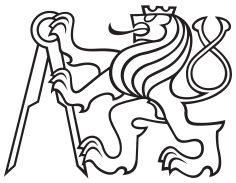
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Konfigurační nástroj pro IIoT platformu SBDH

Filip Barva

Školitel: Ing. Václav Jirkovský, Ph.D.
Květen 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Barva** Jméno: **Filip** Osobní číslo: **483585**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Konfigurační nástroj pro IIoT platformu SBDH

Název bakalářské práce anglicky:

Tool for SBDH IIoT SBDH configuration

Pokyny pro vypracování:

Cílem práce je vytvoření systému pro uživatelskou konfiguraci systému Semantic Big Data Historian (SBDH). Konfigurace spočívá v identifikaci a specifikaci hlavních parametrů následujících frameworků - Apache Storm, Apache Cassandra a InfluxDB. Neméně důležitou částí práce je specifikace a vyhledání připojených zařízení, ze kterých se sbírají produkční data. Specifikace a vyhledání zařízení probíhá pomocí OPC UA discovery serveru, který udržuje informace pro připojení konkrétních zařízení. Definice parametrů a informace o připojených zařízení je v sémantické podobě v ontologii. Sémantická podoba informací umožňuje přesné vyhledávání konkrétních zařízení. Implementace bude pomocí vhodného programovacího jazyka a bude se skládat z implementace uživatelského rozhraní, OPC UA klienta pro komunikaci s OPC UA discovery serverem a s vytvořením konfiguračního souboru pro SBDH.
Student se seznámí se sémantickým webem a se systémy Apache Storm, Cassandra a InfluxDB proto, aby definoval vhodnou množinu parametrů pro konfiguraci daných systémů. Dále implementuje OPC UA klienta (pomocí vhodného OPC UA stacku a OPC UA SDK) komunikujícího s OPC UA discovery serverem pro získání adres zdrojových zařízení a jejich filtrování pomocí požadovaných schopností definovaných v ontologii. Následně implementuje uživatelské rozhraní ve vhodném programovacím jazyce a umožní vytvoření konfiguračního souboru SBDH.
. Navržené a implementované řešení bude ověřeno na datech sbíraných z flexibilní výrobní linky Testbedu pro Průmysl 4.0 ČVUT (alternativně ze senzorů malé vodní elektrárny). Ověřována bude správnost a úplnost vrácených záznamů.

Seznam doporučené literatury:

[1] W. Mahnke a kol. - OPC unified architecture - Springer Science & Business Media, 2009.
[2] J Hendler a kol. - Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL - Morgan & Claypool, 2020.
[3] A. Jain - Mastering apache storm: Real-time big data streaming using kafka, hbase and redis - Packt Publishing Ltd, 2017.
[4] E. Hewitt - Cassandra: the definitive guide - O'Reilly Media, Inc., 2010.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Václav Jirkovský, Ph.D. katedra kybernetiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Václav Jirkovský, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Václavu Jirkovskému, Ph.D. za jeho ochotu pomoci, trpělivé konzultace a vedení. Dále bych chtěl poděkovat své rodině a přátelům za podporu, kterou mi vždy poskytují.

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, 20. května 2022

Abstrakt

Práce se věnuje návrhu a implementaci uživatelského rozhraní pro konfiguraci Semantic Big Data Historianu (SBDH), který je tvořen frameworky Apache Storm, Apache Cassandra, InfluxDB a komunikaci primárně pomocí OPC UA. Každý z nich má specifické parametry, které je třeba nastavit. Uživatelské rozhraní je navrženo pomocí JavaFX a odvíjí se od těchto parametrů. Cílem práce je vytvoření aplikace, která usnadní vytvoření konfiguračního souboru SBDH nebo úpravu již existujícího.

Dále se práce zabývá navrhnutím a vytvořením OPC UA klienta. Ten zde slouží pro ulehčení a zrychlení vytváření konfigurace za pomocí discovery procesu. Díky discovery procesu je klient schopný získat servery zaregistrované na globálním discovery serveru a ty pak nabídnout uživateli k zaznamenání do konfigurace. Servery je možné filtrovat za pomoci uživatelem zvolené ontologie, která je postavená na Semantic Sensor Network Ontology. Ta obsahuje mnoho konceptů a jejich vlastnosti pro popis senzorů. Dotazy na ontologii jsou realizovány pomocí sémantického dotazovacího jazyka SPARQL. Uživateli jsou k dispozici předpřipravené dotazy na základě zvolené ontologie, ze kterých si může vybrat. Je mu také umožněno vytvářet vlastní dotazy za pomoci SPARQL.

Pro ověření funkčnosti je v práci také zohledněn návrh a zprovoznění testovacího prostředí. V této části je dopodrobna popsáno, jaké nástroje byly využity k vytvoření testovacího prostředí a za pomocí jakých nástrojů se dále ověřovala správnost výstupů aplikace.

Klíčová slova: OPC UA, Apache Storm, Apache Cassandra, InfluxDB, Sémantický web, Big Data, Konfigurace, SBDH

Školitel: Ing. Václav Jirkovský, Ph.D.

Abstract

The thesis deals with the design and implementation of a user interface for the configuration of the Semantic Big Data Historian (SBDH), which consists of the frameworks Apache Storm, Apache Cassandra, InfluxDB, and communication primarily using OPC UA. Each of these has specific parameters that need to be set. The user interface is designed using JavaFX and is based on these parameters. The work aims to modify the existing SBDH file.

The work also deals with the design and creation of the OPC UA client. This is used to facilitate and speed up the configuration through the discovery process. Thanks to the discovery process, the client is able to obtain servers registered on the global discovery server and then offer them to the user for recording in the configuration. Servers can be filtered using a user-selected ontology, which is built on Semantic Sensor Network Ontology. It contains many concepts and their properties for the description of sensors. Ontology queries are created using the semantic query language SPARQL. Users are provided with pre-prepared queries based on the selected ontology from which they can choose. It also allows users to create custom queries using SPARQL.

To verify the functionality, the work also takes into account the design and commissioning of the test environment. This section describes in detail which tools were used to create the test and which ones were used to further verify the correctness of the application outputs.

Keywords: OPC UA, Apache Storm, Apache Cassandra, InfluxDB, Semantic web, Big Data, Configuration, SBDH

Title translation: Tool for SBDH IIoT SBDH configuration

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Tato bakalářská práce se zabývá tématem uživatelské konfigurace systému Semantic Big Data Historianu (SBDH). Ten se skládá z několika různých frameworků, které jsou propojeny mezi sebou a je potřeba specifikovat jejich parametry podle konkrétní aplikace (SBDH). V současné době, kdy se průmyslové systémy adaptuji ke klíčovým vlastnostem čtvrté průmyslové revoluce, se tyto systémy stávají velmi komplexními ekosystémy, a proto jejich napojeni na systém, který je odpovědný za sběr dat a jejich řízení v reálném čase, je poměrně složitým problémem. V takto složitém systému uživatel při konfiguraci muže vytvořit mnoho chyb.

Cílem práce je usnadnit uživateli konfiguraci SBDH pomocí aplikace, která umožní uživateli vygenerovat a upravovat konfigurační soubory pro frameworky (Apache Storm, Apache Cassandra a InfluxDB) a komunikaci pomocí OPC UA, které dohromady tvoří SBDH. Pro komunikaci pomocí OPC UA je třeba vyhledat servery, ze kterých jsou sbírána data. Pro usnadnění konfigurace připojených zařízení je možné využít OPC UA globální discovery server, který udržuje detaily pro připojení konkrétních zařízení. V rámci práce bude implementováno základní uživatelské rozhraní aplikace, která umožní načíst, upravit a uložit konfigurační soubor.

Tato práce je členěna následovně: Na úvod jsou v kapitole *Teoretická část* definovány pojmy, které s prací souvisí. *Praktická část* je poté věnována samotné implementaci uživatelského rozhraní a popisu jednotlivých konfiguračních parametrů. Implementace je rozdělena na implementaci grafického rozhraní a OPC UA klienta. Grafické rozhraní je vytvořeno knihovnou JavaFX. Vytvoření klienta bylo realizováno pomocí knihovny Eclipse Milo. Pro rozšíření klienta o komunikaci s ontologií se využívá knihovna Apache Jena, která umožňuje načtení ontologie a následně její dotazování pomocí sémantického dotazovacího jazyka SPARQL. Součástí praktické části je i tvorba testovacího prostředí a následné testování vytvořené aplikace.

Kapitola 2
Teoretická část

Tato kapitola je věnována vysvětlení teoretických pojmů, se kterými se v této práci setkáte. Pro vysvětlení pojmu Semantic Big Data Historian bude nejdříve popsán pojem Big Data (BD). Dále bude popsán pojem NoSQL databáze a stručně představeny způsoby zpracování dat. Architektura SBDH se skládá z Apache Storm pro zpracování Big Data a Apache Cassandra pro ukládání dat. SBDH využívá OPC UA standard jako primární způsob komunikace pro připojení různých zdrojů dat.

2.1 Semantic Big Data Historian

SBDH využívá technologie sémantického webu k vytvoření znalostní báze pro popis zařízení a dat, která jsou naměřena, aby data pomáhala reprezentovat koncepty z reálného světa pro lepší strojové zpracování. Nesprávná interpretace dat by mohla snížit přesnost analýzy, nebo ji dokonce celou zneplatnit. Běžné systémy pro sběr dat využívají relační databáze, které neumožňují snadné úpravy a aktualizaci jejich schemat, což jsou potřebné vlastnosti v kontextu průmyslu 4.0. Proto se v datově náročných aplikacích uchovávají i metadata, jako například původ, přesnost či věrohodnost. [1]

Historian je software sbírající data z vybraných linek a relevantních procesů, která následně ukládá a případně dále zpracovává. Sbírané udaje obsahují kromě sledované hodnoty i informace s danou hodnotou spojenými jako je například čas zařízení nebo status měření. [2]

Semantic Big Data Historian (SBDH) slouží pro zpracování a ukládání velkého množství výrobních dat. Díky využití technologií sémantického webu dovoluje zachytit i informace nad rámec běžných systémů pro sběr dat. Rozšiřuje způsob zpracování dat a jejich analýzu. Kromě umožnění práce s Big Data využívá SBDH dříve zmíněnou sémantiku pro zlepšení interpretace dat během jejich analýzy či prezentace informací pro uživatelské rozhraní. Sémantika, přesněji řečeno ontologie, dále usnadňuje integraci dat ze zařízení vyšší úrovně jako je například MES. SBDH je tvořeno pomocí real-time frameworku ke zpracování dat a distribuované databáze. [3]

2.2 Big Data

Koncept Big Data představuje soubor dat, který neustále roste natolik, že je obtížné ho spravovat pomocí existujících konceptů a nástrojů pro správu databází. Obtížnost může být související se zachycováním, ukládáním, vyhledáváním, sdílením, analýzou a vizualizací těchto dat. [4]

Obecně přijímanou definicí Big Data je tzv. „3V“ definice, která specifikují hlavní charakteristiky:

- **Objem** ***(Volume)*** - Veliký objem dat (terabajty a petabajty).
- **Rychlost** ***(Velocity)*** - Veliká rychlost přijímání a zpracování dat.
- **Rozmanitost** ***(Variety)*** - Různé typy a formáty dat.

Hlavním důvodem k využití frameworků pro zpracování Big Data je, že nejsme schopni takto velké množství dat zpracovat běžnými způsoby a systémy. Z velké množství uložených historických dat můžeme za pomocí metody „data mining“ např. nalézt skryté závislostí v datech. [4]

2.3 Sémantický web

Sémantický web má za cíl, aby data reprezentovaná v počítači byla nejen srozumitelná pro lidi, ale i pro stroje. Toho se snaží docílit vytvořením dobře známých ontologií, což jsou explicitní specifikace sdílené konceptualizace. Díky sémantickému webu a ontologii uživatel nemusí stahovat data z jednotlivých zdrojů, propojovat je a interpretovat, ale udělá to za něj zařízení. Sémantický web využívá řadu různých technologii jako je například RDF, RDFS nebo OWL pro popis a práci s daty. Všechny tyto technologie patří do specifikace W3C. Jako dotazovací jazyk se zde využívá SPARQL.

