Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

DIPLOMOVÁ PRÁCE

OPC UA server pro řídící systém REX

Plzeň, 2009 Jan Krčmář

PROHLÁŠENÍ

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 25. května 2009 .....

Jan Krčmář, v.r.

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce Ing. Pavlovi Baldovi PhD. za cenné připomínky a rady.

Děkuji také všem, kteří mi pomáhali při zpracování a korekturách.

OPC UA Server

Dnešní svět se velmi rychle vyvíjí. Moderní technologie se neustále inovují, což platí především v oblasti výpočetní techniky a kybernetiky. Jednou z oblastí kybernetiky je řízení. Zde se již po dlouho dobu pro komunikaci mezi řídicími a řízenými systémy a vizualizačními programy používá standard OPC. Nedávno byl inovován i tento standard. Jeho nová verze byla vydána pod označením OPC Unified Architecture. Implementací tohoto nového standardu s využitím stávajících technologií Java a rozšířením pro řídicí systém REX se zabývá tato práce. Důraz je mimo jiné kladen na to, aby bylo řešení spustitelné na různých operačních systémech.

Klíčová slova: OPC, OPC Unified Architecture, REX, Java, SOAP, Apache Tomcat, Apache Axis, BSD Licence.

OPC UA Server

Today's world has been going through very fast development. Modern technologies have been continuously innovated, that extremely applies to the field of computer hardware and cybernetics. One of the parts of cybernetics is control. Here, the OPC has been used as a standard for communication among control system and controlled system, and visualization programs. However, even this standard was innovated as well recently. Its new version was published under designation of OPC Unified Architecture. This work is dealing just with implementation of this new standard, using Java existing technologies, and its extending to the REX control system. The emphasis is given to the fact that the solution is applicable for various operation systems.

Keywords: OPC, OPC Unified Architecture, REX, Java, Apache Tomcat, Apache Axis, BSD License.

Obsah

Seznam obrázků

1 Úvod

Na Katedře kybernetiky Západočeské univerzity v Plzni se dlouhodobě k měření a řízení používá systém REX[13]. Jedná se o komplexní systém, který byl několikrát úspěšně nasazen i v průmyslu. Je vhodný pro použití ve vnořených systémech, od jednoúčelových desek až po procesní stanice s operačním systémem Windows, Windows CE a PharLap ETS. Pro přenos dat mezi jednotlivými prvky řízení a vizualizací používá systém REX standard *OPC Data Access*[28] (OPC DA).

OPC DA je starší verze standardu *OPC UA*[1]. Očekává se, že nový standard bude nejméně tak úspěšný jako jeho předchůdce. Avšak přechod na nové technologie probíhá v průmyslu velmi pozvolna a může trvat poměrně dlouho, než bude starý standard nahrazen svým modernějším nástupcem. Svědčí o tom i skutečnost, že ještě stále není vytvořen žádný server ani klient, který by kompletně implementoval všechny funkce popsané v tomto standardu.

Ačkoliv nástup této nové technologie zřejmě nebude příliš rychlý, katedra kybernetiky ji považuje za důležitou. Proto je nutné, aby se stávající projekty neustále vyvíjely a sledovaly moderní trendy. V systému REX je již dlouhou dobu zaimplementováno rozhraní pro komunikaci pomocí standardu OPC. Ale aby byl projekt i nadále atraktivní v oblasti průmyslu, je nutné, aby se REX včas připravil na příchod nového standardu.

A právě dalším rozvojem tohoto systému se bude zabývat tato diplomová práce. V práci se zaměříme na implementaci nového standardu OPC UA, především se bude jednat o OPC UA server. Dále se budeme zabývat jeho propojením s řídicím systémem REX. Specifikace OPC UA je komplexní a velmi rozsáhlá, některé její části dokonce nejsou dosud publikovány. Již nastudování této specifikace je časově náročné a její implementace je velice složitá.

V práci budou nejprve vysvětleny základním pojmy, které budou v práci použity. Tím si vytvoříme základ pro pochopení dalších kapitol. Poté se seznámíme s technologiemi, které byly k realizaci použity. Přiblížíme si současný stav problematiky OPC UA nejen na katedře, ale i ve světě. V další kapitole pak přistoupíme k samotnému řešení problému. Popíšeme základní myšlenky implementace a návaznost na použité technologie. Nastíníme možnosti dalšího rozvoje implementovaného řešení. V závěru stručně popíšeme výsledky a přínos této práce.

2 Základní pojmy

Tato část obsahuje vysvětlení termínů, které se v práci vyskytují.

2.1 OLE

Object Linking and Embedding[29] (OLE) je technologie, která umožňuje vkládání a odkazování z a do dokumentů a dalších objektů vyvinutých firmou Microsoft. OLE objekt implementuje určité definované OLE rozhraní, pomocí kterého lze na tento objekt odkázat z jiného objektu nebo programu. Pro vývojáře navíc přináší možnost implementace vlastních rozhraní. Program, který otevírá dokument s OLE objektem, pak může vloženou část získat od jiného programu, který umí s tímto OLE objektem pracovat. Hlavní výhodou OLE je vizualizace dat získaných z jiných programů, které by nebyl původní program schopen zpracovat. Existují dvě možnosti vkládání dat. Linkování (Linking) znamená, že v uloženém souboru bude odkaz na OLE objekt jiného souboru. Pokud se tento OLE objekt změní, projeví se změny i v souboru, ve kterém je tento odkaz. Druhý způsob je vkládání (Embedding). Tento způsob exportuje data z OLE objektu a výsledek uloží do souboru. Soubor neobsahuje žádný odkaz na původní data. Při změně původního OLE objektu se změny dat neprojeví.

2.2 OPC

OLE for Process Control[28] (OPC) je specifikace původem z průmyslové automatizace. Jedná se o nadstavbu OLE pro účely přenosu dat mezi řídícími prvky v průmyslovém řízení. Mimo jiné se klade důraz na real-time potřeby řídících aplikací. Dalším úkolem OPC bylo propojit aplikace v operačnímu systému Windows s konkrétními hardwarovými signály. Sdružení OPC Foundation mělo v úmyslu touto specifikací určit společný průmyslový standard pro přenos dat. Samo sdružení OPC Foundation se nyní zasazuje o to, aby zkratka OPC dále nebyla zkratkou, ale pojmem. Sdružení nechce, aby bylo spojováno s produkty firmy Microsoft.

2.2.1 OPC DA

Data Access[28] (OPC DA) je původní specifikace. Dříve byla označována pouze jako OPC Specifikace. Vznikem dalších specifikací ji však bylo nutné odlišit od ostatních, a proto se začala označovat jako OPC DA. Je založena na produktech firmy Microsoft jako jsou COM[1][30] a DCOM[2][32]. Tím se stala silně závislou na systémech této firmy. OPC DA definuje pouze práci s real-time daty a nikoliv s historickými daty. Pokud je třeba pracovat s historickými daty, lze použít *Historical data access*[28] (OPC HDA).

2.2.2 OPC UA

Novější specifikace *OPC Unified Architecture*[1] byla vytvořena na základě zkušeností s OPC DA a podle jejích autorů je od ní velice odlišná. Jejími hlavními rozdíly je, že je nezávislá na technologiích firmy Microsoft, jako jsou COM a DCOM, a snaží se přiblížit modernímu komunikačnímu modelu *Service Oriented Architecture* (SOA). Celé OPC aplikace se tak mohou stát multiplatformní. Dalším účelem návrhu této specifikace je zvýšení bezpečnosti a poskytnutí informačního modelu.

Na rozdíl od starší specifikace mohou být aplikace implementovány nejen pro systémy Windows, ale také pro sytémy Linux, Mac a další.

Při absenci modelu DCOM však vzniká nutnost definovat svou vlastní síťovou vrstvu. Implementace OPC UA bude tedy složitější, ale je lépe škálovatelná. Odstraněním závislosti na modelu DCOM odpadá vrstva, která byla pro vývojáře jen černou skříňkou (blackboxem). Protokoly, které mohou nahradit starou komunikační vrstvu, jsou:

- OPC UA Binary TCP

  Binární přenos dat pomocí protokolu *TCP*[15]. Specifikace jasně definuje formát binárního přenosu dat a je také rychlejší, protože byl navrhnut přímo pro účely OPC.

- HTTP/SOAP

[1] Component object model - rozhraní pro komunikaci mezi procesy.

[2] Distributed component object model - rozhraní pro komunikaci mezi procesy po síti.

Protokoly *HTTP* a *SOAP* jsou moderní a dnes velice oblíbené, proto pro ně existuje velké množství knihoven v jazyku Java i .Net. Lze je snadno využít ve webových aplikacích.

Další novou vlastností je jiné pojetí adresního prostoru. V OPC UA dále není hierarchické uspořádání jednotlivých ojektů, ale hierarchie se určuje až podle požadavku klienta. V rámci rozrůstání internetu nadále vzrůstá bezpečnostní riziko pro uložená i posílaná data. Specifikace tedy přidává a doporučuje možnost zabezbečení. Pro binární data zavádí vlastní zabezpečení, pro *XML*[22] doporučuje použít specifikaci *WS-SecureConversation*[27]. Inspiruje se především již známými metodami kryptování a šifrování. Dále se rošiřují možnosti o volbu jedno či více vláknové implementace aplikace.

2.3 REX

Řídící systém REX[13] je používán na katedře kybernetiky pro měření a řízení. Pro simulaci procesů se velmi často používá kombinace programů Matlab a Simulink. Tyto programy se nyní často dostávají z akademické půdy i do praxe. Z tohoto důvodu je systém REX schopen spolupracovat i s těmito programy. Především se jedná o Simulink, ve kterém může probíhat vývoj a simulace řídicích procesů.

Systém REX je otevřený škálovatelný systém, který je vhodný především pro vnořené řízení. Je přenositelný na různé platformy, kde lze pro překlad použít překladač jazyka C nebo C++. REX je možné spustit jak na jednoduchých deskách, tak i na procesních stanicích se standardním operačním systémem.

Systém REX se skládá z několika hlavních částí. Pro vývoj a ladění řídících algoritmů obsahuje REX několik nástrojů.

- *RexView* pro vizualizaci trendů systému a interaktivnímu monitorování chodu systémů.
- *RexDraw* pro návrh řídících algoritmů v grafické formě schémat složených z funkčních bloků.

Vizualizace lze realizovat i pomocí dalších programů, které jsou kompatibilní s technologií DCOM. Pro řízení v reálném čase se využívá řídící program RexCore. Pro komunikaci mezi programem RexCore a jeho nástroji se nejčastěji používá protokol *TCP/IP*.

