VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

OPC UA KLIENT PRO KOMUNIKACI S PLC B&R AUTOMATION

OPC UA CLIENT FOR COMMUNICATION WITH PLC B&R AUTOMATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Lukáš Staněk
AUTHOR

VEDOUCÍ PRÁCE — Ing. et Ing. Stanislav Lang, Ph. D.
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Lukáš Staněk**
Studijní program:	Strojírenství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. et Ing. Stanislav Lang, Ph.D.**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

OPC UA klient pro komunikaci s PLC B&R Automation

Stručná charakteristika problematiky úkolu:

Práce bude věnována problematice datové komunikace prostřednictvím moderního komunikačního protokolu OPC UA. Student si vybere vhodný framework (OPC UA stack) a na něm založí vlastní aplikaci OPC UA klienta (pro běžné PC). Vytvořený OPC UA klient bude sloužit pro čtení a zápis hodnot proměnných na PLC B&R Automation (kde je vestavěný OPC UA server).

Cíle bakalářské práce:

Proveďte rešerši komunikačního protokolu OPC UA.
Zvolte programovací jazyk a vyberte vhodný framework (OPC UA stack) pro tvorbu aplikace.
Vytvořte vlastní aplikaci jednoduchého OPC UA klienta.
Přiložte stručný návod pro ovládání aplikace.

Seznam doporučené literatury:

OPC Foundation. OPC Unified Architecture: Interoperability for Industrie 4.0 and the Internet of Things [online]. [cit. 2020-10-23]. Dostupné z: https://opcfoundation.org/wp-content/uploads/2017/11/OP--UA-Interoperability-For-Industrie4-and-IoT-EN.pdf

B&R Industrial Automation. OPC UA for motion control, safety and real-time applications [online]. [cit. 2020-10-23]. Dostupné z: https://www.br-automation.com/cs/technologie/opc-ua-for-motion-control-safety-and-real-time-applications/

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2020/21

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Tato práce je rešerší komunikačního protokolu OPC UA s následnou ukázkou jeho implementace do vytvářeného klienta. Rešerše začíná historickým vývojem a následně přechází v teoretický rozbor OPC UA jehož výstupy jsou implementovány v aplikaci OPC UA klienta. Tento klient, který je výstupem práce, následně demonstruje praktické využití OPC UA. Současně je poukázáno na vhodné postupy při vývoji aplikace, které jsou platné všeobecně.

ABSTRACT

This thesis is a research of the OPC UA communication protocol with a subsequent example of its implementation in the creating client. The research begins with historical development and then focuses on to the theoretical analysis of OPC UA, these outputs are implemented in the client application OPC UA. This client, which is output of this thessis, then demonstrates the practical use of OPC UA. In this thesis are shown suitable procedures for development of applications, which are generally valid.

KLÍČOVÁ SLOVA

OPC, OPC Foundation, OPC Classic, OPC UA, UA .Net Stack, PLC komunikace, komunikace klient – server, OPC UA klient.

KEYWORDS

OPC, OPC Foundation, OPC Classic, OPC UA, UA .Net Stack, PLC communication, client – server communication, OPC UA client.

BIBLIOGRAFICKÁ CITACE

STANĚK, Lukáš. *OPC UA klient pro komunikaci s PLC B&R Automation,* Brno, 2021, 49 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky.

PODĚKOVÁNÍ

Děkuji Ing. et Ing. Stanislavu Langovi, Ph. D. za jeho ochotu, rady a cenné připomínky, které mi pomohli při tvorbě této práce. Taktéž děkuji mojí manželce a prarodičům mých dětí za neskutečnou podporu při studiu.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením Ing. et Ing. Stanislava Langa, Ph.D. a s použitím literatury uvedené v seznamu literatury.

V Brně dne 22. 5. 2021 ……

Lukáš Staněk

OBSAH

1 ÚVOD

Práce *OPC UA klient pro komunikaci s PLC B&R Automation* pojednává o relativně novém komunikačním protokolu. Autor si toto téma zvolil, jelikož ho zaujala možnost dozvědět se více o aktuálně se rozšiřujícím komunikačním protokolu při současném rozšíření svých programátorských schopností. Zajímavost tohoto protokolu podtrhuje také skutečnost, že se postupně začíná používat v různých odvětvích, ač byl původně určen pro průmyslové využití.

První část této práce pojednává o historickém vývoji OPC UA, které navazuje na svého „předchůdce“, dnes nazývaného OPC Classic. V úvodu této kapitoly jsou historické návaznosti, které předcházely vzniku samotného OPC UA, zejména vznik samotného organizace OPC Foundation stojící za vývojem celého OPC. Část kapitoly rozebírá jednotlivé specifikace OPC Classic používané před vznikem OPC UA.

Druhá část představuje protokol samotný. Začíná popisem komunikačního modelu s následným rozborem jednotlivých vrstev protokolu. Podkapitola Zabezpečení OPC UA je seznámením s principy bezpečné OPC UA komunikace a jednotlivými možnostmi zabezpečení. Je zde také uveden popis získání symetrických klíčů pro šifrování komunikace. Tento teoretický blok uzavírá popis přístupu k informacím v rámci jmenných prostorů na příslušných serverech, a to včetně popisu objektů sloužících k poskytování informací.

Poslední část je praktická a navazuje na teoretický úvod této práce. V úvodu jsou rozebrány vstupní požadavky na klienta na jejichž základě bylo postupováno dále. V rámci popisu zvoleného frameworku je provedeno seznámení s metodami potřebnými pro sestavení spojení, čtení a zápis dat do PLC. Jsou zde popisovány dvě metody pro získávání dat ze serveru, jedna pro jednorázové získání aktuální hodnoty a druhá pro odběr dat na základě jejich změny. Následuje představení vývoje vytvořeného klienta doplněné o ukázky některých již existujících klientů.

2 HISTORIE OPC UA

OPC Unified Architecture (UA), bylo vytvořeno v rámci organizace OPC Foundation. OPC Foundation popisuje OPC UA jako platformově nezávislou architekturu orientovanou na služby, která integruje všechny funkce jednotlivých specifikací OPC Classic do jednoho rozšiřitelného rámce. [1]

2.1 Vznik OPC a OPC FOUNDATION

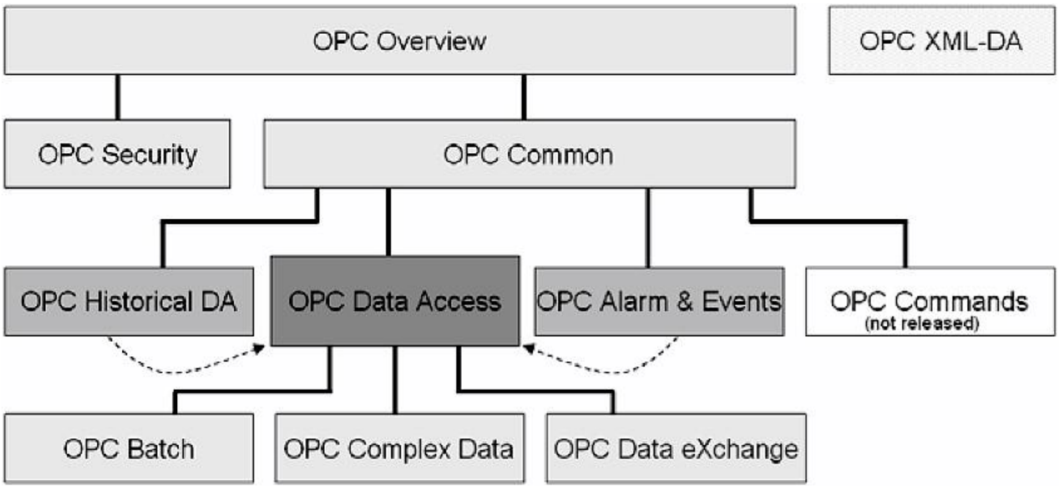
Začátkem devadesátých let došlo v průmyslu k velkému rozšíření automatizačních prostředků založených na PC a softwaru. V zařízeních bylo používáno mnoho různých sběrnic, protokolů a rozhraní a každá aplikace tedy musela mít své vlastní ovladače pro všechna podporovaná zařízení. Tento stav byl problémem zejména pro řídící a vizualizační software, což vedlo ke vzniku pracovní skupiny založené společnostmi Fischer-Rosemount, Rockwell Software, Opto 33, Intellution a Intuitive Technology. Tato pracovní skupina vznikla v roce 1995 a hlavním cílem bylo definovat standard Plug&Play pro ovladače zařízení poskytující standardizovaný přístup k datům automatizace v systémech založených na systému Windows, který byl v té době dominujícím operačním systémem používaným v průmyslu. Pracovní skupina vytvořila standard pro přístup k datům založený na COM (Component Object Model) a DCOM (Distributed Component Object Model), tento standard byl nazvaný OPC, zkratka pro OLE (Microsoft Object Linking & Embedding) for Process Control. V dnešní době je tento standard nazýván jako OPC Classic.

Již během prvního roku se přidává několik dalších prodejců softwaru využívajících OPC jako mechanismus interoperability a v srpnu 1996 pracovní skupina vydává verzi 1.0 zjednodušené specifikace OPC pro Data Access (DA). Ukazuje se nutnost vzniku organizace pro udržování standardu OPC a již v září dochází ke vzniku OPC Foundation. Úkolem OPC Foundation je vytvářet a udržovat specifikace OPC a zajišťovat soulad se specifikacemi OPC prostřednictvím certifikačního testování. [2] [3]

2.2 Specifikace OPC Classic

OPC Foundation definovala řadu specifikací, někdy označovaných jako softwarová rozhraní nebo protokoly, pro standardizaci toku informací z úrovně procesu na úroveň správy. Specifikace OPC jsou vzájemně nezávislé, lze je využívat samostaně a jsou určeny pro použití v různých oblastech průmyslové automatizace. Nejčastěji se používají pro aplikace HMI (Human-machine interface) a SCADA (Supervisory Control and Data Acquisition) k získání dat ze zařízení. Tato data zahrnují aktuální data, historická data a informace o událostech zařízení. Všechny specifikace mají vlastní příkazy pro čtení, zápis atd. a používají při výměně dat osvědčený model klient – server, kdy k jednomu serveru lze připojit i více klientů a současně lze jednoho klienta připojit také k více serverům.

V současné době OPC Foundation na svých stránkách uvádí informace k těmto osmi specifikacím: Data Access, Historical Data Access, Alarm & Event, XML-Data Access, Data eXchange, Complex Data, Security a Batch, přičemž první tři zmiňované jsou nejvíce využívané. Na obrázku 1 je přehled všech specifikací OPC



Obr. 1: Classic OPC specifikace [3]

Data Access (DA) – specifikuje čtení, sledování a zápis proměnných obsahujících aktuální data. Hlavní použití je získávání dat v reálném čase pro různé klienty včetně HMI. Jedná se nejdůležitější specifikaci implementovanou téměř u všech produktů s technologií OPC. Server klientovi odesílá pouze informace o proměnných (položkách OPC), které si klient sám vyžádá. Aby bylo možno identifikovat případnou nepřístupnost dat poskytovaných v reálném čase, jsou data opatřena časovým razítkem a hodnocením kvality dat. Kvalita dat je rozdělena do tří kategorií, dobrá (přesná data), špatná (nedostupná data) a nejistá (neznámá).

Historical Data Access (HDA) – oproti OPC DA poskytuje stejným způsobem přístup k uloženým datům bez ohledu na jejich zdroj, kterým může být například jednoduché protokolování nebo systém SCADA. Využívá hlavně tří způsobů přístupu k datům a to čtení nezpracovaných dat z archivu definovaných proměnných v časové doméně, čtení hodnot proměnných pro zadané časové razítko a čtení agregovaných hodnot z historie pro zadanou časovou doménu. K poskytnutým hodnotám je vždy připojeno časové razítko a údaj o kvalitě. Součástí OPC HDA jsou i metody pro úpravu historické databáze jako mazání, vkládání či nahrazování.

