VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

KÓDOVÁNÍ DAT A FORMÁTY PRO VÝMĚNU INFORMACÍ

DATA SERIALIZATION AND FORMATS FOR INFORMATION EXCHANGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAROSLAV PROCHÁZKA
AUTHOR

VEDOUCÍ PRÁCE Ing. ONDŘEJ MORSKÝ
SUPERVISOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Jaroslav Procházka	***ID:***	106740
Ročník:	3	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Kódování dat a formáty pro výměnu informací

POKYNY PRO VYPRACOVÁNÍ:

Úkolem práce je popsat datové formáty, které je možné použít pro serializaci a deserializaci dat jako například XML, TLV, JSON a další. Student by měl tyto formáty porovnat z hlediska rychlosti počítačového zpracování, paměťové náročnosti, a podobně. Na základě techto znalostí bude navrhnut a implementován vlastní formát pro přenos a ukládání dat.

DOPORUČENÁ LITERATURA:

[1] KOSEK, Jiří. XML pro každého. Grada Publishing, 2000. 164s. ISBN 80-7169-860-1
[2] ENGLANDER, Robert. Java and SOAP. O'Reilly Media, 2002. 276s. ISBN 978-0596001759

Termín zadání:	7.2.2011	***Termín odevzdání:***	2.6.2011

Vedoucí práce: Ing. Ondřej Morský

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ANOTACE

Tato práce se zabývá kódováním dat a datovými formáty, které lze použít pro výměnu informací a je možné je použít pro serializaci a deserializaci dat. V rámci přenosu dat je podrobněji vysvětlen serializační a deserializační proces převodu datových struktur, objektů do sekvence bitů, kde později může být uložen ve vyrovnávací paměti, nebo přenesen počítačovou sítí. Jsou důkladně popsány nejznámější datové formáty XML a JSON. Práce hodnotí jednotlivé formáty a porovnává je z hlediska rychlosti počítačového zpracování, paměťové náročnosti a způsobu provedení jednotlivých formátů. Navržení a aplikace vlastního datového formátu pro přenos a ukládání dat.

KLÍČOVÁ SLOVA

Serializace, deserializace, kódování dat, datové formáty.

ABSTRACT

This thesis deals with a data encryption and data formats that can be used to exchange information and can be used to data serialization and deserialization data. The data is further explained and serialization and deserialization process of converting data structures, objects in a sequence of bits, which can later be stored in the cache memory or transferred to a computer network. They are thoroughly described in the best-known data formats, XML and JSON. The paper evaluates the different formats and compares them in terms of processing speed, memory consumption and execution of each format. Design and application of custom data format for data transfer and storage.

KEYWORDS

Serialization, deserialization, data coding, data formats.

PROCHÁZKA, J. *Kódování dat a formáty pro výměnu informací.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 42 s. Vedoucí bakalářské práce Ing. Ondřej Morský.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Kódování dat a formáty pro výměnu informací.“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce Ing. Ondřej Morský za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

V poslední době zaznamenává celosvětová síť Internet expanzi a tím i mnoho aplikací se sítí spojených. V dnešní době, ve které se stále rychleji rozvíjí internetové technologie, hrají významnou roli informační systémy. Na obecný rozvoj mají podstatný vliv informace a nové technologie, což podporuje rozvoj informačních systémů. V každém dni lidé přistupují prostřednictvím počítačů k aplikacím rozdílných typů. Aplikace nejvíce používané se dají rozdělit na dva odlišné typy. Prvním takovým typem můžeme nazvat desktopové, a neboť to jsou ty, které jsou umístěny a nainstalovány na určitém počítači. Druhým typem aplikace jsou webové, které jsou často umístěny na úplně jiném stroji, a na jiném místě a my k nim přistupujeme prostřednictvím rozhraní jiné aplikace – internetovým prohlížečem. S webovými aplikacemi se setkáváme např. při správě studijního e-learningu, v souvislosti se zábavou, během činnosti v povolání nebo členstvím na sociálních sítích a tak dále. V aplikacích provádíme každým dnem spoustu úkonů, buď rutinních, nebo za dané období jedinečných. Zřizování, úschova a možnost editace dat jednotlivých uživatelských profilů, patří ke každému uživatelskému účtu jednotlivých aplikací. Před několika málo lety, co doba výpočetní techniky debutovala, a i přes její globální rozšíření, jsou stále uživatelé, kteří disponují různou úrovní sekundární gramotnosti. Jsou aplikace intuitivně nenáročné, jako je např. webový e-mailový klient, ale také velmi náročné na ovládání či pochopení a pro neznalého uživatele možná i nepoužitelné. Příkladem muže být Joomla!, kde se nezkušený uživatel velmi snadno ztratí v různém nastavení, nebo rozmanité expertní, vývojářské systémy. Většina uživatelů často ráda ocení, když s aplikací může pracovat intuitivně bez jakéhokoliv zaškolení o problematice. Samozřejmě nejvhodnější a nejpříjemnější řešení by bylo, kdyby nám aplikace usnadňovala rutinní výkony, nejlépe je prováděla automaticky a těmi nejnáročnějšími aplikacemi by nás s co největší trpělivostí provedla. Za největší výhodu webových aplikací lze předpokládat jejich nesmírný potenciál být navržen „na míru“ uživatelům. Existují systémy, jejichž úkolem je usnadnit uživateli práci tím, že přizpůsobují své chování a uživatelské rozhraní potřebám, cílům, znalostem a dovednostem uživatele. Takové aplikace můžeme nazvat adaptivními. Adaptivní aplikace pracují na základě určitých zdrojových dat, konkrétně to mohou být operace, které uživatel provádí, nějakým způsobem přeformulované a dekomponované do elementárních akcí [1]. Tyto aplikace musí však podléhat nějakým pravidlům přenosu počítačovou sítí a dnešní trend je takový vývoj, že se snaží zacílit na rychlost přenosu a čistotu kódu. Kódy jsou většinou velmi složité a proto je v poslední době snaha kódy co nejvíce optimalizovat na co nejjednodušší možné řešení. Při přenosu dat počítačovou sítí v souvislosti s ukládáním dat v počítačích se při bližším zkoumání setkáme s tzv. pojmem – „serializace“. Serializace je proces, který převádí datovou strukturu nebo objekt do posloupnosti bitů, z čehož plyne, že data mohou být uložena v souboru nebo ve vyrovnávací paměti. Takto zpracovaná data jsou přenášena přes síťové připojení z jednoho počítače na druhý. Později, ve stejném nebo v jiném počítačovém prostředí, jsou tato data tzv. „vzkříšena“. Tento proces se nazývá deserializace. Deserializace je tedy opačná operace, jde tam o extrakci datové struktury ze série bajtů [2].

Mým cílem bude navrhnout a implementovat, na základě znalostí o kódování dat a

formátů pro výměnu informací, vlastní formát pro přenos a ukládání dat. Vlastní formát dat je vytvořen pro prostředí Java. Jsou porovnány parametry typu náročnost na výpočetní výkon počítače, rychlost počítačového zpracování a náročnost na paměťový prostor.

V první kapitole jsou popsány teoretické poznatky k serializaci datových formátů. V kapitole je rozepsáno použití serializace, podpora programovacích jazyků, které se využívají k jejich naprogramování či simulaci pro různé výzkymy, a důsledky využití serializace.

Ve druhé kapitole jsou rozepsány poznatky o různých serializacích a jejich druhy. Dále je zde uveden pojem deserializace a následně je tento pojem vysvětlen.

Ve třetí kapitole jsou vysvětleny jednotlivé datové formáty serializace a rozebrány jejich výhody a nevýhody použití nebo nasazení v praxi.

Čtvrtá kapitola pojednává o praktickém řešení projektu. Popisuje vyvinutý simulační program. Je zde uveden popis rozhrání a ovládání programu. Přehled struktury aplikace a popsané třídy tvořící program pro simulaci.

Předposlední kapitola je vyhrazena pro provedené simulace v aplikaci. V této kapitole jsou názorně předvedeny jednotlivé náročnosti časových závislostí všech datových formátů. Dále jsou zde porovnány datové náročnosti a celkové zhodnocení použitých formátů.

Závěrečná kapitola porovná a hodnotí shrnutí všech čtyř datových formátů serializace simulované na vyvinuté aplikaci.

1 SERIALIZACE

Obecně, serializace je proces, který neznamená nic jiného, než seřazení původně rozvětveného kódu. Tento proces je složitější, než by se mohlo na první pohled zdát. Především se musí vyřešit serializace samotných dat. Musejí být pokud možno předem známa pravidla, podle kterých se bude určovat, která data budou patřit za která. Pokud by ale pravidla neexistovala a žádná data by se o druhá data nestarala, velmi brzy by došlo k zablokování celého procesu přenosu informací. Také musí být vymyšlen přechod zpět ze serializovaného stavu. Tedy vlastně opačný proces a to proces deserializace, což přesně znamená, navrácení původního stavu po jeho serializaci. Pokud chceme po serializaci a deserializaci dosáhnout stejného stavu jako předtím, musí být pravidla pro deserializaci právě opačná k pravidlům serializace [2].

1.1 Použití

Serializace stanovuje metodu přetrvávajících objektů, které jsou mnohem pohodlnější, než psaní jejich vlastností do textového souboru na disku a opětovné obnovení souboru zpět. Dále způsob vydávající vzdálené volání procedur, např. jako v SOAP [3]. Metoda pro distribuci objektů, a to zejména v oblasti softwarových komponent, jako je COM, CORBA atd. Metoda pro detekci změn dat v různém čase.

Pro některé z těchto funkcí, aby byly užitečné, musí být zachována nezávislost architektury. Například, pro maximální využití distribuce, je počítač konstruován na různých hardwarových požadavcích a měla by být schopna spolehlivě rekonstruovat serializovaný datový tok, bez ohledu na endianitu. Endianita je v informatice způsob uložení čísel v paměti počítače, který definuje, v jakém pořadí se uloží jednotlivé bajty číselného datového typu. Označuje se také jako pořadí bajtů. To znamená, že jednodušší a rychlejší postup přímého kopírování paměti, rozložené datové struktury, nemohou fungovat spolehlivě na všech počítačových architekturách [4]. Serializovat datové struktury v architektuře nezávislým formátem znamená, že zde nejsou problémy v uspořádání v bajtech, uspořádání paměti, nebo prostě různé způsoby, jak reprezentovat datové struktury v různých programovacích jazycích.