Resource Description Framework neboli RDF je specifikace tvořící model pro ukládání a práci s daty na webu. Data jsou reprezentována pomocí trojic jako objekt a jeho vztah k dalšímu objektu což tvoří orientovaný graf určující vztahy mezi objekty. Takto vytvořený RDF graf lze serializovat například do formátu XML. RDF k reprezentaci dat využívá URI což umožňuje provazování různých data a možnost data jednoduše sdílet napříč aplikacemi. Resource Description Framework Schema (RDFS) je jednoduchý ontologický jazyk specifikující základní vztahy mezi RDF objekty.

Web Ontology Language neboli OWL je rozšířením původního RDFS. Využívá se k vytváření ontologií které popisují entity jejich vlastnosti a vztahy mezi nimi. Jejich hlavním cílem je popsat data tak, aby co nejpřesněji zachycovali koncepty z reálného světa a byli srozumitelné jak pro člověka tak pro stroj. Jednou z výhod ontologii je možnost je rozšiřovat a případně i kombinovat z ostatními ontologiemi. [5]

2.4 Zpracování dat

Hlavními požadavky na zpracování dat pro SBDH jsou co nejmenší latence a co největší efektivita. Existují dva hlavní způsoby zpracování stream dat – real-time a dávkové. Procesům s nízkou latencí (v řádu sekund nebo jejich zlomků) se říká real-time.

„[. . .] Na rozdíl od tradičních systémů pro analýzu dat, které shromažďují a pravidelně zpracovávají veliké – statické – objemy dat, systémy pro streamové zpracování dat se vyhýbají ukládání dat do paměti a zpracovávají je, jakmile jsou dostupná, čímž se minimalizuje čas, který jednotlivá datová položka stráví v procesu zpracování.“ [6]

Pro streamové zpracování dat je charakteristické, že se data nejdříve sbírají, poté se zařadí do fronty, z níž se postupně zpracovávají, a nakonec se uloží či zobrazí.

Kromě čistě streamového zpracování dat existuje i architektonický vzor *Lambda architektura*. Ten kombinuje přístup zpracovávání dat po dávkách s streamovým zpracováním, čímž využívá nejlepší vlastnosti obou těchto přístupů. [6]

2.4.1 Apache Hadoop

Apache Hadoop je framework, který obsahuje open source softwarové komponenty pro práci s BD. Tento framework řeší jak uložení BD pomocí systému HDFS, tak i jejich zpracování pomocí algoritmu Hadoop MapReduce.

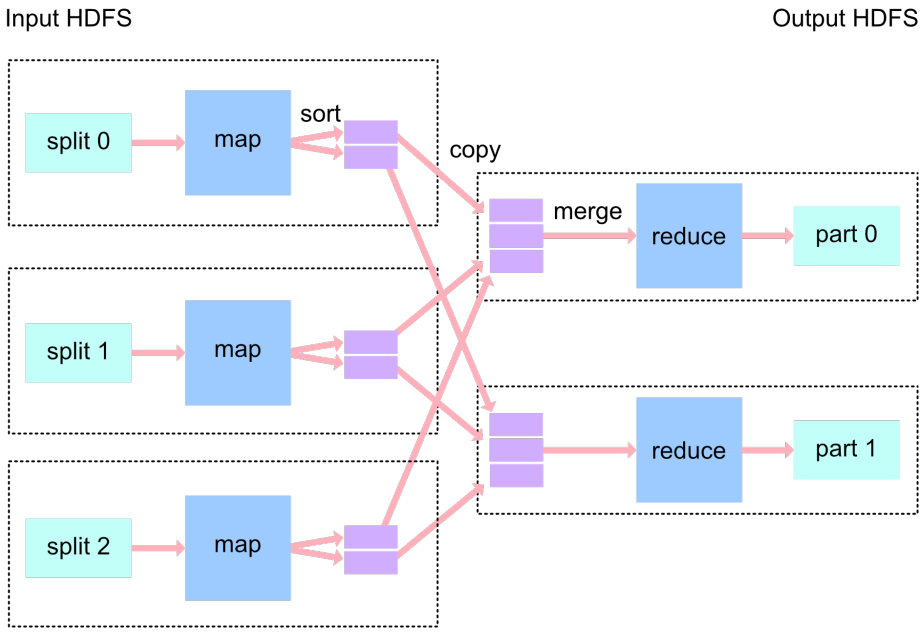
„Hadoop MapReduce je algoritmus, který zpracovává veliké množství dat paralelně na velkých clusterech (tisících uzlů) komoditního hardwaru spolehlivým způsobem odolným vůči chybám.“ [7]

Job v MapReduce rozděluje vstup na nezávislé části, které se poté paralelně zpracovávají v *map tasks*. Jejich výstup pak je posílán do *reduce tasks*. Zmíněné vstupy i výstupy se pak ukládají v souborovém systému. [7]

Služby v Hadoop clusteru se dělí na JobTracker, TaskTracker, NameNode, Secondary NameNode a DataNode. Každý server může obsahovat vícero z daných služeb. JobTracker přiřazuje jednotlivé úkoly TaskTrackeru a ten na základě toho, jestli je daný task *map* nebo *reduce* data buď rozděluje nebo skládá do celkového výsledku. NameNode se stará o metadata systému souborů, pokud se používá Hadoop Distributed File System (HDFS). Secondary NameNode pravidelně kontroluje metadata na NameNode. HDFS soubory se potom ukládají v DataNode, která i vyřizuje požadavky na čtení či zapisování. [8]

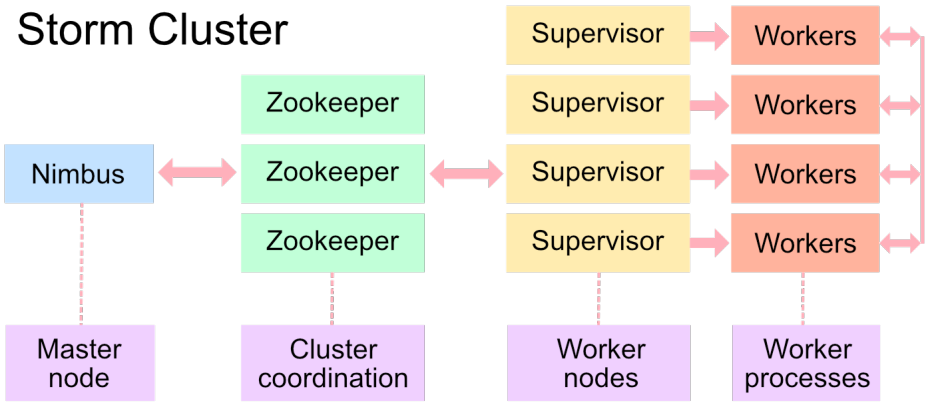
2.4.2 Apache Storm

Apache Storm provádí real-time proudové zpracování dat. Apache Storm je kompatibilní se spoustou programovacích jazyků jako například Java, Python, JavaScript či Ruby. [6]



Obrázek 2.1: MapReduce algoritmus.

Storm má dva typy uzlů. Na hlavním uzlu běží *Nimbus* - proces běžící na pozadí zodpovědný za distribuci kódu napříč klastrem, přiřazování úkolů a sledování poruch. Druhým typem jsou pracovní uzly a v nich běží na pozadí *Supervisor*, který čeká na přiřazení práce od Nimbusu. Pracovní procesy jsou rozprostřené mezi několika zařízeními a každý vykonává určitou část topologie. Veškerou komunikaci mezi Nimbusem a Supervisory zařizuje Zookeeper cluster. Tento cluster můžete vidět na obrázku 2.2. Nimbus a Supervisor jsou bezstavové a často v nich dochází k selháním. Pro případ selhání jsou jejich stavy uloženy v Zookeeperovi, který se současně stará o jejich zpětné spouštění. Tento návrh vede k vysoké stabilitě Storm clusterů.

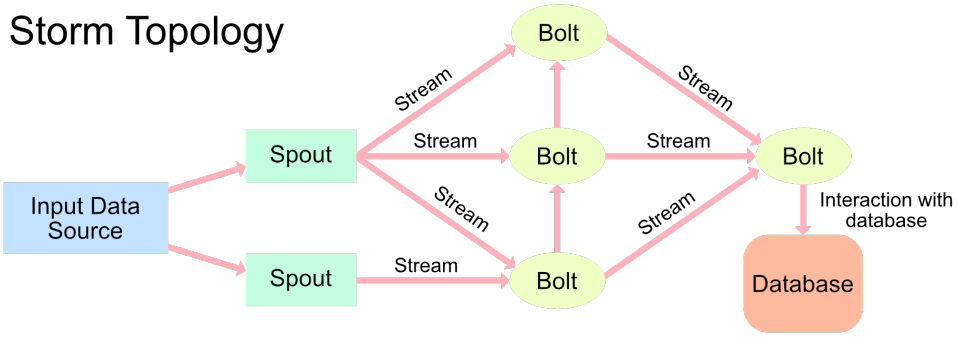


Obrázek 2.2: Storm cluster

Název	Model zpracování	Architektonický vzor	Real-time
Hadoop	Stream	Lambda	Ne
Storm	Stream	Lambda	Ano
Spark Streaming	Micro-batch	Lambda	Ne

Tabulka 2.1: Srovnání frameworků na zpracování dat pro SBDH

Pro zpracovávání se zde používají topologie, které nepřetržitě zpracovávají zprávy. Topologie je acyklický graf definující průběh zpracování dat. Každý uzel v ní má svoji vlastní logiku a propojení mezi jednotlivými uzly definuje, jakým způsobem se data v grafu distribuují. Jednotlivé uzly se dělí na „spouty“ a „bolty“, které transformují proudy dat a mají implementovatelné rozhraní. *Spout* je zdrojem těchto proudů dat a *bolty* tato data zpracovávají a vykonávají různé další logické operace jako například spojování a separace dat či komunikace s databází viz obrázek 2.3. Uzly v topologii je možné spustit paralelně a uživatel může nastavit, jak moc velký paralelismus bude daný uzel mít.



Obrázek 2.3: Storm topologie

„Storm používá n-tice jako svůj datový model. N-tice je pojmenovaný seznam hodnot a pole v n-tici může být objekt libovolného typu. [...] Storm podporuje všechny primitivní typy, řetězce a bajtová pole jako hodnoty polí n-tice. Pro použití objektu jiného typu je možné implementovat serializátor pro daný typ.“ [9]

2.4.3 Apache Spark

Apache Spark je analytický engine sloužící pro práci s BD. Pro zpracování dat se zde využívá komponenta Spark streaming.

Spark streaming je rozšíření Spark API. Je využíván ke škálovatelnému zpracování proudů dat, které je odolné vůči chybám. Data může brát z různých zdrojů a zpracovávat je pomocí různých funkcí, jako například map, reduce a join, a posílat je do souborových systémů či databází. Data zpracovává a posílá po dávkách. [10]

2.5 NoSQL databáze

„[. . .] NoSQL systémy jsou distribuované, nerelační databáze určené pro rozsáhlá úložiště dat a pro masivně paralelní zpracování dat napříč velkým množstvím komoditních serverů. K interakci s daty také používají jiné než SQL jazyky a mechanismy.“ [11]

NoSQL databázové systémy vznikly díky tomu, že velké společnosti schraňující velká množství dat měly problémy se s nimi vypořádat při používání relačních databází. Umí používat technologii OLTP, která slouží pro bezpečné a jednoduché modifikace v prostředí s mnoha uživateli, a data transformovat.

Kvůli zajištění integrity dat se pro běžné relační databáze používá princip transakcí, který zajišťuje atomicitu dat (Atomicity), jejich konzistenci (Consistency), separaci (Isolation) a trvanlivost (Durability). Tyto charakteristiky jsou známé také jako ACID. NoSQL databáze kvůli své architektuře a kvůli způsobu použití nesplňují ACID. Místo ACID charakteristik splňují CAP theorém. Podle CAP theorému není možné, aby bylo při škálování zároveň dosaženo všech těchto tří vlastností:

- **Silná konzistence *(Consistency)*** - Všichni vidí stejnou verzi dat.
- **Vysoká dostupnost *(Availability)*** - Klient dostane odpověď na svůj požadavek, i když je jeden či více uzlů nedostupných.
- **Tolerance rozdělení *(Partition tolerance)*** - Celý systém funguje, přestože nefunguje síť spojující jednotlivé servery.

