Ten na rozdíl od svého předchůdce Medipix2 2.4.3 umožňuje rozšíření naměřené informace i o časovou oblast (viz 2.4.4). To nově umožňuje provozovat detektory v módech TOA⁴ a TOT⁵.

Další změnou oproti svému předchůdci je, že každý detektor obsahuje dva detekční čipy s tloušťkami 300 μm a 500 μm , umístěné předními stranami k sobě - viz 4.1a. To přináší možnost měřit koincidence - když částice projde oběma vrstvami detektoru a zároveň v každé nechá jisté měřitelné množství své energie, je detekována oběma vrstvami a je možné zpětně zrekonstruovat její trajektorii. Tyto koincidence se nejsnáze detekují, pokud oba Timepix čipy pracují v módu TOA - jelikož rychlost částic se blíží rychlosti světla, je vysoce pravděpodobné, že zasažené pixely budou mít stejnou hodnotu.



Obrázek 4.1: Atlas TPX detektor - vrstvy a rozmístění konvertorů

¹V průběhu LS3³ (plánováno 2017 - 2018) je plánováno rozšíření této sítě o nové detektory

²z angl. Large Hadron Collider

³z angl. long shutdown - dlouhodobá technologická přestávka LHC

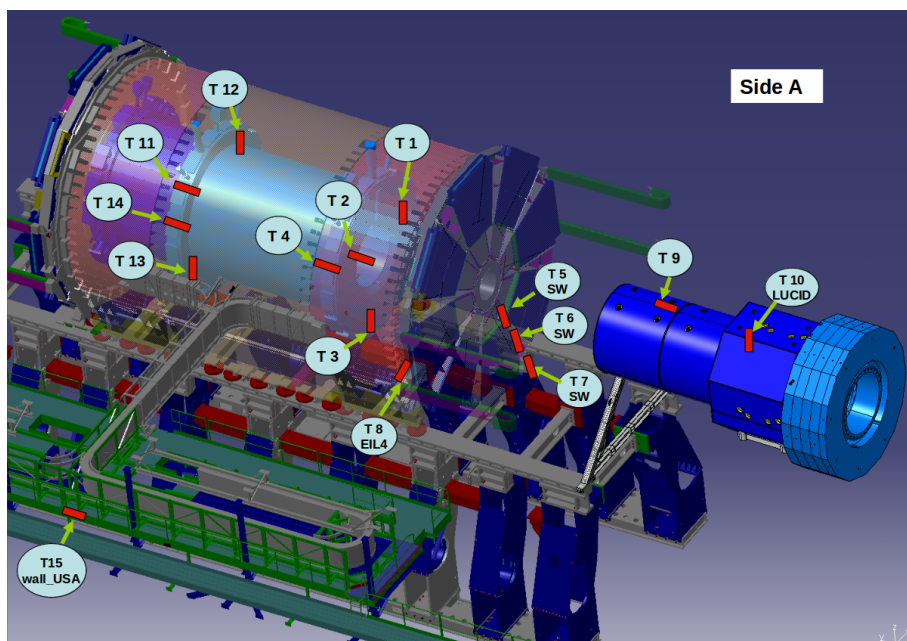
⁴z angl. Time of Arrival - čas přiletu částice v hodinových cyklech detektoru od začátku akvizice

⁵z angl. Time Over Threshold - počet hodinových cyklů, kdy komparační napětí je větší, než referenční (ekvivalent energie deponované částice, viz kapitola 3)

Mezi vrstvami detektoru je umístěn konvertující materiál pro detekci termálních a rychlých neutronů. Rozmístění těchto konvertorů je na obrázku 4.1b.

Hlavním úkolem Atlas TPX experimentu je online monitorování spektrálních charakteristik velice různorodého radiačního prostředí Atlas experimentu, založené na prostorovém uspořádání sítě a (vzhledem k aktuálnímu módu detektoru) i na informaci o deponované energii zainteragovaných částic, či na času jejich interakce.

Detektory, instalované blízko interakčnímu bodu, jsou rovněž použity jako monitory integrované luminozity, což je veličina, která udává počet realizovaných srážek, resp. s intenzitou svazku urychlovače. Podle [?] je to veličina, která v případě srážení dvou proti sobě letících svazků ukazuje, jaký je součin počtů částic v jednotlivých svazcích prolétajících jednotkovou plochou v srážkové oblasti, vynásobený počtem obletů svazků za jednotku času (nejčastěji se vyjadřuje v jednotkách na centimetr čtvereční a sekundu).



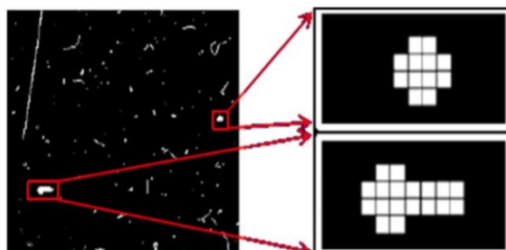
Obrázek 4.2: Atlas TPX - přehledem rozmístění detektorů

4.1 Atlas MPX

Atlas MPX[?] [?] je předchůdcem detektorové sítě Atlas TPX, který je v současné době plně nahrazen. Detektorová síť Atlas MPX se skládala z 16 Medipix2 detektorů, které byly instalovány na různé pozice Atlas detektoru. Hlavním cílem této sítě bylo měření vlastností radiačního pole uvnitř experimentu Atlas, jeho složení, spektroskopických charakteristik a částečně také přispěla k měření neutronů.

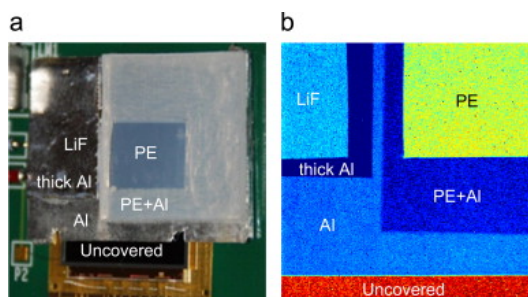
Všechny detektory operovaly v tzv. Medipix módu, který se vyznačuje tím, že v rámci jedné akvizice počítá počet částic, které interagovaly pixelovou maticí detektoru a jejichž deponovaná energie byla vyšší, než prahová. Na obrázku 4.3 je znázorněn snímek z jednoho

detektoru s detailem zachycených částic. Na obrázku vpravo nahoře je částice typu **heavy blob** (těžce nabitá částice, jejíž trajektorie byla kolmá s povrchem detektoru), vpravo dole je pak zachycena částice typu **heavy track** (také těžce nabitá částice, která ale přiletěla pod větším a proto zanechala větší stopu) - více klasifikaci částic se dočtete v podkapitole 4.3.



Obrázek 4.3: Snímek z Atlas MPX detektoru s výřezem zachycených částic (převzato z [?])

Každý z těchto detektorů byl osazen $300\ \mu\text{m}$ tlustým křemíkovým senzorem, který byl pokryt konvertory pro lepší detekční účinnost neutronů (obr. 4.4).

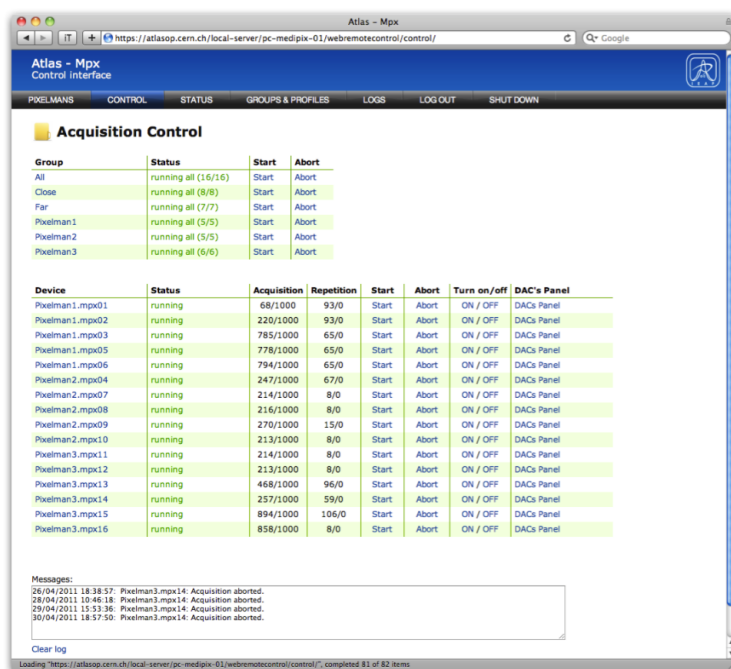


Obrázek 4.4: Fotografie znázorňující Medipix2 detektor s neutronovými konvertory (převzato z [?])