Alarm & Event (A&E) – slouží k přijímání upozornění na události a alarmy a na rozdíl od OPC DA a OPC HDA klient nedává požadavek na konkrétní upozornění, ale jsou mu implicitně poskytována všechna upozornění ze serveru a jejich omezení může klient provést filtrováním, např. podle zdroje upozornění. Jedná-li se o upozornění vyžadující potvrzení, lze je potvrdit pomocí OPC A&E.

XML-Data Access – je specifikace, která se příliš nevyužívá, a to zejména kvůli svým nárokům na přenášená data a omezenému výkonu. Její význam je tedy spíše evolučního charakteru. Jedná se o specifikaci nezávislou na operačním systému a

fyzickém umístění, kdy k přenosu dat dohází ve formátu XML pomocí HTTP a může tak být použita v síti intranetu i internetu. [3] [4] [5] [6]

2.3 OPC UA

Se vzrůstajícím zastoupením jiných operačních systémů, jejichž autorem nebyla firma Microsoft, sílila touha mít produkt nezávislý na systémovém prostředí. Mezi dalšími důvody, které vedly ke vzniku nového komunikačního standardu, bylo přání zbavit se velkého množství jednotlivých specifikací, odstranit poměrně složitý způsob vzájemného propojení mezi systémy a mít velkou bezpečnost přenášených dat.

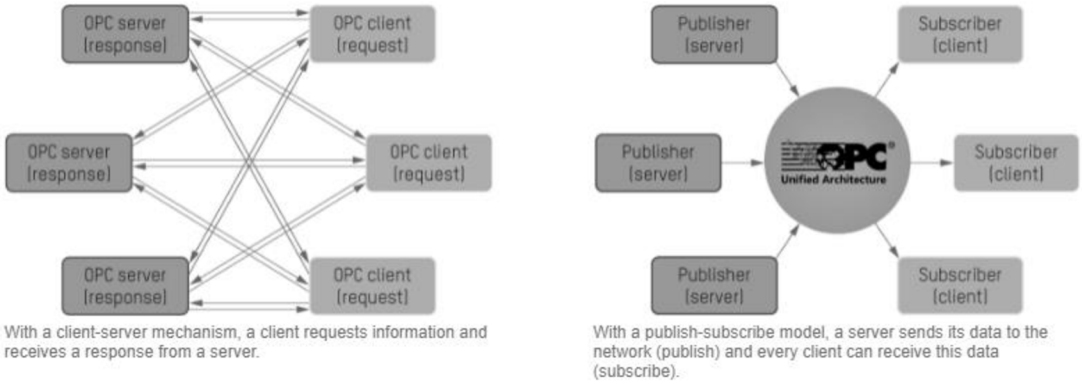
Mezi hlavní požadavky na nový standard mimo systémové nezávislosti patřilo zejména zachování všech funkcí jednotlivých specifikací bez ztráty výkonu, zpětná kompatibilita, budoucí rozšiřitelnost a zabezpečení přenášených dat. Počáteční skupina definující požadavky a případy použití nového standardu byla složena z více než 40 zástupců, mezi nimiž byli nejen členové OPC Foundation, ale také členové normalizačních organizací.

Původní specifikace se začala označovat jako OPC Classic a nový standard dostal název OPC Unified Architecture (UA). Je navržen pro bezproblémovou, bezpečnou a spolehlivou výměnu informací od senzorů až po IT podniky nezávisle na operačních systémech, prodejcích a trzích. Z funkčního hlediska je OPC UA ekvivalentem OPC Classic majícím více schopností.

Velké množství produktů založených na OPC Classic, které měly přejít na novou generaci technologie, vyžadovalo zpětnou kompatibilitu. V letech 2006 a 2007 proběhlo ověření a implementace s následným zveřejněním specifikace OPC UA v roce 2008. OPC UA bylo od počátku navrhováno tak, aby se mohlo stát standardem IEC, v současnosti je znám jako IEC62541. Tento standard a lokalizace do různých částí svět, jako např. Čína a Korea, byly vedeny snahou usnadnit globální přijetí. [7] [3] [8]

3 OPC UA KOMUNIKACE

Protokol OPC UA je od počátku navrhován jako síťová komunikace založená na modelu klient – server, a to na několika úrovních komunikačních kanálů pro zajištění požadavků na bezpečnou, flexibilní a spolehlivou komunikaci. Uvedený model má svá omezení v sítích s velkým množstvím spojení, tato omezení byla důvodem vzniku funkcionality publikování a odběru (publish-subscribe model). Tato funkcionalita byla představena začátkem roku 2018. V původním modelu (který nadále zůstává zachován) klient požádal o informace a následně přijal konkrétně adresovanou odpověď ze serveru. Oproti tomu nový model umožňuje serveru odesílat data do sítě (publikovat) a jsou dostupná pro všechny klienty v síti, kteří je mohou odebrat, viz schéma na obrázku 2. [3] [9] [10]

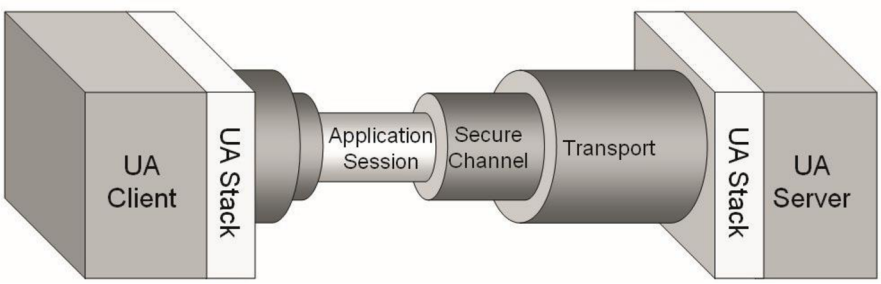


Obr. 2: Pub/Sub model vs Client/Server model [9]

3.1 Spojení Klient – Server

OPC UA aplikace se běžně skládá ze tří softwarových vrstev: Aplikace, OPC UA Client/Server SDK a OPC UA Stack. Pojmem *aplikace* v daném kontextu rozumíme systém, který pracuje s daty prostřednictvím OPC UA a obsahuje specifické funkce pro aplikaci a mapování této funkce na OPC UA Software Development Kit (SDK) či OPC UA Stack. OPC UA SDK obsahuje běžné funkce pro OPC UA, oproti tomu OPC UA Stack implementuje pouze tři komunikační vrstvy, viz obrázek 3. Transportní vrstva (Transport Layer) definuje síťový protokol, k dispozici je UA TCP nebo HTTP(s)/SOAP, a slouží k samotnému obsluhování odesílaných a přijímaných zpráv. Komunikační vrstva představuje zabezpečený kanál (*Secure Channel*) mezi klientem a serverem, přičemž po jeho vytvoření je generován jedinečný identifikátor, který má trvalou platnost, a token s časově omezenou životností. Není-li token pravidelně obnovován, dojde k ukončení spojení. Součástí komunikační vrstvy také bývá serializační vrstva, která provádí kódování a lze ji za určitých podmínek vypustit mezi aplikacemi používajícími stejný OPC UA Stack. Stejně jako jsou dva komunikační protokoly využívané v OPC UA, tak

jsou definována i dvě kódování: OPC UA Binary a XML. Třetí vrstva je aplikační a představuje relaci (Session), každý kanál obsahuje maximálně jednu relaci, kterou musí klient udržovat aktivní pomocí dotazů a požadavků, jelikož po stanovené době nečinnosti komunikace zaniká. V rámci této vrstvy probíhají volání a zpracování služeb. [3] [11]



Obr. 3: Různé úrovně komunikačního kanálu [3]

3.2 Zabezpečení OPC UA

Zabezpečení pro OPC UA bylo navrhováno hned od samého počátku ve více vrstvách. Toto vrstvené zabezpečení umožňuje volbu potřebné míry bezpečnosti na základě požadavků příslušné aplikace. Zabezpečení na nejvyšší úrovni tak lze využívat pro přístup přes veřejný internetový prostor, u něhož je kladen důraz zejména na bezpečnost přenosu, nebo bez zabezpečení na nejnižší úrovni, kde mohou být požadavky na výkon důležitější než bezpečnost přenosu.

Samotné zabezpečení lze rozdělit na vnější a vnitřní. Vnitřní zabezpečení, kam spadá zejména zavírání přebytečných a neužitečných spojení či sledování aktivity na serveru, se netýká protokolu OPC UA a nebude dále řešeno.

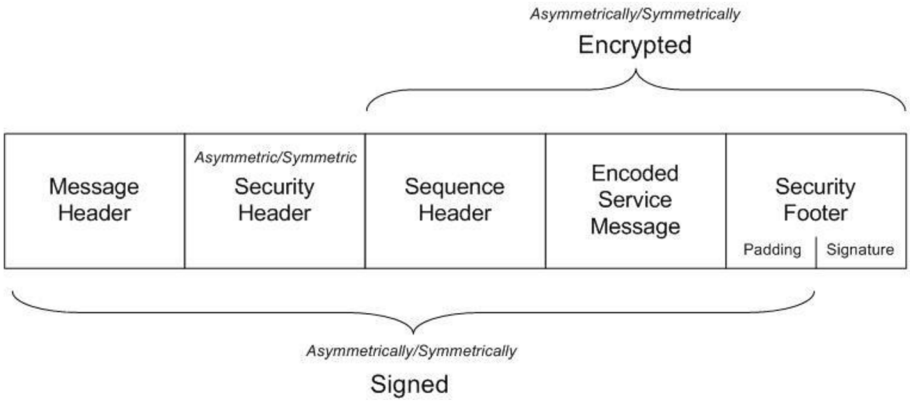
Vnější zabezpečení v rámci OPC UA protokolu se skládá ze tří vrstev. První vrstva, nazvaná Security Mode, určuje zabezpečení zpráv certifikátem. Pro každé spojení je využíván aplikační certifikát, a proto všichni členové OPC UA komunikace (klient, server i gateway) musí mít vlastní instanci aplikačního certifikátu jednoznačně identifikující aplikaci a zařízení, na kterém je spuštěna. Vrstva Security Mode může nabývat tří stavů, v nejzákladnějším režimu zabezpečení, tedy bez autentizace (Security Mode = None), je každý platný certifikát považován za důvěryhodný a slouží pouze k získání informací o druhé straně. Druhý stav využívá autentizaci (Security Mode = Sign) a to ať už jednostranně, tzv. serverová / klientská autentizace, nebo oboustranně. Pro úspěšnou autentizaci je nutné, aby daný certifikát(y) byl v seznamu důvěryhodných certifikátů nebo musí být certifikát vydán důvěryhodnou certifikační autoritou. Pro třetí možnost (Security Mode = SignAndEncrypt) je provedena jak autentizace, tak dodatečné zašifrování veřejným klíčem certifikátu serveru.

Druhá vrstva označující bezpečnostní politiku, Serucity Policy, určuje kryptografické algoritmy používané k šifrování zpráv. OPC UA využívá čtyři způsoby šifrování:

- Security Policy = None
- Security Policy = Basic128Rsa15
- Security Policy = Basic256
- Security Policy = Basic256Sha256

Dvě aplikace OPC UA mohou navzájem komunikovat pouze pokud mají alespoň jednu společnou bezpečnostní politiku. Případně je lze nakonfigurovat tak, aby nepřijímaly určité zásady zabezpečení, i když je podporují (z hlediska implementace).

Poslední vrstva zabezpečení je ověření uživatele. Uživatel může být anonymní, přihlášený jménem a heslem, ověřovanými na serveru při sestavování relace, certifikátem X.509v3. nebo uživatel identifikovaný WS-SecurityTokenem.