NoSQL databáze většinou upřednostňují vysokou dostupnost a toleranci rozdělení před konzistencí. Tyto systémy jsou známé jako BASE (Basically Available, Soft-state, Eventually consistent). [11]

2.5.1 Apache Cassandra

Cassandra je NoSQL distribuovaná databáze umožňující škálování při zatížení aplikace. Je robustní a odolná, protože žádný z uzlů není hlavní (tomuto uspořádání se říká *masterless*). Její uzly mezi sebou komunikují pomocí peer-to-peer protokolu. Na rozdíl od relačních databází je Cassandra snadněji rozšířitelná o další servery a to i za běhu bez nutnosti přerušení služeb. Dle CAP theorému je AP (Available a Partition-tolerant), ale uživatel si může nastavit míru konzistence jako počet uzlů, které musí zaznamenat operaci před tím, než je operace prohlášena za úspěšnou.

Cassandra nabízí jazyk Cassandra Query Language (CQL), což je jazyk podobný SQL. Data v clusterech jsou organizována pomocí Keyspace, v němž jsou uložené jednotlivé Tables, které obsahují Partitions, jež v sobě mají Rows, a ty obsahují Columns. Keyspace navíc obsahuje nastavení replikace datasetu. Table definuje schéma pro kolekci Partitions. Patrition definuje povinnou část primárního klíče Rows. Rows obsahují primární klíče Columns. Columns mají jediný údaj. Cassandra nepodporuje operace napříč Partitions z důvodu velké náročnosti na zpracování tak velkého objemu dat. [12]

2.6 InfluxDB

InfluxDB je tzv. Time Series Databáze (TSDB), což jsou databáze zaměřené na ukládání dat s časovou značkou nebo časových řad. Specializují se hlavně na měření a vyhodnocování změn v průběhu času, což umožňuje časové řady efektivněji ukládat a pracovat s nimi. Tyto databáze jsou kombinací NoSQL a relačních databází. [13] Pro efektivnější a rychlejší zacházení s daty databáze shlukuje data se stejným časovým rozsahem do stejných fyzických částí clusteru. Data se zapisují a ukládají rychle i při velké zátěži, protože jsou TSDB navržené tak, aby se data časových řad zaznamenávala velmi často. Pro docílení takovéto architektury je potřeba dobře navržená komprese dat. Jedna ze základních funkcí, kterou TSDB obsahují je nastavení doby, po kterou jsou data v databázi uchována (tzv. retention policy).

Hlavním konceptem v InfluxDB je čas. Ten se ukládá s každou hodnotou v poli. Každé pole s daty má svůj klíč (název) a hodnotu. Dále jsou tam tagy, které mají také tag key a tag value. N-tice tagů se nazývá tag set. Hlavním rozdílem mezi poli a tagy je to, že tagy jsou indexovány a tudíž se na ně lze rychleji dotazovat. Dalším konceptem je measurement což je v podstatě název tabulky, ve které jsou data uložena, a retention policy pak následně udává dobu, po jejímž uplynutí budou data z databáze smazána. V základu je retention policy nastavena na nekonečno. Pokud daná data mají stejný measurement, tag set a retention policy, shlukují se do sérií.

Jednou z výhod použití InfluxDB je jeho podpora Grafanou, což je software pro analýzu časových řad, který vizualizuje data pomocí dashboardů. [14]

2.7 OPC Unified Architecture

OPC Unified Architecture (OPC UA) je standard, který umožňuje robustní a bezpečnou komunikaci mezi klientem a serverem pomocí zpráv. V modelu Klient-Server definuje OPC UA množiny služeb, které jsou serverem poskytovány klientovy. Z této množiny následně server specifikuje, které jsou podporovány, aby klient věděl které může využít. Dále servery definují modely objektů, na které se klienti mohou dynamicky dotazovat. Servery mohou poskytnout přístup k datům a můžou i notifikovat klienta na změny hodnot v proměnných. OPC UA disponuje protokoly popisující adresní prostor daných serverů, což umožňuje reprezentovat data v binárních strukturách, XML či JSONu. Skrze adresní prostor se mohou klienti dotazovat na metadata popisující formát dat a to i v runtimu. Díky tomu, že OPC UA podporuje spoustu druhů vztahů mezi daty může být hierarchie serveru přizpůsobena požadavkům klienta. [15]

2.7.1 Komunikace

Hlavními prvky komunikace v OPC UA jsou aplikační, komunikační a transportní vrstva, které dohromady tvoří tzv. komunikační zásobník.

Aplikační vrstva je reprezentována pomocí relace. Ta se používá k identifikování komunikace a probíhá v ní volání a zpracování služeb, a když je dlouho nečinná, uzavře se. Ve chvíli, kdy s ní nějaký kanál od stejného klienta naváže spojení, relace se znovu otevře.

Na základě komunikačního protokolu se vytvoří komunikační vrstva, která zprostředkovává zabezpečení daného kanálu. Veškerá komunikace mezi serverem a klientem je identifikována pomocí identifikátoru kanálu a bezpečnostní známky, což zajišťuje důvěryhodnost přenesených dat.

Transportní vrstva je nejnižší vrstvou komunikace. Umožňuje adresovat přímo aplikaci pomocí daného komunikačního protokolu jako je například TCP nebo UDP. [15]

2.7.2 Adresní prostor

Adresní prostor je množina objektů a s nimi spojených informací, které jsou OPC UA serverem zpřístupněny pro klienta.

„Hlavním cílem OPC UA adresního prostoru je poskytnout standardní způsob, jak servery reprezentují objekty pro klienty. OPC UA informační model byl navržen za tímto účelem. Definuje objekty pomocí proměnných a metod. Také dovoluje vyjádřit vztahy k ostatním objektům.“ [15]

Základní prvky adresního prostoru jsou reprezentovány jako uzly. Ty jsou přiřazeny do tříd, kde každá třída definuje jiný prvek objektového modelu. Každý uzel má atributy, které ho popisují, a uchovává si reference na připojené uzly, které vyjadřují vztahy mezi nimi. Uzel uchovávající v sobě referenci na druhý se nazývá zdrojový, uzlu na nějž je odkazováno se říká cílový. OPC UA podporuje velké množství různých typů referencí jako například hasProperty, hasComponent a hasType. Spojení typu reference, zdrojového uzlu a cílového uzlu musí být v adresním prostoru unikátní. [15]

2.7.3 Discovery

Proces discovery pomáhá OPC UA klientům nalézt požadované OPC UA servery na síti a pak zjistit, jak se k nim připojit. Když už má klient tuto informaci, může si ji uložit a příště nemusí provádět discovery proces, když se chce znovu připojit na server.

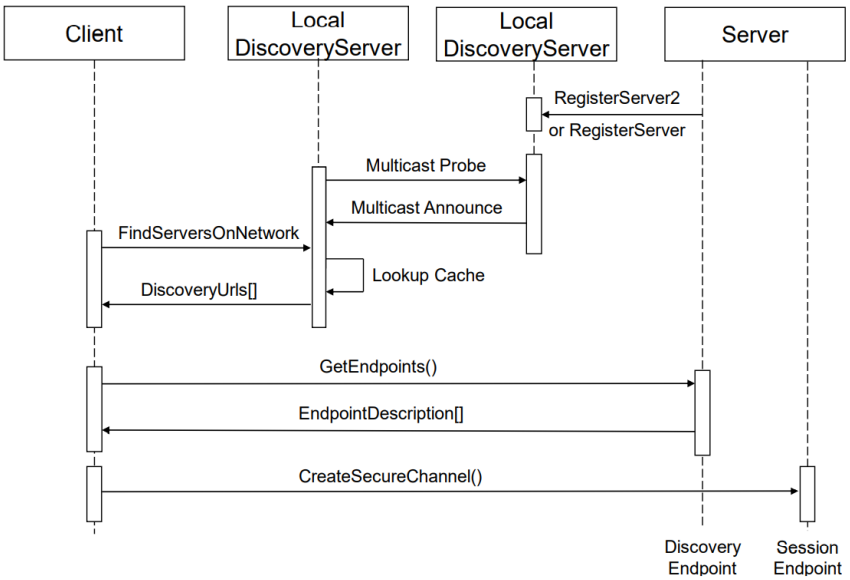
Na discovery server se zaregistrují všechny servery, pro usnadnění jejich nalezení OPC UA klientem. Existují tři typy discovery serverů: lokální (LDS), lokální s MulticastExtension (LDS-ME) a globální (GDS). LDS umožňuje nalézt všechny zaregistrované aplikace na jednom zařízení. LDS-ME je rozšířenou variantou LDS, která poskytuje discovery informace o serverech běžících na stejném MulticastSubnetu. GDS může zaznamenávat a poskytovat informace o serverech napříč celou dostupnou sítí.

Klient má několik přístupů, kterými může nalézt potřebné servery. Pokud již vlastní potřebný endpoint, stačí mu sním jen navázat komunikaci. Ve většině případů ji však nezná, ale ví, který host by daný server mohl mít, a tak musí navázat komunikaci s LDS na hostovi pomocí adresy definované v OPC UA standardu, v některých případech se mohou i lišit, a zavolat metodu

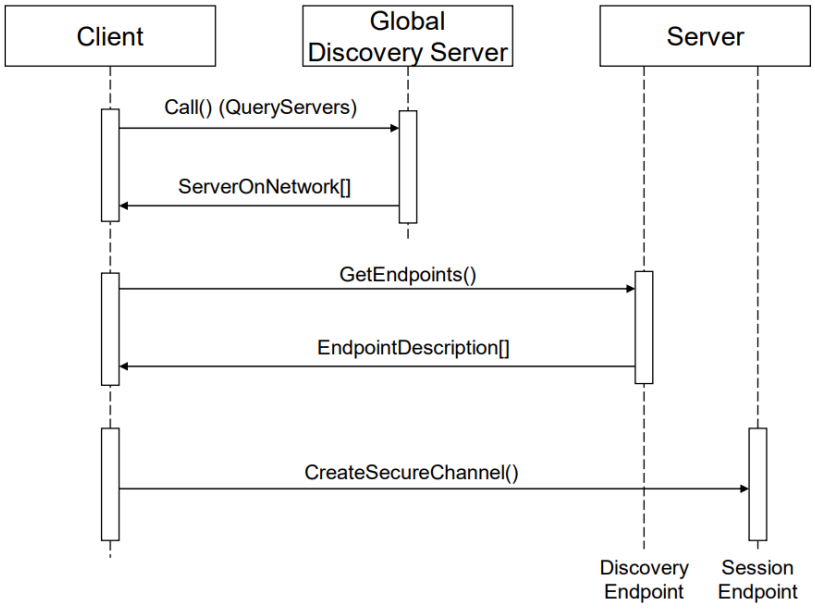
FindServer. Tato metoda následně vrátí DiscoveryUrl všech zaregistrovaných serverů na daném hostovi. Pokud ale klient neví, na kterém hostovi by se server mohl nacházet, musí využít LDS-ME na svém hostovi a pomocí metody FindServersOnNetwork získat všechny zaregistrované servery na místním MulticastSubnetu viz obrázek 2.4. Poslední variantou je, že klient chce server mimo místní MulticastSubnet a tudíž musí využít GDS, který má adresu definovanou pomocí OPC UA standardu, nebo je mu předem známá. GDS umožňuje klientovi prohledávat všechny dostupné servery. Klient naváže komunikaci s GDS pomocí služby Call, která vyvolá metodu QueryApplications. Tato metoda funguje stejně jako FindServers a je rozšířena o některé filtrovací možnosti jako například filtrování pomocí ServerCapability. Průběh tohoto procesu můžete vidět na obrázku 2.5.

Pro registraci serverů na LDS nebo LDS-ME se využívá služba RegisterServer2 definovaná OPC UA standardem, která umožňuje registraci serveru s více informacemi. Pro zpětnou kompatibilitu by servery a aplikace měly podporovat starší službu RegisterServer – například pokud je voláno RegisterServer2 a server vrátí chybu BadServiceUnsupported, tak je možné, že se jedná o starší LDS a bude tedy možné se na něj zaregistrovat alespoň pomocí RegisterServer. Jednou z nejdůležitějších informací, o které je RegisterServer2 rozšířen, je ServerCapability. Jedná se o identifikátory definované OPC UA standardem, které ulehčují filtrování serverů v průběhu discovery procesu. Každý server může mít libovolný počet těchto identifikátorů.