4.1.1 Hardwarová a softwarová architektura sítě Atlas MPX

Tato síť se skládala z 16 Medipix2 2.4.3 detektorů, které byly pomocí USB vyčítacího rozhraní FITPix 2.4.6 připojeny ke třem počítačům (z důvodu distribuce toku dat a výkonu). Na každém počítači se o komunikaci s detektory staral software Pixelman 2.4.7, který řídil akvizici dat, nastavování parametrů detektorů apod.

Pro vzdálené ovládání byl vyvinut plugin pro Pixelman, který umožňoval jeho rozšíření o TCP/IP ovládací vrstvu. Pomocí jednoduchého textového protokolu bylo tedy možné řídit každý ze třech uzlů. Pro tyto účely byla vyvinuta centrální řídicí aplikace [?], pomocí které bylo možné řídit akvizici všech detektorů a nastavovat jejich parametry. Tato aplikace poskytovala webové rozhraní (obr. 4.5), které díky tou dobou méně striktní CERNské politice síťové bezpečnosti bylo možné tento experiment ovládat odkudkoliv z internetu.



Obrázek 4.5: Atlas MPX - řídicí aplikace (převzato z [?])

4.2 Hardwarová architektura

Při návrhu hardwarové architektury sítě Atlas TPX musela být zohledněna zvýšená intenzita radiačního a elektromagnetického pole v prostorách Atlas detektoru. Snahou proto bylo, co nejvíce hardwarových komponent umístit z dosahu tohoto pole. Z pohledu hardwarové instalace této detektorové sítě se prostory Atlas experimentu dělí na dvě části - UX15 a USA15 (viz obr. 4.6). V UX15 se nachází vlastní experiment. V tomto prostoru byly umístěny jen detektory (na obr. 4.6 TPX01 až TPX15) a zbytek sítě byl instalován v USA15, kterou od zbytku experimentu dělí cca 60 m tlustá železobetonová stěna. Tady se nachází vyčítací elektronika a další nezbytný hardware.

Na obrázku 4.7 je fotografie těchto komponent. Jak již bylo zmíněno výše, detektor (z obr. 4.7, na obr. 4.6 jako TPX01 až TPX15) se skládá z dvojice detekčních čipů Timepix, které jsou pomocí LVDS zesilovačů a cca 100 m dlouhých ethernetových kabelů propojeny se zařízením AtlasPix (obr. 4.7 dole), které vzniklo modifikací vyčítacího rozhraní FITPix 2.4.6. Toto zařízení obsahuje FPGA⁶, minipočítač Raspberry Pi a další podpůrnou elektroniku.

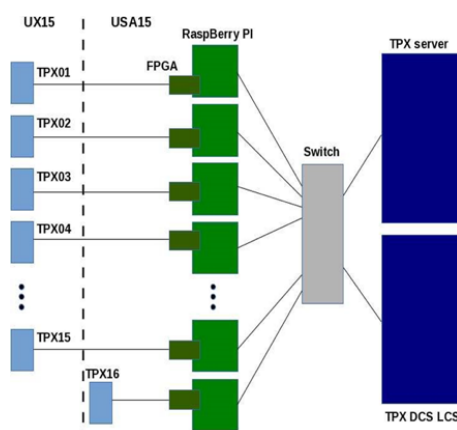
FPGA se stará o komunikaci s Timepix detektory, v rámci které dochází k nastavování řídicích registrů Timepix čipů, ovládání akvizice, vyčítání dat, řízení triggeru⁷ apod.

Dalším článkem tohoto řetězce je minipočítač Raspberry Pi, který plní dvě úlohy. Tou první je komunikace s FPGA pomocí SPI⁸ rozhraní a deserializace (získání dat ze struktury

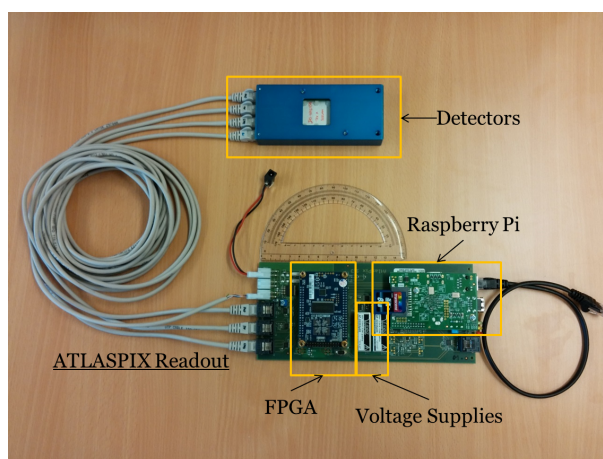
⁶z angl. Field Programmable Gate Array (programovatelné hradlové pole)

⁷řídicí signál, který spouští resp. zastavuje (dle konfigurace) akvizici detektoru

⁸z angl. Serial Peripheral Interface (sériové periferní rozhraní)



Obrázek 4.6: Atlas TPX - diagram hw komponent



Obrázek 4.7: Atlas TPX - fotografie hw komponent

komunikačního protokolu) a derandomizace (není zaručena časová posloupnost) surových dat z FPGA. Druhou úlohou tohoto zařízení je poskytování API⁹ vyšším řídicím vrstvám této sítě pomocí specifikovaného komunikačního protokolu a klasického ethernetového rozhraní.

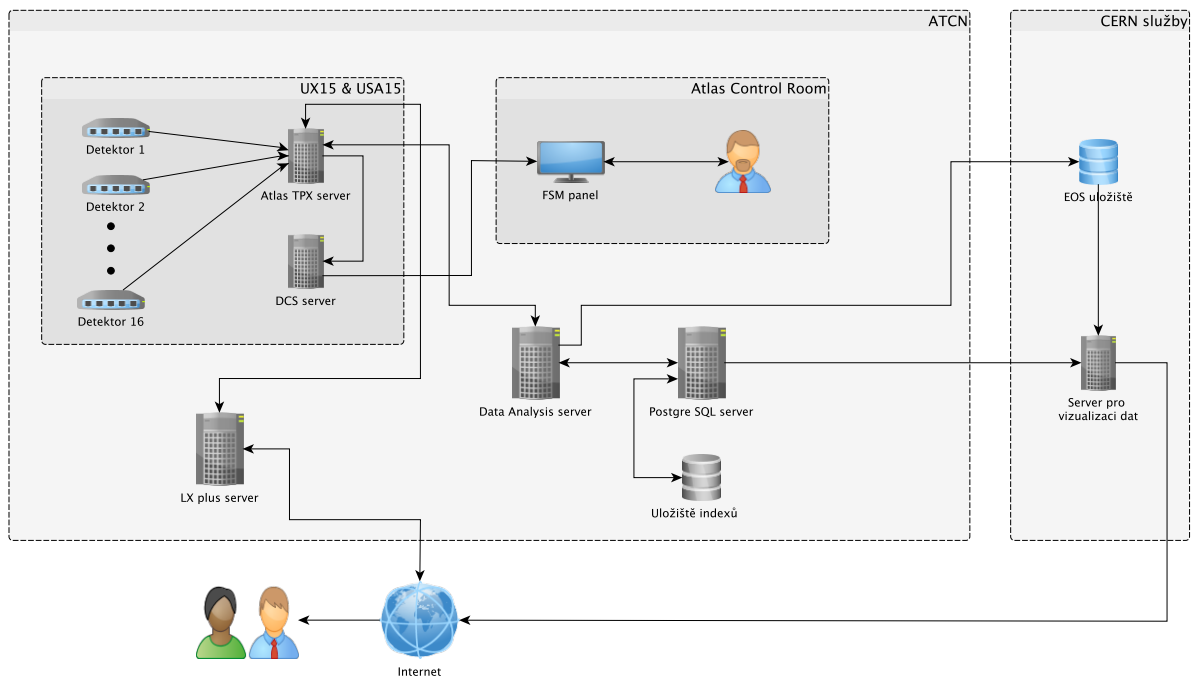
Všechny tyto zařízení jsou pomocí ethernetového switchu propojeny s TPX serverem, centrálním bodem této sítě, který jí pomocí řídicího softwaru 4.4 a komunikačního protokolu 4.4.1 ovládá. Zároveň je k síti připojen i TPX DCS¹⁰ server, pomocí kterého jsou různé stavové informace Atlas TPX sítě předávány CERNu, resp. Atlas experimentu. Tyto stavové informace jsou převážně hardwarového charakteru (na př. napětí, časování apod.), ale také jsou předávána data o počtu pořízených snímků, jejich okupanci apod.