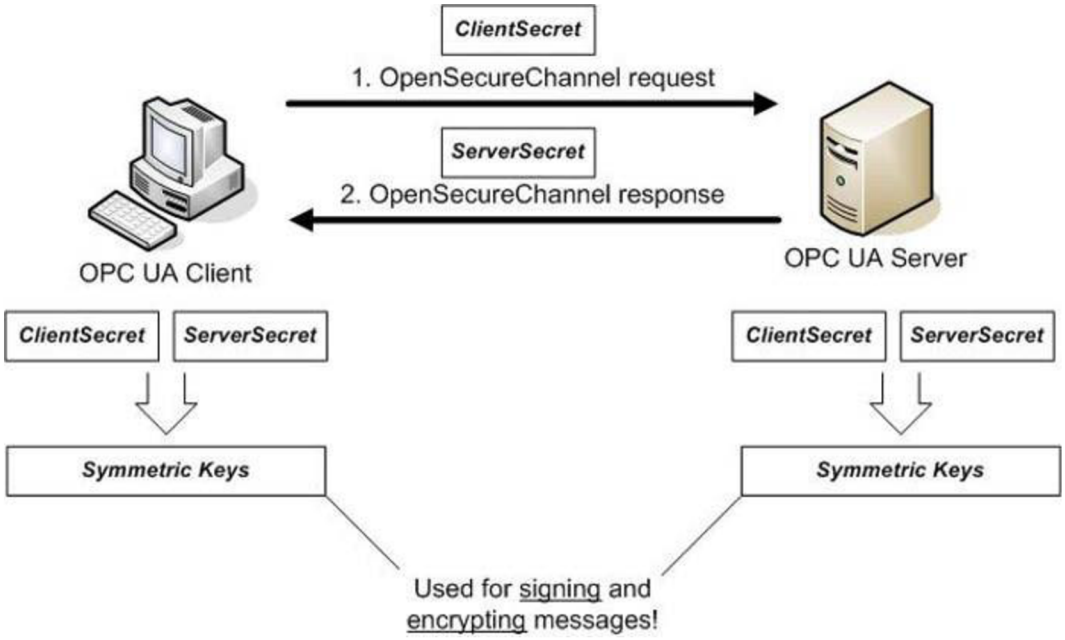


Obr. 4: Struktura bloku zabezpečené zprávy [3]

Blok zabezpečené zprávy se skládá ze záhlaví, samotné kódované servisní zprávy (Encoded Service Message) a zápatí, jeho struktura je na obrázku 4. První část záhlaví, označená jako Message Header, slouží k identifikaci typu zprávy. Hlavička symetrického zabezpečení, Security Header, následuje po Message Header a obsahuje TokenId pro identifikaci sady symetrických klíčů. Výjimkou jsou případy požadavků služby OpenSecureChannel, která je jako jediná kódovaná asymetricky, kdy hlavička obsahuje identifikaci použitých zabezpečovacích algoritmů a použité certifikáty. Sequence Header je třetí částí záhlaví a nemusí se použít vždy, jelikož obsahuje identifikaci bloku v případech, kdy se zpráva musí rozdělit na více bloků. V zápatí se sdělují informace k ověření integrity dat.

Služba OpenSecureChannel se primárně využívá k výměně tajných informací mezi klienty a servery, ty jsou poté použity k odvození symetrických klíčů. Použití symetrických klíčů má výhodu v nižším nároku na výpočet oproti asymetrickému kódování. Až poté, co jsou vytvořeny symetrické klíče u klienta i serveru, dojde k vytvoření zabezpečeného spojení (Security Channel). Za účelem vyšší bezpečnosti mají tyto klíče, a tím i přenosový kanál, časově omezenou životnost a je nutno pravidelně

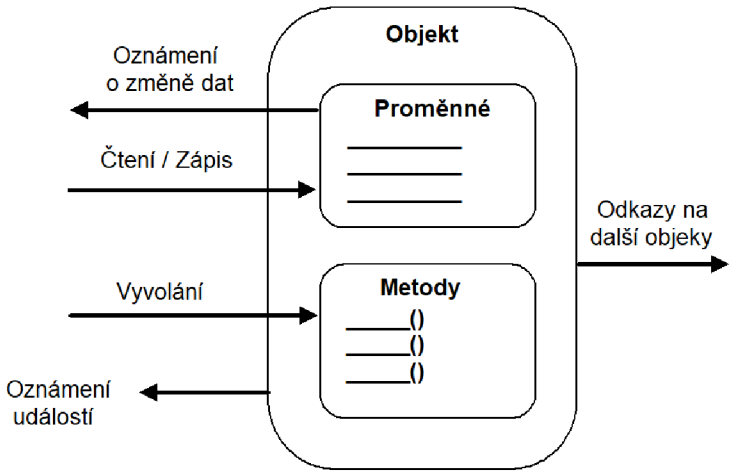
opakovat proces odvození klíčů pro vytvoření nového kanálu. Schéma samotného procesu je na obrázku 5. Tento proces je prováděn ve vrstvě nad relacemi a samotná relace zůstává stejná. [3] [12]



Obr. 5: Schéma vytvoření symetrických klíčů [3]

3.3 Získávání informací ze serveru

Data jsou serverem poskytována jako objekty (jedná se tedy o objektový model), tyto jsou definovány z hlediska proměnných a metod. Na obrázku 6 je ukázka modelu objektu.



Obr. 6: Model objektu OPC UA [3]

Jednotlivé objekty jsou označovány jako uzly (Nodes) a každý uzel je přiřazen ke specifické třídě NodeClass představující jiný prvek modelu. Nalézání dat v adresním prostoru serveru je realizováno hlavně službami Browse a Read. Browse slouží

k procházení adresního prostoru a jejím výstupem jsou zejména Node ID, NodeClass, Reference, Browse name a Display name. Pro přístup k metadatům daného objektu (uzlu) slouží Read, která poskytuje hlavně hodnoty uzlu a časová razítka.

NodeClass – každý objekt patří do jedné z osmi tříd rozlišujících jednotlivé typy objektů. Příslušné třídy jsou OBJECT, VARIABLE, METHOD, OBJECT_TYPE, VARIABLE_TYPE, REFERENCE_TYPE, DATA_TYPE a VIEW.

NodeID – jednoznačný identifikátor uzlu v rámci serveru, který může být ve formátu číselném nebo ve formátu textového řetězce založeném na označení příslušného adresního prostoru a BrowseName. Jelikož jeho textový formát může obsahovat mnoho znaků, je umožněno v rámci každé relace nechat serverem přiřadit příslušnému uzlu číselné NodeID, které je platné jen pro danou relaci. Toto vykonává registrační služba RegisterNodes a využívá se zejména při opakovaném přístupu k danému uzlu pro vyšší efektivitu přenosu. Optimalizace je dosaženo na dvou úrovních. První úroveň je snížení objemu dat při adresování, kdy díky jak kratší formě NodeID, tak velmi efektivnímu přenosu číselných NodeID v binárním protokolu OPC UA, dochází ke snížení množství dat. Druhá úroveň probíhá uvnitř serveru, kdy je server informován o plánovaném častějším přístupu k uzlu a sám interně tento přístup optimalizuje. Zejména s ohledem na zvýšenou zátěž uvnitř serveru má být v případě nepotřebnosti registrace zrušena službou UnregisterNodes.

Reference – popisují vztah mezi dvěma uzly jak v rámci jednoho serveru, tak i mezi uzly různých serverů. Reference je definována zdrojovým uzlem, cílovým uzlem, sémantikou a směrem. Každá reference obsahuje pouze jeden vztah, např. pokud je uzel B „rodičem“ uzlu C i „dítětem“ uzlu A současně, může reference ukázat pouze jeden z těchto vztahů (vztah A-B nebo B-C). Reference mohou být jak nesymetrické („rodič – dítě“), kde záleží na směru, tak symetrické („sourozenec-sourozenec“), kde na směru nezáleží. Jelikož reference je v daný okamžik vázána k jednomu uzlu, tak se může stát, že ukazuje na již neexistující uzel nebo uzel dočasně nedostupný.

BrowseName – je jmenné označení uzlu důležité pro identifikaci v rámci aplikací, ale není určeno pro zobrazování uživatelům.

DisplayName – toto jméno se používá pro zobrazení uživateli a je lokalizováno dle dalších pravidel.

Výše uvedené služby slouží k jednorázovým přístupům, které se mohou libovolně často opakovat, ale nejsou vhodné pro cyklické čtení dat z důvodů velkého zatížení přenosového kanálu. K cyklickému čtení dat slouží vhodnější mechanismy pro odběr dat při jejich změně. [3] [13]

4 OPC UA KLIENT AMALKA

4.1 Analýza požadavků

Základní požadavky na aplikaci vychází ze zadání práce. Dle zadání se má jednat o jednoduchého klienta, který dokáže číst a zapisovat hodnoty proměnných do PLC od firmy B&R Automation s vestavěným OPC UA serverem. Aplikace má být naprogramována s použitím vhodného frameworku a v jazyce, který si zvolí sám autor. Další požadavky vznikly ze strany autora. Jedná se především o zohlednění ekonomického aspektu a právních norem vztahujících se k použitému software. Jelikož výsledkem bude veřejně dostupná nekomerční aplikace pro studijní účely, tak se použití jakýchkoliv finančních prostředků jeví jako neúčelné. Framework tedy musí být dostupný zdarma pro nekomerční účely. Pro použité vývojové prostředí zvoleného jazyka byla taktéž dána podmínka bezúplatné dostupnosti, s případným omezením nelimitujícím tento projekt, např. časový limit použití prostředí (dostatečný na dokončení tohoto projektu), možnost bezplatného využití pouze pro studenty atd.

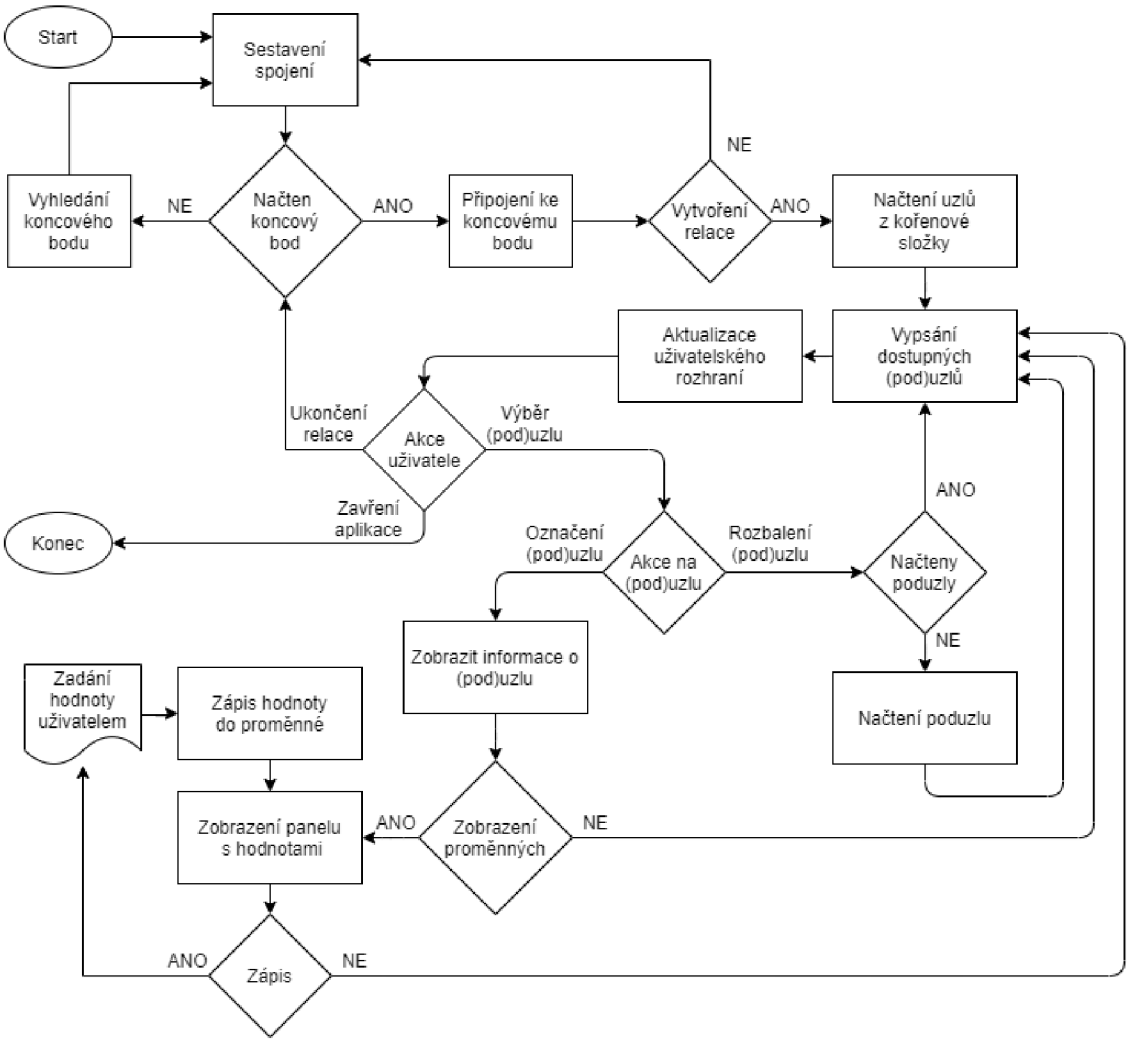
Jelikož se jedná o vysokoškolskou práci vypracovávanou v rámci oboru Aplikovaná informatika a řízení, bylo autorem rozhodnuto, že použije některý z programovacích jazyků vyučovaných v rámci tohoto oboru. Během studia výše zmíněného oboru se studenti seznámili s programováním v Matlabu, C++, C#, ST Language a PHP.