Registrace aplikace na GDS probíhá pomocí metody RegisterApplication. Tato metoda stejně jako RegisterServer2 umožňuje při registraci uložit informace o ServerCapability. Pokud registrovaná aplikace je LDS nebo LDS-ME tak na ní GDS zavolá FindeServers nebo FindServersOnNetwork, aby získal GDS informace ke všem zaregistrovaným aplikacím. Tyto aplikace rovnou registruje sám do sebe. GDS musí pravidelně updatovat svojí databázi voláním FindServersOnNetwork nebo FindServers na zaregistrované LDS. Všechny informace o známých aplikacích jsou organizovány v DirectoryType pod složkou Applications. Tato složka nemusí být přístupná pro procházení, ale přistupují k ní metody, které GDS poskytuje. [15]



Obrázek 2.4: MulticastSubnet Discovery proces



Obrázek 2.5: Globální Discovery proces

Kapitola 3
Praktická část

SBDH jehož konfigurací se tato práce zabývá je tvořen z frameworků Apache Storm, Apache Cassandra, InfluxDB a primárně komunikací pomocí OPC UA. Tato kapitola se zabývá výběrem potřebných parametrů konfigurace jednotlivých frameworků, vytvořením OPC UA klienta umožňujícího jednodušší sběr dat pro konfiguraci a návrhem konfiguračního programu.

3.1 Konfigurační parametry

Nejdůležitější částí při návrhu konfiguračního programu bylo zvolení parametrů potřebných pro konfiguraci jednotlivých částí SBDH. Každá jednotlivá část, ze které se skládá SBDH, má svoji vlastní rozsáhlou dokumentaci popisující vlastnosti a funkce, které umožňuje.

Prvním frameworkem, pro který byly zjišťovány parametry, byl Apache Storm. Pro účely seznámení se s frameworkem byla vytvořena instance Apache Stromu pomocí softwaru Docker[1]. Docker obsahuje již funkční verzi vytvořenou a spravovanou přímo vývojáři, kterou stačí jen stáhnout. Na základě dokumentace a získaných zkušeností s Apache Strom byly pro konfiguraci vybrány dva parametry – jeden určující počet procesů zpracovávajících topologii a druhý definující, zda systém zaznamenává události. Ostatní důležité parametry jsou definovány uvnitř topologie nebo nejsou uživatelem konfigurovatelné.

Pro další dva frameworky Apache Cassandra a InfluxDB byly vybrány konfigurační parametry na základě jejich dokumentace. Pro Apache Cassandra je zapotřebí specifikovat, jakým způsobem se bude s databází komunikovat a následně pod jakým jmenným prostorem budou data v databázi ukládány. Stejně jako pro Apache Cassandra je potřeba pro InfluxDB definovat, jakým způsobem se bude s databází komunikovat. Tato část je ještě rozšířena o přihlašování pomocí uživatelského jména a hesla přímo do databáze. Posledním aspektem konfigurace InfluxDB je způsob, jakým se bude zacházet s daty. Jelikož se jedná o databázi využívanou k vytváření analýzy nad daty, není třeba, aby v databázi byly uchovávána všechna data. Proto je zapotřebí nastavit, po jak dlouhou dobu budou data na databázi uchována. Také je

[1] Docker: https://www.docker.com/

potřeba definovat, pod jakým názvem se data budou ukládat, nebo kolikrát se budou v databázi replikovat.

Z dokumentace ke komunikačnímu protokolu OPC UA vyplynulo, že pro uživatele je jedním z nejpodstatnějších parametrů seznam endpointů, které zprostředkovávají služby na serveru. Endpointy definují s jakými servery se bude komunikovat a umožňují vytvořit subscriptions na potřebná data. Dále je třeba definovat pod jakým jmenným prostorem budou data hledána, a která konkrétní data chce uživatel sbírat.

Všechny tyto parametry jsou následně zaznamenány do konfiguračního souboru. Tento soubor je načten topologii v Apache Stormu a následně použit pro konfiguraci i ostatních frameworků. Příklad konfiguračního soubor můžete vidět na obrázku 3.1.

```
storm.workers.number = 1
storm.debug = true

cassandra.nodes = 127.0.0.1
cassandra.port = 7199
cassandra.keyspace = dev

opcua.endpoints = opc.tcp://192.168.1.3:4841/lego-cm-opcua/server, opc.tcp://192.168.1
 .3:4842/lego-cm-opcua/server, opc.tcp://192.168.1.3:4843/lego-cm-opcua/server,
 opc.tcp://192.168.1.3:4844/lego-cm-opcua/server, opc.tcp://192.168.1
 .3:4845/lego-cm-opcua/server, opc.tcp://192.168.1.3:4846/lego-cm-opcua/server,
 opc.tcp://192.168.1.3:4847/lego-cm-opcua/server, opc.tcp://192.168.1
 .3:4848/lego-cm-opcua/server, opc.tcp://192.168.1.3:4850/lego-cm-opcua/server,
 opc.tcp://192.167.1.3:4850/lego-cm-opcua/server
opcua.namespaces.browse = 2
opcua.subscription.limit = 1000
opcua.variablesToRead = CountersData., SettingsData., RuntimeData.Control.TMU,
 RuntimeData.CM.State

influxdb.permit = true
influxdb.connection.host = http://localhost:8086
influxdb.connection.user = admin
influxdb.connection.password = admin
influxdb.dbName = smartcm
influxdb.retentionPolicy.name = rpCM
influxdb.retentionPolicy.onDb = cm
influxdb.retentionPolicy.duration = 1d
influxdb.retentionPolicy.replication = 1
influxdb.retentionPolicy.shard.duration = 12h
influxdb.tagsToStore = RuntimeData.Control.TMU, RuntimeData.CM.State, CountersData
 .Control.OkCount, CountersData.Control.ItemAfterCounting, CountersData.Control
 .ItemCountToFew, CountersData.Control.ItemCountToMany, SettingsData.CM
 .PortionItemCount, SettingsData.CM.PartNo,SettingsData.CF.CountingSpeed, SettingsData
 .LF.CountingSpeed, SettingsData.Belt.CountingSpeed
```

Obrázek 3.1: Konfigurační soubor

Apache Storm

- **Workers number** - Kolik *worker procesů* se má vytvořit pro topologii napříč stroji v clusteru. Každý worker proces tedy vykonává určitou podmnožinu topologie.
- **Debug** - Pokud je debug mód povolen, zaznamenávají se události. Jsou zaznamenávány ve tvaru časová známka, název komponenty, id komponenty, id zprávy, seznam emitovaných hodnot. Události ze všech boltů jsou posílány do eventlogger boltu, který je uchovává.

Apache Cassandra

- **Nodes** - IP adresa Cassandry.
- **Port** - Komunikační port Cassandry.
- **Key space** - Key space, do kterého se dají data ukládat.

OPC UA

- **Endpoints** - Fyzická adresa aplikace dostupná na síti, která umožňuje klientům přistoupit k jedné či více službám nabízených serverem.
- **Variables to read** - Definuje, která data budou ze serveru získávána.
- **Namespace** - Udává prostor, který se během inicializace prohledávat.
- **Subscription limit** - Omezuje maximální počet hodnot, které mohou být do fronty uloženy (velikost fronty).

Celý přístup funguje tak, že uživatel zadá *namespace* a *variables to read*. Při inicializaci se prohledá celý prostor, naleznou se odpovídající proměnné a na ně se poté vytvoří *subscriptions*.

InfluxDB

- **Permit** - Pokud je zvolen mód Permit, data se také posílají do InfluxDB.
- **Host** - Adresa pro připojení do databáze.
- **User** - Přihlašovací jméno.
- **Password** - Přihlašovací heslo.
- **DB name** - Název databáze.
- **Retention policy** - Způsob, jakým databáze zachází s daty.
  - **Name** - Název příslušného zacházení s daty. Pokud je prázdné, odpovídá základnímu nastavení databáze.
  - **On DB** - Název databáze na kterou je aplikovaná retention policy.

Replication - Počet kopií dat v databázi.

Duration - Nastavení doby, po kterou jsou data v databázi uschována.

Shard duration - Definuje časové okno, ve kterém je *shard group* uschována v databázi. Shard group je struktura pro ukládání shards, které uchovávají samotná time series data.

- **Tags to store** - Protože InfluxDB slouží hlavně pro vizualizaci pomocí Grafana SW, tak umožňuje dodatečnou filtraci dat pomocí Tagu.

3.2 Vytvoření projektu

K vytvoření aplikace byl využit programovací jazyk Java. Ten byl vybrán převážně kvůli knihovnám Eclipse Milo a Apache Jena, které jsou volně distribuované. Eclipse Milo umožňuje práci s OPC UA a byl využit hlavně pro vytvoření OPC UA klienta, který je zapotřebí k usnadnění konfigurace SBDH. Apache Jena slouží pro vytváření a práci s ontologii v jazyce OWL/RDF. V této práci je využívána hlavně pro dotazování ontologie pomocí jazyka SPARQL. Dalším potřebným aspektem této aplikace bylo vytvoření grafického rozhraní. Pro tyto účely byla vybrána nativní Java knihovna JavaFX. Tato knihovna slouží převážně k vývoji tzv. Rich Internet Applications (aplikací běžících na webu s některými funkcionalitami desktopových aplikací), umožňuje však i přímo vývoj desktopových aplikací.

Pro tvorbu aplikací v JavaFX se využívají různá vývojová prostředí (jako například NetBeans IDE, Eclipse nebo IntelliJ IDEA), která značně ulehčují vývoj. K vývoji aplikace bylo vybráno IntelliJ IDEA, protože s prací v něm má autor této práce největší zkušenosti a jinak mezi prostředími moc rozdílů není.

IntelliJ IDEA umožňuje vytvoření JavaFX projektu, ale další práce s takto vytvořeným projektem (jako přidávání knihoven nebo vytvoření spustitelné aplikace) je velmi obtížná. Z tohoto důvodu byl zvolen sestavovací nastroj Maven, který je plně podporován Javou. Tento nástroj slouží pro zjednodušení sestavení projektů a k snadnému přidávání knihoven. Způsob přidávání knihoven probíhá pomocí vytvoření externích závislostí na knihovny v souboru pom.xml. Tento soubor slouží k definování jednotlivých částí projektu. JavaFX umožňuje pomocí Mavenu vytvoření základního projektu, který již obsahuje všechny potřebné části pro zprovoznění grafického rozhraní. Jak vytvořit takový projekt je přímo popsané na stránkách JavaFX[2]. Tento základní program má modulární strukturu, což znamená, že všechny moduly využívané z ostatních knihoven je potřeba zapsat do souboru *module-info.java*.

S takto připraveným projektem byly dále přidány závislosti na knihovnu Eclipse Milo, Apache Jena a byl rozšířen proces sestavení projektu o pluginy potřebné k vytvoření aplikace. Pro vývoj bylo zvoleno Java SDK 15, které podporuje jak knihovny Eclipse Milo verze 0.6.4, Apache Jena verze 4.4.0, tak JavaFX verze 17.