Neodmyslitelnou součástí serializace je definice klíče z důvodu zakódování dat. Vytažení jedné části serializované datové struktury vyžaduje, aby byl celý objekt přečten od začátku do konce, a byl rekonstruován. V mnoha aplikacích tato linearita je přednější, protože to umožňuje jednoduché, společné vstupně/výstupní rozhraní, která mají být využity k předání dat o stavu objektu [4]. V aplikacích, kde vyšší výkon, je problém, může při organizaci složitějších, nelineárních operací vynaložit více úsilí k řešení.

Vzhledem k tomu, jak serializace a deserializace mohou být řízeny z běžných kódů, je možné pro společný kód dělat obojí najednou. Lze zjistit rozdíly mezi objekty a serializovat jejich předchozí kopie a přitom poskytují podklad pro další vstup takové detekci. Není nutné sestavovat dřívější kopii, protože rozdíly mohou být zachyceny „on the fly“ neboli bezmyšlenkovitě. Je to způsob, jak rozumět technice zvané diferenciální provedení. Zmíněná technika je užitečná při programování uživatelského rozhraní,

jehož obsah se v různém čase mění – grafické objekty mohou být vytvořeny, odstraněny, upraveny, aniž by nutně musel psát samostatný kód pro objekty vstupních událostí [5].

1.2 Podpora programovacích jazyků

Několik objektově orientovaných programovacích jazyků podporuje přímo objektovou serializaci. Poskytují k tomu standardní rozhraní jednotlivých programovacích jazyků. Nejznámější z objektově orientovaných programovacích jazyků jsou Python, PHP, Java, C++. Jsou k dispozici také knihovny, přidávající podporu pro serializaci jazyků, které postrádají přirozenou podporu.

1.2.1 Java

Java poskytuje automatickou serializaci, která vyžaduje, aby byl objekt označen serializačním rozhraním. Implementující rozhraní ochranných známek ve třídě jako „serializace v pořádku“, potom zpracuje serializaci v Javě interně. Nejsou žádné serializační metody definované na rozhraní serializace, ale serializační třída může volitelně definovat metody s některými zvláštními jmény a podpisy, že pokud je definována, bude se nazývat jako součást serializace či deserializace procesu [3]. Jazyk také umožňuje vývojářům přepsat serializační proces důkladně pro jiné rozhraní, které obsahuje dvě speciální metody, které se používají k uložení a obnovení stavu objektu.

Standardní kódovací metoda používá jednoduchý překlad polí do přenosu bajtů. Stejně jednoduše jako nepřechodné a nestatické, odkazuje na objekty, které jsou zakódovány do přenosu. Každý objekt, na který odkazuje serializovaný objekt a není označen, jako přechodný, musí být rovněž serializován, a pokud nějaký objekt v kompletním složení odkazuje na objekt nepřechodný, nebyl serializován, poté se serializace nezdaří. Vývojář může ovlivnit toto chování označením objektů jako přechodné, nebo pro nově spuštěnou serializaci zkrátí část kódu tak, aby jeho část nemohla být serializována. Objekty v Javě je možné serializovat pomocí JDBC a ukládat je do databáze [3].

1.2.2 Python

Python realizuje serializaci přes standardní knihovny a moduly. Python využívá nejjednodušší modul pro serializaci objektů, marshal. V modulu marshal existuje proces nazývaný marshaling, při kterém dochází konverze dat z interního do externího tvaru. Unmarshaling je pak proces opačný. Modul marshal používá především jazyk Pythonu pro ukládání kompilovaných souborů. Interní formát není dokumentovaný, protože verzi od verze se mění. Nelze proto spoléhat na jeho zpětnou kompatibilitu. Nicméně je nezávislý na platformě, a tudíž soubory zapsané na jednom počítači může použít i na jiném počítači tutéž verzi Pythonu. Dále je využíván modul Pickle. Tento modul Pickle umožňuje lépe řídit proces serializace objektů. Tento objekt již zaručuje zpětnou kompatibilitu a dokáže serializovat i instance uživatelských tříd a dokonce i třídy a funkce samotné. Modul Pickle si neukládá žádný kód serializace, pouze si uloží jméno modulu a jméno třídy. Modul pickle při vícenásobné serializaci jednoho objektu pouze uloží odkaz na posledně serializovaný objekt. Proto je třeba dávat pozor na změny

objektů mezi jednotlivými serializacemi, pokud k nim dojde. Poslední významný modul pro serializaci dat je modul shelve. Modul shelve je mezistupněm mezi serializací a opravdovými perzistentními objekty. Řeší totiž otázku pojmenování objektů. Shelve používá pro ukládání databázi, která se použije závisle na platformě a dostupných modulech [6].

Python ve vyšších verzích obsahuje již standardní knihovny pro podporu JSON a XML. Nicméně, tyto moduly Pythonu zvládnout jen základní typy řetězců jako celá čísla a sbírky základních typů. Vzhledem k tomu, že modul Pickle je určen pro libovolné objekty.

1.2.3 PHP

PHP realizuje serializaci přes vestavěné serializační a deserializační funkce. V PHP jazyce lze serializovat některé ze svých datových typů, s výjimkou zdrojů (např. soubor ukazatelů či pouzder apod.).

Pro objekty starší verze PHP jsou zde dvě „magické metody“, které mohou být realizovány v rámci definice třídy – *sleep ()* a *wakeup ()* metody. Tyto dvě metody jsou volány v rámci serializace a deserializace, respektive k vyčištění a obnovení objektu. Funkce *sleep()* se spustí na začátku serializace. Může se do ní vložit kód například pro ukončení spojení s databází. Důležité také je, aby funkce *sleep()* vrátila pole, které obsahuje proměnné objektu, ze kterých se vytvoří bajt řetězec. Naproti tomu funkce *wakeup()* se spustí na začátku deserializace. Lze ji využít například pro obnovení spojení s databází [8]. Tyto metody mohou také povolit, které vlastnosti objektu budou serializovány. Existuje zde jeden velmi důležitý bezpečnostní nedostatek. Pokud je řetězec bajtů předáván v session, data jsou zde v nezašifrované formě, takže data může případný útočník velmi jednoduše odchytit a pokud útočník zná strukturu bajt řetězce, která není nijak složitá, nebude problém si data přečíst a dostat se tak do vytvořené databáze. Podmínkou je tedy použití například zabezpečeného protokolu SSL.

1.2.4 C++

Jazyka C++ nabízí poněkud omezenou podporu pro zpracování souborů. Tento problém pravděpodobně vznikl v době, kdy byl programovací jazyk koncipován a již používán. Mnoho jazyků, které byly vyvinuty po C++, jako je Object Pascal, jde o rozšíření programovacího jazyka vlastnostmi objektově orientovaného programování. Dále programovací jazyk Java dokáže poskytnout mnohem lepší podporu, pravděpodobně proto, že knihovny byly realizovány jako jejich přímý požadavek na tento problém. Na základě omezené podpory, C++ podporuje ukládání pouze hodnot primitivních typů, jako jsou int, char, double.

Objekt se skládá ze serializace uložených hodnot, které jsou součástí objektu, většinou hodnotu zdědil od deklarované, proměnné třídy. Současný standard C++, nepodporuje serializaci objektů interně. Pro provedení tohoto typu operace se musí použít technika, známá jako binární serializace. K uložení střední hodnoty musí být využita třída *fstream*, která poskytuje možnost uložit hodnoty v binárním formátu. Tento formát se skládá z úspor každého bajtu na nosiči sladěním bajtů souvislým způsobem, stejným způsobem jsou uloženy v binárních číslech proměnné [7].

1.3 Důsledek

Serializace zbaví neprůhlednost abstraktního datového typu, potenciálně vystavuje soukromé implementační detaily. Tímto dochází k odrazení konkurentů od výroby kompatibilních produktů, formátů. Autoři proprietárního softwaru často udržují podrobnosti serializace svých programů, formátů v tajemství a nelze u nich upravovat zdrojové kódy. Někteří úmyslně popletou, nebo dokonce zašifrují serializovaná data. Přesto, součinnost vyžaduje, aby mohli žádostem porozumět a navzájem serializovat formáty [2]. Proto vzdálené volání metody architektury, jako je CORBA, definuje jejich serializační formáty v detailu a často poskytuje metody ověřování důslednosti některého serializačního přenosu při převodu zpět do objektu.

1.3.1 Člověkem čitelný formát

V pozdních devadesátých letech, poskytuje alternativu ke standardnímu protokolu serializace. Takovým protokolem začal být XML, značkovací jazyk, který je založený na kódování. Tento jazyk je zvyklý na produkci lidsky čitelného textu. Takové kódování může být užitečné pro objekty, které mohou být pochopeny mnohem lépe pro člověka nebo komunikaci k jiným systémům, bez ohledu na programovací jazyk. Má to nevýhodu ztráty kompatibility přenosu bajtů základního kódování, který je obecně více praktický. Řešení tohoto dilema je transparentní, komprimační režim (např. binární XML) [8].

XML je dnes často užitý na asynchronní přenos uspořádaných dat mezi klientem a serverem ve webových aplikacích v Ajax. Alternativa pro tento případ je použití formátu JSON, tento orientovaný serializační protokol je textově lehčí, který používá JavaScript syntax a je podporován ve více jiných programovacích jazycích.

Další alternativa je YAML. YAML je skutečná nadmnožina JSON a obsahuje funkce, které dělají serializace mnohem účinnější, a tím i více přátelské pro člověka a potenciálně kompaktnější [9]. Mezi tyto funkce patří pojem značkovací datové typy, podpora pro nehierarchické datové struktury, možnost datovou strukturu s odsazením a mnoho forem skalárních dat.

2 DRUHY SERIALIZACE

Serializace mohou být následujících typů:

- Binární serializace
- SOAP serializace
- XML serializace

2.1 Binární serializace

Binární serializace je mechanismus, který píše data do výstupního přenosu tak, že to může být používáno automaticky k rekonstrukci objektu. Termín binární v názvu znamená, že potřebné informace, které jsou nutné k vytvoření přesné binární kopie objektu, jsou uloženy na paměťovém médiu. Pozoruhodný rozdíl mezi binární serializací a XML serializací je ten, že binární serializační instance zachovává identitu, zatímco XML serializace nezachovává identitu serializační instance [10]. Jinými slovy, v binární serializaci je celý objekt uložen do stavu, kdyžto v serializaci XML pouze pro některé z objektů jsou data uložena. Binární serializace zvládne data s více odkazy na stejný objekt, oproti tomu XML serializace odkazuje na jedinečný objekt.