⁹z angl. Application Programming Interface (aplikační programovací rozhraní)

¹⁰z angl. Data Control System

4.3 Softwarová architektura

Na obrázku 4.8 je znázorněn diagram návrhu softwarové architektury sítě Atlas TPX z pohledu jejího řízení, vizualizace dat a předávání stavových informací CERNu. Diagram je členěn do dvou základních částí - ATCN¹¹ (technická síť Atlas experimentu, která je oddělena od zbytku Atlas sítě a obsahuje systémy pro vyčítání dat a pro řízení, včetně TDAQ¹² a DSC [?]) a CERN služby, které poskytují perzistentní úložiště dat a web server pro jejich vizualizaci.



Obrázek 4.8: Atlas TPX - diagram softwarových komponent

Popis architektury z pohledu řízení: Na obrázku 4.8 se nachází Atlas TPX server, který je umístěn v serverové místnosti (USA15) Atlas experimentu, umístěné cca 100 m pod zemským povrchem. Tento server pomocí komunikačního protokolu (specifikovaném v 4.4.1) řídí činnost detektorů (nastavování parametrů, ovládání akvizice apod.). Zároveň pomocí JSON REST API poskytuje rozhraní pro své řízení a předávání stavových informací (více v 4.4.2). Díky tomuto rozhraní je možné činnost serveru řídit z ATCN sítě. Pro potřeby vzdáleného ovládání mimo síť ATCN slouží LX plus server, který zajistí spojení vytvořením SSH tunelu.

Předávání stavových informací zajišťuje DCS server, který je od Atlas TPX serveru získává pomocí jeho API. Hlavním úkolem DCS je zajištění získávání stavových informací ze všech experimentů a detektorů homogenním způsobem a interakce s LHC

¹¹z angl. ATLAS Technical Control Network

¹²z angl. Trigger and Data Aquisition (trigger a akvizice dat)

(předávání dat luminozité, stavu svazku urychlovače, radiační pozadí apod.). Tato data jsou dále předávána do Atlas Control Room, která se nachází na povrchu. Tam jsou tato data operátorů prezentována pomocí FSM panelu, což je aplikace, která vizualizuje stromovou strukturu všech systému a detektorů Atlas experimentu. Každý list této stromové struktury (detektor, senzor atd.) má několik proměnných, z nichž každá má předem definované intervaly s příslušnými stavy (OK, WARNING, ERROR, FATAL atd.). Výhodou této struktury je, že pokud kterýkoliv list změní svůj stav, tak se tato informace propaguje přes všechny nadřazené uzly, tudíž odhalení případné chyby je pro operátory mnohem snazší.

Popis architektury z pohledu analýzy a vizualizace dat: Když kterýkoliv detektor dokončí akvizici snímku, tak vygeneruje a pošle asynchronní událost Atlas TPX serveru s informací, že data jsou připravena k vyčtení. Následně server vyčte snímek z detektoru (i s jeho metadaty), zpracuje a připojí k němu informace o nastavení detektoru. Poté je třeba data přenést do Data analysis serveru, což v principu je možné dvěma¹³ způsoby:

1. Atlas TPX server uloží získaná data v textové podobě do lokálního (či síťového) datového úložiště, odkud jsou přenesena do Data analysis serveru pomocí automatického kopírovacího skriptu.
2. Druhou možností je přenesení dat pomocí JSON REST API protokolu, který je Data analysis serverem implementován. Tento druhý způsob je výhodnější, neb minimalizuje prodlevu mezi dobou pořízení snímku a následným zpracováním Data analysis serverem a dostupnosti jeho vizualizace pomocí web serveru a zároveň přináší úsporu objemu přenesených dat.

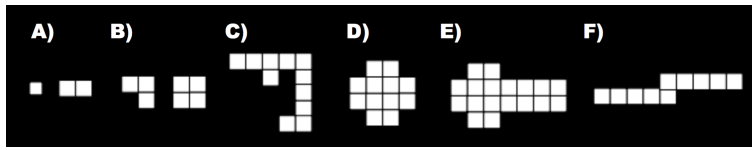
Hlavní úlohou Data analysis serveru je provedení tzv. Cluter analýzy. Jde o proces, při kterém jsou z každého snímku získány shluky sousedních pixelů (tzv. clusterů), které mají nenulovou hodnotu. Z těchto clusterů, resp. z jejich tvaru a celkové deponované energie částice (pokud zasažení pixely operovaly v TOT módu) je možné zjistit typ částice, která danou událost způsobila. Na obrázku 4.9 můžete vidět 6 základních dělení clusterů, kde

- (a) je tzv. DOT, způsobený fotony, či elektrony o energii do 10 keV
- (b) je tzv. SMALL BLOB, způsobený fotony, či elektrony s energií většinou nad 10 keV
- (c) je tzv. CURLY TRACK, způsobený elektrony do 10 MeV
- (d) je tzv. HEAVY BLOBS, způsobený těžce nabitými částicemi (na př. alfa)
- (e) je tzv. HEAVY TRACK, způsobený těžce nabitými částicemi (na př. protony)
- (f) je tzv. STAIGHT TRACK, způsobený těžce nabitými částicemi (muony apod.)

Po dokončení analýzy dat, jsou data uložena do ROOT¹⁴ souborů (do souborové struktury po detektorech a hodinách). ROOT je framework, který je vyvíjen v CERN a je určený pro

¹³V současné době je možný pouze první způsob, protože díky politice CERNské síťové bezpečnosti a dlouhými schvalovacími termíny je Data analysis server a všechny s ním související systémy (web server, databáze s indexem a vlastní úložiště dat) prozatím umístěn v ÚTEF ČVUT v Praze.

¹⁴<https://root.cern.ch/>



Obrázek 4.9: Cluster analýza - 6 základních typů clusterů (převzato z [?])

ukládání dat a jejich analýzu. Vygenerované soubory jsou ukládány do EOS úložiště, což je služba pro perzistentní ukládání ROOT souborů provozovaná CERN.

Jelikož každý ROOT soubor má velikost řádově v jednotkách GB , jakékoliv operace nad nimi (na příklad vyhledávání) jsou velice časově náročné. Z tohoto důvodu vznikla PostgreSQL databáze s indexem na jednotlivé clustery, obsažené v ROOT souborech. Pro vizualizaci dat slouží web server, který pomocí databáze s indexem a ROOT souborů poskytuje online výsledky cluster analýzy a další informace.