Konfigurace systému probíhá pomocí schémat vytvořených v programu RexDraw nebo v systému Simulink. Tyto soubory mají zpravidla příponu `*.mdl`. Pro překlad do konfigurace pro RexCore se používá program RexComp. Vznikne tím soubor s příponou `*.rex`. Tento soubor lze načíst do RexCore, který vykonává funkci dle vytvořených schémat.

2.4 ISO/OSI model

Open Systems Interconnection[15] (ISO/OSI) referenční model byl vypracován ve snaze standardizovat počítačové sítě. V roce 1984 byl přijat jako mezinárodní norma. Tato norma nespecifikuje konkrétní implementaci jednotlivých systémů, ale popisuje sedmivrstvou síťovou architekturu, její funkce a služby. Vrstvou se rozumí sada funkcí, kterými poskytuje daná vrstva služby vrstvě výšší a kterými využívá služby z nižší vrstvy.

Jednotlivé vrstvy jsou:

1. Fyzická – Přenos signálu.
2. Spojová – Adresování síťových rozhraní v síti.
3. Síťová – Určení logické cesty paketů v síti.
4. Transportní – Poskytuje jasný a spolehlivý přenos dat pro vyšší vrstvy.
5. Relační – Zajišťuje kontinuální spojení mezi počítači v síti.
6. Prezentační – Reprezentace dat, zabezpečení dat, kryptování.
7. Aplikační – Poskytuje konkrétní data aplikacím.

2.4.1 IP

Internet Protocol[15] (IP) je datový protokol používaný v *packet-switching*[3] sítích. Posílání dat je realizováno pomocí bloků zvaných datagramy (nebo pakety[4]). Každé rozhraní komunikující přes tento protokol má přiřazený jednoznačný identifikátor, IP adresu. V současné době se využívají především verze IPv4 a IPv6. Hlavní rozdíl mezi těmito verzemi je, že IPv4 používá pro adresování rozhraní 32 bitovou adresu, IPv6 používá 128 bitovou adresu. IPv6 má tedy větší adresní prostor, což je jeden z hlavních důvodů jeho zavedení.

2.4.2 TCP

Transmission Control Protocol[15] (TCP) je z pohledu ISO/OSI modelu na čtvrté vrstvě. Spolu s IP protokolem vytváří základy internetu. Obecně se často mluví o TCP/IP protokolu. TCP je prostřední vrstvou mezi IP vrstvou pod ním a aplikací nad ním. Tento protokol poskytuje spolehlivé datové spojení mezi jednotlivými počítači nebo programy a poskytuje služby pro přenos dat. Aplikace protokolu TCP pouze předá data, která chce přenést, a cílovou IP adresu. TCP poté převede data do formátu pro vrstvu IP a předá je.

2.4.3 HTTP

Hypertext Transfer Protocol[16] (HTTP) je protokol na úrovni aplikační vrstvy. Sloužil původně k přenosu HTML[5] dokumentů. Ty mohou obsahovat propojení do dalších zdrojů (dokumenty, nebo multimédia). Spolu s poštovním protokolem se zasloužil o obrovský rozmach internetu. Obsahuje standardní model propojení klienta a serveru. Klient na server posílá požadavky a server posílá zpět odpověď. Klient je v tomto případě webový prohlížeč a server je webový prostor. Standardní komunikační port pro HTTP má číslo 80.

[3] Packet-Switching (Přepojování paketů) Síťová rozhraní posílají pakety přes přenosovou síť. Pakety jsou přenášeny přes trvalý virtuální okruh nebo přes přepínaný virtuální okruh.

[4] Sekvence bajtů skládající se z hlavičky a těla. Hlavička obsahuje informace pro přepínače a v těle jsou data, která chceme přenést.

[5] HyperText Markup Language – značkovací jazyk používaný pro přenos dokumentů v internetu.

2.5 XML

Extensible Markup Language[22] (XML) je značkovací jazyk, který má širokou škálu použití. Lze ho použít jak pro přenos a uchování dat, tak i pro mnoho dalších účelů, jako je například popis stavby dokumentu. Jeho hlavním účelem je uchovat nebo serializovat data pro přenos. Svým způsobem zobecňuje původní strukturu HTTP dokumentů. Neurčuje konkrétní značky, ale pouze obecný vzor dokumentu. Jednotlivé značky (elementy) jsou hierarchicky seřazeny a měl by existovat vždy jeden hlavní kořenový (root) element. Každý element může mít atributy a obsahovat data. Základní vzor XML vypadá takto:

```
<element_name attribute_name="attribute_value">data</element_name>
```

2.6 SOA

Service Oriented Architecture (SOA) je architektura poskytující metody pro vývoj systémů a jejich integraci do systémů, kde se aplikace rozdělují do menších částí. Každá část plní příslušnou skupinu funkcí celé aplikace. Jednotlivé části mezi sebou musí komunikovat a tím zajistit celou funkčnost aplikace. Tyto části se nazývají služby (services). Aplikace získá nové funkce jednoduše tím, že se implementují nové služby.

2.7 SOAP

Simple Object Access Protocol[25] (SOAP) je protokol, který využívá pro přenos zpráv aplikační vrstvu HTTP. Formát SOAP zpráv je založený na XML. Pro komunikaci přes SOAP lze využít šablony, podle kterých se serializují posílaná data. SOAP zpráva se skládá z několika částí.

- SOAP Envelope je obálka zprávy.
  - SOAP Header je hlavička dané části zprávy.
  - SOAP Body je obsah zprávy.
- SOAP Attachment je příloha zprávy.

SOAP má největší využití při implementaci RPC[6]. V případě SOAP se také často mluví o XML-RPC, tedy o vzdáleném voláním procedur pomocí zpráv zakódovaných v XML. Klient SOAP vyšle zprávu na server a tím spustí danou proceduru na serveru. Server odpovídá zprávou, ve které je výstup dané procedury.

Pro specifikaci OPC UA poskytuje OPC Foundation šablony na svých stránkách. Slabou stránkou tohoto protokolu je vyšší objem dat pro přenos stejné informace než například při použití binárního přenosu.

2.8 XSD

XML Schema Definition[23] (XSD) je schéma, které popisuje XML strukturu dokumentu. Definuje přípustný obsah dokumentu. To znamená pořadí a počty elementů, včetně jejich formátu. Součástí formátu elementu je například definice jejich datových typů, atributů a přípustných potomků.

2.9 WSDL

Web Services Description Language[24] (WSDL) je jazyk založený na XML, kterým lze popsat webové služby. Tento jazyk zavádí při popisu služeb jejich abstraktní a konkrétní definice a tím přesně popisuje, jakým způsobem lze službu volat a co daná služba nabízí. WSDL se často používá v kombinaci s protkolem SOAP a XML schématem a tím může poskytovat služby přes internet. Struktura WSDL dokumentu se dělí na následující části.

- Service je kontejner sady funkcí, které jsou vystaveny přes webové rozhraní.
- Port je adresa, ze které je daná služba zpřístupněna. Obvykle je reprezentována *URL*[7] adresou.
- Binding určuje vazbu mezi XML dokumentem a SOAP protokolem.
- Port Type definuje webovou službu, operace, které po ní mohou být požadovány, a zprávy, které tyto operace provádí.

[6] Remote procedure call - vzdálené volaní procedur.

[7] Řetězec znaků s definovanou strukturou, který přesně určuje umístění zdroje na internetu.

- Operation je abstraktní popis chodu aplikace podporované danou službou.
- Message je XML zpráva, která je při příchodu mapována na operaci.
- Types kontejner pro definice datových typů, například XSD.

2.10 Framework

Framework je sada nástrojů, které poskytují soubor funkcí. Tyto funkce přebírají typické problémy dané oblasti, čímž usměrní vývoj tak, aby implementace konkrétního problému byla usnadněna. Framework je podobný sofwarovým knihovnám, ale vybrané části lze překrýt vlastní specifickou funkcionalitou. Řízení programu probíhá z frameworku na rozdíl od knihoven, kde tok programu určuje uživatel. Nevýhodou je, že frameworky bývají komplexní a tím dochází ke zpomalení celé aplikace. Při použití frameworku se také musí věnovat čas jeho nastudování. Pokud se však stejný framework používá v různých aplikacích, dojde k výrazné úspoře času.

2.11 JSP

JavaServer Pages[19] (JSP) je technologie, která umožňuje vkládat do statických HTML stránek dynamicky generovaný kód použitím programovacího jazyka Java. Podobně se pro generování stránek používá například skriptovací jazyk *PHP*. JSP soubory jsou zkompilovány na Java servlety, které pak lze umístit na server. Dnes se používá převážně pro programování aplikací využívajících *MVC*[8] architekturu.

2.12 Licence

Pojem licence může mít několik významů. V této práci znamená termín licence dokument, kterým autor uděluje práva, jimiž určuje, jakým způsobem může uživatel s jeho dílem nakládat. Na akademické půdě se na licencování softwaru obvykle používají free software a open-source licence. Zmíněny budou pouze tři nejpoužívanější licence pro svobodný software.

[8] Třívrstvá architektura moderních webových aplikací.

2.12.1 BSD Licence

Berkley Software Distribution (BSD) je jedna z nejsvobodnějších softwarových licencí. Byla pojmenována podle operačního systému BSD[9]. Typická BSD licence obsahuje tři hlavní klauzule. Ty dovolují neomezenou redistribuci díla v binární i zdrojové podobě za jakýmkoliv účelem, pokud jsou dodrženy všechny licenční klauzule.

- Dílo musí obsahovat poznámku copyrightu, seznam klauzulí a dodatek.
- Redistribuce v binární formě musí obsahovat poznámku copyrightu, tento seznam klauzulí a následující dodatek v dokumentaci a/nebo v dalších materiálech poskytovaných spolu s dílem.
- Jméno organizace ani jména přispěvatelů nesmí být použita k propagování produktů díla ani z tohoto díla odvozeného bez předchozího svolení.

Dodatek obsahuje informace o tom, že se autor zříká zodpovědnosti za škody vzniklé použitím jeho díla.

2.12.2 GPL Licence

General Public License (GPL) byla napsána Richardem Stallmanem pro projekt *GNU*. Hlavní myšlenkou je, že dílo musí být šířeno včetně zdrojového kódu. Narozdíl od BSD licence GPL neumožňuje šíření díla v binární formě. K distribuci díla musí být vždy dostupný i zdrojový kód. Dalším rozdílem je, že strikně vyžaduje, aby všechna odvozená díla byla licencována stejnou licencí.