2.1.1 Výhody a nevýhody binární serializace

Jednou z hlavních výhod použití binární serializace v řízeném prostředí je to, že objekt může být deserializován od stejného data, kde byl serializován. Kromě toho, další výhodou binární serializací je vyšší výkon, protože je rychlejší a ještě silnější v tom smyslu, že poskytuje podporu pro komplexní objekty, vlastnosti pouze pro čtení a dokonce i cyklické odkazy. Nicméně nevýhodou je, že tato binární serializace není snadno přenosná na jinou platformu.

2.2 SOAP serializace

Protokol SOAP je ideální pro komunikaci mezi aplikacemi, které používají heterogenní architektury. Aby bylo možné používat SOAP serializace v .NET musíme přidat odkaz na *System.Runtime.Serialization.Formatters.Soap* do žádosti. Základní výhodou je přenositelnost SOAP serializace. Formát *SoapFormatter* serializuje objekty ve zprávě SOAP, nebo analyzuje SOAP zprávy a výpisy objektů ze zprávy serializace [3].

2.3 XML serializace

Podle MSDN, XML serializace konvertuje *public* oblasti a vlastnosti objektů nebo parametry a vrátí jejich hodnoty metodou přenosu XML, který odpovídá na konkrétní schéma jazyku XML definované jazykem XSD dokumentu. Výsledky XML serializace jsou silné ve třídě s *public* vlastnostmi a polích, která jsou převedena na sériový formát, v tomto případě XML, pro ukládání nebo přenos. Protože XML je otevřený standard, může XML přenos zpracovávat libovolné aplikace podle potřeby, bez ohledu na

platformu. XML serializaci provádí v .NET [8], které je poměrně jednoduché. Základní třída, která se musí použít jak pro serializaci, tak i pro deserializaci, je *XmlSerializer*. Webové služby používající protokol SOAP pro komunikaci a návratové typy s parametry jsou pomocí třídy *XmlSerializer* serializovaná. Serializace XML je však mnohem pomalejší v porovnání s binární serializací.

2.4 Práce s formátovači (formatters)

Formátovač je použit pro určení serializačního formátu pro objekty. Jinými slovy, formátovač se používá na kontrolu serializace objektu při přenosu. Jsou to objekty, které se používají ke kódování a serializaci dat do vhodnějšího formátu, než jsou přenášeny po síti. Vyskytující se na rozhraní nazvané *IFormatter* rozhraní. *IFormatter* rozhraní je významné pro metody serializace a deserializace, které vykonávají aktuální serializaci a deserializaci. V tomto rozhraní jsou dvě formátovací třídy poskytované v rámci .NET, *BinaryFormatter* a *SoapFormatter* [4]. Obě tyto třídy rozšiřují možnosti *IFormatter* rozhraní.

2.4.1 Binární formátovač

Binární formátovač poskytuje podporu pro serializaci pomocí binárního kódování. *BinaryFormater* třída je zodpovědná za binární serializaci a je běžně používáná v .NET technologii. Tato třída není vhodná, pokud mají být data přenášena přes firewall.

2.4.2 SOAP formátovač

Formátovač SOAP poskytuje formátování, které může být použito k serializaci objektů pomocí protokolu SOAP. To se používá k vytvoření SOAP obálky, která používá jako objekt graf vygenerovaných výsledků. Tato obálka zapříčiní serializaci objektů do zpráv SOAP, nebo rozbor zpráv SOAP a dokáže ze SOAP zprávy extrahovat data z již zaserializovaných objektů [3]. SOAP formátovače v .NET jsou široce využívány ve webové správě.

2.5 Deserializace

Následující informace platí pro všechny třídy, které dědí z *XmlObjectSerializer*, včetně *DataContractSerializer* a *NetDataContractSerializer* tříd. Nejzákladnější způsob jak rekonstruovat objekt ve Visual Basic .NET je zavolat jednu z metod přetížení *readObject***.** K dispozici jsou tři přetížení, každá z nich pro čtení s *XmlDictionaryReader*, *XmlReader*, nebo *Stream*. Přetížení *Stream* vytvoří textové *XmlDictionaryReader***,** které není chráněno žádnými kvótami, a mělo by být používáno pouze pro čtení důvěryhodných dat [11]. Důležité také je, že objekt *readObject* vrací metodu na příslušný typ. Následuje výpis z kódu konstrukce instance *DataContractSerializer* a *XmlDictionaryReader*, poté deserializuje *Person* instanci. Obrázek 2.1 Instance DataContractSerializer a XmlDictionaryReader.

```
Dim dcs As New DataContractSerializer( GetType (Person))
Dim fs As New FileStream(path, FileMode.Open)
Dim reader As XmlDictionaryReader = _
   XmlDictionaryReader.CreateTextReader(fs, New XmlDictionaryReaderQuotas())

Dim p As Person = CType (dcs.ReadObject(reader), Person)
```

Obrázek 2.1 Instance DataContractSerializer a XmlDictionaryReader

Před voláním metody *readObject* je pozice prvku XML z obálky nebo neobsahuje žádný obsahový uzel, který předchází prvku z obalu. Lze to provést tím způsobem, že se zavolá čtecí metoda a to *XmlReader* nebo jeho odvození a testuje typ uzlu, jak je uvedeno v následujícím kódu. Obrázek 2.2 Odvození typu uzlu.

```
Dim ser As New DataContractSerializer( GetType (Person), "Zákazník", "http://www.contoso.com" )
Dim fs As New FileStream(path, FileMode.Open)
Dim reader As XmlDictionaryReader = XmlDictionaryReader.CreateTextReader(fs, New XmlDictionaryReaderQuotas())

While reader.Read()
    Select Case reader.NodeType
        Case XmlNodeType.Element
            If ser.IsStartObject(reader) Then
                Console.WriteLine( "Nalezen  element" )
                Dim p As Person = CType (ser.ReadObject(reader), Person)
                Console.WriteLine( "{0} {1}" , _
                                   p.Name, p.Address)
            End If
            Console.WriteLine(reader.Name)
    End Select
End While
```

Obrázek 2.2 Odvození typu uzlu

Při použití jednoho z jednoduchých *ReadObject* přetížení, deserializace hledá výchozí název a jmenný prostor prvku z obálky a vyvolá výjimku, pokud zjistí neznámý prvek. *IsStartObject* se nazývá metoda k ověření a která je umístěna na prvku a je pojmenovaná podle očekávání [11]. Existuje způsob, jak zakázat této obálce ověřit název prvku, z některých přetížení *readObject* metody a přijmout logické parametry *verifyObjectName,* které jsou nastaveny na hodnotu pravda ve výchozím nastavení. Pokud je nastavena na nepravdu, název a jmenný prostor prvku z obalu je ignorován. To je užitečné pro čtení XML, který byl napsán již dříve s použitím mechanismu serializace krok za krokem.

3 DATOVÉ FORMÁTY SERIALIZACE

Serializační formáty jsou využívány na načítání a ukládání dat, na nižší úrovni potřeby náročnosti využití počítačového výkonu. Různé serializační formáty se liší v tom, co se serializuje (soubor, spojení, vektory) a zda jsou určeny k serializaci jednoho objektu nebo více objektů pohromadě, typicky to může být pracovní prostor. Data poté uloží do souboru a na začátek záhlaví vloží uvedený formát serializace.

Serializace musí brát v úvahu, že objekty mohou obsahovat i odkazy na prostředí, které potom musí přiložit i toto prostředí. Zde je pod pojmem prostředí považováno jako balíčky nebo jmenný prostor, kde jsou uložena data podle názvu. Dále jsou zde referenční objekty, které nejsou duplikovány na kopírování a měly by zůstat na sdílení deserializaci. Jedná se o slabé reference, externí odkazy, jmenné prostory, prostředí jiné než ty, spojené s balíčky. Referenční objekty jsou řešeny pomocí hash tabulek a odkazy po první referenční značce, jsou zapsány položkou v tabulce.

3.1 Formát JSON

JSON (JavaScript Object Notation) je oproti XML využívaný především moderními AJAX aplikacemi. Vznikl jako odlehčená náhrada za XML. JSON využívají například aplikace Twitter, Facebook, Delicious. Je schopný pojmout pole hodnot, objekty, řetězce a čísla, tedy prakticky bez jakéhokoliv omezení. Tento formát JSON lze jednodušeji analyzovat, je snadno čitelný i zapisovatelný člověkem a je i velmi efektivně strojově generovatelný. Je založený na programovacím jazyce JavaScript. JSON je zcela nezávislý datový formát. Formát je textový, využívající však úmluvu dobře známou programátorům jazyků rodiny C, Java, JavaScript, Python a Perl [12]. Pomocí těchto programovacích jazyků je formát JSON pro výměnu dat opravdu ideálním jazykem.

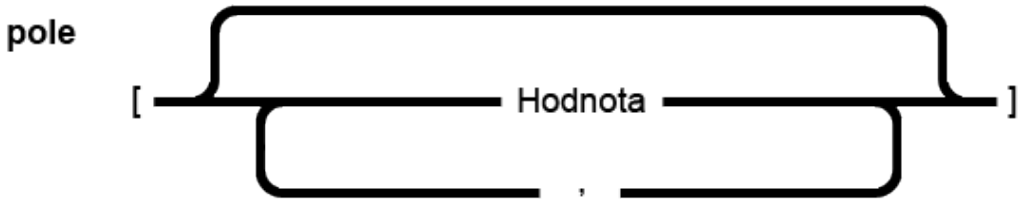
JSON je založen na dvou základních strukturách. Soubor párů název – hodnota. Tento soubor párů bývá v rozdílných jazycích uskutečněn jako *objekt*, který obsahuje položky typu záznam, struktura, slovník, hash tabulka, klíčový seznam nebo asociativní pole. Druhou strukturou JSON je tříděný seznam hodnot. Ten je ve většině jazyků realizován jako *pole*, obsahující vektor, seznam nebo posloupnost.

Existuje několik způsobů, jak ověřit strukturu a datové typy uvnitř JSON, podobně jako u struktury XML. JSON struktura je specifikace pro JSON, založený formát pro definování struktury dat JSON. Data JSON jsou vyžadována pro dané aplikace a jsou plně modifikovatelná, stejně jako tomu je u struktury XML, který stanovuje formát XML. JSON struktura je určena poskytnout validaci, dokumentaci a interakční kontrolu dat JSON. Tato struktura je nejvíce podobná struktuře XML, ale i založené na pojetí struktur *RelaxNG* a *Kwalify*. Základy určené JSON, jsou data ve formě určité struktury a mohou být používaná k určení JSON dat [12]. Stejně tak může být využito u nástrojů k serializaci a deserializaci dat.