4.4 Řídicí software a jeho implementace

Tato kapitola je věnována návrhu a implementaci řídicího software sítě Atlas TPX, který je nasazen na Atlas TPX serveru (viz obr. 4.8). Úkolem tohoto software se zajištění komunikace s detektory, zejména pak řízení akvizice, nastavování parametrů a vyčítání dat - tato komunikace bude popsána v kapitole 4.4.1. Další úlohou tohoto software je poskytování rozhraní pro řízení své činnosti a pro předávání stavových informací o Atlas TPX síti. Toto rozhraní je implementováno pomocí JSON REST API a detailně bude popsáno v kapitole 4.4.2.

TODO

4.4.1 Řízení detektorů

Jak již bylo zmíně výše, detektor se skládá ze dvou Timepix detekčních čipů, FPGA a minipočítače Raspberry Pi, který implementuje komunikační protokol, díky kterému umožňuje řídicímu software ovládání činnosti detektoru. V podkapitole 4.4.1.1 naleznete popis tohoto komunikačního protokolu.

4.4.1.1 Komunikační protokol

Tabulka 4.1 znázorňuje strukturu komunikačního rámce (tzv. paketu) tohoto protokolu pomocí posloupnosti bytů z pohledu Atlas TPX serveru, resp. z pohledu řídicího software. V horní části se nachází struktura odchozího paketu, kde:

0x55¹⁵ je tzv. START BYTE, který značí začátek paketu,

CMD je typ příkazu (tzv. COMMAND TYPE) - viz tabulka přehledu příkazů 4.2,

¹⁵značení v hexadecimální soustavě

SIZE 1..4 je pole vždy o velikosti čtyřech bytů značících velikost (resp. počet bytů) položky DATA, zakódovaných v BIG ENDIAN,

DATA 1 .. DATA n je pole vlastních přenesených dat o velikosti n ,

0xAA STOP BYTE, který značí konec paketu.

Struktura odchozího paketu je velice podobná, až na byte ERR, který oproti příchozího paketu obsahuje. Tento byte může nabývat hodnoty 0x00 (když zpracování požadavku serveru detektorem proběhlo bez chyby), nebo 0x01 (jinak).

0x55	CMD	SIZE 1	SIZE 2	SIZE 3	SIZE 4	DATA 1 .. DATA n		0xAA	} odchozí paket	
0x55	CMD	ERR	SIZE 1	SIZE 2	SIZE 3	SIZE 4	DATA 1 .. DATA n		0xAA	} příchozí paket

Tabulka 4.1: Komunikační protokol - struktura paketů z pohledu serveru

Hodnota příkazu	Název příkazu
0x01	Ping
0x02	Get status of the detector
0x03	Reset of the device
0x04	Set Bias and Timepix Clock
0x05	Get Bias and Timepix Clock
0x06	Set Pixel Configuration
0x07	Get Pixel Configuration
0x08	Set DAC
0x09	Get DAC
0x0A	Perform Digital Test
0x0B	Perform Acquisition
0x0C	Readout Measured Data
0x0D	Direct FITPix Command
0x0E	Stop acquisition
0xFD	Asynchronous Event from device
0xFE	Reboot device
0xFF	Shut down

Tabulka 4.2: Komunikační protokol - přehled příkazů

4.4.1.2 Popis příkazů komunikačního protokolu

Následuje stručný popis příkazů komunikačního protokolu z pohledu obsahu vlastních přenesených dat odchozího a příchozího paketu (viz tabulka 4.1)

0x01 - Ping : Příkaz pro ověření spojení s detektorem. Na základě rozdílu času odeslání a přijetí paketu je vypočtena prodleva spojení v *ns* (tzv. ping)

Odchozí data: nic

Příchozí data: nic

0x02 - Get status of the detector : Příkaz pro zjištění stavu detektoru.

Odchozí data: nic

Příchozí data (2 B): GST MST

- GST - General Status (obecný status)
 - 0x00 - Ok
 - 0x01 - Chyba detekčního čipu
 - 0x02 - Obecná chyba
- MST - Measurement Status (měřicí status)
 - 0x00 - Nečinný
 - 0x01 - Probíhá akvizice
 - 0x02 - Čekání na trigger
 - 0x03 - Data připravena k vyčtení
 - 0x04 - Chyba akvizice

0x03 - Reset of the device : Tento příkaz slouží pro vyresetování FPGA a dalších řídicích struktur detektoru.

Odchozí data: nic

Příchozí data: nic

0x04 - Set Bias and Timepix clock : Příkaz nastavující napětí (Bias) na obou Timepix čipech a jejich měřicí frekvenci (Timepix clock)

Odchozí data (24 B): BIAS1(8 B) BIAS2(8 B) CLK(8 B)

- BIAS1, BIAS2 - hodnota napětí pro Timepix čipy (zakódovaná jako 64-bitové double-precision číslo dle standartu IEEE 754)
- CLK - Měřicí frekvence obou Timepix čipů (zakódovaná jako 64-bitové double-precision číslo dle standartu IEEE 754)

Příchozí data: nic

0x05 - Get Bias and Timepix clock : Příkaz pro vyčtení úrovně napětí z obou Timepix čipů a jejich měřicí frekvenci

Odchozí data: nic

Příchozí data (24 B): BIAS1(8 B) BIAS2(8 B) CLK(8 B)

- BIAS1, BIAS2 - hodnota napětí pro Timepix čipy (zakódovaná jako 64-bitové double-precision číslo dle standartu IEEE 754)
- CLK - Měřicí frekvence obou Timepix čipů (zakódovaná jako 64-bitové double-precision číslo dle standartu IEEE 754)

0x06 - Set Pixel Configuration : Příkaz pro nastavení konfigurace pro každý pixel detektoru

Odchozí data $((1 + (65536 \text{ nebo } 2 * 65536)) B)$: TYPE(1 B) PIXCFG((65536 nebo 2*65536) B)

- TYPE - Výběr čipu, pro který je konfigurace určena
 - 0x00 - Konfigurace určena oboum čipům (délka PIXCFG je 2 * 65536 B)
 - 0x01 - Konfigurace určena prvnímu čipu (délka PIXCFG je 65536 B)
 - 0x02 - Konfigurace určena druhému čipu (délka PIXCFG je 65536 B)
- PIXCFG - Pole konfiguračních bytů (každý byte je pro jeden pixel) pro jednotlivé pixely. Délka tohoto pole je závislá na parametru TYPE. Struktura bytu pro konfiguraci pixelu je následovná:
MASK_BITE(1 b) TEST_BITE(1 b) THL(4 b) MODE(2 b)
 - MASK_BITE - Pixel je aktivní při hodnotě 1, zamaskovaný při 0.
 - TEST_BITE - **TODO**
 - THL - Tyto čtyři bity udávají číslo od 0 do 15, které je použito pro posun globální hodnoty THL
 - MODE - Mód pixelu (0 - Medipix, 1 - Time-Over-Treshold, 2 One-Hit, 3 - Time-of-Arrival)

Příchozí data: nic

0x07 - Get Pixel Configuration : Příkaz pro vyčtení konfigurace pixelů detektoru.

Odchozí data (1 B): TYPE - Výběr čipu, pro který je konfigurace určena (viz předchozí příkaz)

Příchozí data ((65536 nebo 2 * 65536) B): PIXCFG (viz předchozí příkaz)

0x08 - Set DAC : Příkaz pro nastavení hodnot DAC převodníků detektoru.

Odchozí data (3 B): DAC_IDX DAC_VAL DAC_VAL

- DAC_IDX - DAC index
- DAC_VAL - DAC hodnota

Příchozí data: nic

0x09 - Get DAC : Příkaz pro vyčtení hodnoty DAC převodníků detektoru.

Odchozí data (1 B): DAC_IDX

- DAC_IDX - DAC index

Příchozí data (2 B): DAC_VAL DAC_VAL

- DAC_VAL - DAC hodnota

0x0A - Perform Digital Test : Příkaz pro vykonání testu digitálních částí detektoru.

Odchozí data: nic

Příchozí data (3 B): BPC BPC BPC

- BPC - Počet chybných pixelů detektoru

0x0B - Perform Acquisition : Příkaz pro zahájení akvizice snímku/snímků.