2.12.3 Apache Licence

Apache licence je jedna z free software licencí, kterou používá především nadace Apache Software Foundation *(ASF)*. Tato licence je velice podobná BSD licenci. Umožňuje použití díla pro jakékoliv účely, jeho šíření, modifikaci a šíření modifikovaného díla. Licence nevyžaduje, aby modifikované verze programu byly šířeny pod stejnou licencí. Tato licence má jediný požadavek - součástí díla musí být informace o tom, že byl použit kód podlehající Apache licenci.

[9] Unix-like opračí systém vyvinut University of California, Berkeley.

3 Softwarové komponenty

V dnešní době se většina aplikací zakládá na již existujících projektech. Programátor tak nemusí řešit již jednou vyřešené postupy a může se plně věnovat samotnému vývoji a funkčnosti aplikace. Jednotlivé části aplikace se nazývají komponenty. V následujících kapitolách jsou představeny komponenty použité k implementaci OPC UA serveru v této práci.

3.1 Tomcat

Tomcat[17] je implementace technologií Java Servlet a JavaServer Pages podle specifikací od společnosti Sun Microsystems. Tento server je napsán ryze v jazyce Java nadací Apache Software Foundation, která je známa především díky své implementaci HTTP serveru. Produkt je licencován již zmíněnou Apache licencí, což umožňuje jeho masivní vývoj open source komunitou. Uživatelům je k dispozici zdrojový kód i binární distribuce serveru.

Vzhledem k tomu, že je tento server napsán v jazyce Java, má několik důležitých vlastností. Je multiplatformní, takže ho lze jednoduše nainstalovat na různé operační systémy, které Java podporuje. Aplikace, které na serveru poběží, se tímto také stávají multiplatformními. Tyto aplikace jsou programované v Javě a běží na Java platfomně, čímž se výrazně urychlí, například oproti modulům původního HTTP serveru.

Server se skládá z několika částí.

- Catalina je servlet container. Ten zajišťuje běh a správu všech aplikací běžících na serveru.
- Coyote je HTTP server, který předává požadavky další vrstvě Tomcat, která je zpracuje, a posílá zpět odpovědi.
- Jasper je Tomcat vrstva, která zpracovává požadavky a stará se o kompilaci JSP souborů na servlety.

3.2 Axis

Apache Axis[18] je open source framework pro webové služby využívající XML. Implementace jsou dostupné v jazyce C a Java. Obsahuje implementaci SOAP serveru, několik nástrojů pro práci s WSDL definičními soubory a API pro generování a umístění služeb na web. Axis umožňuje vývojářům vytvářet distribuované webové aplikace, které spolu mohou spolupracovat pomocí protokolu SOAP.

Nyní je Axis dostupný jako Axis2, což je původní Axis přepsaný podle nového softwarového návrhu. Nemusí sloužit nutně jen jako rozhraní webových služeb, ale může také fungovat jako samostatná serverová aplikace. Tato verze dále podporuje REST[10] tím, že upravuje zprávy SOAP. Má podporu pro *Spring*[11] framework.

Vystavení Java programu lze provést dvěma způsoby. Prvním způsob je přejmenování daného zdrojového souboru a zkopírování do patřičného adresáře. Druhým je vytvoření balíčku s přeloženým souborem, kde je nadefinováno, jaké služby budou poskytovány. Axis slučuje jednotlivé služby do skupin (Service group).

3.3 Java

Java je programovací jazyk vyvinutý ve společnosti Sun Microsystems. Má podobnou syntax jako jazyk C nebo C++. Aplikace napsané v Javě jsou obvykle zkompilovány do takzvaného byte kódu. Takto zkompilovaný software ke svému běhu potřebuje interpreta, na kterém poběží. V případě Javy se jedná o JVM. Takový interpret je většinou multiplatformní a tak mohou tyto aplikace běžet na těch operačních systémech, na kterých běží jejich interpret.

3.4 Eclipse IDE

Eclipse IDE je vývojové prostředí původně napsané pro vývoj aplikací v jazyce Java. Tento program je open source a obsahuje podporu zásuvných modulů (pluginů), které výrazně rozšiřují jeho možnosti použití. Pro Eclipse byly postupně napsány i moduly pro vývoj v jiných programovacích jazycích jako například C, Python a PHP.

[10] Je styl softwarové architektury pro distribuovaná hypermédia.
[11] Oblíbený open source aplikační framework pro Javu a .Net.

4 Současný stav řešené problematiky

Tato kapitola přiblíží historii a řešení OPC UA v daných oblastech.

4.1 OPC Foundation

Organizace OPC Foundation začala pracovat na specifikaci OPC UA již v roce 2004. Tehdejším úmyslem bylo vytvořit novou architekturu, která by OPC udržela v čele technologií, a poskytnout univerzální framework, který by byl použitelný minimálně na 10 let. Vydání prvních částí proběhlo již v roce 2006.

Jak bylo již řečeno, původní OPC specifikace založená na technologii COM sloužila již přes 10 let a začínala být zastaralá. Během této doby se technologie, na kterých byla specifikace navržena, značně vyvinula. Z tohoto důvodu se nová specifikace musela vzniklým změnám prizpůsobit. Původní OPC bylo navrženo jako celek. Nová specifikace zavádí vrstvy. Zavedení vrstev umožňuje značnou nezávislost OPC na okolních technologiích, kterými je implementována.

OPC specifikace se neustále vyvíjí. Na stránkách jsou v nepravidelých intervalech vystavovány jednotlivé verze částí OPC specifikací. OPC Foundation implementuje také vlastní framework. V současné době je většina specifikace již implementována v jazyce C a .Net. Prozatím však chybí implementace v Javě a XML formáty přenosu dat. Je tedy zřejmé, že se autoři zaměřili na funkčnost původního OPC a implementovali v jazycích C a pro přenos dat použili binární přenos.

4.2 V průmyslu

Firmy zabývající se OPC ostře sledují vývoj OPC UA specifikace i aplikací. Mnohé z nich mají své zástupce na nejvyšších místech v OPC Foundation. To jim zaručuje, že mohou ovlivňovat budoucí vývoj specifikace nebo jim to může přinést určité výhody a náskok před ostatními firmami. Jedná se především o firmy, které již dlouho působí v oblasti automatizace, jako jsou Iconics a Matrikon.

Iconics představil své OPC UA knihovny a OPC UA Server na začátku roku 2009. Tento server je certifikován pro Windows Vista a Windows Server 2008. Matrikon prozatím nemá své vlastní řešení. Vývoj OPC UA aplikací také probíhá v německé firmě

Unified Automation, která nabízí demoverze svých produktů zdarma ke stažení na svých webových stránkách. Lze získat OPC UA Server i OPC UA Klienta. Obě představená řešení OPC UA však prozatím neobsahují kompletní OPC aplikaci.

Všichni zmínění se orientují směrem ke staršímu návrhu OPC. To znamená, že implementují binární přenos dat a pro implementaci používají jazyk C a jeho odnože.

4.3 Na katedře

Na katedře kybernetiky se zatím s novým OPC UA nesetkáme. Katedra je však členem OPC Foundation a pozorně sleduje nové trendy v této oblasti řízení. Standard OPC UA je na katedře považován za velmi důležitý a je tedy snaha o jeho implementaci pro řídící systém REX, který je na katedře používán. Tento systém má podporu pouze pro komunikaci s původním OPC postaveném na technologiích COM nebo DCOM. Implementací nového standardu pro řídící systém REX se zabývá právě tato práce.

5 Řešení

Tato část obsahuje podrobný popis postupu, kterým byl implementován OPC UA server. V jednotlivých podkapitolách je vysvětlena návaznost na použité technologie a implementace modulu pro řídící systém REX. OPC UA specifikace umožňuje přenos dat binární formou nebo pomocí XML schématu využitím protokolu SOAP. Obě možnosti jsou podrobně popsány.

Pro implementaci řešení byl použit jazyk Java verze 1.6.0_13 z několika důvodů. Jedním z důvodů byla existence rozhraní pro připojení na systém REX právě v jazyce Java. Java je moderní jazyk, který svou architekturou umožňuje použít aplikace na různých operačních systémech. Jedním z hlavních úkolů této práce je to, aby bylo možné OPC UA server spustit na různých platformách. Pro jazyk Java také existuje řada knihoven a frameworků, které budou využity při implementaci OPC UA serveru.

Jako vývojové prostředí byl použit nástroj Eclipse verze J2EE Ganymede-SR2. Tento nástroj je plnohodnotným, dnes velice rozšířeným, vývojovým prostředím především pro vývoj aplikací v jazyce Java. Vzhledem k tomu, že je projekt open source, existuje pro něj řada pluginů, které pomáhají vývojářům při programování různých aplikací. Pro implementaci tohoto serveru byl navíc použit plugin oXygen, což je plugin pro editování XML. Tento plugin umožňuje editaci, zasílání a příjem zpráv SOAP. Dále obsahuje rozšíření WSDL pro vzdálené volání služeb definovaných jazykem WSDL.

Práce byla realizována v operačním systému GNU/Linux, distribuce Gentoo. Verze jádra byla 2.6.26-gentoo-r3 a architektura procesoru byla i686.

5.1 Binární přenos dat

Pro binární přenos dat bylo nutné nejdříve naprogramovat vlastní síťové rozhraní. To se rozpadá na několik částí.

5.1.1 Síťová vrstva

Prvním úkolem bylo pro OPC UA server s použitím binárního přenosu dat implementovat síťovou vrstvu. Síťová vrstva se pro binární přenos dělí na několik částí:

- Transport reprezentuje příjem a posílání dat síťovým rozhraním.

- Zabezpečený kanál reprezentuje šifrování dat.
- Serializace převede data do tříd.

Implementace transportní vrstvy v Javě je v balíku

`opc.ua.mappings.transport.opcuatcp`.