Jedná se o univerzální datovou strukturu a v podstatě všechny moderní programovací jazyky je v nějaké uspokojivé formě podporují. Je tedy velmi praktické, aby na tomto formátu JSON byl založen na jazyce nezávislý výměnný formát. V datovém formátu JSON jsou tyto struktury realizovány s využitím následujících

konstrukcí. *Objekt* je netříděná množina párů název – hodnota. Objekt je uvozen znakem *{* levá složená závorka a ukončen znakem *}* pravá složená závorka. Každý název je následován znakem *:* dvojtečka a páry název – hodnota jsou pak odděleny znakem *,* čárka, jak je zřejmé na obrázku Obrázek 3.1 Struktura object. Struktura *Pole* je setříděnou sbírkou hodnot. Začínajícím znakem *[* levá hranatá závorka a končící znakem *]* pravá hranatá závorka. Hodnoty jsou odděleny znakem *,* čárka, pro lepší pochopení nastíněné situace, zobrazeno na obrázku Obrázek 3.2 Struktura pole.

Obrázek 3.1 Struktura object



Obrázek 3.2 Struktura pole

Hodnotou rozumíme *řetezec* uzavřený do dvojitých uvozovek, *číslo*, `true`, `false`, `null`, *objekt* nebo *pole*. Tyto struktury mohou být vnořovány. Dále *řetězcem* je nula nebo více znaků kódování Unicode, uzavřených do dvojitých uvozovek a využívající únikových sekvencí s použitím zpětného lomítka. Znak je reprezentován jako řetězec s jediným znakem. Řetězec je velmi podobný řetězcům z jazyků C nebo Java. *Číslo* je podobné číslům z jazyků C a Java. Jedinou výjimkou je, že není používán oktanový ani hexadecimální zápis. Mezi jednotlivé syntakční znaky a hodnoty lze vkládat bílé znaky, takzvané whitespace. Až na pár výjimek týkajících se kódování je tímto jazyk kompletně popsán.

3.1.1 Výhody a nevýhody JSON oproti XML

Hlavní výhoda formátu JSON oproti XML formátu je menší velikost přenášených dat. Úspora v zapsání stejných dat ve formátu JSON oproti XML formátu, může být až 40%. Tento fakt je dán zřejmě tím, že se nevyužívají párové tagy. Na menších souborech tento rozdíl není až tak zřejmý, avšak na větším objemu dat je tento rozdíl již rapidní. Uvedená výhoda se projeví ve finanční úspoře na vynaložené prostředky k přenosu dat. Další nespornou výhodou je jednoduchý formát, určený a vyvinutý přímo pro výměnu dat, který je dobře čitelný pro člověka a velmi stabilní.

Za nevýhodu u formátu JSON lze možná považovat nemožnost definovat znakovou sadu přenášeného obsahu, což samo o sobě ale nemusí být jako nevýhoda chápáno. Výchozí kódování je moderní UTF-8.

3.2 Formát YAML

YAML nebo také YAML Ain't Markup Language. Formát YAML je určen pro serializaci dat textových souborů. V tomto méně známém formátu je velká škála možností. Strukturovaná data se dají snadno v aplikaci načíst nebo uložit [13]. Jsou zde dva způsoby záznamů, které může zvolit a to interní reprezentaci dat binární formou nebo může použít pouze čistě textově strukturovaný formát. Každá možnost z těchto způsobů má své pro i proti. Binární formát se ukládá i načítá většinou nejrychleji a většinou zabere nejméně místa. Ovšem tento binární formát bez použití speciálních editorů, není obvykle zpracovatelný mimo aplikaci. Formát textový zabere víc místa, ale jeho výhodou je, že textový formát lze zpracovat i obyčejným textovým editorem.

Textové formáty pro ukládání dat mohou být různě čitelné. Srovnání s daty serializované pomocí PHP funkce „serialize“, už tato data nejsou příliš srozumitelná. Datový soubor, který prošel PHP serializací, nelze editovat jednodušším editorem, je to téměř nemožné. Oproti formátu XML, který lze relativně snadno číst i ukládat, lze formát XML i poměrně jednodušším textovým editorem editovat. Na druhou stranu úprava XML souboru vyžaduje poměrně slušné znalosti jeho syntaxe. Zpracování v aplikaci není úplně triviální. Někde uprostřed, co do složitosti editace a zpracování stojí jednoduché formáty typu formátu INI souborů.

YAML je formát, který nabízí jednoduchost souborů INI, bez složitých konstrukcí jakými jsou tagy, uzavírání elementů, únikových znaků a zároveň formát YAML je schopen vyjádřit v čistém textu i složitější konstrukce, např. struktury nebo pole. S formátem YAML se můžeme setkat v současnosti stále častěji, důvěrně známý bude pravděpodobně vývojářům, kteří používají Ruby, ale knihovny pro zpracování YAML souborů existují téměř pro každý používaný programovací jazyk [13].

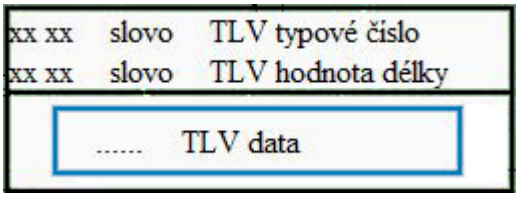
3.2.1 YAML a jeho největší výhody

YAML je mnohem mocnější formát, než by se mohlo na první pohled zdát. Disponuje snazší čitelností kódu, možností snadné editace. Formát YAML dovoluje zapsat strukturovaná data, obdobně jako u formátu XML, s menší režií a s vyšší uživatelskou přívětivostí, než je zápis v XML formátu. Umožňuje vytvářet mnohem složitější struktury pomocí tzv. aliasů, operátory & a *, umožňuje vložit binární data jakožto různé obrázky v kódování BASE64, umožňuje spojovat údaje z různých dokumentů, nebo nastavovat defaultní hodnoty. YAML je označován jako „serializační formát“, myšleno jako v podobě textového souboru strukturovaných dat.

3.3 Formát TLV

Formát TLV je velmi pohodlný a efektivní způsob úpravy dat do organizovaného formátu a to zejména s proměnnou délkou řetězce. V datovém komunikačním protokolu mohou být zakódovány nepovinné informace jako typ, délka, hodnota. Zkratka TLV vyjadřuju přesně tyto parametry v datovém protokolu. Typ a délka pole jsou vyjádřeny ve velikosti obvykle 1-4 bajtů a hodnota pole je proměnné velikosti. V části komunikační zprávy obsahuje pole blok označený jako typ, tento blok obsahuje číselný kód, který označuje typ pole [14]. Pro představu, může tato část zprávy představovat

textový řetězec. Dalším blokem v tomto formátu je délka. Číslo udávající velikost dat v daném bloku, typicky se udává v bajtech. Poslední důležitý člen v poli je blok nazvaný hodnota. Data daného bloku jsou variabilní velikosti bajtů, která obsahuje údaje pro tuto část zprávy. V této části bloku je již vyjádřena podstata zprávy. Grafické zobrazení TLV struktury na TLV struktura obvykle používá binární formát dat a často jsou zaměřené na optimalizaci objemu a rychlosti zpracování dat. Používají se také pro přenos nekonečného proudu dat, například v telekomunikacích [14].



Obrázek 3.3 Struktura TLV

3.3.1 Výhody použití TLV

Formát TLV snadněji vyhledává sekvence pomocí zobecněné analýzy funkcí. Nové prvky zprávy, které jsou přijímány na starší uzel, mohou být bezpečně přeskočeny a zbytek zprávy lze analyzovat. Například v přijímané zprávě se vyskytne neznámá značka a tato značka bude bezpečně přeskočena a analýza zprávy bude pokračovat dále. TLV formát používá prvky obvykle v binárním formátu, který umožňuje rychlejší analýzu a větší úsporu dat.

4 PRAKTICKÉ ŘEŠENÍ PROJEKTU

V této části je představena aplikace, která byla vyvinuta jako hlavní cíl této práce. Aplikace je pojmenovaná DataCoding – Kódování dat a formátů. Je zde vysvětlení funkcí aplikace a popsaný způsob použití aplikace DataCoding.

Tato kapitola bude vhodná i jako manuál ke správnému rozjetí a následnému použití aplikace. Popis struktury programu, jednotlivých částí a následně popsány děje při ovládání simulace.

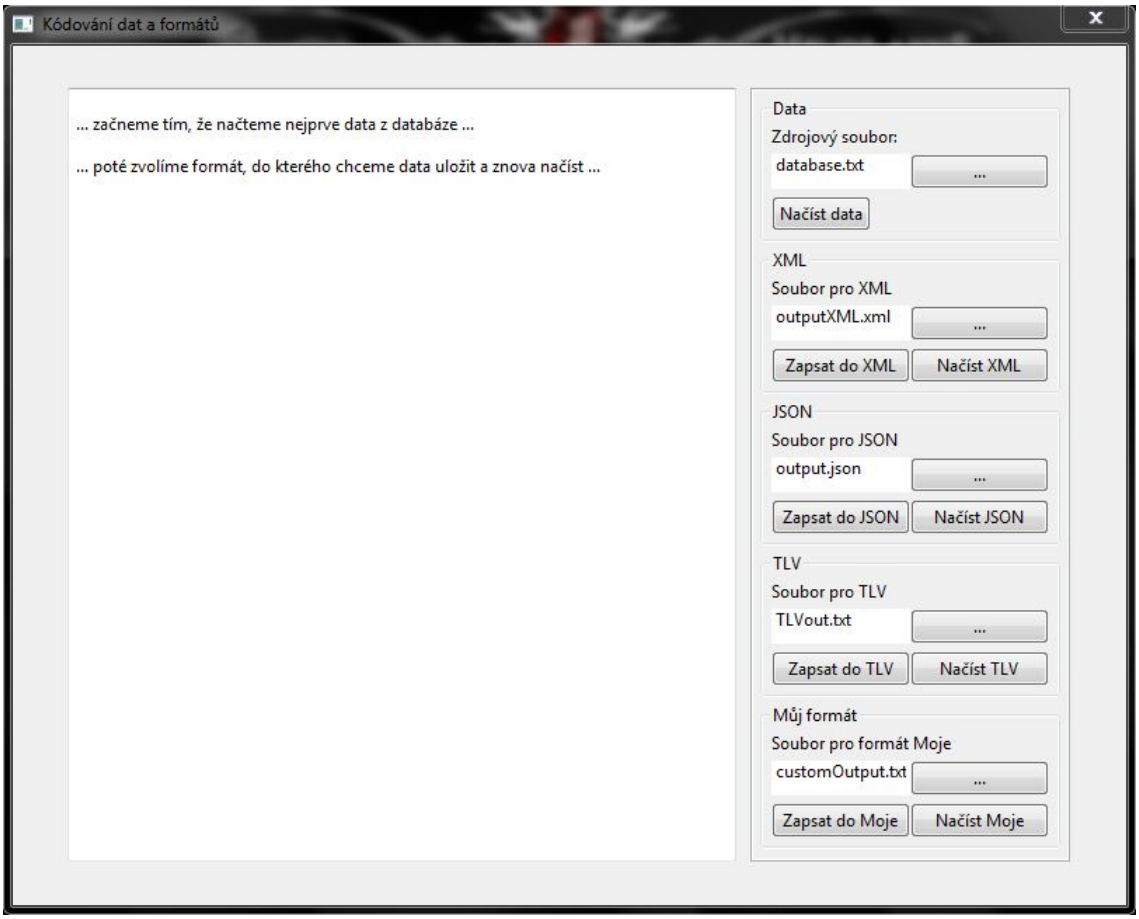
4.1 Vytvořený simulační program

V bakalářské práci bylo cílem popsat datové formáty, které je možné využít v serializaci a deserializaci dat a na základě těchto informací navrhnout a implementovat vlastní formát pro přenos dat. K těmto účelům byla vyvinuta aplikace, která se jmenuje DataCoding – Kódování dat a formátů.