Odchozí data (13 B): ACQTM(8 B) ACQCNT(4 B) TGRMOD

- ACQTIM - Akviziční čas v sekundách (zakódovaný jako 64-bitové double-precision číslo dle standartu IEEE 754).
- ACQCNT - Počet snímku (pokud je 0, detektor bude opakovat akvizici s těmito parametry až do její zastavení)
- TRIGMOD - Mód triggeru
 - 0x00 - bez triggeru
 - 0x01 - akvizice zahájena na signál triggeru
 - 0x02 - akvizice ukončena na signál triggeru

Příchozí data: nic

0x0C - Read Measured Data : Tento příkaz slouží pro vyčtení naměřených dat z detektoru.

Odchozí data (6 B): DET TYPE FRAME_ID FRAME_ID FRAME_ID FRAME_ID

- DET - selektor Timepix čipu
 - 0x00 - oba čipy
 - 0x01 - jen první čip
 - 0x02 - jen druhý čip
- TYPE - typ vyčítaných dat
 - 0x00 - deserializovaný a derandomizovaný snímek
 - 0x01 - surová data z FPGA
 - 0x02_ID - metadata k danému snímku (akviziční čas, napětí apod.)
- FRAME_ID - ID naměřeného snímku

Příchozí data: Velikost a typ příchozích dat se liší dle použitých parametrů DET a TYPE

- Pro TYPE 0x00 přijde pole bytů hodnot čítačů jednotlivých pixelů (hodnota každého pixelu je reprezentována pomocí dvou bytů)
- Pro TYPE 0x01 přijdou blíže nespecifikovaná surová data z FPGA
- Pro TYPE 0x02 přijdou metadata, vázající se k danému snímku s následující strukturou:
 - UNIX časové razítko začátku akvizice [*ms*] (5 B)
 - Bias1 - měřící napětí prvního čipu [*V*] (8 B, double-precision dle IEEE 757)
 - Bias2 - měřící napětí druhého čipu [*V*] (8 B, double-precision dle IEEE 757)
 - Měřící frekvence [*Hz*] (8 B, double-precision dle IEEE 757)
 - Doba akvizice [*s*] (8 B, double-precision dle IEEE 757)

0x0D - Direct FPGA Command : Příkaz pro přímé poslání dat do FPGA a získání odpovědi.

Odchozí data: Dle použitého příkazu

Příchozí data: Dle použitého příkazu

0x0E - Stop acquisition : Příkaz pro zastavení aktuálně probíhající akvizice.

Odchozí data (1 B): TYPE

- TYPE: Typ zastavení
 - 0x00 - Zastavení akvizice po dokončení aktuálně pořizovaného snímku
 - 0x01 - Bezprostřední zastavení akvizice

Příchozí data: nic

0xFD - Asynchronous Event From Device : Tento typ příkazu je asynchronní a detektor ho posílá samovolně dle nastalé události - na příklad dokončení akvizice.

Příchozí data (5 B): EVID VAL VAL VAL VAL

- EVID - Typ nastalé události (na př. 0x00 pro událost dokončení akvizice)
- VAL - Doplnující data (na př. ID snímku pro událost dokončení akvizice)

0xFE - Reboot of the Device : Příkaz pro restartování detektoru.

Odchozí data: nic

Příchozí data: nic

0xFF - Shutdown of the Device : Příkaz pro vypnutí detektoru.

Odchozí data: nic

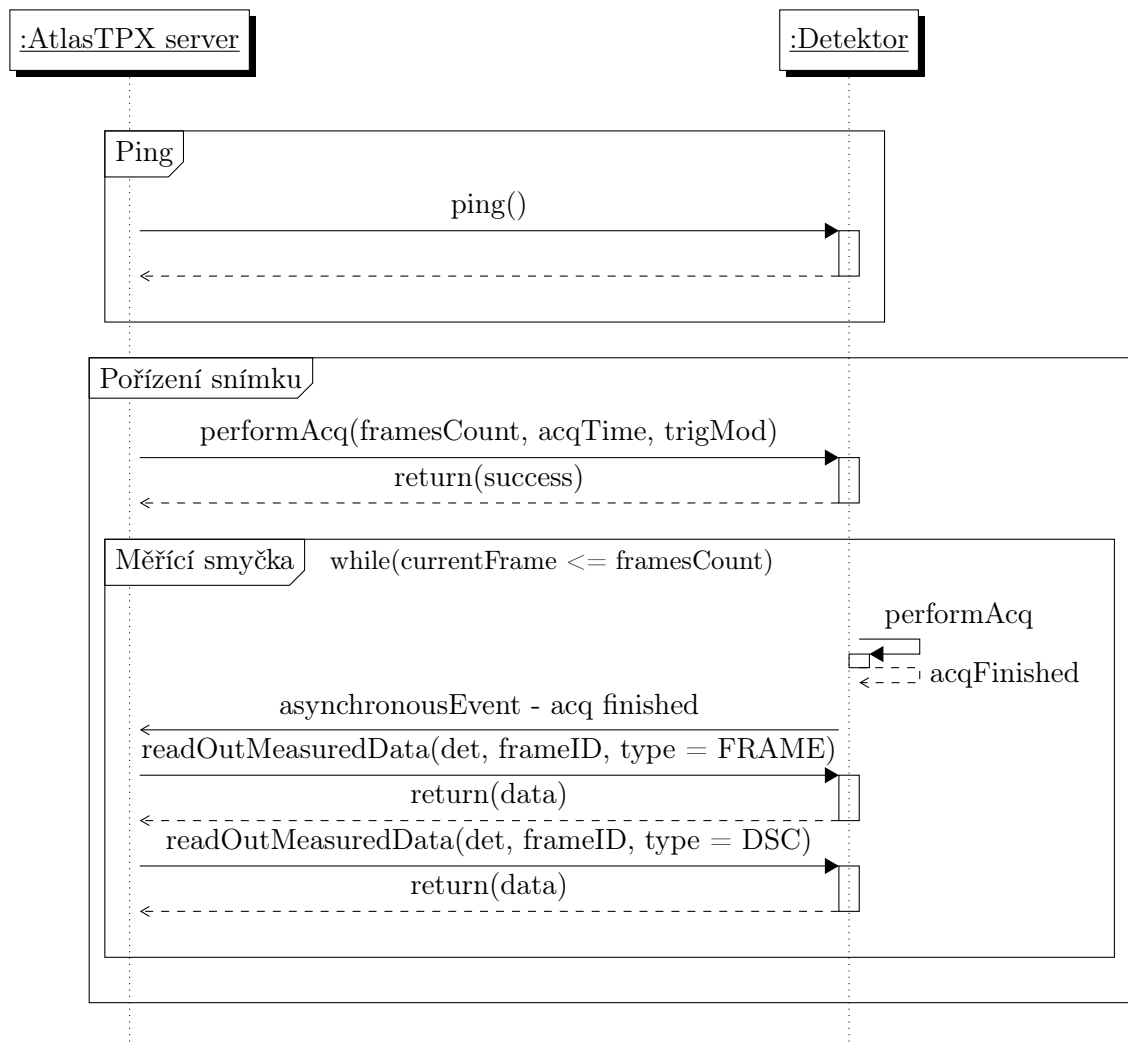
Příchozí data: nic

4.4.1.3 Synchronní a asynchronní příkazy komunikačního protokolu

Téměř všechny příkazy komunikačního protokolu jsou synchronní, tzn. server vyše odchozí paket (s typem příkazu a jeho daty), který detektor zpracuje a bezprostředně vygeneruje a pošle odchozí paket (s příslušnými daty, typem příkazu a informací, zda-li při jeho zpracování došlo k chybě). Příkladem této synchronní komunikace je příkaz ping na obrázku 4.10 nahoře.

Vedle synchronních příkazů existuje i jeden asynchronní - 0xFD - Asynchronous Event From Device. V současné verzi komunikačního protokolu je tento příkaz využit jen pro oznámení serveru, že detektor dokončil akvizici a má data připravena k vyčtení.