[2] JavaFX dokumentace: https://openjfx.io/openjfx-docs/

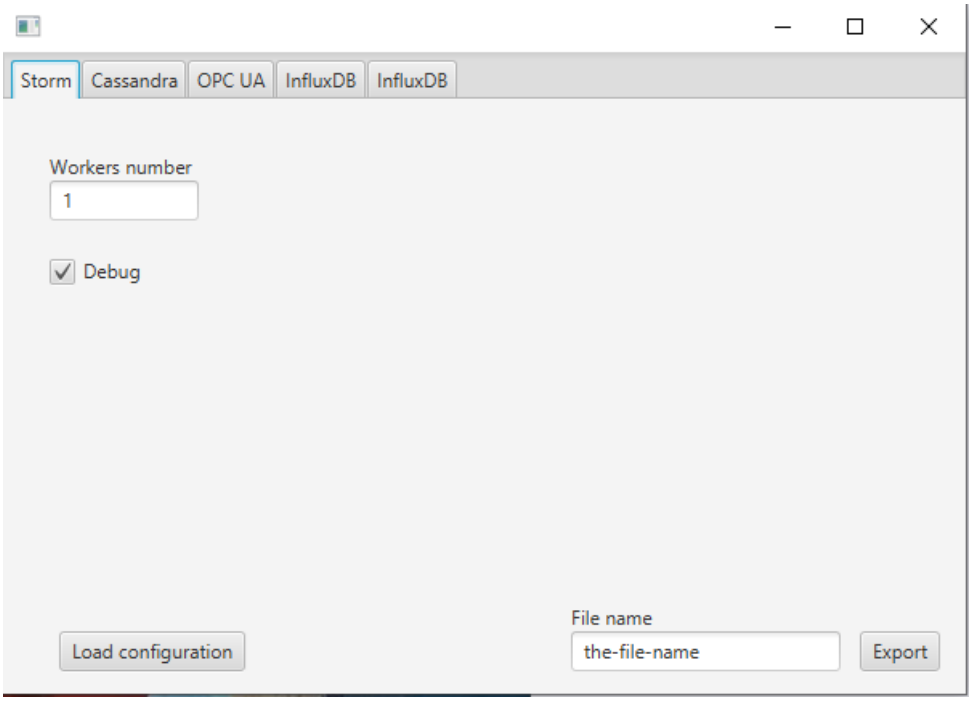
3.3 Uživatelské rozhraní

Uživatelské rozhraní bylo vytvořeno pomocí výše zmíněné Java knihovny JavaFX verze 17. Tato knihovna umožňuje dva způsoby vytváření oken, která se uživateli zobrazí. Prvním je vytvoření rozvržení oken přímo v programu, kam se pak pomocí kódu přidávají tlačítka a další prvky okna. A druhým je za pomocí FXML. Jedná se o jazyk odvozený od XML, který slouží k návrhu jednotlivých oken. Návrh okna se píše do speciálního souboru s koncovkou .fxml, který se dá následně v programu načíst jako nové rozvržení okna, a pak zobrazit uživateli. Pro přidání funkcionality jednotlivým prvkům, se do kódu FXML musí zapsat cesta ke skriptu, který následně implementuje funkcionalitu jednotlivých prvků. Tyto skripty se běžně nazývají *Controller*. Pro provázání jednotlivých proměnných a funkcí z FXML souboru s controllerem je potřeba do controlleru zapsat proměnné a funkce se stejným názvem a připsat k nim dekorátor FXML. Pro návrh oken v FXML existuje grafický program *Scene Builder*, který byl využit pro vytvoření jednotlivých FXML souborů pro konfigurační program. Scene Builder je schopný načíst FXML soubor a zobrazit jej v grafické podobě s možností jej upravovat. Dále umožňuje přiřazovat jednotlivým prvkům v souboru pojmenování a funkce, které se pak dají provázat s controllerem.

3.3.1 Vizuální část

Vizuální návrh hlavního okna se hlavně opíral o množství jednotlivých parametrů pro konfiguraci a jejich rozložení do tříd podle frameworku nebo protokolu, ke kterému patří. Původní návrh rozděloval každou část konfigurace do samostatného okna, mezi nimiž mohl uživatel volit pomocí myši. Jelikož různé části konfigurace nemají stejné množství parametrů, tak některá okna působila velmi prázdně, což můžete vidět na obrázku 3.2 . Také orientace v programu byla nepřehledná kvůli tomu, že uživatel při hledání specifické části konfigurace musel hledat parametr napříč různými záložkami. Z tohoto důvodu byla vizuální část přepracována, tak aby byly všechny parametry vidět na jedné obrazovce. Tato hlavní obrazovka byla rozdělena do jednotlivých segmentů podle toho, na co se daná konfigurace zaměřuje. Do jednotlivých částí byly přiřazeny potřebné parametry pro konfiguraci. Výsledný stav můžete vidět na obrázcích 3.3 a 3.4. Hlavní panel je rozdělen na dva obrázky pro zvýšení čitelnosti. V aplikaci jsou obě části (obr. 3.3 a 3.4) spojené.

Dále bylo pro práci s OPC UA klientem vytvořeno nové okno, které se uživateli zobrazí po zvolení možnosti vyhledávat endpointy v hlavním okně. Po zobrazení se nezavře původní hlavní okno, aby mohl uživatel sledovat data, která pomocí OPC UA klienta přidal. Vzhled tohoto okna můžete vidět na obrázku 3.5. Hlavními aspekty okna je seznam vypsaných serverů a možnost zadat GDS URI. Ostatní grafické položky přibývaly v závislosti na rozšíření funkce OPC UA klienta. Pro zvolení endpointů z jednotlivých nalezených serverů bylo vytvořeno okno s jejich výpisem a možností je uložit.



Obrázek 3.2: Starší verze konfigurace Stormu

Poslední grafickou stránkou je okno pro práci s ontologií. Toto okno je rozděleno do dvou částí. První, která se uživateli zobrazí, je část s jednoduchým výběrem serverů z ontologie viz 3.6. Tato část byla navržena pro ulehčení práce uživatele s ontologií. Druhá část umožňuje uživateli napsat vlastní dotaz v jazyce SPARQL pro získání potřebných dat z ontologie. Toto okno je vyplněno základním dotazem pro ulehčení orientace uživatele v programu viz 3.7.

3.3.2 Implementační část

Implementace byla rozdělena na tři hlavní controllery, kde se každý stará o jedno okno vytvořené pomocí FXML. Komunikace mezi controllery je realizována pomocí předávání závislosti při vytvoření nového controlleru. Všechny controllery pouze reagují na požadavky uživatele a na jejich základě pak komunikují s komplexnějšími částmi kódu a mění vizuální aspekty aplikace.

Controller pro hlavní okno. Zaměřuje se hlavně na možnosti editace a zobrazení zadaných dat pro konfiguraci. Tato data následně na základě uživatelova rozhodnutí posílá nebo načítá ze skriptu pro práci se soubory. Byl vytvořen vlastní prvek umožnující přidávat data do zobrazeného seznamu dat a odebírat je. Controller se pak stará o propojení a správnou funkčnost těchto prvků. Při spuštění vyhledávání endpointů uživatelem je controllerem vytvořena nová scéna a načten rozložení okna z FXML souboru pro práci s OPC UA klientem.

Controller pro práci s OPC UA klientem. Přeposílá uživatelem zadaná data do OPC UA klienta a následně zobrazuje navrácená data z klienta. Jelikož při zadání špatných dat do klienta může dojít chybě, byla implementována vyskakovací okna s upozorněním, která uživatele upozorňují na jednotlivé chyby, jako je například, že žádaný GDS je nedostupný. Tento controller vytváří dvě další okna. Jedno na práci s ontologii a druhé, kde se zobrazuje výpis endpointů na uživatelem vybraném serveru.

Controller pro práci s ontologii. Při spuštění vytvoří vyhledávací okénko a vyžaduje po uživateli zvolení používané ontologie. Následně tuto adresu zašle OPC UA klientovi, který si načte model ontologie. Na začátku je automaticky vytvořen dotaz na ontologii, který vrátí všechny podtřídy senzorů ontologií definované. Tyto podtřídy jsou poté uživateli zobrazeny a on může zvolit ty, které chce vyhledat. Pokud uživatel zvolí možnost vyhledat, controller zašle všechny zvolené podtřídy klientovi, který pak vrátí seznam nalezených serverů. Toto okno také nabízí možnost vytvoření vlastního dotazu pomocí jazyka SPARQL. Z důvodů volnosti při psaní SPARQL dotazů je potřeba zadat název proměnné z klauzule SELECT daného dotazu, která reprezentuje chtěné typy zařízení. Tento parametr a dotaz jsou následně zaslány do OPC UA klienta. Klient SPARQL dotazem zjistí z ontologie požadované typy zařízení. Ty se následně vyhledají na GDS a vrátí se jejich seznam uživateli.

3.4 Tvorba OPC UA klienta

OPC UA klient pro tuto aplikaci byl vytvořen za pomoci knihovny Eclipse Milo. Tato knihovna implementuje komunikaci pomocí OPC UA protokolu podle standardu OPC Foundation. Pro tuto knihovnu neexistuje žádná dokumentace, což značně ztěžuje proces vývoje. Celá knihovna lze stáhnout přímo z github stránky projektu[3], tudíž je možné si celý kód knihovny projít a zorientovat se v něm i bez dokumentace pomocí popisků u jednotlivých funkcí. Na daném githubu je dostupná řadu ukázkových příkladů, které také značně napomohly seznámení se s touto knihovnou.

Pro komunikaci s GDS dostane klient konfiguračního souboru od uživatele GDS URI. Poté si klient načte adresář pro ukládání certifikátů. Pak je vytvořen certifikát pro klienta a nalezen endpoint pro komunikaci se serverem. Pokud se nepovede navázat spojení s GDS pro získání endpointu, pak klient oznámí chybu, že server je nedostupný. Po získání všech potřebných dat je vytvořena instance klienta připravená pro komunikaci se serverem v anonymním režimu.

Před zahájením komunikace je ještě zapotřebí ověřit, zda klient vyžádanému GDS důvěřuje. To je vyhodnoceno podle toho, zda má klient certifikát GDS ve svém seznamu důvěryhodných certifikátů. Když tomu tak není, klient certifikát vloží do svého seznamu důvěryhodných certifikátů, pokud mu to uživatel dovolil. Jestli tato možnost uživatelem nebyla zvolena, klient certifikát GDS vloží do své zabezpečené složky mezi odmítnuté certifikáty a oznámí

[3]Eclipse Milo github: https://github.com/eclipse/milo

chybu, že GDS není důvěřováno. Z té to složky jej může následně uživatel přemístit do složky s důvěryhodnými certifikáty. Pokud je serveru důvěřováno, klient s daným serverem naváže spojení.

Po zahájení komunikace klient vyžádá na serveru metodu poskytující seznam registrovaných serverů. Nalezení této metody počítá s tím, že GDS je implementováno podle norem definovaných standardem OPC UA. Na základě tohoto standardu se tato metoda nachází vždy na stejném uzlu v hierarchii serveru. Tato metoda umožňuje filtrovat servery podle různých parametrů. Pro konfigurační program je nejdůležitějším parametrem filtrace pomocí product URI. Tato metoda neumožňuje zaslat vícero product URI najednou, tudíž je klientem zavolána tolikrát, kolik product URI uživatel zadal. GDS jako odpověď vrací popis s informacemi o registrovaných serverech. Tato struktura se nazývá *ApplicationDescription*. Po získání všech požadovaných serverů zašle klient informace o nich controlleru, který je uživateli zobrazí.

Další informace o endpointech na jednotlivých registrovaných serverech jsou získávány pomocí discovery klienta implementovaného v přímo v knihovně Eclipse Milo. Tento klient umožňuje získání endpointů s LDS daných serverů bez nutnosti navazovat komunikaci pomocí certifikátů. Pokud je server nedostupný, je uživateli zobrazeno chybové okno.

Pro větší usnadnění práce s aplikací byla v klientovi implementována možnost získávat potřebné informace z ontologií, kterou si uživatel zvolí. Toto rozšíření bylo provedeno pomocí knihovny Apache Jena. Knihovna se zde využívá k načtení modelu ontologie a následnému zasílání dotazů na ni pomocí jazyka SPARQL. Následně jsou URI navrácené z ontologie rozděleny a jednotlivě používány pro filtraci serverů registrovaných na GDS, se kterým klient komunikuje. Uživateli je umožněno napsat vlastní dotaz, nebo použít jeden z předdefinovaných programem.

3.5 Testování OPC UA klienta

3.5.1 Vytvoření testovacího prostředí

Pro testování OPC UA klienta bylo nutné vytvořit funkční GDS, se kterým by klient mohl komunikovat, dále vytvořit LDS, které se musely do GDS zaregistrovat. Pro tyto účely byl využit veřejně dostupný projekt od OPC Foundation s názvem UA-.NETStandard-Samples[4]. Tento projekt je vytvořen pomocí vývojové platformy .NET, což je vývojová platforma s velkým množstvím nástrojů a knihoven určených k vývoji aplikací primárně určených pro MS Windows. Pro vývoj v .NET není vyžadován žádný specifický programovací jazyk, jelikož je původně napsaná aplikace vždy přeložena do Common Intermediate Language. Nejčastěji využívané programovací jazyky jsou C# nebo Visual Basic .NET. UA-.NETStandard-Samples je napsán pomocí jazyka C#.

[4]Stránka na githubu: https://github.com/OPCFoundation/UA-.NETStandard-Samples

Pro vývoj aplikací pomocí .NET vzniklo vývojové prostředí Visual Studio obsahující všechny potřebné pluginy. K zprovoznění projektu UA-.NET-Standard-Samples bylo zapotřebí nainstalovat Visual Studio 2019 s rozšířením o vývoj desktopových aplikací pomocí .NET a pluginem pro ukládání a zpracovávání dat. Tento plugin byl zapotřebí kvůli tomu, že daný projekt využívá SQL Server Data Tools. Posledním krokem k zprovoznění projektu bylo externí nainstalování vývojářského balíčku .NET verze 4.6.2. který byl projektem přímo vyžadován.