Vytvořená aplikace je určená pro použití pod systémem Windows Seven, 32 bitový operační systém. Na jiných operačních systémech nebyla aplikace zkoušena, ale je předpoklad, že i na jiných systém bude aplikace plně funkční, jelikož je aplikace napsána v Javě z důvodu přenositelnosti na jiné operační systémy. Naprogramování bylo provedeno v programovacím jazyce Eclipse SDK, verze programovacího jazyku byla 3.6.1. Pro účely jednoduchého ovládání programu bylo vytvořeno prostředí GUI, pro pohodlné načítání dat, následného ukládání a výpisu požadovaných informací. Vytvořená aplikace DataCoding je spustitelná java aplikace z prostředí Eclipse. Aplikace je připravena na formu spustitelného souboru JAR (Java archiv), nejde však o klasický spustitelný soubor (přípona exe), k použití bez nutnosti mít nainstalované vývojové prostředí a knihovny. Pro bezproblémové spuštění z vývojového prostředí je požadována knihovna GUI „swt“, kterou lze zdarma stáhnout ze stránek pro podporu programovacího jazyka Eclipse. Je to jediná externí knihovna, která byla přidána. Vše ostatní je již součástí knihoven, které jsou již implementovány s instalací programovacího jazyka. K práci jsou přiloženy i zdrojové kódy ve formě vytvořeného projektu programovacího jazyka Eclipse.

Objasnění možností aplikace, které zobrazuje a nabízí několik možných formátů, již zmíněných a známých serializačních formátů a jeden vytvořený formát jako cíl této práce, pro uložení a znovu načtení dat a výpis informací do okna aplikace. Konkrétně jde o formáty XML serializace, JSON, binární serializace v podobě TLV a vlastní navržený serializační formát.

Zajisté je i příležitost načtení již předpřipraveného modelu dat ze souboru a jeho případné modifikace a uložení. Aplikace dokáže exportovat data jednotlivých formátů a zapsat jejich hodnoty do předem připravených souborů. Vzhled hlavního okna aplikace je zobrazen na Obrázek 4.1 Vzhled hlavního okna aplikace DataCoding – Kódování dat a formátů.



Obrázek 4.1 Vzhled hlavního okna aplikace DataCoding – Kódování dat a formátů

4.2 Popis rozhraní a ovládání

4.2.1 Základní struktura rozhraní

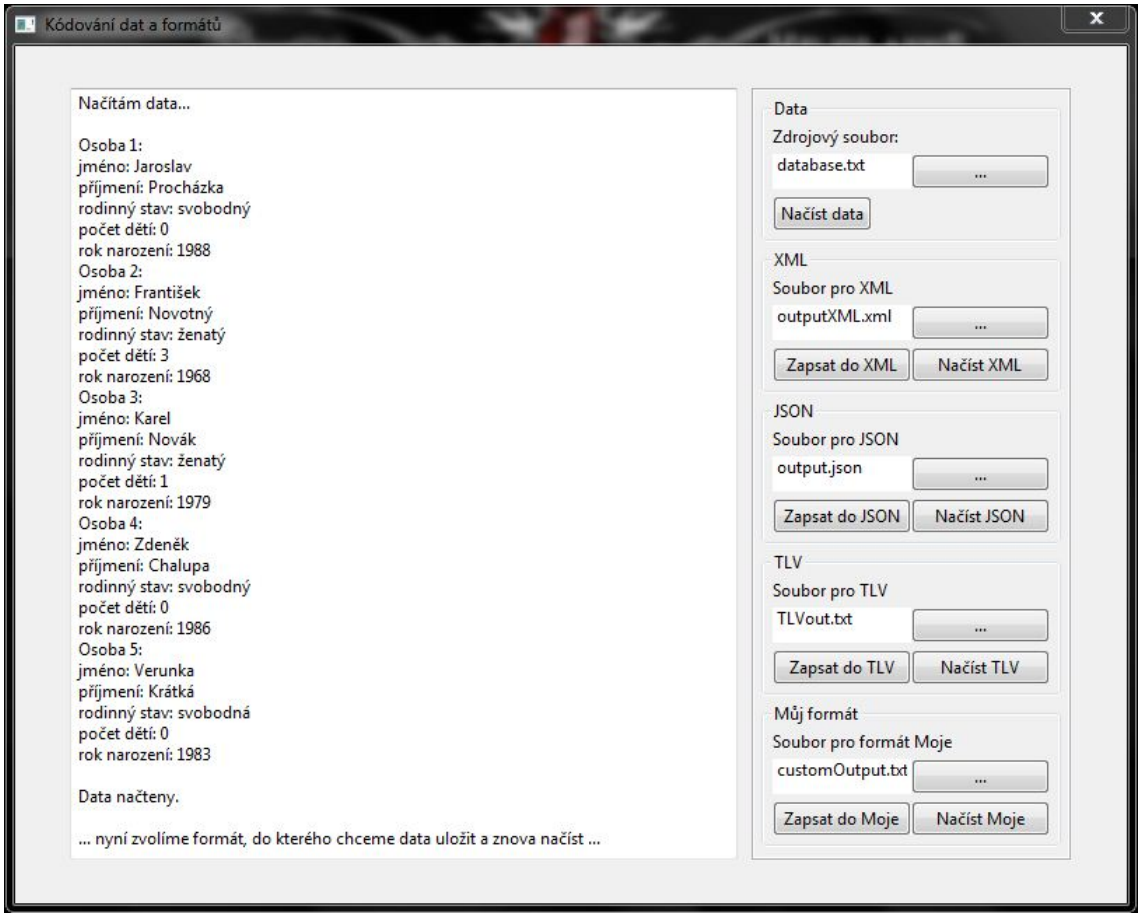
Aplikace je pro jednoduchost ovladatelná z jejího hlavního okna, zachycená na Obrázek 4.1 Vzhled hlavního okna aplikace DataCoding – Kódování dat a formátů. Znázorněné hlavní okno obsahuje několik tlačítek, které velmi efektivně zpřístupňují veškeré funkce aplikace. V pravém sloupci v okně *Data* obsahuje tlačítka pro výběr zdrojové databáze, ze které jsou čerpána data pro následné druhy serializace, a *Načíst data*, které načte hodnoty z vybraného souboru do levého výstupního okna aplikace.

Hlavní rozhraní doplňuje čtyři druhy formátů serializace. U každého okna je tlačítko pro výběr souboru, do kterého se následně data ukládají, nebo načítají zpět do výstupního okna. Dále je tu u každého jednotlivého formátu tlačítko pro zápis dat v uvedeném formátu. Pro formát dat XML je to tlačítko s funkcí – *Zapsat do XML*. Aplikace umožňuje zpětné načtení serializovaných dat, tedy deserializaci dat a to funkcí *Načíst XML*, pro konkrétní případ formátu XML. Tyto funkce jsou stejné pro všechny druhy formátů dat.

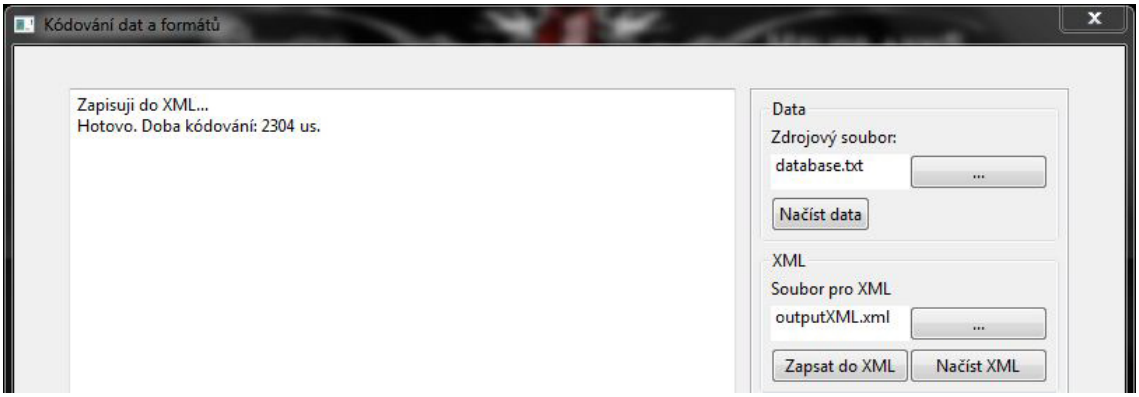
4.2.2 Načtení připraveného modelu dat

Pokud je databáze dat připravena a dostupná ze souboru, lze tuto databázi dat jednoduše načíst pomocí tlačítka *Načíst data* v okně *Data*. Pokud je vše v pořádku, načtená data ze zdrojového souboru se objeví ve výstupním okně nalevo. Pomocí tlačítka ve stejném okně, které je vedle názvu zdrojového souboru, lze po reakci v klasickém dialogovém okně vybrat zdrojový soubor s daty. Po zvolení požadovaného zdrojového souboru a odsouhlasení dialogu jsou data načtena a je možné s nimi dále pracovat. Je možné, pokud půjde o rozsáhlejší soubor dat a informací, že načtení dat může nějakou chvilku zabrat. Ovšem na výkonných sestavách by časová prodleva neměla být nikterak velká.