Rozhodovací proces výběru z výše uvedených jazyků byl založen na dostupnosti frameworků pro dané jazyky. Při hledání frameworků se autorovi podařilo nalézt volně dostupné frameworky pro programovací jazyky C, C++, C#, Golang, Java, JavaScript, Python, Rust a TypeScript. Průnikem těchto dvou výběrových množin se staly jazyky C++ a C#. Posledním kritériem, na jehož základě byl vybrán jazyk C#, byla skutečnost, že framework pro jazyk C# je poskytován v rámci licence GPL přímo organizací OPC Foundation, která stojí za vývojem OPC UA, a tím je dána garance implementace všeho potřebného. Následná volba vývojového prostředí padla na Visual Studio používané při výuce jazyka C# v rámci oboru, jelikož splňovalo podmínku bezplatné dostupnosti a autorovi již bylo známé z předchozího studia.

Definice „jednoduchý klient, který dokáže číst a zapisovat hodnoty proměnných do PLC …“, stanovovala funkční požadavky na aplikaci. Aplikace tedy musí, na daném PLC, být schopna: vytvořit spojení, procházet jmenný prostor, rozlišit jednotlivé objekty, načítat hodnoty proměnných a také zapisovat nové hodnoty s ověřením správnosti zápisu. Na zpracování uživatelského rozhraní žádné požadavky stanoveny nebyly a autorovi byla ponechána volná ruka.

4.2 Funkce UA klienta

Po spuštění aplikace je nutno sestavit spojení s příslušným serverem, v našem případě je server součástí PLC B&R Automation. Podmínka aktivní relace, bez které není možno načítat jakékoliv informace pomocí OPC UA, je také dobře patrná na vývojovém diagramu na obrázku 7. Jakmile je relace aktivní, provede aplikace načtení informací z kořenového jmenného prostoru serveru. V pozadí programu dojde k uložení příslušných NodeID a uživateli se v zobrazí příslušná DisplayName. S přihlédnutím k principům OPC UA jsou DisplayName vypsána ve stromovém zobrazení, čímž uživatel získává přehled o závislostech mezi jednotlivými uzly.



Obr. 7: Vývojový diagram klienta

Byť se tento klient zaměřuje na čtení a zápis hodnot proměnných, tak varianta zobrazení informací jen o uzlech patřících do třídy proměnných se jeví jako zmatečná a aplikace proto načítá, a případně zobrazuje, i základní informace o ostatních uzlech zahrnutých do stromové struktury. Teoretická varianta průchodu celého jmenného prostoru a následné selekce jen na uzly proměnných je z principu koncepce modelování OPC UA nesmyslná a vyžadovala by nemalé nároky na data během prvotního procházení celého jmenného

prostoru. Pokud by se jednalo o jedno samostatné PLC není to tak výrazně viditelné jako v případě, kdy připojené PLC, fungující jako server pro naši aplikaci, současně funguje jako klient připojený k dalším serverům. Jelikož každý server se může v rámci svých referencí odkazovat na uzly dalších serverů, může tak vzniknout teoreticky nekonečné řetězení. Konečnost tohoto řetězení je dána systémem a provozem, v nichž jsou servery provozovány. Vyhledávání uzlů s proměnnými tak bude ponecháno na uživateli formou procházení stromové struktury. Pro případy opakovaného přístupu k určité proměnné lze takový uzel zobrazit i přímo zadáním příslušného NodeID, které uživatel zná (od správce serveru, z dokumentace k PLC apod.) nebo jej nalezne v informacích o uzlech v rámci jejich procházení.

Výše zmíněná datová náročnost při procházení celého jmenného prostoru je současně důvodem, aby ve stromové struktuře po sestavení spojení došlo k automatickému načtení pouze kořenového adresáře. Další uzly jsou načítány až poté co jsou vybrány uživatelem, při současném uložení základních informací o uzlech do vnitřních proměnných programu. Následné zobrazování načtených uzlů už probíhá v rámci klienta bez přístupu na server. V případě, že dojde se strany uživatele k požadavku na zobrazení detailnějších informací nebo načtení hodnot proměnných, jsou tyto znovu načteny ze serveru.

Jsou-li v uživatelském rozhraní zobrazeny hodnoty proměnných, je dána možnost hodnotu přepsat a uložit. Jako kontrola správně zapsaných dat je provedeno zobrazení znovu načtených hodnot ze serveru. Přímý zápis hodnot bez jejich vizuálního zobrazení není v aplikaci podporován, jelikož se jedná o přístup typický pro automatizační prostředky, a ne pro lidského uživatele.

4.3 Implementace metod z UA .NET Standard

OPC Foundation UA .Net Standart Stack je framework vytvořený OPC Foundation a poskytovaný jako Open Source. Jak je již z názvu patrno, tento stack je vyvíjen za použití .NET Standardu. Jeho vývoj probíhá v jazyce C# a celý kód je dostupný na https://github.com/OPCFoundation, a to včetně podpůrných ukázkových příkladů. Případnou implementaci lze provést také s využitím balíčků NuGet. Díky .NET Standardu lze tento stack použít pro vývoj aplikací cílících na různé platformy a mezi jeho velké výhody patří pravidelné testování organizací OPC Foundation, což zaručuje jeho správnou funkčnost a kompatibilitu. [14]

Z důvodu jednoduchosti byly k implementaci frameworku využity balíčky NuGet ve verzi 1.4.365.48 publikované počátkem března 2021. V tomto sestavení se k vytvoření relace nabízí asynchronní metoda *Create()* instance *Session* vyvolávající službu *CreateSession*. Metoda má čtyři přetížení, z nichž dvě jsou pro reverzní spojení (spojení iniciované ze strany serveru) a dvě pro použití v klientech. Obě metody, určené pro klienty, mají parametry shodné, pouze parametr, určující požadavek na shodu domény certifikátu a koncového bodu, je součástí pouze jednoho. Vstupní parametry metod využívají běžné datové typy (k určení jména, časové životnosti, nastavení případného

updatu nastavení koncového bodu) seznam preferovaných lokalizací a tři instance (*ApplicationConfiguration, ConfiguredEndpoint, UserIdentity*).

ApplicationConfiguration – je instance konfigurace příslušné aplikace. Tuto konfiguraci můžeme vytvořit přímo v aplikaci nebo načíst ze souboru. Je-li uložená v souboru, lze ji sdílet mezi aplikacemi. Ve své minimální podobě obsahuje identifikační údaje aplikace (např. jméno, typ aplikace, identifikátor URI apod.), informace o certifikátech, komunikační kvóty a konfiguraci klienta/serveru. [15]

ConfiguredEndpoint – definuje koncový bod, tuto instanci vytvoříme konstruktorem na základě instancí *EndpointConfiguration* a *EndpointDescription*. Přičemž k vytvoření *EndpointConfiguration* je použit defaultní konstruktor *new()* s následným přiřazením potřebných hodnot. *EndpointDescription* je získáván ze serveru a obsahuje informace o možnostech spojení s daným koncovým bodem. Mezi tyto informace patří zejména adresa serveru a možnosti zabezpečení. V případě klientů pravidelně přistupujících k neměnnému koncovému bodu lze tuto konfiguraci uložit bez nutnosti jejího opakovaného získávání. [3]

UserIdentity – tato instance nese informace pouze o uživateli a v rámci serveru přiřazuje přidružená práva uživatele k příslušné relaci. Lze použít bezparametrický konstruktor pro anonymního uživatele nebo s parametry pro identifikaci uživatele. Potřebnými parametry jsou dva textové řetězce (jméno a heslo), instance certifikátu nebo instance tokenu.

Z výše popsané metody je patrno, že pro vytvoření relace je nutno mít informace o koncovém bodu uložené v příslušné instanci. Situaci, že máme tuto instanci k dispozici již při spuštění aplikace nebudeme předpokládat. Tento klient slouží jako univerzální klient pro jakýkoliv server OPC UA a uložení informací o několika málo koncových bodech dostupných autorovi by nepřineslo žádné reálné výhody. K získání (v rámci OPC UA se tento proces nazývá Discovery) příslušných instancí popisu koncových bodů můžeme přistoupit třemi způsoby:

```
C#
string url = "opc.tcp://127.0.0.1:4840";
Uri uri = new Uri(url);
DiscoveryClient discoveryClient = DiscoveryClient.Create(uri);
EndpointDescriptionCollection endpointDescriptions;
endpointDescriptions = discoveryClient.GetEndpoints(null);
discoveryClient.Close();
discoveryClient.Dispose();
```

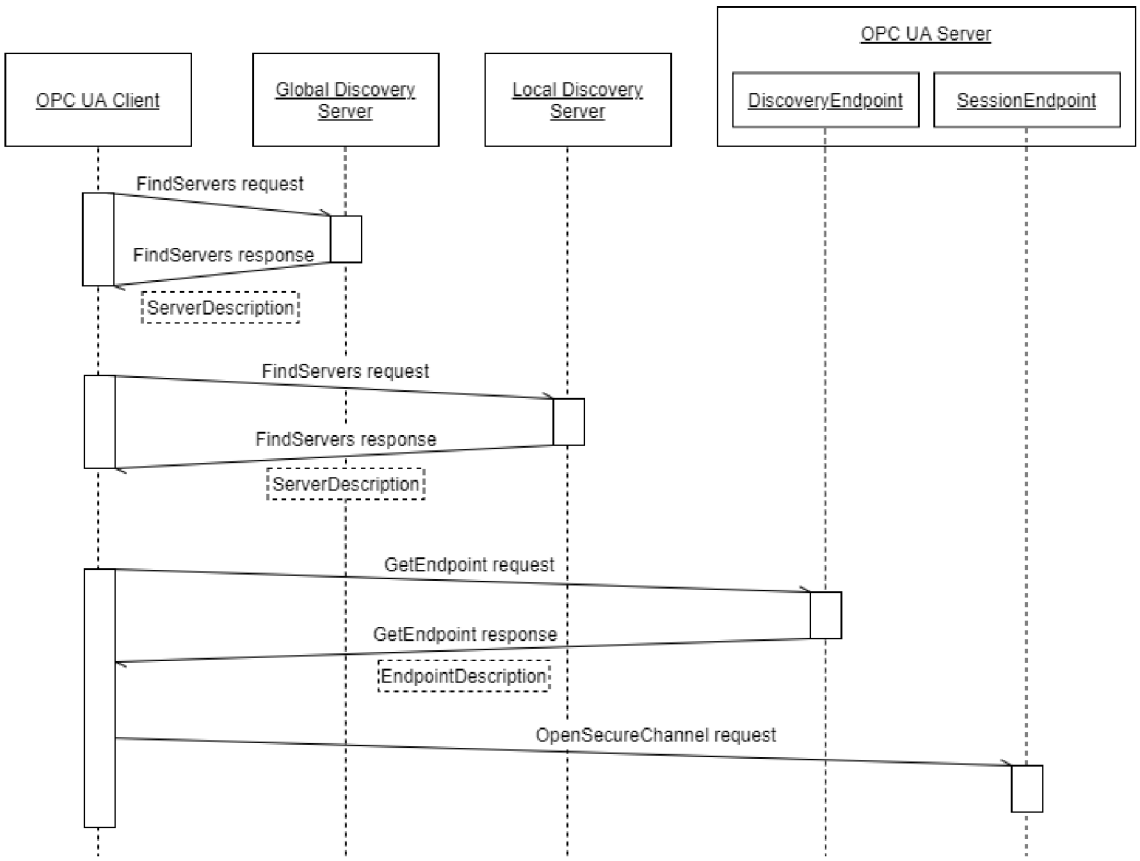
Obr. 8: Ukázka kódu pro získání EndpointDescription

Přímé zadání adresy – je nejjednodušším způsobem. Do konstruktoru instance *DiscoveryClient,* za účelem získání informací o dostupných koncových bodech, vložíme přímo adresu serveru. Tato adresa může ukazovat na jakýkoliv server v síti dostupné

klientovi včetně internetu, je-li k dispozici. Následně je na příslušném *DiscoveryClientovi* vyvolána metoda *GetEndpoints*. Ukázka tohoto jednoduchého kódu je na obrázku 8.