Celý projekt obsahuje velké množství OPC UA serverů a klientů. Pro účely testování byl využit Global Discovery Client (GDC) a Global Discovery Server (GDS). GDC zde funguje pro registraci serverů do GDS. GDS obsahuje tři různé přihlašovací účty, z nichž každý má jiná oprávnění. Pro registraci serverů je vyžadováno mít oprávnění administrátorské.

K připojení klienta na server byl využit účet s přihlašovacím jménem appadmin a heslem demo, který vlastní administrátorské oprávnění. Tyto přihlašovací údaje byly získány z dokumentace projektu. Po navázání komunikace mezi klientem a serverem klient poskytuje různé typy registrace, z nichž byl použit registrační typ Server Pull Management. Pro registraci serveru je potřeba definovat jednotlivé informace o něm. Na obrázku 3.8 můžete vidět registrační okno klienta. Parametry potřebné pro registraci budou dále popsány v kapitole Popis registračních parametrů.

Klient umožňuje i načíst konfiguraci pro registraci z xml souboru, nebo také vytvořit konfigurační soubor z dat zadaných uživatelem. Po vyplnění registračních parametrů se zvolí možnost Registrovat a klient zobrazí uzel, do kterého by měla být uložena registrace serveru na GDS. Tento uzel se ještě v průběhu registrace může změnit. V následujícím kroku je potřeba vygenerovat certifikát, který bude využívat GDS ve svém seznamu důvěryhodných serverů – takzvaném Trust Listu. Posledním krokem v registraci je spojení vygenerovaného certifikátu s certifikáty již vlastněnými GDS.

Jedním z registrovaných serverů byl UA Sample Client, který je součástí již výše zmíněného projektu od OPC Foundation. Tento klient měl již vytvořený konfigurační soubor pro registraci, takže byla využita možnost načtení registračních parametrů pomocí něj. Dalším registrovaným serverem bylo veřejně dostupné LDS (také od OPC Foundation). Jedná se o instalovatelnou aplikaci běžící na pozadí operačního systému. Tento server byl přímo vyvinut pro OS Windows a je jednoduše ovladatelný pomocí administrátorské příkazové řádky Windows. Jednotlivé informace pro registraci tohoto serveru byly zjištěny z dokumentace.

Klient SBDH konfiguračního programu využívá vyhledávání serverů na základě product URI. Tyto URI je možné napsat buď ručně, nebo je vyhledat ve zvolené ontologii pomocí dotazu v jazyce SPARQL. Klient zohledňuje, že daná ontologie využívá Semantic Sensor Network Ontology. Pro účely testování byla zvolena COCI ontology, která popisuje řadu různých senzorů pro malou vodní elektrárnu. Pro možnost otestovat správnost dotazů na ontologii bylo registrovaným serverům změněno product URI na URI definované v ontologii COCI.

Po registraci obou serverů byla úspěšnost registrace ověřena pomocí programu UAExpert. Jedná se o OPC UA klienta navrženého k testování, který umožňuje velké množství funkcí definovaných OPC UA standardem. Celý program má velmi rozsáhlé grafické zpracování a značně tak ulehčuje orientaci při procházení kontrolovaného serveru. Ověření proběhlo pomocí připojení na GDS jako běžný uživatel bez specifických oprávnění. Následně na základě znalosti architektury GDS dle OPC UA standardu byla nalezena poskytovaná metoda QueryServer. Tato metoda slouží k získávání registrovaných serverů na daném GDS. Této metodě je třeba určit, od kolikátého záznamu má data vracet, a kolik maximálně jich má být. Další parametry volání jsou nepovinné. Po provolání této metody bylo zjištěno, že oba výše zmíněné servery jsou zaregistrované na daném GDS, a že jejich registrační data odpovídají datům při registraci.

3.5.2 Popis registračních parametrů

- **Application ID** - Unikátní identifikátor přiřazený aplikaci po registraci daným GDS.
- **Application Name** - Základní název aplikace.
- **product URI** - URI dané aplikace, které je také uloženo v jejím certifikátu.
- **Discovery URLs** - Seznam obsahující všechny URL adresy, pomocí kterých je daná aplikace vyhledatelná.
- **Server Capabilities** - Seznam všech *Server Capabilities*, což jsou identifikátory definované OPC UA standardem určující funkce serveru. Jsou využívané pro filtrování aplikací v průběhu fáze *discovery*.
- **Certificate Store Path** - Místo uložení certifikátu. Pro lokální uložení se využívá adresa *CurrentUser\UA_MachineDefault*, ale lze zvolit i umístění do uživatelem specifikované složky.
- **Certificate Subject Name** - Jméno pro certifikát odlišující ho od ostatních certifikátů.
- **Trust List Store Path** - Volitelný parametr. Slouží pro zkopírování veřejného certifikátu do důvěryhodné složky.
- **Issuer List Store Path** - Volitelný parametr. Slouží pro zkopírování veřejného certifikátu do složky vydavatele.
- **Domains** - Název domény rozšiřující certifikát. Může se jednat o IP adresu nebo jméno hosta.

3.5.3 Průběh testování

Pro testování OPC UA klienta bylo navrženo devět testovacích scénářů, které pokrývají všechny možnosti, co daný klient poskytuje. Průběh testování se celý odehrával v testovacím prostředí, jehož vznik a ověření správnosti byly popsány v kapitole Vytvoření testovacího prostředí. Všechny scénáře začínají poté, co uživatel zapne aplikaci a zvolí si možnost nalezení endpointů pomocí OPC UA klienta. Všechny zmíněné scénáře prošli úspěšně.

Ke GDS se nelze připojit. Problémy s připojením ke GDS mohou vzniknout ze dvou různých důvodů. Jedním je buď špatně zapsané nebo neexistující discovery URI GDS. Druhým, že dané GDS zrovna není aktivní. Klient si ověřuje možnost připojení ke GDS ve chvíli, kdy se na něm uživatel snaží vyhledat servery, nebo se snaží pro vyhledávání použít ontologii. K otestování této funkce byl vypnut používaný GDS v testovacím prostředí a následně se uživatel pokusil jak najít servery na GDS, tak najít servery s pomocí ontologie. V obou případech program zobrazí chybové okénko s oznámením, že se nepodařilo k serveru připojit.

Klient GDS nedůvěřuje. Před připojením na GDS má uživatel možnost si zvolit, zda bude danému serveru důvěřovat. Znamená to, že si uloží certifikát daného serveru do svého seznamu důvěryhodných serverů (trust listu). Pro připojení k serveru je potřeba, aby klient měl server v seznamu důvěryhodných serverů, jinak nedojde k navázání spojení. Při testování uživatel zadal GDS URI a nezvolil možnost důvěřovat serveru. Program správně zobrazil chybovou hlášku, že daný server není důvěryhodný. Následně uživatel ve složce obsahující certifikáty nalezl certifikát v podsložce odmítnutých certifikátů. Ten dále přesunul do složky s důvěryhodnými certifikáty a opětovně se pokusil připojit ke GDS. Další pokus o připojení proběhl bez problému.

Klient GDS důvěřuje. Uživatel zvolil v programu možnost důvěřovat GDS při připojení a zadal GDS URI. Klient se bez problémů připojil k danému GDS. Následně bylo i zkontrolováno, že se certifikát GDS opravdu nachází v bezpečnostní složce klienta.

Nalezení všech serverů v GDS. Uživatel zadal GDS URI a zvolil možnost věřit serveru. Následně zmáčkl tlačítko nalézt. Po připojení na GDS se vypsaly všechny zaregistrované servery na GDS. Tento výpis byl následně ověřen pomocí programu UaExpert.

Nalezení serverů po zadání product URI. Z minulých částí testování byl zvolen jeden z vypsaných serverů a bylo použito jeho product URI pro filtraci. Po stisknutí tlačítka „Find“ byl vrácen pouze daný server. Následně bylo zapsané neplatné product URI a GDS nevrátil žádný záznam.

Nalezení serverů pomocí ontologie z výběru senzorů. Uživatel po vyplnění GDS URI klikl na tlačítko „Find product URI using ontology“. Otevřelo nové okno požadující vybrání dané ontologie. Po jejím výběru se uživateli

zobrazily všechny podtřídy senzorů, které daná ontologie obsahuje. Uživatel zvolil jednu z podtříd, která obsahuje product URI přiřazené jednomu ze serverů registrovaných na GDS. Po stisku tlačítka „Find“ se okno samo zavřelo a v původním okně se vypsal zaregistrovaný server se správným product URI, které odpovídalo zvolené podtřídě z ontologie.

Nalezení serverů pomocí vlastního dotazu na ontologii. Stejně jako v minulém scénáři si uživatel zvolil možnost nalézt servery pomocí ontologie. Jelikož soubor s ontologií zvolil již v minulém kroku, nebyl dotázán na její opětovné zvolení a byla vybrána ta samá ontologie, co minule. Uživatel v horní liště překlikl na záložku s dotazy. V tomto okně si uživatel může sám napsat dotaz pomocí jazyka SPARQL. Dané okno již obsahuje základní dotaz na jednu podtřídu definovanou v ontologii. Uživatel tento dotaz přepsal tak, že si vyžádal jinou podtřídu. Následně zapsal výstupní proměnnou daného dotazu do kolonky pro výstupní proměnnou a klikl na tlačítko „Find“. Okno se samo zavřelo a správně se vypsal server odpovídající dotázané podtřídě z ontologie.

Nalezení endpointu pomocí připojení na LDS vybraného serveru. Uživatel nalezl všechny servery registrované na GDS a zvolil jeden z nich. Následně klikl na tlačítko umožnující se připojit na LDS daného serveru a nalézt jeho endpointy. Klient se úspěšně připojil k LDS a vypsal všechny endpointy, které dané LDS obsahovalo. Výpis byl znovu ověřen pomocí programu UaExpert.

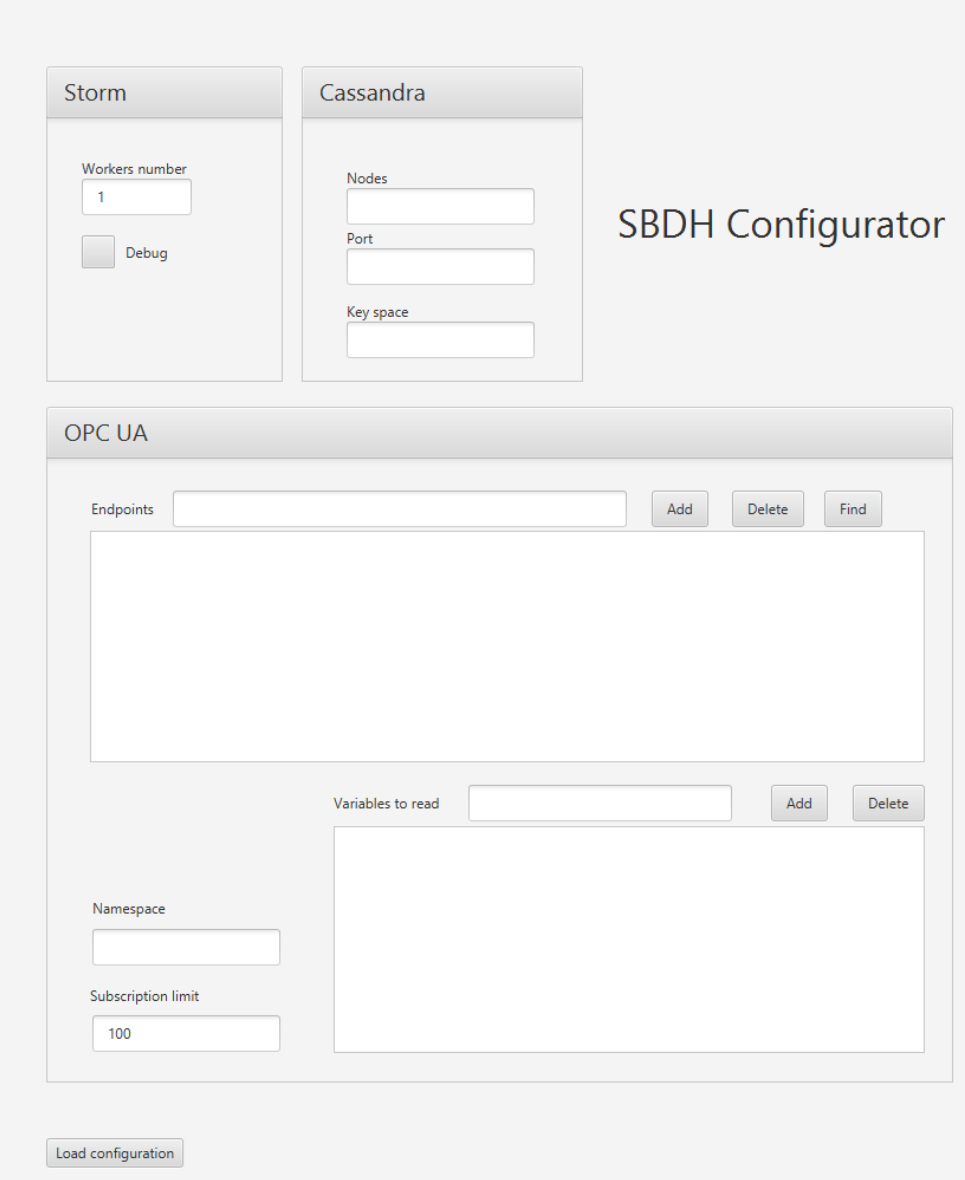
Nelze se připojit k LDS vybraného serveru. Pro tento test bylo zapotřebí vypnout vybraný server. Jelikož GDS uchovává data o registrovaných serverech bez nutnosti aktivity daných serverů, tak i přes vypnutí serveru se po vyhledání všech registrovaných serverů zobrazí ve výpisu. Následně si klient vybral tento neaktivní server a zvolil možnost nalezení endpointů na LDS. Kvůli nemožnosti připojení klienta k serveru program zobrazil chybové okno oznamující, že je daný server nedostupný.