Na obrázku 4.10 je znázorněn příklad kombinace synchronní a asynchronní komunikace pro pořízení snímku. Nejprve server vyše synchronní příkaz s žádostí o provedení akvizice (s parametry: doba akvizice, počet snímků a mód triggeru) na který hned dostane odpověď. Následuje vyčítací smyčka - detektor udělá akvizici snímku (pokud již všechny neudělal) a



Obrázek 4.10: Příklad použití komunikačního protokolu

vyšle serveru asynchronní příkaz s kódem právě dokončené akvizice a s ID¹⁶ snímku. Tuto asynchronní zprávu server zachytí a provede vyčítací sekvenci (pomocí příkazu `0x0C` - `Read Measured Data`), jejíž kroky se mohou lišit dle konfigurace. Ve výchozím nastavení tato sekvence vypadá následovně:

1. Vyčtení vlastního snímku (hodnot jednotlivých pixelů).
2. Vyčtení metadat (tzv. DSC). Tato data obsahují dodatečné informace ke snímku, jako na příklad přesnou dobu akvizice, čas začátku akvizice a měřicí napětí a frekvenci Timepix čipů.

¹⁶unikátní číslo snímku od spuštění detektoru

4.4.1.4 Implementace modulu pro řízení detektorů na straně serveru

V této podkapitole bude popsána implementace řízení detektorů v řídicím softwaru sítě Atlas TPX. Celý software byl implementován v jazyce JAVA za pomoci build nástroje Maven¹⁷ a knihoven Dropwizard¹⁸, Retrofit¹⁹ a RxJava²⁰.

Jak již bylo zmíněno výše, detektory jsou připojeny k Atlas TPX serveru přes ethernetové rozhraní a pomocí TCP/IP protokolu je vlastní komunikace realizována pomocí výše popsaného komunikačního protokolu. Z pohledu navazování spojení byla použita architektura klient-server tak, že detektor plní roli serveru a Atlas TPX server zase klienta. Tato architektura bylo použita s ohledem na robustnost a stabilitu této sítě a aby úloha navazování spojení zůstala v kompetenci Atlas TPX serveru. Když by na příklad došlo k přerušení napájení nebo jiné chybě jednoho z detektorů, Atlas TPX server by nebyl schopen se pokusit o znovu navázání spojení.

Na obrázku 4.11 můžete vidět diagram tříd modulu pro práci s detektory řídicího softwaru. Tento diagram není úplný a obsahuje jen několik nejdůležitějších tříd, zbytek je k nalezení na příloženém CD.

Při návrhu tohoto modulu bylo vycházeno z návrhového vzoru Model-View-Controller [?], resp. z jeho modifikace Model-Controller, protože tento modul žádnou prezentační vrstvu nemá.

Začněme popisem balíčku `model`. Jeho nejdůležitější třídou je třída `DetectorUnit`, která uchovává všechny informace o jednom detektoru, jako na příklad název, `tpxID`, `unitID`, ip adresu, port, `SettingsStorage` (úložiště nastavení detektoru, jako na příklad `bias`, `clock`, parametry akvizice, DAC hodnoty atd.) a `HwStatus` (tento objekt nese informace o posledních naměřených údajích získaných z detektoru, jako třeba aktuální hodnoty `bias`, `ping`, obecný a měřící status apod.).

Pro uchovávání všech instancí třídy `DetectorUnit` slouží singleton [?] třída `DetectorUnitStorage`, která tyto instance uchovává v kolekci `HashMap`, jejímž klíčem je `TpxID` (`Integer`). Instanci tohoto singletonu je možné získat pomocí třídy `DetectorUnitStorageFactory`, resp. pomocí její statické metody `getStorage()`. Před prvním použitím je třeba nejprve toto úložiště inicializovat pomocí statické metody `DetectorUnitStorageFactory.initStorage(String initFilePath)`, které se coby parametr předá cesta v souborovém systému k "*.csv" souboru s tabulkou s detektory. Každý řádek tohoto souboru reprezentuje jeden detektor a je v následujícím formátu:

```
název_detektoru;ip_adresa;unit_id;tpx_id;port
```

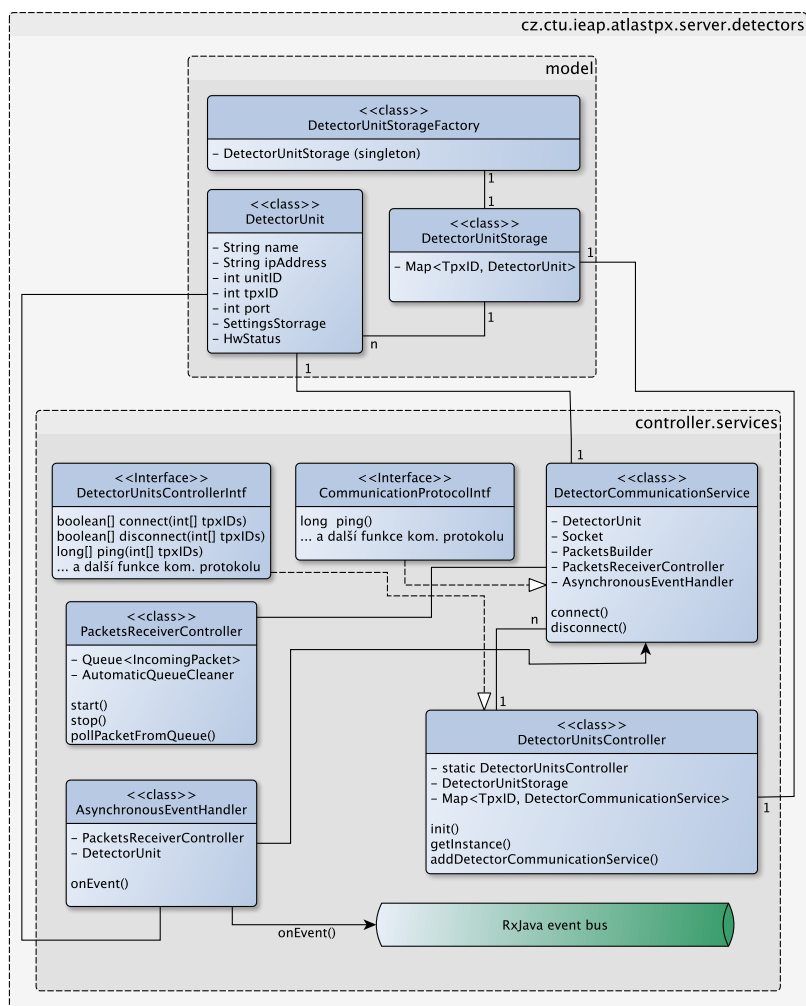
V balíčku `controller.services` se nachází vlastní logika tohoto modulu. Třída `DetectorCommunicationService` se stará o vlastní komunikaci s detektorem (jedna instance = jeden detektor). Tato třída obsahuje instanci třídy `DetectorUnit` (předané v konstruktoru), která mimo jiné obsahuje IP adresu a port příslušného detektoru. Tyto parametry se používají v metodě `connect()`, ve které se vytvoří spojení s detektorem (`socket`, `BufferedOutputStream`, `BufferedInputStream` apod.). Pro odpojení detektoru zase slouží metoda `disconnect()`.

¹⁷ <<https://maven.apache.org/>>

¹⁸ <<http://www.dropwizard.io>>

¹⁹ <<http://square.github.io/retrofit/>>

²⁰ <<https://github.com/ReactiveX/RxJava>>



Obrázek 4.11: Diagram tříd modulu pro ovládání detektorů

Krom metod pro síťovou komunikaci obsahuje i všechny metody implementované z interface `CommunicationProtocolIntf` - všechny synchronní metody komunikačního protokolu.