V situaci, kdy potřebný zdrojový soubor nemáme, vytvoříme si vlastní zdrojový soubor dat nebo si jej necháme vytvořit. Poté je postup následovný, vybereme pomocí dialogového okna vytvořený zdrojový soubor, potvrdíme a pomocí tlačítka načteme data. Data se nám načtou v okně nalevo a můžeme zkontrolovat správnost načtených dat a instrukce, která nás navádějí, jak dále postupovat s načtenými daty. Tyto popsané informace jsou dobře patrné z Obrázek 4.2 Načtení dat z databáze.



Obrázek 4.2 Načtení dat z databáze



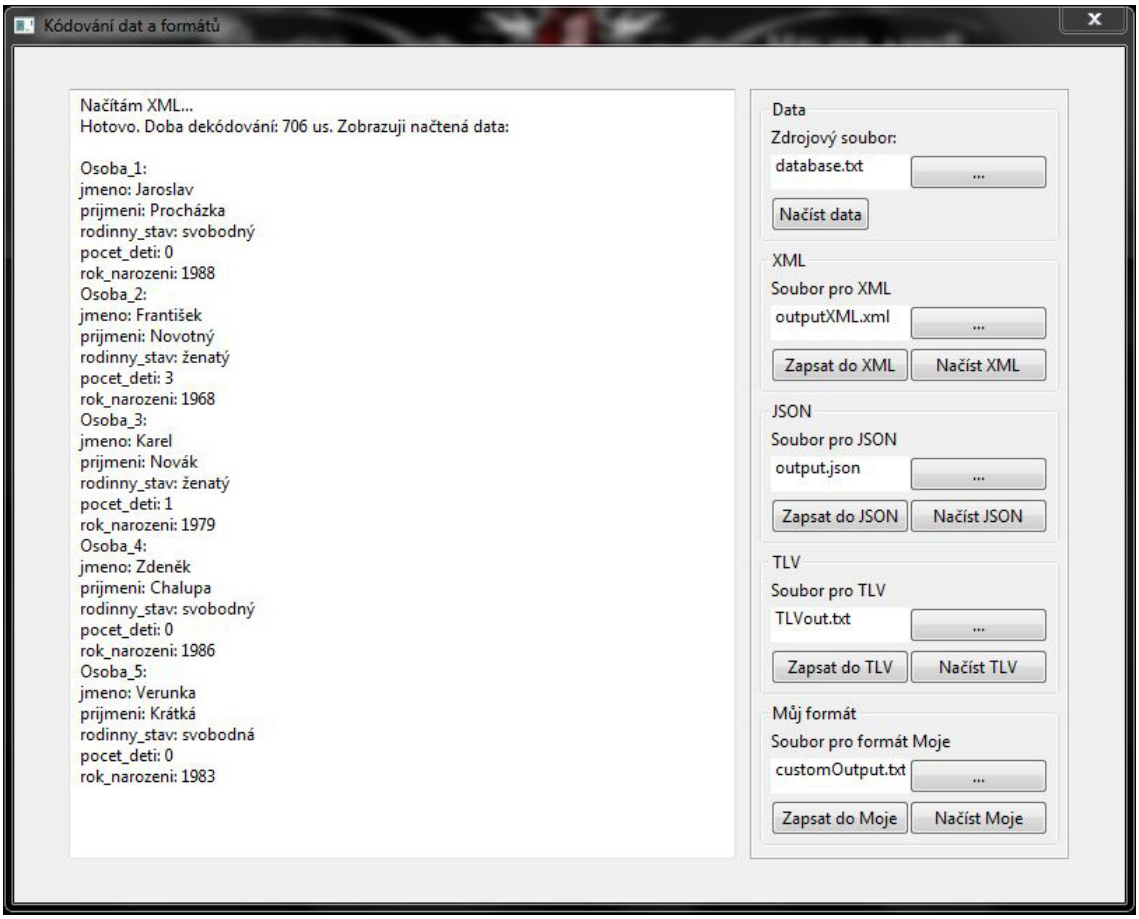
Obrázek 4.3 Zápis dat do formátu XML

4.2.3 Spuštění a ovládání aplikace

Aplikace dokáže s načtenými daty pracovat a serializovat data do čtyř druhů formátů, které si uživatel může zvolit z nabídky vpravo. Při zapsání do jakéhokoliv druhu formátu je poté zobrazeno potvrzení správně zakódovaného formátu a doba kódování, kterou tento proces zabral. Pro úplnost zobrazeno na Obrázek 4.3 Zápis dat do formátu XML. Opětovné načtení serializovaných dat se provede tlačítkem *Načíst ...* a zvolený stejný typ formátu dat, kterým se data zapsala. Ve výstupním okně se zobrazí potvrzení o deserializaci dat, správně načtená data a doba dekódování, kterou tento proces zabral. Výsledek je zobrazen na Obrázek 4.4 Načtená data formátu XML. Zobrazená data si poté můžeme zkontrolovat, zdali nedošlo k nějaké chybě při zapisování dat do patřičného formátu dat a její serializace. Deserializace dat je opětovné načtení serializovaných dat a tento proces převede zakódovaný tok informací na původní hodnoty a zobrazí ve výstupním okně původní hodnoty zapsané konkrétní metodou zápisu formátu. Tento postup serializace dat a následné deserializace dat se opakuje u všech typů serializačních formátů. Aplikace stále vrací do výstupního okna informace, potřebné k dalším postupům, či informace o provedené činnosti námi spuštěné.

Aplikace dokáže zachytit chyby způsobené například neexistujícím souborem a vyvolat upozornění na tyto chyby či omyly. Při načítání databáze se při chybě vyvolá chybové upozornění na stav aplikace. Pokud by došlo k chybě při zapisování dat ze čtyř volených formátů, je zobrazena chybová hláška, která upozorní na chybu a vypíše její plné znění. Stejná situace je i v případě, jde-li o proces deserializace, načítání dat, u všech dostupných formátů.

Výstupní soubor těchto metod je taktéž zobrazen u každého formátu a je možné si opět vybrat pomocí dialogového okna zvolený výstupní soubor. Ve výstupním souboru je po zapsání určité serializace dat forma kódování, kterou byl procesem zapsán do patřičného druhu serializace kódu.



Obrázek 4.4 Načtená data formátu XML

4.2.4 Uložení dat programu

Data v průběhu simulace programu se zapisují do souborů, které si navolíme. Soubor pro ukládání je přehledně vypsán v okně pod konkrétním názvem formátu serializace. Tlačítkem pro výběr souboru, do kterého zapisovat serializovaná data, je možné pouze pro konkrétní přípony k tomu určené. Nelze si tedy splést jiný soubor pro to určený. Pokud máme zvolená data vybrána a následně načtena do aplikace, která je vypíše do výstupního okna, můžeme přejít k následnému kroku a zapsat zvolená data do čtyř různých formátů serializace.

První druh formátu serializace v aplikaci je XML. Po zakódování dat v tomto formátu, je výsledná serializace zapsána do výstupního souboru. Ve výstupním souboru jsou data serializovaná podle druhu a zvolené serializace. XML je velmi známé a proto je zřejmé, jak by tento tok informací měl vypadat. Na Obrázek 4.5 Serializace XML - začátek a konec výpisu, je názorná ukázka, jak serializace XML vypadá a do jakého toku dat je zakódována původní struktura dat. V párovém tagu je vyjádřena první položka na pozici řádku dva. Tento tag je uzavřen uzavíracím tagem na osmnáctém řádku v zobrazeném souboru. V první položce jsou další atributy této položky. Prvek

obsahující atribut *jmeno* je uveden tagem na řádku třetím a mezi uzavíracím tagem, který je na pátém řádku, je hodnota tohoto prvního prvku, první položky. Druhý prvek je vytvořen úplně stejně identicky s tím rozdílem, že dochází ke změně hodnot v tomto druhém prvku, který je závislý na datech zapsaných v databázi zdrojových informací. Po naplnění první položky a následné uzavření této položky, obsahující atribut *Osoba_1*, bude na řadě druhá položka z databáze a soubor se bude serializací XML podobně plnit. Takto opakující se proces naplní celý výstupní soubor, vyjadřující serializaci dat XML.

```
1  <root>
2  <Osoba_1>
3  <jmeno>
4  Jaroslav
5  </jmeno>
6  <prijmeni>
7  Procházka
8  </prijmeni>
9  <rodinny_stav>
10 svobodný
11 </rodinny_stav>
12 <pocet_deti>
13 0
14 </pocet_deti>
15 <rok_narozeni>
16 1988
17 </rok_narozeni>
18 </Osoba_1>
19 <Osoba_2>
20 <jmeno>
21 František
22 </jmeno>
23 <prijmeni>
24 Novotný
25 </prijmeni>
```

```
64 0
65 </pocet_deti>
66 <rok_narozeni>
67 1986
68 </rok_narozeni>
69 </Osoba_4>
70 <Osoba_5>
71 <jmeno>
72 Verunka
73 </jmeno>
74 <prijmeni>
75 Krátká
76 </prijmeni>
77 <rodinny_stav>
78 svobodná
79 </rodinny_stav>
80 <pocet_deti>
81 0
82 </pocet_deti>
83 <rok_narozeni>
84 1983
85 </rok_narozeni>
86 </Osoba_5>
87 </root>
88
```