Využití LocalDiscoveryServeru(LDS) – tento přístup se také označuje jako *normální*. Tato situace předpokládá, že na příslušném stroji (počítači) je instalován server LDS, ke kterému jsou zaregistrovány všechny dostupné servery v rámci stroje. Aplikace službou *FindServers* požádá server o adresy dostupných serverů a následně na přijaté adrese / přijatých adresách uplatní předchozí přístup.

Využití GlobalDiscoveryServeru(GDS) – tato varianta, označovaná jako hierarchická metoda, se používá v rámci sítě a je velice podobná předchozímu přístupu. Od ní se odlišuje první fází, ve které se služba *FindServers* uplatňuje u GDS, a odpovědí jsou adresy příslušných LDS. Následný postup je totožný s předchozím přístupem, tedy odeslání požadavku na LDS. Schéma tohoto principu je na obrázku 9. [3]



Obr. 9: Hierarchický přístup [3]

Pro tvorbu klienta byl zvolen přístup vyhledávání pomocí LDS, tento přístup nepřímo umožňuje i přístup přímo zadanou adresou. Je-li služba *FindServers* adresována na LDS, tak odpovědí jsou adresy registrovaných serverů. V případě, že je *FindServers* adresováno na konkrétní server, vrátí vlastní adresu. Tato aplikace tedy vyzve uživatele k zadání adresy koncového bodu a na ni poté směřuje službu *FindServers*. Následně, pomocí *GetEndpoints*, na všech dostupných adresách získáme všechny dostupné instance *EndpointDescription*.

Volba příslušného koncového bodu je na uživateli a po provedené volbě aplikace vytvoří aktivní relaci. Jelikož samotné vytvoření relace není pro uživatele viditelné, následuje vyvolání služby *Browse* za účelem nalezení uzlů v kořenovém adresáři pro jejich následné zobrazení uživateli. Uživateli je dána možnost zobrazení dalších poduzlů a zobrazení informací o jednotlivých uzlech/poduzlech. V případě, že daný uzel dosud nemá načteny své poduzly, opětovně se vyvolá služba *Browse* za účel hledání poduzlů. Poduzly jsou v rámci této aplikace myšleny uzly podřízené na základě referencí nadřazenému uzlu.

V rámci zvoleného frameworku se služba *Browse* volá metodou *Browse()* na instanci *Session*, ukázka této metody je na obrázku 10. Pro účel získání referencí na další uzly a jejich *NodeID*, v rámci naší aplikace, jsou parametry *RequestHeader* a *ViewDescription* nadbytečné a kritéria pro vyhledávání byla dány v rámci ostatních parametrů. Jedním z důležitých parametrů je *NodeID* reprezentující procházený uzel, v případě procházení kořenové složky odpovídá identifikátoru *ObjectIds.RootFolder*. *NodeID* reprezentují typ referencí je dalším nezbytným parametrem a je definováno identifikátory instance *ReferenceTypeIds*. Parametr určující směr reference je v klientovi používán pouze v dopředném směru, jelikož načtené uzly se i s referencemi ukládají do interní paměti programu a případný opačný směr reference je tak odvoditelný bez dotazu na server. Výstupní parametr *byte[]* udává pokračovací bod pro metodu *BrowseNext()*, která se cyklicky opakuje a prochází příslušný jmenný prostor dokud není hodnota parametru *byte[]* nulová (jsou načteny všechny reference dle zadaných kritérií).

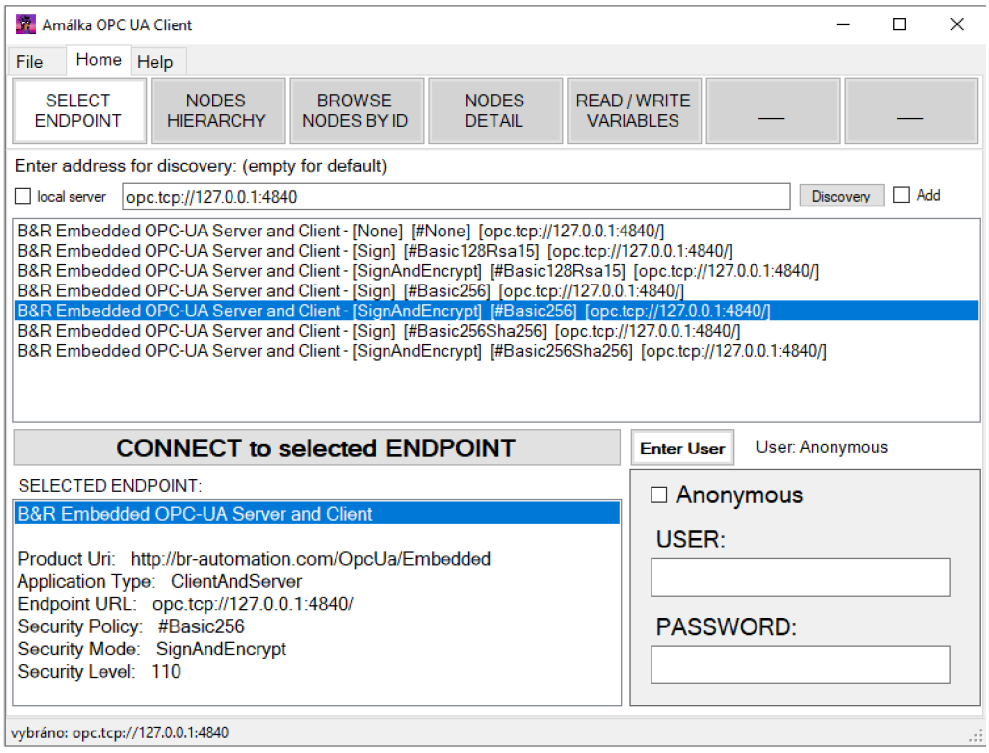
```
C#
// Předpokládá se aktivní Session session.
NodeId nodeToBrowse = ObjectIds.RootFolder;
NodeId referenceTypeId = ReferenceTypeIds.Organizes;
session.Browse(null, null, nodeToBrowse, 0,
               BrowseDirection.Forward, referenceTypeId,
               true, 0, out byte[] continuationPoint,
               out ReferenceDescriptionCollection references);
```

Obr. 10: Ukázka kódu pro načtení referencí z kořenové složky

Pro získání informací vztažených k zadanému uzlu je v rámci frameworku více metod. S přihlédnutím k předpokládanému malému počtu načítaných položek se neuplatnila možnost dávkového načítání a jsou použity metody načítající hodnoty jednotlivě. Pro načítání informací o jednotlivých uzlech je využita metoda *ReadNode()* a pro načítání hodnot proměnných metoda *ReadValue()*, obě tyto metody jsou na instanci Session. V případě první metody je návratovým typem instance *Node* a u druhé metody se jedná o instanci *DataValue*. Pro obě zmiňované metody je společné použití jediného parametru, *NodeID*. Použití obou metod je triviální a na rozdíl od metody *Write()*, sloužící pro zápis nových hodnot, nepotřebuje další parametry než *NodeID*. Metoda *Write()*, jako základní

parametr, vyžaduje kolekci instancí *WriteValue*, která mimo jiné obsahuje *NodeID*. Podmínkou změny hodnoty v zapisovaném uzlu je dodržení typu proměnné a jelikož všechny hodnoty získávané od uživatele jsou získány formou textových řetězců, je nutno tyto hodnoty transformovat na správný datový typ před vložením hodnoty v instanci *WriteValue*.

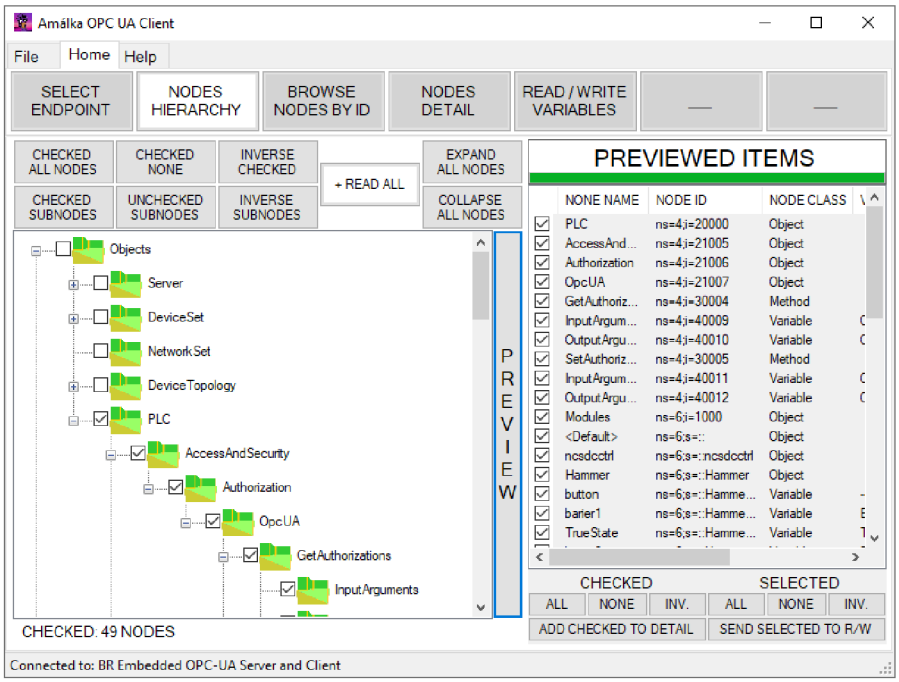
Ke splnění základních požadavků na klienta byly výše uvedené metody sice dostatečné, ale během testování se ukázaly jako nedostatečující. Tato nedostatečnost spočívá v omezené možnosti zobrazování načtených hodnot, jelikož umožňovala pouze statické zobrazení číselného údaje. Zobrazení aktuální hodnoty u dynamických proměnných sice bylo s výše uvedenými metodami řešitelné za pomoci cyklických smyček, ale tento přístup je nevhodný při využívání OPC UA. Pro tyto situace je implementována bezparametrická metoda *Create()* instance *Subscription* sloužící k odběru dat ze serveru. Vytvořenou instanci *Subscription*, po nastavení základních vlastností, přidáme k aktivní relaci její metodou *AddSubscription()* s parametrem *Subscription*. Vytvoření odběru dat ze serveru samo o sobě žádná data neposkytuje, aby sever začal patřičná data odesílat, je nutno instancemi *MonitoredItem* definovat položky k odběru. Pro správnou funkci odběru dat je nutno u každé *MonitoredItem*, mimo nastavení vlastností, zaregistrovat událost *Notification*, která vrací příslušnou *MonitoredItem* s novou sledovanou hodnotou.