Nalezení endpointů vybraného serveru z GDS. V případě, že je vybraný server nedostupný, může uživatel zvolit možnost nalezení endpointů uložených pro daný server v GDS. Tento seznam nemusí být úplný a proto je lepší dotazovat se přímo LDS. Uživatel po vybrání serveru a možnosti nalezení endpointů na LDS zjistil, že se k serveru nedá připojit, a tak zvolil možnost nalezení endpointů pomocí GDS. Zobrazilo se okno s výpisem všech zaregistrovaných endpointů pro daný server v GDS. Výpis byl znovu zkontrolován s daty v programu UaExpert.

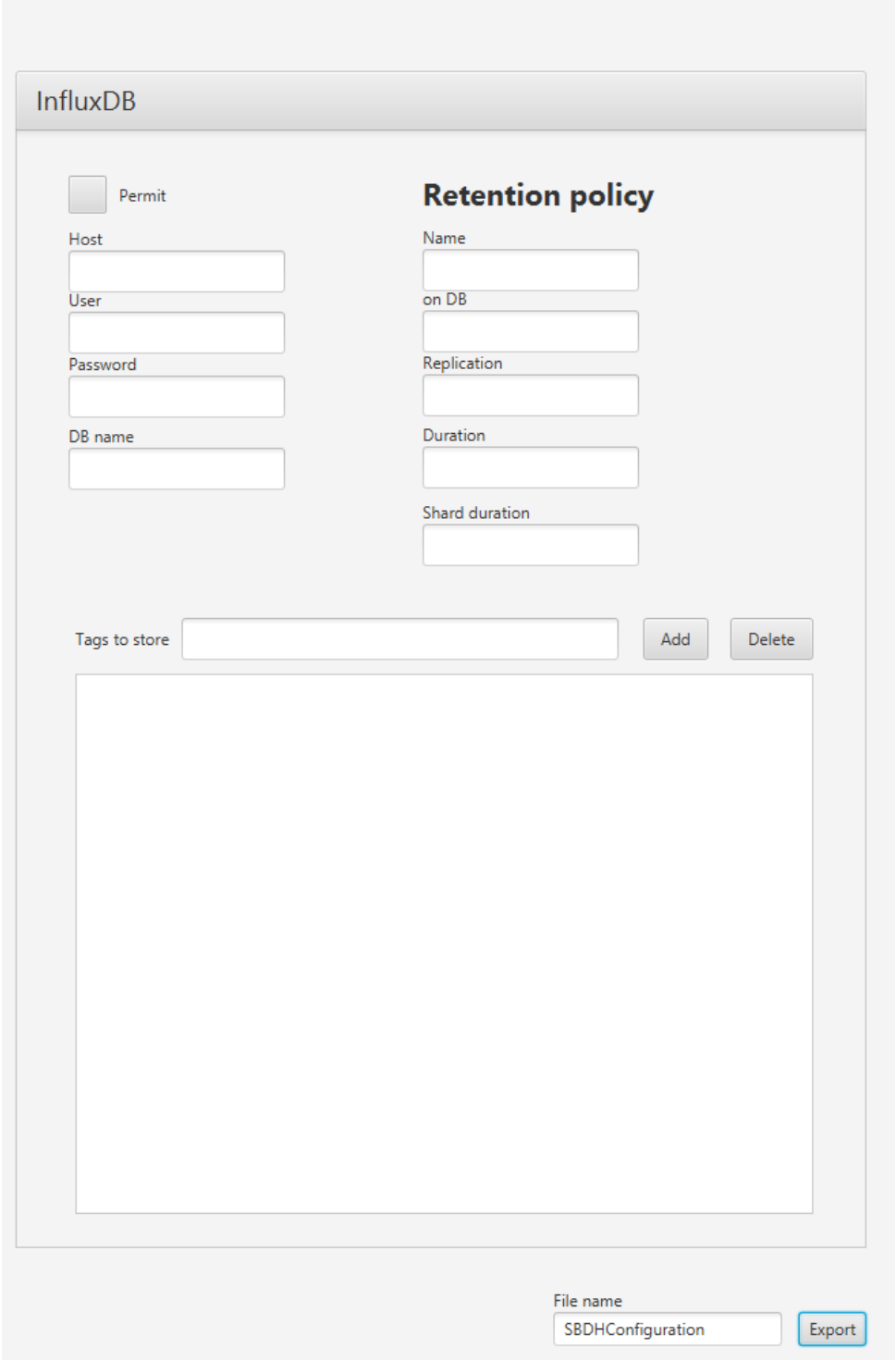
3.6 Testování práce s textovými soubory

Testování práce s textovými soubory probíhalo na třech částech. Jejich načtení, úprava a uložení. Pro načtení byl využit konfigurační soubor používaný pro konfiguraci SBDH využívaného pro výrobní linku specializující se na balení lego elementů. Tento konfigurační soubor byl přepsán, aby odpovídal aktuálnímu formátu konfiguračních souborů generovaných programem pro

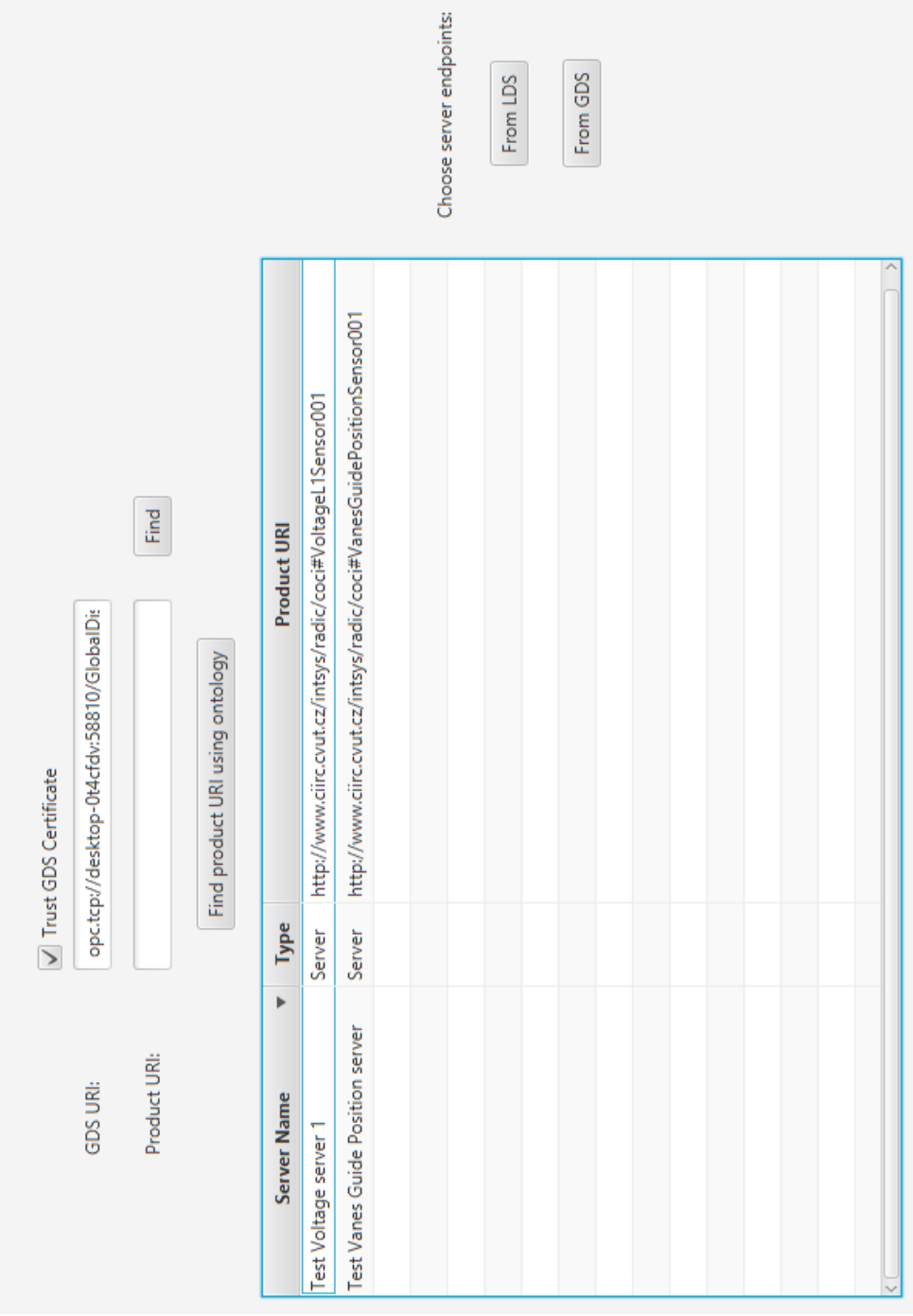
konfiguraci SBDH. Načtení souboru prošlo bezchybně a všechny udaje uložené v konfiguračním souboru byli rozepsány do jednotlivých částí konfiguračního programu. Následně byla zkontrolována správnost údajů oproti samotnému souboru. Všechny části konfigurace byli pomocí konfiguračního souboru buď úplně přepsány nebo částečně upraveny. Pro kontrolu uprav byl poté vyexportován nový konfigurační soubor pod jiným názvem než původní a bylo zkontrolováno že jsou data přepsána. Celé testování proběhlo správně a podle zadané specifikace.