Obrázek 4.5 Serializace XML - začátek a konec výpisu

Dalším druhem formátu je serializace JSON. Tento druh serializace je velmi podobný výše uvedenému XML. Liší se pouze tím, že neobsahuje párové tagy a tudíž nemusí tak zdvojnásobovat svůj objem dat o tyto položky. Je i lépe čitelný a snadný v orientaci ve zdrojovém souboru. Výhodou této serializace je i to, že tento formát JSON je méně náročnější při strojovém generovaní kódu. Pro lepší znázornění, jak tento výsledný serializační kód vypadá, je zde Obrázek 4.6 Serializace JSON. Tento serializační formát se místo tagů vkládá do složených závorek a jednotlivé atributy jsou vloženy do uvozovek. Složené závorky jsou mnohem lépe přenositelné v kódu a mnohem lépe strojově generovatelné a navíc se neztrácí přehlednost v kódování serializace. Celý proces začíná vložením první levé složené závorky a po ní je vložena první položka z databáze do uvozovek. Hodnota první položky v uvozovkách je oddělena interpunkčním znaménkem dvojtečka a opět vložena levá složená závorka pro vstup

prvního prvku. Znázorněný první prvek a jeho hodnota je uložena do uvozovek. Hodnota prvního prvku je oddělena interpunkčním znaménkem dvojtečky a také vložena mezi dvoje uvozovky. Řádek je ukončen znaménkem čárky znázorňující konec prvního prvku a jeho hodnoty. Na dalším řádku může následovat druhý prvek s hodnotou ve stejné struktuře, jako tomu bylo u prvního prvku v první položce. Všechny tyto prvky jsou následně ukončeny znaménkem čárka a to i první položka. Na Obrázek 4.6 Serializace JSON, je vidět na osmém řádku uzavírající první položku pravou složenou závorkou a po pravé složené závorce následuje interpunkční znaménko čárka. Následný proces aplikace vloží další položku z databáze, pokud nějakou položku ještě obsahuje zdrojová databáze dat. Postup strukturovaného zápisu v případě druhé položky serializace dat JSON je stejný jako v první položce souboru. Celý soubor je tedy ukončen dvěma pravými složenými závorkami. Na konci již není interpunkční znaménko čárka, jelikož již nenásleduje žádná položka a tudíž je proces ukončen a uzavřen.

```
1  {
2  "Osoba 1":{
3  "jméno":"Jaroslav",
4  "příjmení":"Procházka",
5  "rodinný stav":"svobodný",
6  "počet dětí":"0",
7  "rok narození":"1988"
8  },
9  "Osoba 2":{
10 "jméno":"František",
11 "příjmení":"Novotný",
12 "rodinný stav":"ženatý",
13 "počet dětí":"3",
14 "rok narození":"1968"
15 },
```

Obrázek 4.6 Serializace JSON

Třetím druhem serializace v aplikaci je číslicová serializace typu TLV. TLV postrádá veškerou nadbytečnou textovou výplň a je vše řešeno pomocí číselného kódování. Serializace pomocí TLV je zobrazena na Obrázek 4.7 Serializace TLV. Je zde patrné, že zde již nejsou vůbec žádné párové tagy i jiné textové hodnoty atributů. Vše je překládáno do číselného formátu, kde první číslo bloku značí položku. Druhé číslo vyjadřuje celkový počet proměnných, kterých se v první položce vyskytuje. Za druhým číslem následuje již první hodnota požadované první položky v této struktuře, kde číslo za touto první hodnotou vyjadřuje vnitřní počet proměnných hodnot v této první hodnotě položky. Takto jsou dále po sobě vkládány další hodnoty první položky a stejně generovány jejich proměnné. Druhá položka je oddělena dvojitým interpunkčním znakem – lomítko (/). Následná struktura zakódovaného toku informací je opakující. Aplikace provede zápis druhé položky a následuje celkový počet proměnných v této druhé položce. Poté následuje stejný sled operací, jako tomu bylo u první položky a to takových, že bude naplněna první hodnota druhé položky a vypsán její počet vnitřních proměnných hodnot. Aplikace pokračuje takto dále na konec, až dojde k serializaci celé

zdrojové databáze dat tímto stylem.

```
1/45/1/8/Jaroslav/2/9/Procházka/3/8/svobodný/4/1/0/5/4/19
88//2/42/1/9/František/2/7/Novotný/3/6/ženatý/4/1/3/5/4/1
968//3/36/1/5/Karel/2/5/Novák/3/6/ženatý/4/1/1/5/4/1979//
4/41/1/6/Zdeněk/2/7/Chalupa/3/8/svobodný/4/1/0/5/4/1986//
5/41/1/7/Verunka/2/6/Krátká/3/8/svobodná/4/1/0/5/4/1983
```

Obrázek 4.7 Serializace TLV

Posledním druhem serializace v uvedené aplikaci je mé vlastní navržené řešení daného úkolu bakalářské práce. Výpis dat ze serializace mým navrženým zakódováním je zobrazen na Obrázek 4.8 Navržená serializace dat. Jedná se o strukturovaně stejnou vazbu jako je formát TLV. Rozdíl v těchto dvou serializací je ten, že mnou navržená serializace se liší zápisem hodnot v přesném zadání a nikterak se nepřevádí do číselného kódování. Dalším rozdílem je nevypisování nadbytečného počtu proměnných, které následující položka nebo prvek této položky vyjadřuje. Od serializace XML se tento druh liší zcela znatelně a ve všem. Nevyskytují se tu párové tagy jako takové u serializace XML. V mém vyjádření vlastního řešení jsou velmi efektivně zjednodušeny tyto prvky a to vede k optimálnější struktuře dat i jejího objemu. Po stránce strojové je tento serializační formát kódu méně náročný z hlediska počítačového zpracování. Jediným podobným prvkem jsou opět obě složené závorky, které jsou použity u serializačního formátu JSON. Tyto složené závorky slouží k vymezení začátku a konce každé položky, která se vyskytne zapsaná v databázi. Levá složená závorka značí začátek atributu položky, jednotlivého prvku položky a hodnoty daného prvku v položce. Pravá složená závorka ukončuje hodnotu daného prvku. Jednotlivé položky této serializace jsou taktéž uzavřeny pravou složenou závorkou. Při serializaci dat do tohoto formátu je číselně zobrazena první položka z databáze. Po tomto kroku následuje naplnění první položky příslušnými prvky spojené s první položkou dat. Tyto prvky se číselně vyjadřují od jedničky až po maximální hodnotu dané položky. Po vypsání prvního čísla prvního prvku je ve složených závorkách uvedena hodnota tohoto prvku. Uzavření hodnoty prvku nastane pravou složenou závorkou, ale pokud daná položka obsahuje další prvky, číselná hodnota předešlého prvku se inkrementuje. Proces poté opět vloží do složených závorek přesně danou hodnotu prvku a uzavře. Takto serializační kód pokračuje, dokud z databáze nevyčerpá veškerá data. K uzavření celého řetězce dojde taktéž pravou složenou závorkou. Dojdeme tak v závěru, že se tímto serializačním formátem přispělo k dobrému výsledku.

```
{1{1{Jaroslav}2{Procházka}3{svobodný}4{0}5
{1988}}2{1{František}2{Novotný}3{ženatý}4{3}5
{1968}}3{1{Karel}2{Novák}3{ženatý}4{1}5{1979}}
4{1{Zdeněk}2{Chalupa}3{svobodný}4{0}5{1986}}5
{1{Verunka}2{Krátká}3{svobodná}4{0}5{1983}}}
```

Obrázek 4.8 Navržená serializace dat

4.3 Přehled struktury programu

Zdrojový kód programu je rozvržen do několika hlavních částí, neboli tříd programovacího jazyka. Všechny třídy programu jsou v hlavním balíčku, který nese název *cz.dataCoding*. V tomto hlavním balíčku jsou obsaženy všechny důležité třídy ke správné funkci programu. Jednotlivé názvy důležitých tříd:

- *DataCoding* – hlavní třída projektu, zde je hlavní aplikační logika
- *DataProvider* – stará se o načítání dat ze zdrojového souboru
- *DataStructure* – třída prvků stromu
- *DataToXML* – převede data do formátu XML a zápis do souboru
- *DataToJSON* – převede data do formátu JSON a zápis do souboru
- *DataToTLV* – převede data do formátu TLV a zápis do souboru
- *DataToCustom* – převede data do vlastního formátu a zápis do souboru
- *escapeType* – escape sekvence povolených znaků

Programovací jazyk Eclipse SDK, ve kterém je kód napsán, již vše potřebné měl, tedy programovací jazyk, implementováno a kromě grafického rozhraní GUI nebylo potřeba žádné další knihovny přidávat.

Knihovnu s GUI je potřeba stáhnout ze stránek[1] přímo pro podporu programovacího jazyka Eclipse. Po stažení příslušné knihovny *swt* pro konkrétní typ operačního systému už stačí pouze tuto knihovnu nahrát do programovacího jazyka Eclipse, nikoliv do projektu vytvořené aplikace.

4.3.1 Knihovna swt

Grafické rozhraní aplikace umožňuje mnohem lepší orientaci a snadnější ovládání požadovaných serializačních metod. Knihovnu *swt* po stažení nijak neupravujeme a ani nemusíme rozbalovat z archivačního souboru. Možností je také rozbalit archiv souboru a poté nahrát knihovnu do Eclipse. Stačí pouze otevřít programovací jazyk Eclipse SDK. Programovací jazyk Eclipse je v anglickém jazyce, proto jsou uvedeny korektně přesné názvy položek menu a rozhraní v anglickém jazyce. Po otevření programovacího rozhraní zvolíme z hlavní nabídky hned první rolovací menu *File* a v tomto menu najdeme položku *Import …* a zvolíme tuto položku. Otevře se klasické dialogové okno, ve kterém zvolíme *Existing Projects into Workspace* ve složce *General* a potvrdíme tlačítkem *Next*. Nyní máme na výběr ze dvou možností.

První možností je vybrat si *Select root directory*, kde si poté vybere z dialogového okna grafickou knihovnu *swt*. V této první možnosti musí být zvolena knihovna *swt*, která byla rozbalena z archivu souboru. Potvrdíme dialogové okno s vybranou knihovnou. Poté programovací jazyk Eclipse zkontroluje, zdali je vše v pořádku a pokud ano, dovolí knihovnu *swt* přidat a potom odklikneme finální tlačítko *Finish*.

Druhou možností je zvolit si z možné nabídky *Select archive file* a zvolíme staženou knihovnu *swt* v archivu a potvrdíme dialogové okno s vybranou knihovnou. Dále proběhne kontrola knihovny, jeli vše v pořádku, programovací jazyk Eclipse dovolí knihovnu *swt* přidat a pak zvolíme finální tlačítko *Finish* k provedení příkazu.

[1] http://www.eclipse.org/swt/

Grafická knihovna *swt* se poté tváří jako jeden z možných projektů a je i vidět v liště s vytvořenými projekty, pokud jich je více. Tato jednoduchá úprava stačí, aby fungovalo bezchybně grafické prostředí GUI pro aplikaci DataCoding – Kódování dat a formátů.

4.4 Charakteristika vybraných tříd

Důležité třídy tvořící aplikaci byly vypsány. Nyní konkrétní popis jednotlivých tříd, které byly použity.