Obr. 11: OPC UA Client Amálka verze 1

4.4 Aplikace verze 1

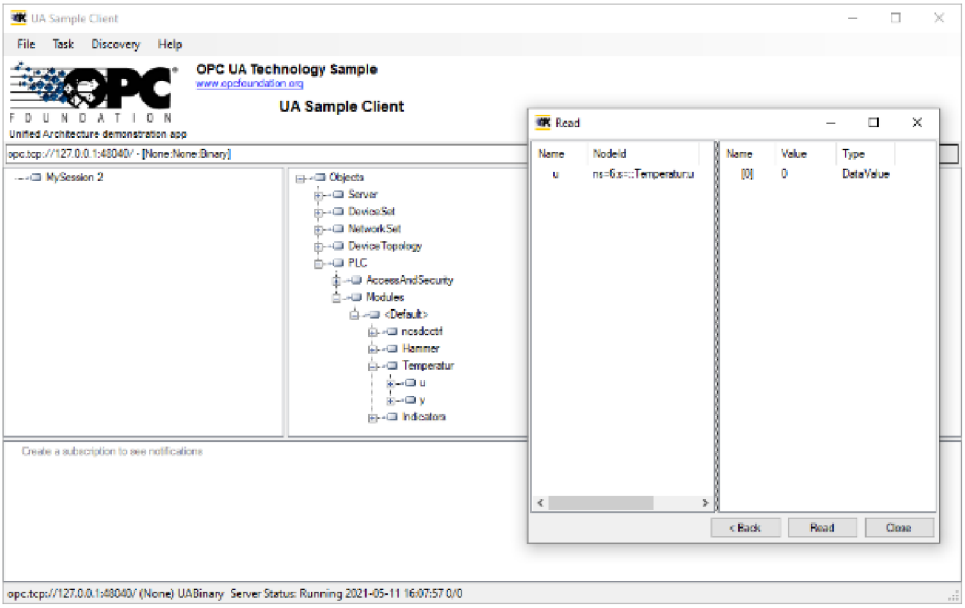
Zadání nestanovovalo žádné požadavky na uživatelské rozhraní. Volba mezi textovým a grafickým rozhraním byla také ponechána na autorovi. Autor se pro grafické rozhraní rozhodl na základě svých osobních preferencí a následné zvolení Windows 10 jako operačního systému pro aplikace bylo dáno technickými prostředky dostupnými autorovi během vývoje aplikace.
Uživatelské rozhraní první verze aplikace vzniklo bez jakékoliv koncepce, a to na principu přidávání dalších ovládacích prvků či záložek v souvislosti s psaním kódu a jeho testováním. Spousta ovládacích prvků byla přidávána bez jakéhokoliv smysluplného využití uživatelem aplikace a jejich reálnou funkcí bylo pouze usnadnit testování aplikace. Tímto způsobem se podařilo vytvořit sice plně funkční, ale uživatelsky velice nepřívětivou aplikaci. Pro demonstraci jsou přiloženy snímky první verze na obrázku 11 a obrázku 12. Tato aplikace je jednoduchým klientem v rámci svých funkcí, ale nedá se o ní mluvit jako o jednoduché aplikaci pro uživatele, aplikace je též v příloze.



Obr. 12: OPC UA Client Amálka verze 1

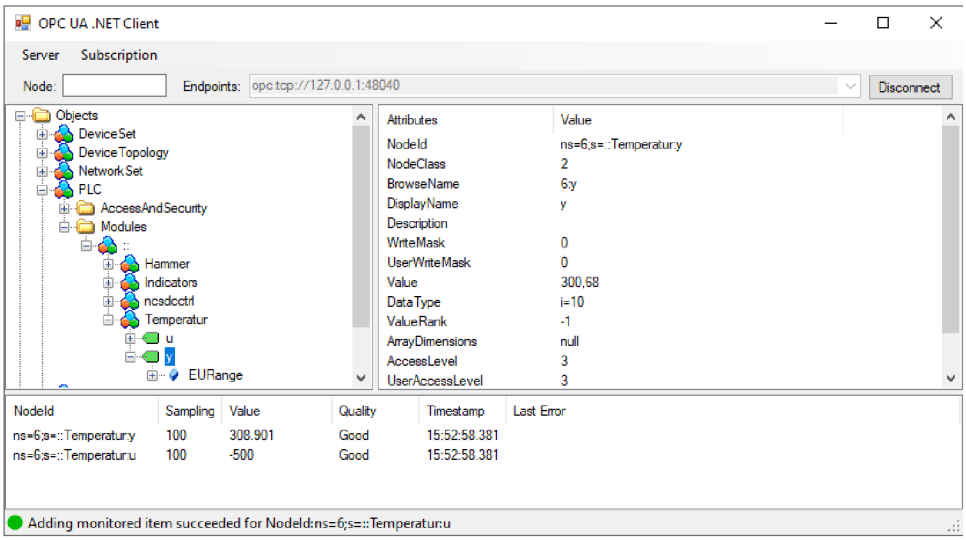
4.5 Návrh uživatelského rozhraní pro verzi 2

Vývoj aplikace bez jakékoliv koncepce uživatelského rozhraní se ukázal jako značně nevhodný a vyžádal si druhou verzi. Aby nedošlo ke vzniku koncepčně nekonzistentní aplikace, bylo nutno před vznikem druhé verze vytvoření konceptu uživatelského rozhraní. Hlavním požadavkem na toho rozhraní byla jednoduchost při zachování potřebné funkčnosti. Prvotní inspirace, vzatá ze známého prostředí operačního systému Windows 10, byla rozšířena o poznatky získané z jiných OPC UA klientů.



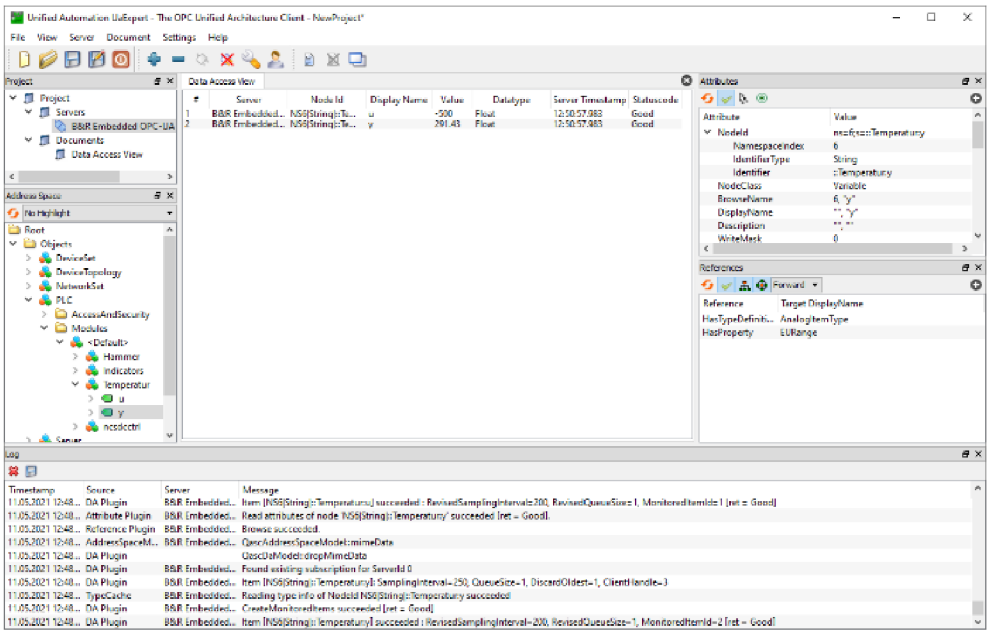
Obr. 13: OPC Foundation: UA Sample Client

V současné době je na internetu dostupných několik různých freewarových klientů. Jako první ukázka byl zvolen klient od OPC Foundation. Tento strohý klient, zobrazený na obrázku 13, nabízí v hlavním okně pouze zadání adresy, procházení stromu uzlů a okno pro zobrazování oznámení. Další informace zobrazuje v externě otevíraných oknech.



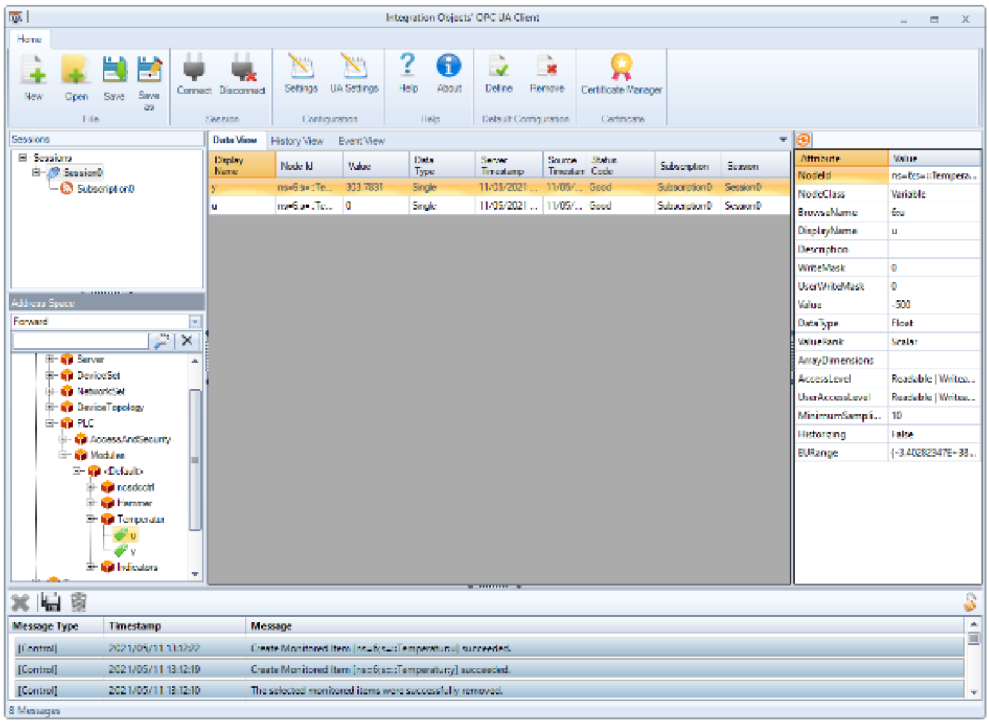
Obr. 14: Siemens: OPC UA .NET Client

Na obrázku 14 je náhled na dalšího jednoduchého klienta, ukázkový klient od firmy Siemens, který veškeré informace zobrazuje přímo v hlavním okně. V levé části zobrazuje strom uzlů, v pravé části jsou zobrazeny detailní informace o zvoleném uzlu a spodní část je vyhrazena odebíraným proměnným.



Obr. 15: Unified Automation: UaExpert

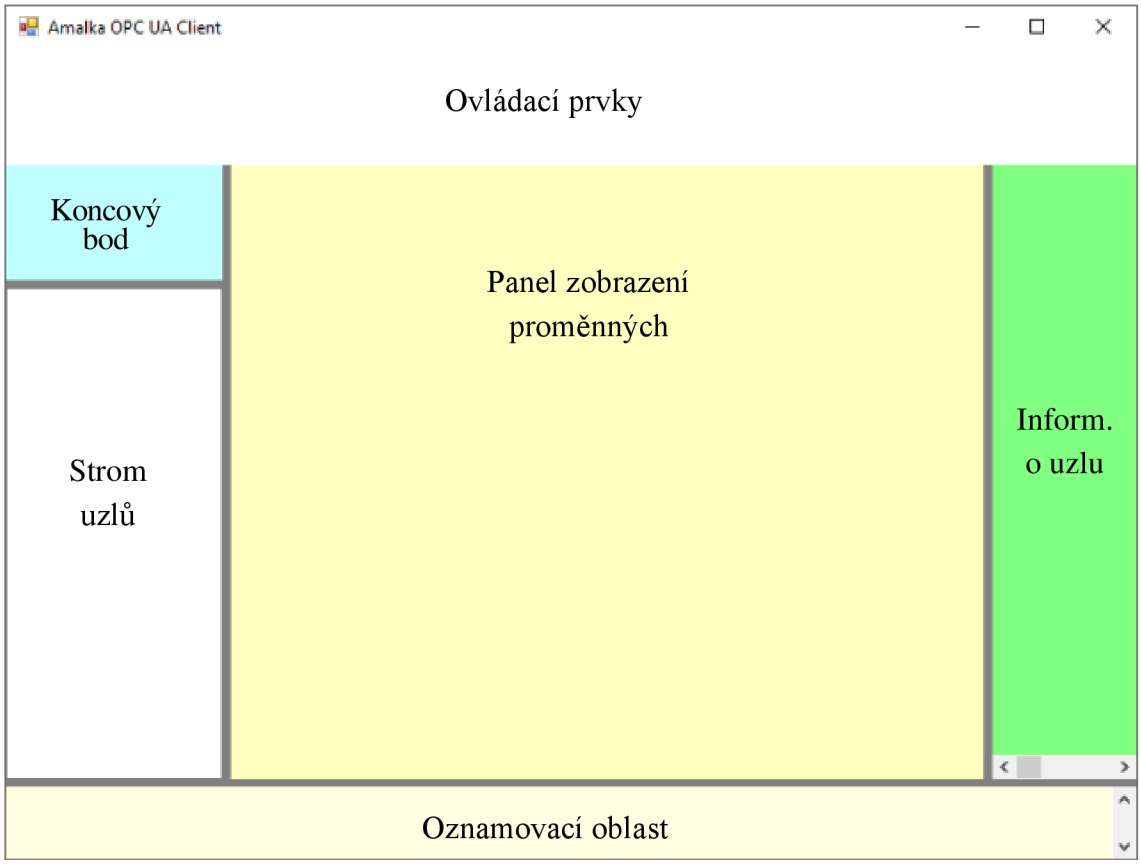
Jako ukázky sofistikovanějších volně dostupných OPC UA klientů byly zvoleny snímky aplikací UaExpert – The OPC Unified Architecture Client od firmy Unified Automation na obrázku 15 a na obrázku 16 OPC UA Client jehož autorem je Integration Objects. Obě zmíněné společnosti patří mezi korporátní členy OPC Foundation.