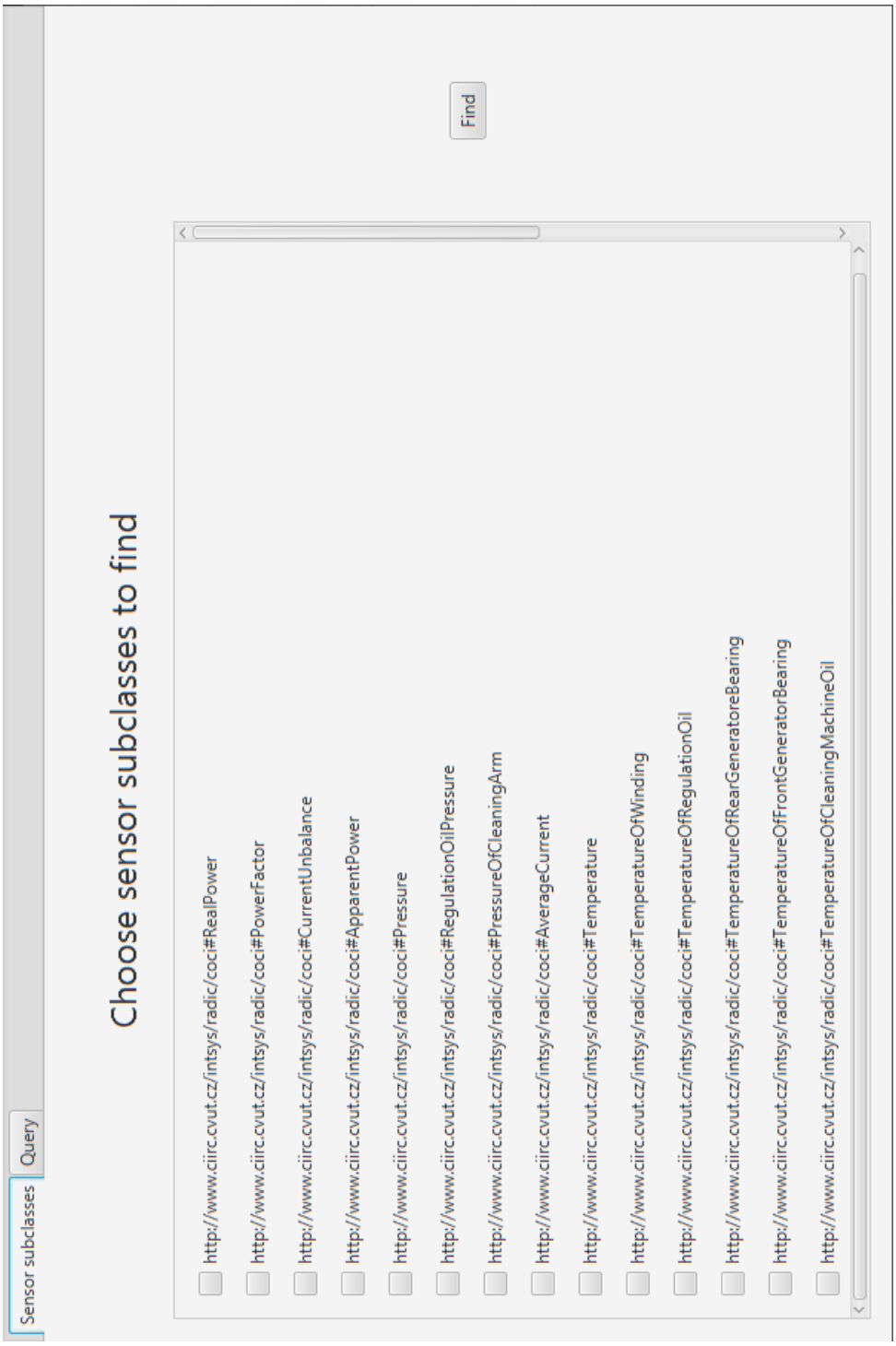
Obrázek 3.3: SBDH hlavní panel - levá část



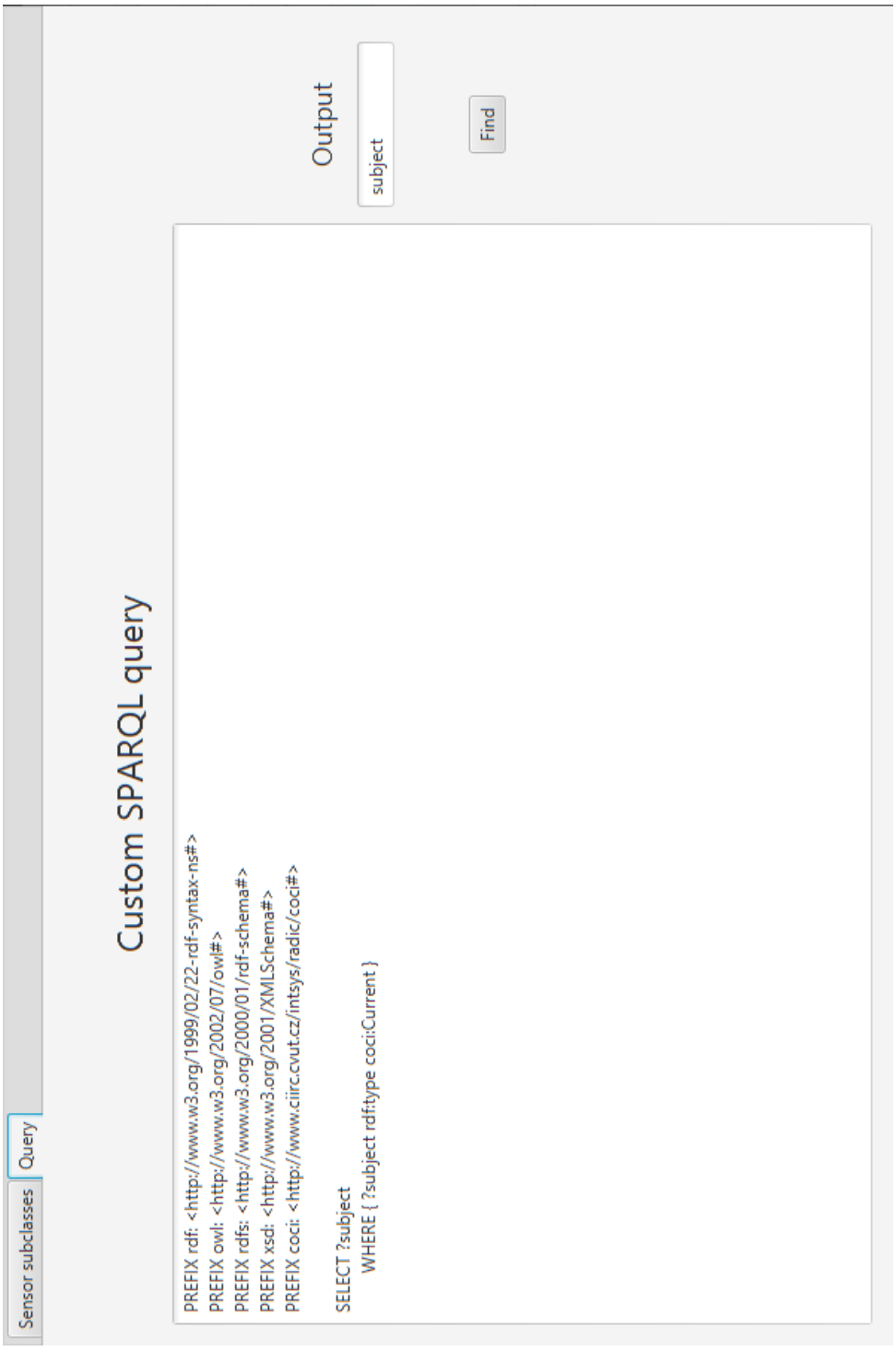
Obrázek 3.4: SBDH hlavní panel - pravá část



Obrázek 3.5: Panel OPC UA klienta



Obrázek 3.6: Panel pro ontologii s připravenými dotazy



Obrázek 3.7: Panel pro ontologii s vlastním dotazem

Obrázek 3.8: Klient pro registraci serverů do GDS

Kapitola 4

Závěr

Cílem této práce bylo vytvořit aplikaci ulehčující konfiguraci frameworků Apache Storm, Apache Cassandra, InfluxDB a komunikačního protokolu OPC UA. Výsledné rozhraní načítá konfigurační soubory, umožňuje upravit, přidat či odebrat parametry konfigurace a ty pak uložit zpět do souboru. Toto rozhraní bylo vytvořeno pomocí knihovny JavaFX a aplikace Scene-builder.

Dále byl implementován OPC UA klient, který umožňuje komunikaci s GDS za účelem získání registrovaných serverů s možností filtrace pomocí product URI. Toto filtrování bylo napojeno na komunikaci s uživatelem zvolenou ontologií, oproti které je možná zaslat vlastní dotaz, nebo vybrat z dotazů předdefinovaných. Na základě odpovědi z ontologie se následně filtrují vyhledávané servery. Klient umožňuje také komunikaci s LDS nalezených serverů, čímž lze získat jejich endpointy. Tento klient byl realizován pomocí knihoven Eclipse Milo a Apache Jena.

V průběhu vytváření aplikace bylo realizováno testovací prostředí pro implementovaného OPC UA klienta. Testovací prostředí se skládá z jednoho GDS a dvou LDS. Při testování dotazů na ontologii byla využita COCI ontologie definující senzory pro malou vodní elektrárnu pomocí Semantic Sensor Network Ontology. V tomto prostředí byla aplikace odladěna a bylo ověřeno, že funguje správně.

Aplikace by do budoucna šla rozšířit o možnost prohledávání informačních modelů dostupných na nalezených OPC UA serverech. To by sloužilo pro snazší nalezení proměnných, které chce uživatel sbírat. Dále by také bylo vhodné vytvořit více uživatelsky přívětivé prostředí pro vytváření dotazů na ontologii. To by se mohlo skládat z předdefinovaných dotazů, které by si uživatel mohl volit a kombinovat, a tak vytvořit svůj vlastní dotaz.

Literatura

[1] CERAVALO, P. et al. Big Data Semantics. 2018. Journal on Data Semantics 7, 65–85. Dostupné z: https://doi.org/10.1007/s13740-018-0086-2

[2] Oden Technologies. Oden Technologies [online]. New York, NY 10011: Oden Technologies, 2020 [cit. 2021-12-22]. Dostupné z: https://oden.io/glossary/data-historian/

[3] OBITKO, Marek; JIRKOVSKÝ, Václav. Big data semantics in industry 4.0. In: International conference on industrial applications of holonic and multi-agent systems. Springer, Cham, 2015. p. 217-229.

[4] SINGH, Sachchidanand a Nirmala SINGH. Big Data analytics [online]. 2012. 2012 International Conference on Communication, Information & Computing Technology (ICCICT): IEEE, 2012 [cit. 2021-12-21]. ISBN 978-1-4577-2078-9. Dostupné z: https://ieeexplore.ieee.org/document/6398180

[5] OWL Web Ontology Language Overview. W3C [online]. International: W3C, 2004 [cit. 2022-04-20]. Dostupné z: https://www.w3.org/TR/2004/REC-owl-features-20040210/

[6] WINGERATH, Wolfram, et al. Real-time stream processing for Big Data. it-Information Technology, 2016, 58.4: 186-194.

[7] Apache Hadoop. Apache Hadoop [online]. U.S.A.: Apache Software Foundation, 2020 [cit. 2021-12-22]. Dostupné z: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Overview

[8] Intel Corporation. Optimizing Hadoop Deployments. Technical report, 2010. Dostupné z: http://www.intel.com/content/dam/doc/white-paper/cloud-computing-optimizing-hadoop-deployments-paper.pdf

[9] Tutorial. Apache Storm [online]. Apache Software Foundation, 2022 [cit. 2022-05-02]. Dostupné z: https://storm.apache.org/releases/current/Tutorial.html

[10] Apache Spark. Apache Spark [online]. U.S.A.: Apache Software Foundation, 2021 [cit. 2021-12-22]. Dostupné z: https://spark.apache.org/docs/latest/index.html

[11] MONIRUZZAMAN, A. B. M.; HOSSAIN, Syed Akhter. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. arXiv preprint arXiv:1307.0191, 2013.

[12] The Apache Software Foundation. Apache Cassandra [online]. U.S.A.: The Apache Software Foundation, 2021 [cit. 2021-12-22]. Dostupné z: https://cassandra.apache.org/

[13] BADER, Andreas. Comparison of time series databases. 2016. Master's Thesis.

[14] NAQVI, Syeda Noor Zehra; YFANTIDOU, Sofia; ZIMÁNYI, Esteban. Time series databases and influxdb. Studienarbeit, Université Libre de Bruxelles, 2017, 12.

[15] OPC Foundation. OPC Foundation [online]. USA: OPC Foundation, 2021 [cit. 2021-12-22]. Dostupné z: https://opcfoundation.org

Příloha A

Obsah elektronické přílohy

1. Documentation - Tato složka obsahuje vygenerovaný JavaDoc popisující nejdůležitější části finálního konfiguračního programu.
2. LaTeXSource - Tato složka obsahuje zdrojový soubor bakalářské práce ve formátu LaTeX.
3. SBDHConfigurator - Tato složka obsahuje zdrojový kód finálního konfiguračního programu.
4. ConfigurationFile.txt - Konfigurační soubor vygenerovaný pomocí konfiguračního programu.