Pro příjem příchozích paketů z detektoru vznikl `PacketsReceiverController`. Ten ve vlastním vlákne vyčítá proud bytů, přicházejících z detektoru a následně provádí jejich parsování, jehož výsledkem je instance objektu `IncomingPacket`, které obsahuje typ příkazu, příznak chyby a vlastní data. Takto vzniklý paket je zařazen do fronty příchozích paketů detektoru (implementované jako `ConcurrentLinkedQueue<IncomingPacket>`). Nad touto frontou pracují jednak všechny metody se synchronními příkazy komunikačního protokolu (které sledují, zda-li se v ní v daném časovém intervalu objeví odpověď), ale také instance objektu `AsynchronousEventHandlerController`. Ten ve vlastním vlákne tuto frontu sleduje a objeví-li se paket s typem příkazu `0xFD` (`Asynchronous Event From Device`), tak ho z fronty odebere, jeho vlastní data rozparsuje a dále zpracuje. Pokud se na příklad jedná o událost dokončené akvizice, tak `ReadOutService` snímek z detektoru vyčte a pomocí `RxJava event`

bus vyšle asynchronní zprávu napříč celou aplikací s vyčteným snímkem. Zde byl použit návrhový vzor Producer - Consumer, kde `AsynchronousEventHandlerController` představuje Producer a na kterémkoliv jiném místě aplikace se Consumer (možno i více Consumerů) může zaregistrovat ke sledování událostí v event bus. Příkladem takového Consumera může být `FrameSaverController`, který se postará o uložení získaného snímku (viz kapitola 4.4.3).

Aby bylo možné pohodlně ovládat více detektorů současně, vznikl `DetectorUnitsController`. Ten implementuje interface `DetectorUnitsControllerIntf`, jehož metody pokrývají veškerou funkcionalitu detektoru, jako třeba metody k navázání a ukončení spojení, ale také všechny metody komunikačního protokolu. Všechny tyto metody mají jeden společný parametr - `int[] tpxIDs` (pole `TpxID` detektorů). Tento parametr určuje, nad kterými detektory se má daná metoda hromadně vykonat. Jejich výstupem je zase pole, jehož typ se liší dle příkazu (na příklad pro metodu `connect` je to pole boolean proměnných - značící úspěch připojení, pro `ping` zase double čísel, udávajících odezvu spojení v *ns*). Tento objekt je zase typu singleton a jeho instanci lze získat pomocí metody `getInstance()`. Po spuštění aplikace je však třeba tento singleton nainicializovat pomocí metody `init()`, která pomocí `DetectorUnitStorageFactory` vytvoří datovou strukturu s `DetectorCommunicationService` jednotlivých detektorů.

4.4.1.5 Emulátor detektoru

Pro účely vývoje a testování řídicího software pro Atlas TPX server byl vyvinut emulátor detektoru, který plně emuluje jeho činnost. Emulátor byl rovněž napsán v jazyce JAVA a jeho zdrojové kódy a spustitelný `jar` soubor naleznete na přiloženém CD.

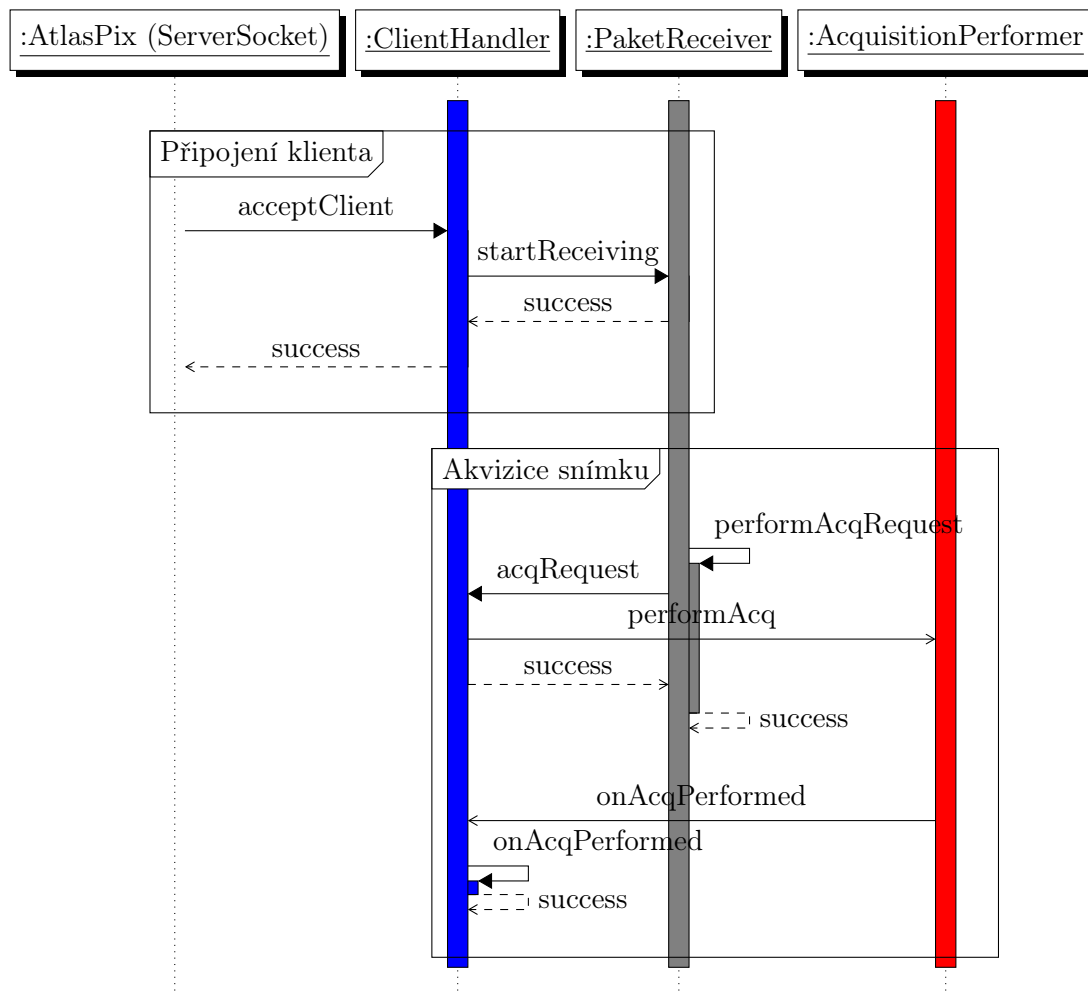
Emulátor se spouští se dvěma povinnými parametry, kde

1. je port, na kterém server emulátoru naslouchá (celé číslo),
2. je pravděpodobnost, s jakou při zpracovávání požadavku bude simulována chyba - 0 znamená bez chyb, 1 samé chyby (desetinné číslo).

Na obrázku 4.12 je zobrazen sekvenční diagram dvou případů užití tohoto emulátoru. První případ užití znázorňuje připojení klienta (na př. Atlas TPX serveru) a druhý pak proces zpracování žádosti o snímek a jeho vygenerování.

Pojďme si nyní popsat první část - připojení klienta. Ve třídě `AtlasPix`, resp. v její metodě `startServer()` se nachází nekonečná smyčka, ve které je pomocí instance třídy `ServerSocket` navázáno spojení s novými klienty. Když spojení s klientem je navázáno, je vytvořen klientský socket, pomocí kterého dojde v vytvoření instance třídy `ClientHandler`. Ta se v samostatném novém vláknu stará o obsluhu nově připojeného klienta. Zároveň `ClientHandler` vytvoří instanci třídy `PaketReceiver`, který čte proud bytů přijatých od klienta a parsuje jej do objektů typu `IncomingPacket`.

Tím se dostáváme k druhému příkladu případu užití - akvizici snímku. Poté, co `PaketReceiver` přijme nový paket, tak ho vloží do fronty paketů. Tuto frontu z druhé strany čte a zpracovává `ClientHandler`. Všechny příchozí pakety od klienta mohou obsahovat jen synchronní příkazy komunikačního protokolu, takže může být okamžitě nasimulován příslušný stav emulátoru a vygenerována odpověď klientovi.