Obr. 16: Integration Objects: OPC UA Client

Při pohledu na sofistikovanější klienty je patrno, že obsahují mnoho funkcionalit, které v jednoduchém klientovi, určeném k zápisu a čtení proměnných, budou působit spíše rušivě. V kontrastu s tím byl vzorový klient od OPC Foundation, jehož strohost a otevírání spousty dalších oken působí uživatelsky nepřívětivě. Jako vhodný koncept bylo zvoleno řešení využívají zjednodušené rozhraní sofistikovanějších klientů, založené na uživatelské koncepci používané v posledních letech. Uživatelské rozhraní se skládá z několika částí sloužících k různým účelům a předpokládá se obsluha aplikace pomocí myši.

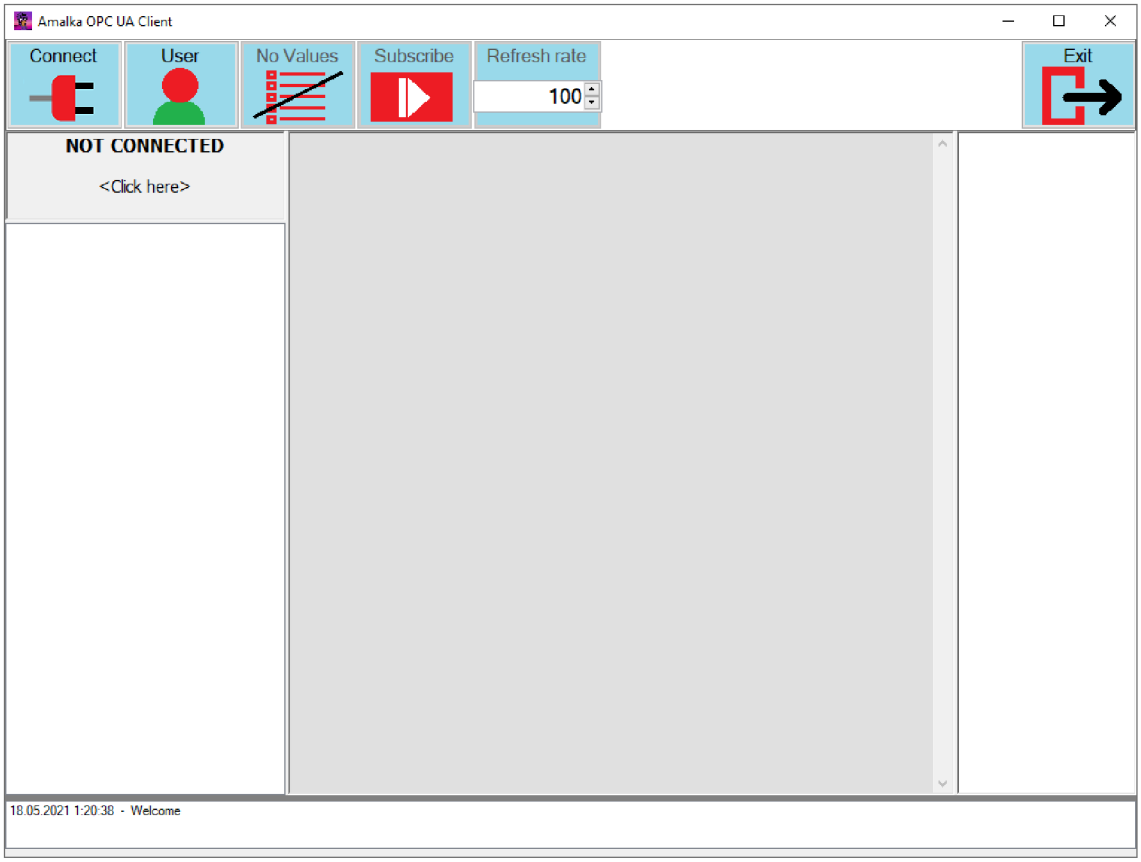
Horní část obsahuje hlavní lištu, která uživateli poskytuje nezbytné ovládací prvky. Ve spodní části je oznamovací oblast a prostor mezi nimi vyplňují tři sloupce. Prostřední sloupec, dominující celé aplikaci, slouží k zobrazování načtených hodnot proměnných s možností zápisu nových hodnot. Levý sloupec vyplňuje okno s označením koncového bodu a jeho stromu uzlů. Poslední, pravý sloupec, zobrazuje doplňkové informace o zvoleném uzlu, jsou-li k dispozici. Grafické znázornění konceptu je na obrázku 17.



Obr. 17: Koncept uživatelského rozhraní aplikace

4.6 Aplikace verze 2

Druhá verze aplikace vznikla na základech verze první, ale shoduje se pouze v rámci použití stejných metod z frameworku. Vizuální změna vyžadovala přepracování velké části syntaxí, aby je bylo možno aplikovat do navrženého konceptu, a zapracování zvýšené podpory pro myš. Zejména přidávání zobrazovaných proměnných v hlavním panelu působí v první verzi značně zmatečně, a proto bylo přepracováno. V první verzi se zobrazování realizovalo formou výběru položek ve stromové struktuře, či jiném seznamu, a následné odeslání proběhlo aktivací příslušného tlačítka na panelu. Odstraněná tlačítka, používaná v první verzi, byla nahrazena kontextovou nabídkou a možností přetažení uzlu ze stromu pomocí myši. Při spuštění aplikace je uživateli k dispozici pět tlačítek a jeden ovládací prvek k nastavení obnovovací frekvence. Uspořádání těchto prvků je ukázáno na obrázku 18.

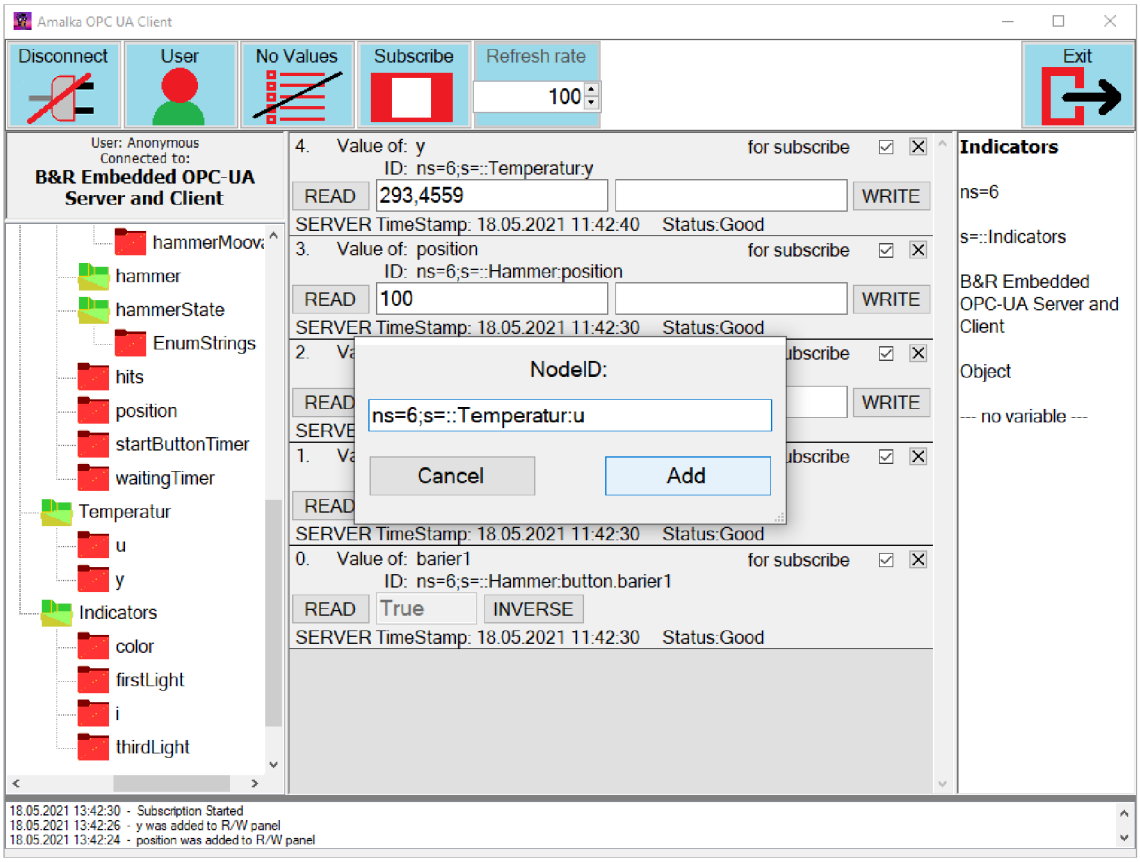


Obr. 18: Úvodní zobrazení aplikace Amálka OPC UA Client verze 2

Z pohledu této práce stojí za zmínku tlačítko *User*, ostatní prvky jsou popsány v návodu k aplikaci v příloze. Tlačítko *User*, sloužící k zadání uživatele přistupujícího na server, bylo v počáteční fázi vývoje aplikace k dispozici pouze v případě neaktivního spojení. Postupné testování aplikace ukázalo, že toto řešení není nejvhodnější a byla přidána možnost volby uživatele i během aktivního spojení. Tento proces vedl k novému požadavku na používané metody z použitého frameworku. Doplněná metoda umožňující změnu uživatele *UpdateSession()* je metodou instance *Session* a umožňuje nejen změnu

aktuálně přihlášeného uživatele, ale také změnu lokalizace. Jelikož nebylo pro testování k dispozici PLC či jiný server s vícero lokalizacemi, nebyla tato možnost implementována.