Hlavní třída `TcpServerSocket` obsahuje instanci `ServerSocket`, která vytvoří pro každé nové spojení `Socket`. Při novém spojení je vytvořeno a spuštěno vlákno `TcpThread`, které se dále stará o komunikaci. Pro třídu `TcpThread` je vytvořena právě jedna instance třídy s rozhraním `SecureChannel`. Konkrétní třída `SecureChannel` musí být přiřazena metodou `addHandler`. Při ukončení cyklu se uzavře server socket a tím i všechny již otevřené sockety. Zdrojový kód pro příjem nového spojení výše popsaného algoritmu může vypadat následovně:

```
// create socket
srvSocket = new ServerSocket(port);
while (run) {
  // accept
  Socket socket = srvSocket.accept();
  // start new tcp listening thread for each client
  log.fine(this, "starting new thread to client");
  new Thread(new TcpThread(socket, secchan.getClass().newInstance(), true)).start();
}
srvSocket.close();
```

Třída `TcpThread` implementuje rozhraní `Runnable` a obsahuje socket, který jí byl předán třídou `ServerSocket`. Při spuštění se nejdříve inicializují vstupní a výstupní proudy dat. Dále se zpracovává každá příchozí zpráva. Ta je načtena do pole bajtů. Délka zprávy není předem známa, proto je její velikost zjištěna po přijetí a následně je zkrácena na správnou velikost. Pokud je délka zprávy záporná znamená to, že došlo k chybě a komunikace je přerušena. Zkrácená zpráva je zaslána do další vrstvy. Výstup další vrstvy je poslán zpět.

```
srvOutput = socket.getOutputStream();
srvOutput.flush();
srvInput = socket.getInputStream();
byte[] in;
byte[] tmpin;
byte[] out;
while (listen) {
  // read the input byte buffer
  int length = srvInput.read(tmpin);
  // copy the real message using the correct length
  if (length < 0) {
    throw new TcpMessageInvalidLengthException("Got length: " + length);
  }
  in = new byte[length];
  System.arraycopy(tmpin, 0, in, 0, length);
  // get the response
  out = (byte[])secchan.makeRequest(in);
  srvOutput.write(out);
  srvOutput.flush();
}
srvOutput.close();
srvInput.close();
socket.close();
```

5.1.2 Zabezpečený kanál

Další vrstvou je podle specifikace OPC UA zabezpečený kanál. Na této vrstvě probíhá otevření zabezpečeného kanálu a šifrování dat. Třídy této vrstvy jsou v balíku

`opc.ua.mappings.security.opcuasecure`.

Hlavní třídou je `OPCUASecuredChannel`, která implementuje rozhraní `SecureChannel`. Tato třída je volána metodou `makeRequest`, která vrací serializovaný a zašifrovaný obsah zprávy. Třída navazuje spojení pro jednotlivá vlákna. Navázání spojení probíhá podle OPC UA v několika krocích:

1. Klient otevře nový socket.

2. Klient pošle zprávu `HEL` (Hello) serveru.
3. Server odpoví zprávou `ACK` (Acknowledgement).
4. Klient pošle zprávu `OPN` (Open secure channel request). V této části komunikace se klient a server domluví na formě a parametrech zabezpečení.
5. Server odpoví zprávou `OPN` (Open secure channel response).
6. Další komunikace probíhá pomocí zpráv `MSG`. Klient posílá požadavek, server posílá odpověď.
7. Pro ukončení komunikace klient zašle zprávu `CLO` (Close secure channel). Tím se uzavře socket.

Pokud nastane chyba, odešle se zpráva `ERR` a ukončí se spojení.

Pro testování serveru jsme použili beta verzi OPC UA klienta od firmy Unified Automation. Během testování bylo dosaženo bodu 4. Server umí přijmout `HEL` zprávu, odpovědět `ACK` zprávou, přijme `OPN` zprávu, kterou je server schopen deserializovat do objektu. Server vytvoří a zašle `OPN` odpověď, ale klient ji není schopen deserializovat.

V tomto bodě bylo provedeno mnoho testů. Zprávy byly zachytávany pomocí programu Wireshark, což je analyzátor pro síťový provoz, a podrobně zkoumány. Nebyla však nalezena příčina, pro kterou nebyl klient schopen zprávu deserializovat a pokračovat dále v komunikaci. Klient po neúspěchu při deserializaci zprávy ukončil spojení. Vývoj OPC UA serveru s binárním přenosem byl přerušen a bylo rozhodnuto, že vývoj serveru bude dále pokračovat s využitím XML komunikace.

5.2 HTTP/SOAP

Pro síťovou komunikaci pomocí XML byly použity dostupné definiční soubory OPC specifikace. Na stránkách OPC Foundation jsou ke stažení XSD soubory, které popisují datové typy, i WSDL soubory popisující OPC UA služby. Protože je server psán v jazyce Java, nabízelo se použití již nějakých existujících nástrojů, které by zajistily HTML/SOAP komunikaci, popřípadě i šifrování. Pro tento účel byl vybrán již dříve zmiňovaný framework Axis.

Framework Axis je vhodný i pro zpracování definičních souborů WSDL, protože obsahuje sadu skriptů pro práci s těmito soubory. Přístup ke službám je umožněn přímo přes Axis. Axis obsahuje také webové rozhraní, kterým lze snadno spravovat jednotlivé poskytované služby. Dále bylo nutné vybrat HTTP server, na kterém Axis framework poběží. Axis je programován jako JSP servlet, proto je nutné použít java servlet container. Jako vhodný se ukázal aplikační server Tomcat.

Server Tomcat byl představen již výše. Obě technologie jsou dílem nadace ASF a obě jsou programovány v jazyce Java. Spolu tvoří silný celek, který bude využit jako základ pro OPC UA server s rozhraním SOAP. Licencovány jsou pod Apache licencí, takže je můžeme bezplatně využívat. V tomto řešení byl použit Axis2 verze 1.4.1 a Tomcat verze 6.0.18.

5.2.1 Síťová vrstva

Apache Tomcat a Apache Axis zajišťují celou síťovou vrstvu. To znamená transport dat, zabezpečený kanál i serializaci a deserializaci zpráv. Zprávy jsou pak předány Axisem přímo API serveru.

Komunikace klienta a serveru probíhá následovně.

1. Klient zašle na server zprávu SOAP, která obsahuje požadavek HTTP POST a zprávu s požadavkem na určitou službu. Jednotlivé služby jsou dostupné podle umístění na serveru a přístupový bod k nim je určen pomocí URL.

2. Server (v našem případě Tomcat) zprávu přijme jako zprávu HTTP, podle URL rozpozná, že se jedná o zprávu určenou pro Axis a přepošle mu její obsah, což je zpráva SOAP.

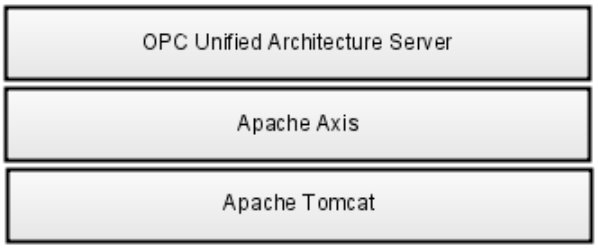
3. Axis opět podle URL rozpozná, že se jedná o zprávu pro určitou sadu služeb (Service set) a předá ji této sadě deserializovanou do Java objektů, na které je převede podle definice v souboru WSDL. V tomto případě je zpráva předána serveru OPC UA, který běží jako služba na Axis servletu.

4. Server zprávu vyhodnotí a odešle odpověd vrstvě Axis jako Java objekty.

5. Ta zprávu převezme a serializuje objekty do zprávy SOAP opět podle definice WSDL. Zprávu SOAP předá Tomcatu.

6. Tomcat server převezme SOAP zprávu a odešle ji zpět klientovi jako zprávu HTTP.

Na obrázku (1) je zobrazeno pořadí jednotlivých vrstev.



Obrázek 1: Uspořádání technologií ve vrstách

5.2.2 Služby

Aby bylo možné do Axisu zaregistrovat služby OPC UA, je nutné vytvořit všem službám třídy podle specifikace OPC UA. K tomu slouží právě pomocné skripty Axisu. Axis vygeneruje podle definičních souborů WSDL celý balík Java souborů, který se pak použije při programování aplikace. Nemusí se nutně jednat pouze o balík, který chceme použít pro server, ale je možné vygenerovat balík i pro klienta. V obou případech se použije stejný soubor WSDL.

Axis neumí pouze generovat Java soubory z souborů WSDL. Obsahuje i nástroje, které ze zdrojových Java kódů serveru nebo klienta vytvoří definiční soubor WSDL. Ten lze použít při vývoji dalších klientů nebo serverů.

Postup pro vytvoření tříd popisujích služby OPC UA pro server je následující.

1. Vytvoříme adresář, kde chceme balík služeb vygenerovat.

2. Zavoláme Axis skript pro generování Java kódu z WSDL s příslušnými parametry.

Pro vygenerování těchto tříd byl vytvořen skript ve skriptovacím jazyce Bash. Tento skript se spustí v adresáři, kde chceme strukturu tříd vytvořit. Vznikne adresář build/server, který obsahuje vygenerované třídy v připravené struktuře pro kompilaci a vytvoření balíku služeb, který použijeme při vývoji serveru.

```
#!/bin/bash
WSDLURI="http://www.opcfoundation.org/UA/2008/02/Bindings.wsdl"
TARGET="build/service"
$AXIS2_HOME/bin/wsdl2java.sh -uri $WSDLURI -d adb -s -ss -sd -ssi -o $TARGET
```

Pro běh skriptu je nutné nastavit globální proměnnou `$AXIS2_HOME`, která obsahuje umístění Axisu na disku. Definiční soubor je v tomto případě stažený přímo z internetu skriptem `wsdl2java.sh`. Není nutné nic stahovat ručně.

Nyní je připravena struktura pro napsání OPC UA serveru.

Při použití Axisu se vyskytl problém při nasazení služeb OPC UA serveru. Axis při požadavku na jakoukoliv službu OPC UA odpovídal zprávou:

Internal server error.

Po prozkoumání logovacích souborů Tomcatu bylo zjištěno, že chyba nastává v samotném jádře Axisu. Proto byly staženy zdrojové kódy, úspěšně nalezeno místo, kde chyba vznikala, a napsána oprava (patch) pro opravení funkčnosti. Tento modul Axisu byl pak zkompilován nástrojem *Maven*[21]. Patch je k dispozici na CD přiloženém k této práci.

5.2.3 Server API

Vygenerováním souborů z předchozí kapitoly vzniknou třídy představující požadavky a odpovědi jednotlivých služeb, vyjímky, které mohou být vyvolány jednotlivými službami, a datové typy definované OPC UA. Vzniknou také rozhraní pro jednotlivé sady služeb a třídy, které tyto rozhraní implementují. Z této sady byl sestaven balík nástrojem *Ant*[20]. Build skript pro tento nástroj byl vytvořen spolu s třídami, při generování výše zmíněným skriptem. Balík byl poté vložen do projektu prostředí Eclipse spolu s balíky Axisu a JavaRex[14]. Z balíku

`org.opcfoundation.ua._2008._02.endpoints_wsdl`

jsou pro implementaci OPC UA serveru nejdůležitější tyto dvě třídy:

- `UAServiceSkeleton` jsou služby UA[4].
- `UADiscoveryServiceSkeleton` jsou služby UA Discovery.