Obrázek 4.12: Emulátor detektoru - sekvenční diagram připojení klienta a pořízení snímku

V našem případě akvizice snímku je situace trochu složitější. `PaketReceiver` odchytí nový paket, obsahující žádost o provedení akvizice, který předá do fronty příchozích paketů. Když se `ClientHandler` dostane k jeho zpracování, tak vytvoří objekt typu `AcquisitionRequest` a předá ho instanci třídy `AcquisitionPerformer`. Ten žádost o akvizici zařadí do příslušné fronty, kde čeká, až samostatné akviziční vlákno se dostane k jejímu zpracování. Při zpracování žádosti o akvizici je vygenerováno náhodná matice hodnot pixelů (o velikosti 256×512) a také příslušná metadata, jako na příklad akviziční čas, bias obou čipů (s přičtenou náhodnou veličinou, simulující fluktuaci napětí na křemíkovém povrchu detekčních čipů) apod. Po vygenerování těchto hodnot se akviziční vlákno uspí na dobu akvizice, aby se chování emulátoru co nejvíce přiblížilo skutečnému detektoru.

Následně je připojenému klientovi poslána asynchronní zpráva o dokončené akvizici. Tato zpráva obsahuje příznak, že naměřená data jsou připravena k vyčtení a také ID vygenerovaného snímku.

Poté klient pošle příkaz pro vyčtení naměřených dat (resp. dva příkazy - jeden pro vlastní

snímek a druhý pro metadata), což bylo popsáno v kapitole 4.4.1.3 - viz obr. 4.10.

4.4.2 REST API server

Řídící software se svému ovládání poskytuje rozhraní přes JSON REST²¹ API. Pro toto rozhraní byla použita knihovna Dropwizard[?], která vznikla složením několika dalších knihoven, zejména pak:

Jetty je open-source software, v současné době vyvíjen Eclipse Foundation. Tato knihovna obsahuje Java HTTP webový server, který na rozdíl od běžných webových serveru (které většinou přes HTTP protokol poskytují soubory koncovým uživatelům) byl navržen pro strojově orientovanou komunikaci.

Jersey je open-source knihovna (která dle [?] vychází z JAX-RS²²) a byla použita pro zajištění REST API. Tato knihovna umožňuje elegantně pomocí anotací mapovat HTTP dotazy na jednoduché Java objekty.

Jackson je výkonný nástroj pro práci s JSON²³ objekty v jazyce Java. Tato knihovna umožňuje ergonomicky mapovat data v JSON do Java objektů pomocí anotací.

V této podkapitole bude popsána implementace této knihovny do Atlas TPX serveru a její provázanost na modul pro řízení detektorů (4.4.1.4).

4.4.2.1 Konfigurace a spuštění serveru

Server je možné spustit z příkazové řádky pomocí příkazu "java -jar atlasTpxServer.jar server config.yml", kde atlasTpxServer.jar je spustitelný binární soubor serveru, server je povinný parametr (znamenaající spuštění programu v módu server) a config.yml, což je také povinný parametr, udávající cestu v souborovém systému ke konfiguračnímu souboru serveru.

V příloze A strukturu tohoto konfiguračního souboru, ve kterém je možné nastavit následující:

Umístění tabulky s detektory (detectorsConfigPath - viz A řádek 8)

Tento parametr udává cestu v souborovém systému ke konfiguračnímu "*.csv"souboru, obsahující tabulku všech detektoru a jejich parametrů, popsanou v 4.4.1.4.

Nastavení webového serveru (server - viz A řádek 11)

V rámci tohoto objektu je možné nastavit použitý typ protokolu spojení (HTTP/HTTPS), port, počty vláken a další.

²¹z angl. Representational State Transfer

²²<<http://jcp.org/en/jsr/detail?id=311>>

²³JSON je v dnešní době standardem pro zápis a výměnu strukturovaných, člověkem i strojem čitelných dat.

Logování (logging - viz [A](#) řádek 30)

Pomocí tohoto parametru je možné nastavit úroveň a cíl vytvářených logů. V příloženém konfiguračním souboru bylo použito dvojího logování a to na standardní výstup a do souboru (oba úrovně INFO). Při logování do souboru je možné navíc nastavit automatickou archivaci.

Automatické vyčítání snímků (readOutDataAutomatically - viz [A](#) řádek 54)

Tento parametr typu `boolean` udává, když přijde asynchronní událost z detektoru o datech připravených k vyčtení, zda-li budou vyčtena automaticky (při hodnotě parametru `true`), nebo manuálně (při hodnotě `false`).

Výstupní adresář (outputDir - viz [A](#) řádek 57)

Parametr udávající cestu v souborovém systému pro ukládání dat z detektorů.

Formát výstupních dat (outputFramesType - viz [A](#) řádek 68)

Tento parametr obsahuje pole výstupních formátů dat, ve kterých získaná data z detektorů budou ukládána. V této verzi programu je podporovaný jediný formát - MULTIFRAME.

Konfigurace data serveru (v [A](#) od řádku 77)

Na tomto místě je možné nastavit parametry (url a port) serveru, sloužícího pro příjem a zpracování naměřených dat a zda-li má být použit. Více o tomto serveru bude zmíněno v [4.4.3.1](#).

4.4.2.2 Metody poskytované serverem

4.4.2.3 Výkonost

4.4.3 Zpracování a ukládání dat

4.4.3.1 Data server

Kapitola 5

Závěr

Příloha A

Konfigurační soubor Atlas TPX serveru

```
1 #####
2 # Basic configuration
3 #####
4 # Path to config csv file with detecotrs config
5 # File structure: 1 detector struct per line
6 # - detector_name[String];detector_ip[String];unit_id[int];tpx_id[int]
7 #       ;port[int]
8 detectorsConfigPath: data/detectors2.csv
9
10 # Controll REST API server configuration
11 server:
12   applicationConnectors:
13     - type: http
14       port: 9179
15       outputBufferSize: 32KiB
16       idleTimeout: 30 seconds
17       minBufferPoolSize: 64 bytes
18       bufferPoolIncrement: 1KiB
19       maxBufferPoolSize: 64KiB
20       acceptorThreads: 1
21       selectorThreads: 2
22       acceptQueueSize: 1024
23       reuseAddress: true
24       soLingerTime: 600s
25   adminConnectors:
26     - type: http
27       port: 9180
28
29 # Logging settings
30 logging:
```

```
31 level: INFO
32 appenders:
33   - type: file
34     currentLogFilename: log/atlastpx_server_app.log
35     threshold: ALL
36     archive: true
37     archivedLogFilenamePattern: log/atlastpx_server_app-%d.log.gz
38     archivedFileCount: 5
39     timeZone: UTC
40 logging:
41   level: INFO
42   appenders:
43     - type: console
44       threshold: ALL
45       timeZone: UTC
46       target: stdout
47
48
49 #####
50 # Data readout & saving configuration
51 #####
52 # If true asynchronous event handler will takes care about automatical data
53 # (frame + DSC) readout from the detector.
54 readOutDataAutomatically: true
55
56 # Directory for saving output data
57 outputDir: ../frames
58
59 # Local data saving configuration
60 # Allowed types:
61 # - MULTIFRAME: Pixelman's multiframe format (3 files per hour).
62 #       Output files:
63 #       1) Frames: Line [X, C], where X is pix. index and
64 #       C pix value. Frames are separated by '#'
65 #       2) Description file ( DSC)
66 #       3) Index file (indexing file 1 and 2)
67 # Example - outputFramesType: [MULTIFRAME]
68 outputFramesType: [MULTIFRAME]
69
70
71 #####
72 # Data server configuration
73 #####
74 # If true all data (configs on change, performed frames) will be automatically
75 # send to the data server
76 # and also data server status will be available by info endpoint.
```

```
77 useDataServer: true
78
79 # URL of dataServer
80 dataServerBaseUrl: atlastpx.utef.cvut.cz
81
82 # port of dataServer
83 dataServerPort: 8042
```


Příloha B

Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce...
[?] fasfassaafsaf



Obrázek B.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případně index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.