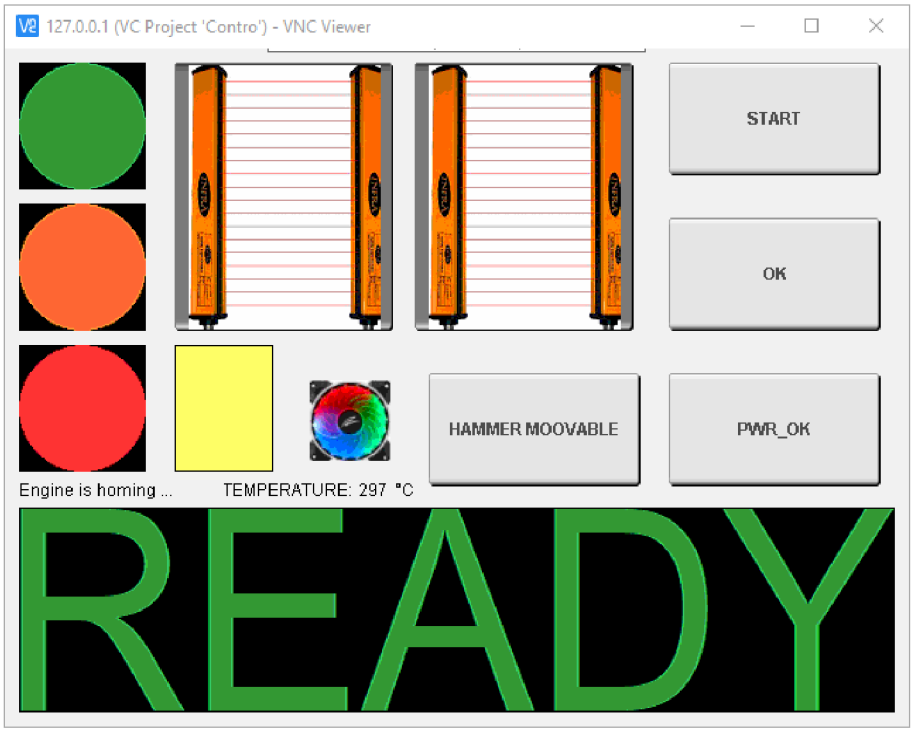
Na Obr. 19: je ukázka spuštěné aplikace, ta je v daný okamžik připojena k simulátoru PLC B&R Automation se spuštěným odběrem dat ze serveru. Odebíraná data, v tomto klientovi, jsou hodnoty proměnných příslušných uzlů. Seznam sledovaných uzlů je vytvořen uživatelem formou výběru ve stromové struktuře. Pro demonstraci přístupu k jednotlivým uzlům bez procházení stromové struktury je k dispozici kontextová nabídka. Pomocí této nabídky je vyvoláno okno, do nějž uživatel zadá přímo NodeID patřičného uzlu. Vyvolané okno je taktéž ukázáno na obrázku 19.



Obr. 19: Spuštěná aplikace Amálka OPC UA Client verze 2

5 TESTOVÁNÍ KLIENTA

Pro zahájení testování bylo nezbytné připojit PLC k PC, na němž docházelo k testování klienta. Vlivem vnějších okolností nebyla varianta použití fyzického PLC k dispozici, proto se připojení PLC realizovalo formou simulace. K simulaci PLC byl využit projekt vytvořený v prostředí B&R Automation Studia, jenž vznikl v rámci dřívějšího studia, na obrázku 20 je snímek z probíhající vizualizace připojeného PLC.



Obr. 20: Snímek VNC Vieweru připojeného k simulovanému PLC

Součástí tohoto projektu nebyl aktivní OPC UA server a bylo nutné jej aktivovat. Vlivem nedostatku zkušeností bylo prvotní nastavení nefunkční, neboť kolidovalo se spuštěným Local Discovery Serverem, běžícím na pozadí jako služba Windows. Tento problém se podařilo odhalit až po instalaci a spuštění klienta třetí strany. Následná nemožnost nalézt simulovaný server prokázala skutečnost, že chyba není ve vyvíjeném klientovi, což byla autorova prvotní domněnka, ale že problém spočívá v konfiguraci na straně simulace.

Samotné testování klienta bylo prováděno autorem již během vývoje. Klient bezchybně načítal veškeré dostupné proměnné a současně umožňoval jejich korektní zápis. Na základě výsledků průběžného testování byly upravovány některé vlastnosti aplikace. Mezi tyto úpravy patří např. odstranění automatického načtení hodnoty po jejím zápisu za podmínky, že je tato položka přihlášena k odběru. Tato úprava byla dána viditelným zpožděním zobrazované hodnoty, jež bylo zapříčiněno duplicitním opakovaným zobrazením tohoto údaje. V rámci testování nebylo možno simulovat dočasné výpadky spojení, což bylo dáno nedostatečnými technickými prostředky.

V případě dalšího vývoje by bylo vhodné disponovat fyzickým PLC za účelem širšího testování komunikace v rámci sítě. Dosud bylo testování omezeno pouze na jeden stroj (autorovo PC), což autora omezilo ve vývoji některých funkcí komunikace. Pokud by bylo k dispozici reálného PLC, které řídí určitý proces, mohl by se klient více přizpůsobit specifickým požadavkům řízeného procesu. Jednou z takových alternativ by mohlo být grafické zobrazení hodnot v návaznosti na jejich reálný základ, např. teplota zobrazovaná v množině jejího reálného rozsahu. Dalším zajímavým příkladem by mohlo být provádění vizualizací probíhajícího procesu na základě získávaných hodnot proměnných (možnost zobrazit polohu apod.).

6 ZÁVĚR

Výstupem práce je OPC UA klient pro komunikaci s PLC B&R Automation včetně jednoduchého návodu k jeho ovládání. Tento výstup nebyl jediným cílem práce, dalšími cíli bylo poskytnou přehled o protokolu OPC UA, zvolit vhodné prostředky pro tvorbu klienta a také rozvoj znalostí a dovedností autora.

Počáteční část, věnovaná historickému vývoji OPC UA, nebyla podstatná pro vznik aplikace klienta ve smyslu implementace poznatků, ale byla obrovským přínosem autorovi po stránce logických návazností. Tyto návaznosti dávají lepší představu o tom, jak je protokol OPC UA využíván a jaké může být jeho budoucí využití. Z časového hlediska je popsaná historie relativně krátká, což je dáno nedávným vznikem OPC UA, ale velkou mírou zaujme progresivnost jeho vývoje v současnosti. Progresivita vývoje je dána požadavky současného rychle se vyvíjejícího průmyslu, který stále ve větší míře využívá síťové přístupy, a také veřejně přístupným otevřeným kódem protokolu OPC UA.

Díky přístupnosti kódu bylo dostupné množství vzorových implementací protokolu, což společně s druhou částí práce poskytlo základy pro tvorbu OPC UA klienta. Druhá část se zaměřuje na popis protokolu samotného, zejména principy zabezpečení a postupy při získávání dat ze serveru. Na základě těchto poznatků byly posléze, ze zvoleného frameworku, vybrány vhodné metody pro sestavení spojení, pro načítání, zápis a odběr dat ze serveru. Zmíněná teorie byla pro autora mnohonásobně přínosnější, než na počátku jejího studování očekával a ukázala se jako nezbytný základ, bez nějž by bylo velmi obtížné naprogramovat funkčního OPC UA klienta. Framework použitý pro aplikaci byl vybrán na základě získaných poznatků o OPC UA a předurčil výběr programovacího jazyka pro aplikaci.

V závěrečné části práce je popisován vývoj aplikace samotné, včetně popisu použitých metod z frameworku. Výsledná aplikace je plně funkčním OPC UA klientem pro načítání a zapisování hodnot proměnných, a to nejen pro PLC B&R Automation, ale také pro jakýkoliv jiný OPC UA server. Výsledná aplikace, v pořadí druhá, vznikla na základech první verze, u které byla soustředěna pozornost na funkčnost a nezohledňoval se komfort uživatelského rozhraní. U druhé verze byl kladen důraz na přívětivé uživatelské rozhraní, odpovídají současným standardům, při současném zachování plné funkčnosti. Oproti první verzi zde byla také implementována možnost odběru dat ze serveru s volbou obnovovací frekvence. Odběrem dat ze serveru je v daném kontextu myšlena automatická aktualizace hodnot načtených proměnných na základě jejich změny. Vývoj více verzí klienta obohatil autora nejen o znalosti vhodnějších postupů při návrhu aplikací, ale také pozvedl jeho programátorské schopnosti o úroveň výše.

Poslední kapitola, s názvem Testování klienta, je zakončením třetí části této práce a zabývá se nejen samotným testováním funkčnosti vytvořené aplikace, ale také rozebírá možnosti dalšího rozvoje aplikace.

7 SEZNAM POUŽITÉ LITERATURY

[1] Unified Architecture. *Unified Architecture - OPC Foundation* [online]. Scottsdale: OPC Foundation, 2021 [cit. 2021-5-19]. Dostupné z: https://opcfoundation.org/about/opc-technologies/opc-ua/

[2] History. *History - OPC Foundation* [online]. Scottsdale: OPC Foundation, 2021 [cit. 2021-5-19]. Dostupné z: https://opcfoundation.org/about/opc-foundation/history/

[3] MAHNKE, Wolfgang. *OPC Unified Architecture*. 2009. ISBN 9783540688990.

[4] OPC and OPC UA explained. *Novotek* [online]. Glasgow: Novotek, 2020 [cit. 2021-5-19]. Dostupné z: https://www.novotek.com/uk/solutions/kepware-communication-platform/opc-and-opc-ua-explained/

[5] OPC v průmyslové komunikaci. *AUTOMA: časopis pro automatizaci* [online]. [2004], **2004**(6) [cit. 2021-5-19]. Dostupné z: https://automa.cz/cz/casopis-clanky/opc-v-prumyslove-komunikaci-2004_06_32378_2345/

[6] Classic. *Members* [online]. Scottsdale: OPC Foundation, 2021 [cit. 2021-5-19]. Dostupné z: https://opcfoundation.org/developer-tools/specifications-classic

[7] OPC UA VÁM ZJEDNODUŠŠÍ ŽIVOT, ALE... *FOXON* [online]. Liberec: FOXON, 2021, 4. září 2019 [cit. 2021-5-19]. Dostupné z: https://foxon.cz/blog/ostatni/454-opc-ua-vam-zjednodusi-zivot-ale

[8] *OPC Unified Architecture Interoperability for Industrie 4.0 and the Internet of Things* [online]. 11. Scottsdale: OPC Foundation, 2020 [cit. 2021-5-19]. Dostupné z: https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf

[9] OPC UA for motion control, safety and real-time applications. *B&R Industrial Automation* [online]. Eggelsberg: B&R Industrial Automation, 2021 [cit. 2021-5-19]. Dostupné z: https://www.br-automation.com/cs/technologies/opc-ua/opc-ua-for-motion-control-safety-and-real-time-applications/

[10] PubSub SDK. *Unified Automation* [online]. Kalchreuth: Unified Automation, 2021 [cit. 2021-5-19]. Dostupné z: https://www.unified-automation.com/products/pubsub-sdk.html

[11] VOJÁČEK, Antonín. Průmyslová komunikace OPC UA - 1.díl - popis protokolu. *Automatizace.hw.cz: rady a poslední novinky z oboru* [online]. Praha 4 - Kateřinky: HW server, c2014, 22. července 2020 [cit. 2021-5-19]. Dostupné z: https://automatizace.hw.cz/prumyslova-komunikace-opc-ua-1dil-popis-protokolu.html

[12] Komunikace přes rozhraní OPC UA. *PROMOTIC: SCADA visualization software* [online]. Ostrava - Vítkovice: Microsys [cit. 2021-5-19]. Dostupné z: https://www.promotic.eu/cz/pmdoc/Subsystems/Comm/OPC/OPCUA.htm

[13] *OPC UA Online Reference: Online versions of OPC UA specifications and information models.* [online]. Scottsdale: OPC Foundation, 2021 [cit. 2021-5-19]. Dostupné z: https://reference.opcfoundation.org/

[14] OPC UA .NET. *Opcfoundation.GitHub.io* [online]. Scottsdale: OPC Foundation, 2016 [cit. 2021-5-19]. Dostupné z: http://opcfoundation.github.io/UA-.NETStandard/#

[15] Creating of OPC UA clients with .NET and helper class. *Siemens* [online]. München: Siemens Aktiengesellschaft, 2021 [cit. 2021-5-19]. Dostupné z: https://support.industry.siemens.com/cs/document/109737901

8 SEZNAM OBRÁZKŮ

9 SEZNAM PŘÍLOH

Příloha 1: Návod k aplikaci OPC UA Client Amalka

Nečíslované přílohy:

- Client_Amalka_v1, projekt Visual Studio 2019 bez balíčků NuGet
- Client_Amalka_v2, spustitelný program
- Client_Amalka_v2, projekt Visual Studio 2019 bez balíčků NuGet
- Soubor readme.txt s návodem na spuštění simulátoru PLC