Metody třídy `UADiscoveryServiceSkeleton` jsou volány Axisem při příchodu discovery požadavku. Metody druhé třídy `UAServiceSkeleton` jsou volány v ostatních případech. Axis rozděluje služby do sad (Service set). V tomto případě se jedná o sady pojmenované podle těchto tříd. Podle URL, kterou volá klient, Axis pozná, která třída danou službu poskytuje. Jednotlivé metody mají název podle služby, kterou poskytují. Při příchozí zprávě, která má URL

http://localhost:8080/axis2/services/UAService/AddNodes

Axis pozná, že se jedná o sadu služeb *UAService* a o službu *AddNodes*. Zavolá tedy metodu `UAServiceSkeleton.AddNodes()`.

Parametrem každé metody je požadavek klienta a výstupem je odpověď služby. Každá metoda navíc může vyvolat vyjímku, která má název podle služby. Níže je uveden příklad definice hlavičky metody pro službu AddNodes.

```
public org.opcfoundation.ua._2008._02.types_xsd.AddNodesResponseE AddNodes
(
  org.opcfoundation.ua._2008._02.types_xsd.AddNodesRequestE addNodesRequest64
)
throws AddNodes_FaultMessage;
```

Každé službě je pak nutné doprogramovat funkcionalitu. Implicitně jsou všechny služby vygenerovány tak, aby vyvolaly vyjímku s oznámením, že volaná služba není implementována. Například pro službu *TranslateBrowsePathsToNodeIds* je vyjímka vyvolána následovně.

```
throw new  java.lang.UnsupportedOperationException("Please implement " +
  this.getClass().getName() + "#TranslateBrowsePathsToNodeIds");
```

V dalším odstavci je popsáno, jakým způsobem způsobem je konkrétně implementováno API serveru OPC UA. Nejdříve popíšeme implementaci služeb UAService.

Sada služeb UAService je implementována v souboru `UAServiceSkeleton.java`. Pro každou službu musí být definována metoda, kterou bude Axis volat při požadavku na službu. OPC UA server se skládá z dvou hlavních částí. Server API, přes který se na server přistupuje, a adresní prostor[3] (Address Space), kde se uchovávají data. Aby API mohlo přistupovat na data uložená v adresním prostoru, musí obsahovat instanci adresního prostoru. Adresní prostor je definován pomocí rozhraní `AddressSpaceInterface`. Všechny klientské aplikace mají mít přístup ke stejným datům. Proto je adresní prostor definován staticky. Instanci adresního prostoru získává API od třídy `AddressSpaceFactory` při své inicializaci. API také obsahuje informaci o vytvořených sezeních (Session). Pro tento účel je zavedena instance `Sessions`, která je opět definována staticky a získá se při inicializaci od třídy `SessionsFactory`. Na úrovni API je také modul pro komunikaci s řídícím systémem REX. Stejně jako v přechozích případech je nutné, aby byl modul společný pro všechny klienty. Implementace rozhraní pro REX modul je ve třídě `RexModuleController`. Instance je rovněž definována staticky a je získána od třídy `RexModuleControllerFactory`. Všechny třídy `Factory` implementují metodu `newInstance()`, která vrací instanci dané třídy. Třída `UAServiceSkeleton` implementuje následující služby.

- *CreateSession* je služba, která vytváří sezení a přiřazuje sezení unikátní identifikátor, kterým se klient prokáže při přístupu k ostatním službám. Tento identifikátor je pak posílán v hlavičce každého požadavku. Vytvořené sezení může být používáno až po jeho aktivaci.

- *ActivateSession* je služba, která je volána, pokud chce klient aktivovat sezení. Sezení musí být předem vytvořeno.

- *CloseSession* je služba, která ukončí sezení. Pro identifikaci sezení, které má být ukončeno postačuje prázdná zpráva. Identifikátor sezení je zaslán v hlavičce zprávy.

- *AddNodes* je služba, která do adresního prostoru přidá jeden nebo více uzlů (Node). Povinné parametry pro vytvoření nového uzlu jsou `NodeClass` a

`NodeAttributes`. Důležitý nepovinný parametr je `NodeId`, který je důležitý při implementaci modulu REX.

- *DeleteNodes* je služba, která smaže uzel z adresního prostoru. Povinnými parametry jsou `NodeId` určující uzel, který se má smazat, a `DeleteTargetReferences`. Poslední parametr určuje, jestli se mají smazat i reference uzlu. Podle specifikace však server nemusí zaručit, že budou opravdu všechny reference smazány.

- *Read* je služba pro čtení dat z adresního prostoru. Hlavními parametry pro čtení je identifikátor uzlu (`NodeId`) a identifikátor atributu uzlu (`AttributeId`).

- *Write* služba zapíše data do adresního prostoru. Povinné parametry jsou `NodeId`, kterým se určí uzel, kam se má zapsat, `AttributeId`, který určuje který atribut má být zapsán, a `Value`, což jsou konkrétní data určená k zapsání.

Server běží na frameworku Axis, proto jsou všechny tyto služby při implicitním nastavení serveru, dostupné na adrese

http://localhost:8080/axis2/services/UAService.

Hlavičky všech odpovědí jsou téměř stejné, proto třída navíc implementuje metodu `newDefaultResponseheader`. Tato metoda má parametr `RequestHeader`, ze které sestaví hlavičku `ResponseHeader` pro zprávu odpovědi a vrátí ji.

V dalším kroku je popsána sada služeb UADiscoveryService, které jsou implementovány ve třídě `UADiscoveryService.java`. Kompletní popis discovery služeb je v části specifikace OPC UA, která nebyla dosud vydána. Proto je implementována pouze základní funkčnost. Sada discovery implementuje službu `GetEndpoints`, která vrací seznam všech koncových bodů poskytovaných OPC UA serverem. Metoda `GetEndpoints` třídy odpovídá názvu dostupné služby. Koncové body jsou uloženy v adresním prostoru. Instance adresního prostoru je získána od třídy `AddressSpaceFactory`.

5.2.4 Addresní prostor

Adresní prostor je místo, kam se ukládají všechny koncové body (`Endpoint`) a uzly (`Node`). Třída, která adresní prostor implementuje, je `AddressSpace` a nachází se v balíku

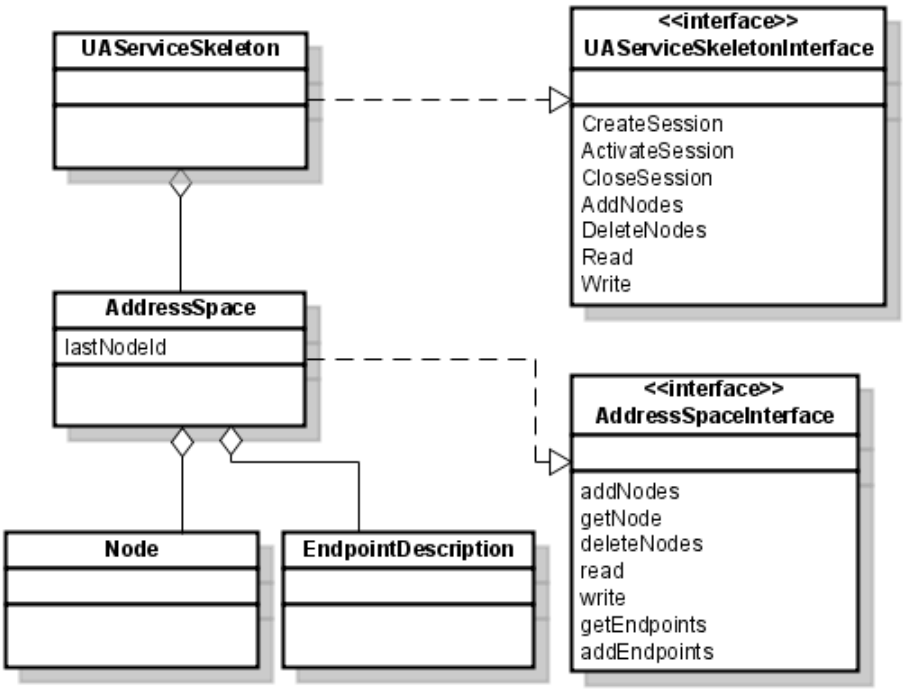
`cz.zcu.kky.opcua.application`

Tato třída implementuje rozhraní `AddressSpaceInterface`. Rozhraní definuje základní operace s uzly a koncovými body:

- *addNodes* přidá uzly do adresního prostoru předané v parametru jako seznam `NodeId`. Vrací `true`, pokud bylo vložení všech uzlů úspěšné.
- *addNode* přidá uzel předaný v parametru jako `NodeId` do adresního prostoru. Vrací `true`, pokud bylo vložení uzlu úspěšné.
- *getNode* vrátí uzel, který nalezne podle identifikátoru v parametru. Pokud není uzel nalezen, vrátí `null`.
- *deleteNodes* smaže uzly podle seznamu `NodeId` v parametru. Vrací `true`, pokud byly všechny uzly z parametru nalezeny a úpěšne smazány.
- *deleteNode* smaže uzel podle identifikátoru v parametru. Vrací `true`, pokud byl uzel nalezen a úspěšně smazán.
- *read* nalezne uzel podle `NodeId` v parametru a vrátí hodnotu jeho atributu podle `AttributeId` v parametru. Pokud není uzel nalezen, vrátí pouze prázdnou hodnotu.
- *write* podle parametru `NodeId` nalezne uzel a nastaví atribut určený parametrem `AttributeId` na hodnotu, která je předána v parametru. Vrací `true`, pokud byl uzel nalezen.
- *getEndpoints* vrací všechny koncové body serveru uložené v adresním prostoru.
- *addEndpoints* přidá koncový bod do adresního prostoru.

Diagram (2) zobrazuje vztahy mezi třídami v adresním prostoru.

Třída `AddressSpace.java` implementuje všechny funkce rozhraní podle návrhu. Všechny uzly jsou uloženy ve statické struktuře `HashMap`. Klíčem pro každý prvek je identifikátor uzlu. Tím je zajištěno, že ve struktuře nebude více uzlů se stejným identifikátorem, což je hlavním předpokladem specifikace adresního prostoru OPC UA.



Obrázek 2: Diagram úspořádání adresního prostoru

Všechny koncové body jsou uloženy ve statické struktuře `List`. Protože parametr `NodeId` není při vytváření nového uzlu povinný, zavádí třída vlastní číselník, podle kterého určí identifikátor nového uzlu. Pro čtení a zápis dat byla napsána pomocná statická třída `NodeHelper`, která serializuje data z a do XML formátu. Jejími metodami jsou:

- *getAttribute* má parametry `Node` a `AttributeId`. Podle *AttributeId* vybere parametr, který se má vrátit, vytvoří z něj objekt `DataValue` a vrátí ho. Pokud nelze atribut převést, vrátí prázdnou hodnotu a chybovým status kódem.
- *setAttribute* má parametry `Node`, `AttributeId` a `DataValue`, která má být do uzlu zapsána. Metoda implementuje pouze změnu hodnoty `Value` pro uzel typu `VariableNode`.

Aby byla zajištěna vždy pouze jedna instance adresního prostoru pro server, byla napsána pomocná třída `AddressSpaceFactory`. Ta obsahuje instanci `AddressSpace`,

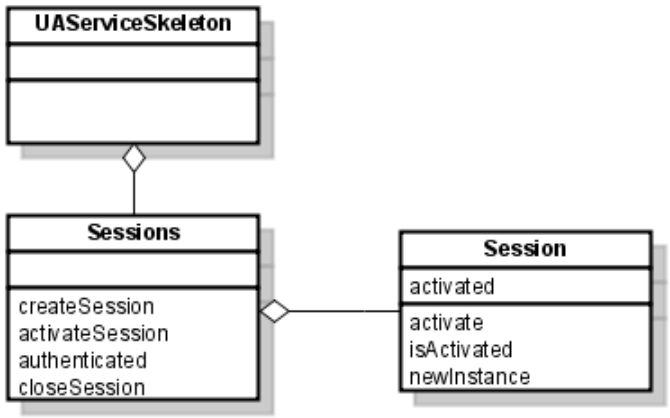
kterou vrací při volání metody `newInstance`.

5.2.5 Sezení

Transportní vrstvu zajišťuje protokol HTTP, který je bezstavový. Proto je zavedeno sezení[4], kterým server udržuje informace o připojených klientech. První služba, kterou musí klient při připojení zavolat je `CreateSession`. Tím server zaregistruje nového klienta a uloží si informace o jeho připojení. Správu sezení provádí třída `Sessions`. která je v balíku `cz.zcu.kky.opcua.application`. Jednotlivá sezení jsou uložena do struktury `HashMap`, která je opět definována staticky. Klíčem je autentizační token, který je unikátní pro každého klienta. Tento token je náhodné celé kladné číslo. Hodnota je instance třídy `Session`, která obsahuje informace o připojeném klientovi. Třída implementuje následující metody:

- `createSession` vytvoří nové sezení.
- `activateSession` aktivuje sezení, které bylo vytvořeno.
- `authenticated` vrací `true` pokud je sezení autentifikováno.
- `closeSession` uzavře sezení.

Diagram (3) zobrazuje vztahy mezi třídami v implementaci sezení.



Obrázek 3: Diagram pro sezení

Při zavolání metody `createSession` je vytvořena nová instance třídy a vygenerován token. Ten je metodou vrácen zpět API serveru a ten ho pošle klientovi zpět. Další službu, kterou musí klient zavolat je `ActivateSession`. Tím se sezení aktivuje. API aktivuje sezení voláním metody `activateSession`. Klient se při každém volání služby prokáže svým tokenem. API autentizaci provede voláním metody `authenticated` ve třídě `Sessions`. Uzavření sezení se provede voláním služby `CloseSession`. Tím se odstraní `Session` včetně tokenu ze seznamu sezení.

Třída `Session` obsahuje proměnnou `activated` typu `boolean`, která nabývá hodnoty `true`, pokud je sezení aktivováno. Obsahuje metody:

- *activate* změní stav sezení na aktivní
- *isActivated* vrací true, pokud je sezení aktivováno.
- *newInstance* vrací novou instanci třídy `Session`, která je zkonstruována s parametrem `CreateSessionRequest`.

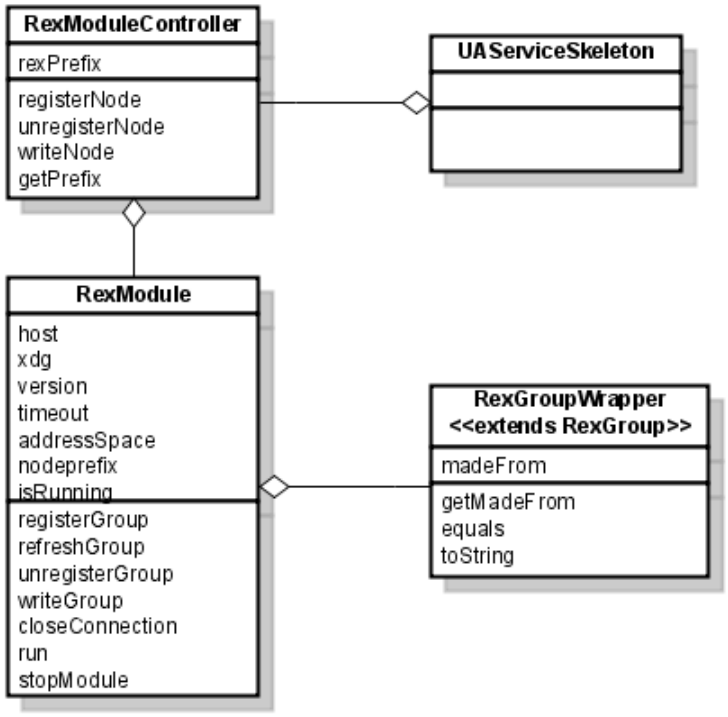
5.2.6 REX modul

Modul pro komunikaci se systémem REX byl navržen tak, aby byl napojen na API vrstvu serveru. Konkrétní implementace je v balíku `cz.zcu.kky.opcua.rex`. Tento balík obsahuje několik tříd.

- RexModule rozšiřuje třídu `Thread` a využívá balík nástrojů JavaRex pro komunikaci s řídícím systémem REX.
- RexModuleController je řídící třída nad třídou `RexModule`. Tato třída poskytuje rozhraní pro API serveru.
- RexModuleControllerFactory obsahuje statickou instanci `RexModuleController`.
- RexGroupWrapper je rozšíření třídy `RegGroup`, která je součástí balíku JavaRex.

Diagram (4) zobrazuje vztahy REX modulu.

API serveru obsahuje instanci třídy `RexModuleController`, kterou získá od třídy `RexModuleControllerFactory` voláním funkce `newInstance()`. Server volá



Obrázek 4: Diagram pro modul REX

`RexModuleController`, pokud mu přijde požadavek na přidání nebo odebrání uzlu nebo pokud je požadován zápis dat do uzlu. Aby se odlišilo, jestli se jedná o uzel, který je v adresním prostoru serveru, nebo se jedná o uzel, který má být předán REX modulu, musí mít uzel určité vlastnosti. S ohledem na to, aby byl identifikátor REX uzlu dobře odlišitelný od ostatních, byl zvolen prefix `opc.rex://`. Identifikátor dále obsahuje informaci o tom, na kterém serveru lze tuto proměnnou získat a vlastní název proměnné. Formát identifikátoru vypadá takto:

```
opc.rex://<server>/<variable>
opc.rex://147.228.47.41/mtuner.PIDMA:hv
```

Pokud příchozí uzel odpovídá této podmínce, je předán ke zpracování třídě `RexModuleController`, která obsahuje i prefix. Od této třídy získávají prefix i ostatní třídy. Tato třída obsahuje seznam vláken, která jsou spuštěna a komunikují s daným

REX serverem. Seznam vláken je ve struktuře `HashMap`, kde je klíčem URL serveru a hodnotou je instance daného vlákna `RexModule`. Metody pro přístup k modulu REX jsou:

- `registerNode` je metoda, která provede registraci nového uzlu. Je volána službou *AddNodes*. Zkontroluje, jestli je spuštěno vlákno, které je připojeno ke vzdálenému serveru REX. Pokud ne, vytvoří ho a předá mu zprávu o registraci nového uzlu. Pokud vlákno již existuje, pouze mu předá zprávu o registraci nového uzlu.
- `unregisterNode` je metoda, která provádí odregistraci registrovaného uzlu. Je volána službou *DeleteNodes*. Metoda nejdříve zkontroluje, jestli existuje vlákno komunikující s daným serverem REX a pokud ano, předá mu požadavek o odregistraci proměnné.
- `writeNode` je metoda, která na vzdálený server REX zapíše požadovanou hodnotu. Metoda zkontroluje, jestli existuje vlákno, které komunikuje s daným serverem a předá

Hlavní třídou, která se stará o komunikaci se vzdáleným systémem REX je `RexModule`. Při inicializaci se vytvoří spojení se serverem REX. K tomu se využije třídy `XDG`, která je v balíku JavaRex. Třída udržuje seznam o registrovaných skupinách na serveru. Tento seznam je implementován strukturou `List`, která obsahuje instance registrovaných tříd `RexGroupWrapper`. Tato třída rozšiřuje `RexGroup` o pole řetězců, které byly pro její vytvoření použity. Hlavní funkcí vlákna `RexModule` je cyklické čtení dat zaregistrovaných skupin ze serveru a ukládá jejich hodnoty do OPC UA adresního prostoru. Identifikátory uzlů, které obsahují data z REX serveru, obsahují hodnoty jmenného prostoru 1000 a řetězcovou hodnotu odpovídající názvu proměnné na vzdáleném REX serveru. Hodnota 1000 je vyššího řádu a byla zvolena tak, aby nebyla konfliktní s dalšími potencionálními adresními prostory. Identifikátor uzlu může vypadat takto:

`ns=1000;s=<string>`

`ns=1000;s=mtuner.PIDMA:hv`

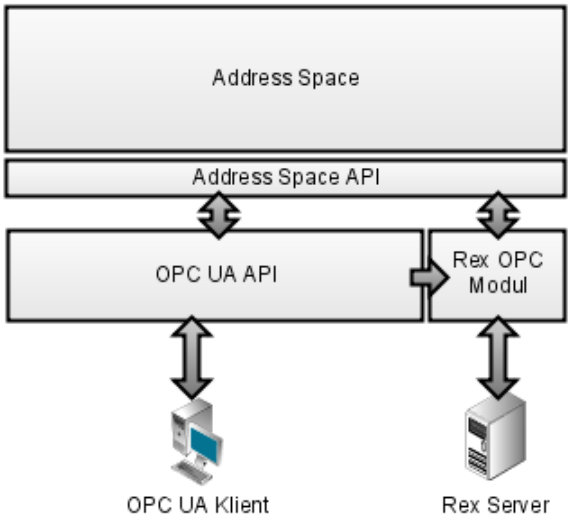
Tímto identifikátorem si lze vyžádat prvek z OPC UA serveru přímo přes OPC UA rozhraní. Uzly vytvořené REX modulem jsou typu *VariableNode* a hodnota získaná ze serveru je v atributu *Value*.

5.2.7 Sestavení a zkompilování programu

Pro zkompilování aplikace byl použit nástroj Ant. Po naprogramování OPC UA serveru byl upraven skript, který byl připraven generátorem tříd tak, aby sestavil a zkompiloval celou aplikaci do Axis archivu. Build skript pro nástroj Ant má název `build.xml`. Změny, které bylo nutné udělat, aby kompilace proběhla úspěšně, jsou přidání balíku JavaRex do seznamu knihoven a zkopírování zdrojových kódů serveru ke zdrojovým kodům vygenerovaným skriptem Axisu. Sestavení aplikace se provede následujícím příkazem.

```
ant jar.server
```

Diagram (5) zobrazuje kompletní schéma pro OPC UA server, který je sestaven výše zmíněným příkazem.



Obrázek 5: Zobrazení celé aplikace.

5.3 Profily

Specifikace OPC UA definuje další důležitou novinku a tou jsou profily[7]. Profily ve smyslu OPC je seznam funkcí, které popisují, jaké služby daná OPC UA aplikace podporuje. Profily jsou specificky určené pro server nebo klienta. Každá OPC UA aplikace

poskytuje kombinaci těchto profilů. Je specifikována minimální konfigurace klienta i serveru. Aby bylo možné server považovat za funkční aplikaci, je nutné, aby podporoval minimálně jeden *Transport Profile* a *Core Server Facet*. V tomto případě podporuje server transport v kombinaci *SOAP-HTTP WS-SC UA XML*. Zabezpečení WS Security lze zajistit modulem *Rampart*[18]. Je to modul dostupný na stránkách Axisu, kde je popsáno, jakým způsobem se modul instaluje. Profil *Core Server Facet* obsahuje několik částí, které jsou implementovány úplně, částečně nebo vůbec.

- *Security Policy - None* implementováno plně.
- *Address Space Base* implementováno plně.
- *Attribute Read* implementován bez parametru *IndexRange*
- *Attribute Write Index* nepovinný, je implementován plně.
- *Attribute Write Values* implementován plně pro uzly typu *VariableNode*.
- *Base Info Core Structure* nepodporuje *Server Object*
- *Discovery Find Servers Self* implementováno plně.
- *Discovery Get Endpoints* implementováno bez filtrování.
- *Security (Administration, User Name Password, User X509)* služby jsou na úrovni frameworku Axis.
- *Session Base* implementováno plně.
- *Session General Service Behaviour* implementováno bez parametru *timeoutHint*.
- *Session Minimal 10 Parallel* implementováno plně.
- Služby *View (Basic, Minimum Continuation Point 01, RegisterNodes, Translate Browse Path)* nejsou implementovány.

5.4 Instalace

Instalace serveru je rozdělěna na několik částí.

1. Tomcat
2. Axis
3. UA Server

Postup instalace je popsán pro operační systém Linux. Instalace v ostatních operačních systémech se od toho postupu téměř neliší.

5.4.1 Tomcat

Balík serveru Tomcat je ke stažení z jeho domovských stránek

http://tomcat.apache.org/.

Stáhneme tedy komprimovaný archiv instalace Tomcat. Vybereme adresář, kam budeme Tomcat instalovat. Vhodný adresář je `/opt`. Do tohoto adresáře rozbalíme instalaci Tomcatu. Vznikne adresář `/opt/apache-tomcat-<verze>`, který budeme dále označovat jako `$TOMCAT_HOME`.

Konfigurační soubory Tomcatu jsou v adresáři `$TOMCAT_HOME/conf`. Nejdůležitější je soubor `$TOMCAT_HOME/conf/tomcat-users.xml`, kde musíme vytvořit nového uživatele s administrátorskými právy a nastavit mu heslo. To může vypadat například takto:

```
<user username="tomcat" password="secPass" roles="admin,manager"/>
```

Pro spuštění serveru spustíme skript `$TOMCAT_HOME/bin/startup.sh`. Tomcat naslouchá implicitně na portu 8080. V prohlížeči zobrazíme hlavní stránku zadáním adresy

http://localhost:8080/.

5.4.2 Axis

Framework Axis je také volně ke stažení na svých domovských stránkách

http://ws.apache.org/axis2/.

Stáhneme distribuci určenou pro běh na aplikačním serveru. Z nabízených možností vybereme *WAR (Web Archive) Distribution*. Po stažení se přihlásíme do administračního prostředí Tomcatu (odkaz Tomcat Manager na hlavní stránce Tomcatu). V tabulce *Deploy* zvolíme soubor, který obsahuje aplikaci. V tomto případě stažený *WAR* soubor Axisu. Tím vystavíme Axis na Tomcat server zpřístupněný na adrese

http://localhost:8080/axis2/.

Axis je tímto způsobem nainstalován do adresáře

`/opt/apache-tomcat-6.0.18/webapps/axis2`,

který budeme dále označovat

`$AXIS2_HOME`.

Konfigurace je uložena v souboru

`$AXIS2_HOME/WEB-INF/conf/axis2.xml`.

V tomto souboru můžeme nastavit uživatele Axisu a jeho heslo následujícími elementy.

```
<parameter name="userName">admin</parameter>
<parameter name="password">axis2</parameter>
```

5.4.3 UA Server

V prvním kroku zkopírujeme balík JavaRex do adresáře

`$AXIS2_HOME/WEB-INF/lib/`.

Na přiloženém CD je připravený Axis archiv `UAServer.aar`, který obsahuje zkompilovaný server připravený k nasazení. Jeho instalace se provádí přes webové rozhraní Axisu,

http://localhost:8080/axis2/axis2-admin/,

kde zvolíme *Upload Service*.

Druhým způsobem je zkopírování archivu přímo do adresářové struktury Axisu

`$AXIS2_HOME/WEB-INF/services/`

a následné restartování Tomcat serveru, aby se projevily změny konfigurace.

Server je v této chvíli nastartován a připraven k použití. Služby jsou dostupné na URL

http://localhost:8080/axis2/services/UAService

nebo Discovery služby na adrese

http://localhost:8080/axis2/services/UADiscoveryService.

5.5 Licence

Licence byla zvolena s ohledem na možnosti dalšího rozvoje aplikace a komerčního využití. Pro další rozvoj je vhodné použít open source licenci, což se ale plně neslučuje s komerčním záměrem. Proto byla vybrána BSD Licence, která umožňuje šíření v binární formě i formou zdrojových kódů. Konkrétně byla vybrána dvoubodová modifikace BSD licence, která vynechává třetí kluzuli o propagaci odvozeného díla s využitím jména organizace či přispěvovatelů.

5.6 OPC UA Klient

Pro server nebyl nalezen vhodný klient, kterým by se dala aplikace testovat. Proto byl opět použit Axis. Tentokrát pro vytvoření API pro klientskou aplikaci. Do vhodného adresáře byl zkopírován soubor, který byl použit pro vygenerování API serveru. V tomto skriptu byly změněny parametry skriptu pro generování Java kódu z definičních souborů WSDL tak, aby vznikly Java třídy pro implementaci klienta. Pro běh skriptu je opět nutné nastavit globální proměnnou `$AXIS2_HOME`.

```
#!/bin/bash
WSDLURI="http://www.opcfoundation.org/UA/2008/02/Bindings.wsdl"
TARGET="build/client"
$AXIS2_HOME/bin/wsdl2java.sh -uri $WSDLURI -d adb -s -o $TARGET
```

V adresáři `build/client` se vygenerují soubory `UADiscoveryServiceStub.java` a `UAServiceStub.java`. Tyto třídy poskytují rozhraní pro implementaci OPC UA klienta. Pro implementaci klienta v prostředí Eclipse je vhodné importovat do projektu balíky obsahující API serveru a Axisu. Balík pro implementaci klienta se opět vytváří nástrojem Ant. Spuštěním níže uvedeného příkazu v adresáři, do kterého se vygenerovaly třídy (`$TARGET`), vznikne balík, který obsahuje API klienta.

```
ant jar.client
```

Příklad klienta je na přiloženém CD. Klient vytvoří a aktivuje nové sezení na serveru, zaregistruje dvě hodnoty ze vzdáleného serveru REX a cyklicky nastavuje parametry na vzdáleném systému REX. Po vteřině vypisuje obě hodnoty z UA serveru, které jsou čtené REX modulem.

5.7 Testování

V této části jsou popsány provedené testy a zhodnocena jejich úspěšnost. Implementace byla ověřena na několika operačních systémech, novém portu serveru REX pro Linux a novém protokolu IPv6.

5.7.1 Linux

Testy probíhaly na operačním systému Linux, který byl použit i při implementaci serveru. Verze jádra byla 2.6.26-gentoo-r3. Server byl spolehlivě nainstalován a spuštěn. Na spuštěném serveru byly provedeny následující testy:

- Připojení klienta a vytvoření sezení bylo úspěšné.
- Aktivace sezení byla úspěšná.
- Vytvoření uzlu bylo úspěšné.
- Čtení dat z uzlu bylo úspěšné.
- Zaregistrování uzlu na REX modulu bylo úspěšné
- Čtení dat z uzlu bylo úspěšné včetně správnosti hodnot získaných REX modulem.
- Zápis dat na vzdálený server REX pomocí REX modulu modulu byl úspěšný.
- Odregistrace uzlu z REX modulu bylo úspěšné.
- Uzavření sezení bylo úspěšné.

Ověření správnosti dat získávaných REX modulem bylo provedeno nástrojem RexView. Testované proměnné měly stejnou hodnotu.

Testování pro operační systém Linux proběhlo úspěšně.

5.7.2 Windows

Testování proběhlo na operačním systému Windows XP SP2. Server byl úspěšně nainstalován a spuštěn. Na spuštěném serveru proběhly následující testy.

- Připojení klienta a vytvoření sezení bylo úspěšné.
- Aktivace sezení bylo úspěšné.
- Zaregistrování uzlu na REX modulu bylo úspěšné

- Čtení dat z uzlu bylo úspěšné včetně správnosti hodnot získaných REX modulem.
- Uzavření sezení bylo úspěšné.

Testování na systému Windows bylo úspěšné.

5.7.3 Mac OS X

Testování proběhlo na operačním systému Mac OS X verze 10.5.7 Leopard. Při instalaci OPC UA serveru se objevily problémy s verzí Javy. Mac OS X používá starší verzi jazyka Java. Proto byl server OPC UA překompilován na operačním systému Mac OS X. Po překompilování a sestavení balíku byl server úspěšně nainstalován a spuštěn. Na spuštěném serveru proběhly následující testy.

- Připojení klienta a vytvoření sezení bylo úspěšné.
- Aktivace sezení byla úspěšná.
- Zaregistrování uzlu na REX modulu bylo úspěšné
- Čtení dat z uzlu bylo úspěšné včetně správnosti hodnot získaných REX modulem.
- Uzavření sezení bylo úspěšné.

Testování na systému Mac OS X bylo úspěšné.

5.7.4 Port REX serveru pro Linux

Testování bylo provedeno i pro port serveru REX pro Linux. Původní REX server byl portován na Linux v rámci diplomové práce Bc. Romana Pišla. V tomto případě nebylo nutné testovat služby sezení, ale pouze funkčnost REX modulu oproti tomuto serveru. OPC UA server běžel na operačním systému Linux stejně jako řídící server REX. Následují vybrané testy pro ověření funkčnosti REX modulu.

- Zaregistrování uzlu na REX modulu pro portovaný server REX bylo úspěšné.
- Čtení dat z uzlu bylo úspěšné včetně správnosti hodnot dat získávaných REX modulem.

OPC UA server je tedy kompatibilní i s portovaným řídícím systémem REX.

5.7.5 IPv6

Protokol IP se používá již velice dlouho. Téměř celý internet je dnes postavený na verzi IPv4. Tato verze má však malý adresní prostor, který nedostačuje rostoucím potřebám internetu a stále sílí snaha o zavedení novější verze IPv6. OPC UA server byl testován i pro tuto novější verzi protokolu IP. Testování proběhlo na operačním systému Linux a bylo úspěšné. Protokoly TCP/IP navazují na aplikační server Tomcat, který je programován v Javě. Knihovny v Javě mají již implementovány třídy pro komunikaci pomocí protokolu IPv6.

6 Možnosti dalšího rozvoje

OPC UA server by se měl rozšířit hlavně o podporu služeb *View*. Tím by se dosáhlo výrazného navýšení funkčnosti. Dále by se měla serveru dopnit podpora pro *References*. Tím by byly uzly svázány. Implementace *References* podmiňuje implementaci *View*. Pokud by byly tyto dvě služby implementovány, jednalo by se již o aplikaci, kterou lze dále snadno rozšiřovat o podporu dalších funkcí.

Modul pro komunikaci s řídícím systémem REX by mohl být rozšířen tak, aby podporoval všechny služby, které jsou dostupné balíkem JavaRex.

OPS UA server nyní podporuje komunikaci pouze přes rozhraní SOAP. To usnadňuje komunikaci především internetovým a vizualizačním aplikacím. Aby mohlo být prováděno i řízení, bylo by vhodné, aby byl server rozšířen i o podporu komunikace v binární formě. To by zajistilo rychlejší přenos dat a tím i lepší řízení.

Projekt je licencován BSD licencí, a proto je možnost jeho vystavení na internetu spolu se zdrojovým kódem. Tímto způsobem se často vyvíjejí open source aplikace. Pokud je vystavený projekt na internetu dostatečně atraktivní, získá si komunitu, která ho bude dále rozšiřovat. Vystavení zdrojového kódu na internetu může způsobit, že chyby nalezené uživatelem může on sám opravit. Tyto opravy pak může zaslat tvůrci programu, který je obvykle použije, a tím odstraní chyby rychleji, než kdyby hledal a opravoval všechny chyby sám.

7 Závěr

V této části bude provedeno zhodnocení celé práce. Budou zde rozebrány úspěchy i neúspěchy práce a diskutováno splnění zadání.

K pochopení problematiky OPC UA byla nejdříve prostudována literatura o komunikačním standardu OPC Unified Architecture. Byly přečteny všechny vydané části specifikace. Seznámení s řídícím systémem REX proběhlo již dříve při práci na semestrálních pracích. Nově byly nastudovány informace o balíku JavaRex a zjištěny užitečné funkce pro implementaci OPC UA serveru. Nejprve byl vytvořen jednoduchý návrh pro OPC UA server, který popisoval rozhraní API serveru a adresního prostoru. Prvním krokem byla implementace OPC UA serveru pomocí specifikace TCP Binary, která nebyla úspěšná. Vyskytl se problém při testování komunikace s OPC UA klientem od firmy Unified Automation. Po tomto neúspěchu bylo rozhodnuto dále pokračovat v implementaci serveru s komunikačním rozhraním HTTP/SOAP. K implementaci byly využity technologie Tomcat a Axis. Tyto technologie zajistily celou komunikační vrstvu pro OPC UA server.

Pro implementaci API serveru bylo použito rozhraní vygenerované nástroji Axisu. Při použití balíku Axis byla objevena chyba ve funkci Axisu, která byla opravena. Implementace adresního prostoru proběhla úspěšně a téměř se nelišila od jeho návrhu. Jednotlivé uzly jsou uloženy ve struktuře, kterou lze velice rychle prohledávat vzhledem k nárokům na adresní prostor, a tím efektivně manipulovat s uzly. Modul pro řídící systém REX byl implementován použitím balíku JavaRex, který zprostředkovává komunikaci se serverem REX. Tento modul má implementovány základní funkce pro komunikaci OPC UA serveru se vzdáleným řídícím systémem REX. V práci je krátce popsán postup pro vytvoření OPC UA klienta.

Testování proběhlo na různých operačních systémech. Testy na operačních systémech Windows a Linux proběhly bez problémů. Na operačním systému Mac OS X se objevily problémy s verzí Javy, které byly odstraněny použitím nižší verze kompilátoru. Server byl následně úspěšně testován i na tomto operačním systému, čímž byl splněn jeden z hlavních cílů práce.

Server nelze považovat za plnohodnotný OPC UA server. Implementace byla zaměřena především pro použití s řídícím systémem REX. Tento cíl byl úspěšně splněn.

Seznam zkratek

API	Application programming interface
AFS	Apache Software Foundation
BSD	Berkley Software Distribution
COM	Component Object Model
DCOM	Distributed Component Object Model
FSF	Free Software Foundation
GNU	GNU's Not Unix
GPL	General Public License
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
ISO/OSI	Open Systems Interconnection
JSP	JavaServer Pages
JVM	Java Virtual Machine
MVC	Model-view-controller
OLE	Object Linking and Embedding
OPC	OLE for Process Control
OPC DA	OPC Data Access
OPC UA	Unified Architecture

PHP	PHP Hypertext Preprocessor
REST	Representational state transfer
RPC	Remote procedure call
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
WAR	Web ARchive
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Seznam literatury

[1] *OPC UA Specification: Part 1 – Concepts, Version 1.01*,
`http://www.opcfoundation.org/UA/Part1/`,

[2] *OPC UA Specification: Part 2 – Security Model, Version 1.01*,
`http://www.opcfoundation.org/UA/Part2/`,

[3] *OPC UA Specification: Part 3 – Address Space Model, Version 1.01*,
`http://www.opcfoundation.org/UA/Part3/`,

[4] *OPC UA Specification: Part 4 – Services, Version 1.01*,
`http://www.opcfoundation.org/UA/Part4/`,

[5] *OPC UA Specification: Part 5 – Information Model, Version 1.01*,
`http://www.opcfoundation.org/UA/Part5/`,

[6] *OPC UA Specification: Part 6 – Mappings, Version 1.0*,
`http://www.opcfoundation.org/UA/Part6/`,

[7] *OPC UA Specification: Part 7 – Profiles, Version 1.0*,
`http://www.opcfoundation.org/UA/Part7/`,

[8] *OPC UA Specification: Part 8 – Data Access, Version 1.01*,
`http://www.opcfoundation.org/UA/Part8/`,

[9] *OPC UA Specification: Part 9 – Alarms and Conditions, Draft Version 0.62*,
`http://www.opcfoundation.org/UA/Part9/`,

[10] *OPC UA Specification: Part 10 – Programs, Version 1.01*,
`http://www.opcfoundation.org/UA/Part10/`,

[11] *OPC UA Specification: Part 11 – Historical Access, Version 1.01*,
`http://www.opcfoundation.org/UA/Part11/`,

[12] Pavel Herout, *JAVA - Bohatství knihoven*,
ISBN 80-7232-368-5,

[13] REX Controls, *Funkční bloky systému REX*,
Plzeň 2008,

[14] Balda Pavel, Čech Martin, *Java interface to REX control system*,
`http://www.rexcontrols.cz/downloads/clanky/JavaRex.pdf`,

[15] Petr Břehovský, *Praktický úvod do TCP/IP*,
ISBN 80-85828-18-9,

[16] *HTTP: RFC 2616 - Hypertext Transfer Protocol - HTTP/1.1*,
`http://www.ietf.org/rfc/rfc2616.txt`,

[17] Apache Software Foundation, *Apache Tomcat 6.0*,
`http://tomcat.apache.org/tomcat-6.0-doc/index.html`,

[18] Apache Software Foundation, *Apache Axis2/Java*,
`http://ws.apache.org/axis2/`

[19] Sun Microsystems, Inc., *JavaServer Pages Technology*,
`http://java.sun.com/products/jsp/`,

[20] Apache Software Foundation, *Apache Ant*,
`http://ant.apache.org/`,

[21] Apache Software Foundation, *Maven*,
`http://maven.apache.org/`,

[22] *XML Schema Part 1: XML Schema Part 1: Structures*,
`http://www.w3.org/TR/xmlschema-1/`,

[23] *XML Schema Part 2: XML Schema Part 2: Datatypes*,
`http://www.w3.org/TR/xmlschema-2/`,

[24] *Web Services Description Language (WSDL) 1.1*,
`http://www.w3.org/TR/wsdl`

[25] *SOAP Part 1: SOAP Version 1.2 Part 1: Messaging Framework*,
`http://www.w3.org/TR/soap12-part1/`,

[26] *SOAP Part 2: SOAP Version 1.2 Part 2: Adjuncts*,
http://www.w3.org/TR/soap12-part2/,

[27] *WS Secure Conversation: WS Secure Conversation 1.3*,
http://docs.oasis-open.org/ws-sx/
ws-secureconversation/v1.3/ws-secureconversation.html,

[28] OPC Foundation, *What is OPC?*,
http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp,

[29] *Specification of generic 'OLE' interface and mechanisms Version - 1.00*,
http://wss.co.uk/pinknoise/Docs/Arc/Apps/OLESpec.html,

[30] Microsoft, *COM: Component Object Model Technologies*,
http://www.microsoft.com/COM/,

[31] Microsoft, *DCOM Technical Overview*,
http://msdn.microsoft.com/en-us/library/ms809340.aspx,

[32] Free Software Foundation, *BASH - GNU Project*,
http://www.gnu.org/software/bash/.