Hlavní třída projektu **DataCoding**. Zde je použita hlavní aplikační logika aplikace. V této hlavní třídě je nadefinováno grafické rozhraní aplikace, které je potom patrné při spuštění aplikace. Jsou zde definována grafická pole, vytyčená pro všechny čtyři druhy serializací. V každé skupině serializačního formátu je struktura stejná a jde pouze o grafický vzhled aplikace. Tento grafický vzhled definuje popisek jednotlivých formátů serializace. Textbox se jménem souboru, do kterého se data zapisují. Tlačítko pro změnu souboru, do kterého se data zapíší. Jsou zde také definovaná tlačítka pro načtení souboru a uložení do souboru. Toto celé ještě upřesňuje oblast, rozmístění jednotlivého rozhraní aplikace. Všechny tyto tlačítka a textboxy jsou definované stejně, pouze se mění hodnoty pro konkrétní druh serializace.Třída je typu *final*, aby se nedala dále dědit. Kořenový prvek těchto dat je typu *DataStructure*, je to vlastní nadefinovaná třída. Atribut *private* omezuje přístup k této proměnné, není vhodné, aby s touto proměnnou někdo operoval z vnějšího prostředí. Dále je v této třídě metoda, která navrací referenci na náš kořenový prvek. Tím umožní dostat naše data do vnějších tříd. Zároveň nám nikdo nemůže naše data přepsat, pokud sami nechceme. Potom je volána metoda *readFile()*, třídy *DataProvider*, a její návratová hodnota je přiřazena do *root*. Dojde k přepisu našich dat, ale momentálně na začátku procesu přepsat data chceme, protože zatím žádné nemáme. Poslední, co se v této třídě vyskytuje, je escape sekvence. Jedná se tedy o zobrazení parazitních znaků, které se v určitém řetězci znaků mohou vyskytovat, na znaky povolené.

Další významnou třídou projektu je **DataProvider**. Tato třída se stará o načítání dat ze zdrojového souboru. Načítání naší zvolené *database.txt* provádí soukromá proměnná *databasefile* a určuje cestu k tomuto souboru. Obsahuje dále metodu *readFile*. Metoda *readFile* načítá data ze zadaného souboru a parsuje tyto data podle určeného formátu. Parsování se dá vysvětlit následovně. Dostaneme vstupní řetězec dat programu, která obsahují zakódované informace. Tyto vstupní řetězce, které jsou naplněny informacemi, je třeba rozložit do patřičných proměnných. Parsováním z takto zakódovaných informací dostaneme potřebné hodnoty do proměnných. Metoda *readFile* ukládá tyto data do stromové struktury definované třídou *DataStructure*. Navrací kořenový prvek, kterému data podléhají. Proměnná typu *BufferedReader* ukládá referenci souboru, vytvořený nový kořenový prvek proměnnou *data* používá jako referenci na prvky stromu. Důležitou částí je zde blok zachycení jakékoliv chyby. Program se pokusí provést blok *try* pokud nastane a program vyhodnotí výjimku (například v důsledku nějaké chyby, události nebo neplatné operace), pak se provádění zastaví a přeskočí se do bloku *catch*. Výjimka většinou značí, že se stalo něco, co se stát nemělo (příkladem může být nenalezení souboru s daty), a proto vykonávání dalších příkazů, které jsou na tomto závislé, by vedlo k dalším a často vážným chybám. Z toho

důvodu jsou přeskočeny. Všechny výjimky mají svůj typ, který určuje, co jsou zač. Všechny dědí vlastnosti od třídy *Exception* – pod tuto třídu spadají všechny výjimky. Zachycení výjimky se provede pro patřičnou výjimku pouze jednou, a to je vždy první blok *catch*, kterému výjimka typově vyhovuje. Například výjimka neexistujícího souboru spadá pod *IOException*, ale i pod *Exception*, proto platí pravidlo, že konkrétnější bloky *catch* jsou uvedeny první. Tím je zajištěno, že výjimka neexistujícího souboru bude odchycena blokem *IOException*. Ostatní výjimky, které nespadají pod I/O odchytí *Exception*.

Třída projektu **DataStructure** je třída prvků stromu. Obsahuje *DataStructure parent* – reference na rodičovský (nadřazený) prvek, dále je to název tagu *String name*, hodnota obsažená v tagu *String value* a v poslední řadě to je seznam referencí na vnořené (dceřiné) prvky – tagy. Bez-parametrický konstruktor, musí být tento konstruktor definován, protože definuje parametrické konstruktory. Parametrickým konstruktorem je nastavena reference na rodiče do proměnné instance *parent*, je nastaveno jméno a zbytek bude vynulován. Tento parametrický konstruktor je používán při vytváření potomku. Dále nastavuje pouze referenci na rodičovský prvek. Použit je v případě, když název prvku není znám v době vytváření. Pokud je při vytváření potomka jeho název i hodnota známa, pak se použije tento konstruktor – *DataStructure(DataStructure parent, String name, String value)*. Metoda *getName()* nám vrací obsah soukromé proměnné *name*, což je název prvku. Metoda setName() umožní změnit hodnotu soukromé proměnné *name* z vnějšku. Parametr *String getValue()* je obdoba metody *getName()*, vrací řetězec s hodnotou, kterou obsahuje *value*. Metoda *DataStructure addChild(String name)* přijme název nového prvku v parametru *name* a vytvoří potomka použitím parametrického konstruktoru. Potomci jsou sdruženi v lineárním seznamu. Následuje metoda *DataStructure addChild(String name, String value)*. Tato metoda je obdoba předchozí metody pomocí tzv. přetížení. Pokud je znám jak název, tak i hodnota nového prvku, tak použijeme tuto metodu, která správně přidá nového potomka s použitím adekvátního konstruktoru. Metodou *hasChildren()* zjistí, jestli má prvek potomky. *LinkedList<DataStructure> getChildren()* touto metodou z vnějšku získá referenci na seznam potomků určitého prvku. Poslední metoda v této třídě vrací referenci na rodičovský prvek a proměnná *parent* je soukromá, znemožnění manipulace s proměnou.

Třída projektu **DataToXML** je třídou, která převede data do formátu XML a zapisuje do souboru. Soukromá proměnná *xmlFilename* určuje cestu k souboru, kam se budou data zapisovat. Proměnná *output* ukládá referenci na výstupní datový soubor. Metoda *write(DataStructure)* v parametru přijme kořenový prvek datového stromu. Poté si vytvoří iterátor nad jeho potomky a tyto postupně vypisuje metodou *inOrderWrite*. Dochází zde k měření doby, po kterou probíhalo kódovaní. Metoda *preOrderWrite(DataStructure data)* rekurzivně vypíše stromové struktury metodou pre-order. Principiálně vstupní parametr je prvek stromu, vypíše se jeho jméno mezi < a > (otvírací tag). Zkontroluje, jestli prvek má vnořené prvky (tagy), pokud ano, vytvoříme iterátor nad těmito prvky a rekurzivně je zavolána tato metoda a předává jí referenci na potomka (vnořený prvek). Pokud prvek nemá potomky, vypíše se jeho hodnota. Pokud prvek má potomky a po ní následuje uzavírací tag, tato metoda končí a navrací se z rekurze, pokud k ní došlo. Metoda *read()* načte specifikovaný soubor cestou *xmlFilename* do datového stromu a vrátí referenci na kořenový prvek tohoto stromu. Přitom měří čas načítaní a převodu. Metoda *setOutputFile()* převezme řetězec obdržený

v parametru *filepath* a referenci na něj uloží do členské proměnné *xmlFilename*. Tato metoda umožní změnit proměnnou *xmlFilename* z vnějšku, přestože k ní není přístup. Následují poslední dvě metody této třídy a to metoda nastavující výstupní soubor a metoda vracející celkový čas serializace/deserializace.

Třída projektu **DataToJSON** a **DataToTLV**. Jedná se prakticky o stejnou převodní třídu jako u XML. Jsou zde pozměněny soukromé proměnné *jsonFilename*, která určuje cestu k souboru, kde se data budou ukládat. Pozměněná logika struktury zápisu serializace a deserializace dat pro daný formát. Struktura třídy jako taková je zachována. Proměnná *output* ukládá referenci na výstupní datový soubor, taktéž jako u formátu XML. Podobně jsou na tom i metody *write(DataStructure)*, *preOrderWrite(DataStructure data)*, *read()*, *setOutputFile()*, ve kterých se mění proměnné a struktura logiky převádění serializace a následné deserializace na základě daného formátu.

Třída projektu **DataToCustom** je vlastní navržený kód serializačního formátu. Soukromá proměnná tady v této třídě je *customFilename*, určující cestu k souboru, kde jsou data zapsána. Taktéž i tady proměnná *output*, odkazuje na výstupní datový soubor. Metoda v tomto druhu serializace, *write(DataStructure root)*, přijme kořenový prvek datového stromu, měří počáteční a koncový čas. Otevře výstupní soubor a uloží počáteční čas v nanosekundách. Poté předá referenci metodě root *preOrderWrite()*. Po skončení uloží aktuální čas, provede rozdíl počátečního a koncového času a vypíše její hodnotu. Tímto uzavře výstup. Dále provádí zachycení všech výjimek, případně vypíše chybovou hlášku se zněním obsažené chyby. V Metodě *preOrderWrite* vpisuje do souboru, takže může dojít k chybě při zápisu do souboru, proto je vše v bloku *try*. Pokud má prvek potomky, vypíše otevírací tag. Vytvoříme iterátor nad těmito potomky a poté prochází jednotlivé potomky. Obsahuje pomocnou referenci na aktuálního potomka. Rekurzivně volá metodu s referencí na potomka a nakonec vypíše uzavírací tag. Pokud prvek nemá potomky, pak obsahuje nejspíše hodnotu a vypíše tuto hodnotu. Nakonec v této metodě dojde k flushnutí, neboli k uložení výstupu. Jistota, aby proud výstupu byl zapsán do souboru a nebyl pouze jen v zásobníku paměti. Jako poslední se provede zachycení výjimek, zjednodušeně zachytí všechny výjimky jedním blokem. V metodě *read()* proměnná ukládá referenci na vstupní soubor, poté spustí počáteční a koncový čas. Otevře vstupní soubor a vytvoří načítací zásobník typu *StringBuffer*. Deklaruje potřebné proměnné a uloží počáteční čas. Dále načítá znaky ze souboru. Pokud načte znak (\) a není nastaven *escapedChar*, pak ho nastaví. Jestli je další znak konec souboru, program skončí, poněvadž je soubor nejspíše chybný. V jiném případě resetuje *escapedChar*. Jeli nastavení *storeData* na hodnotu true, tak program uloží data do *readBufferu*. Nyní vybere stav procesu a načítá na začátku znaky. Načetl-li znak ({) přejde na další stav a vytvoří kořenový prvek. Poté zvýší počet otevřených prvků a následuje-li stav *Item* a načtený znak ({), bude přidávat nového potomka. Pokud už prvek potomky obsahuje, dojde k načtení jejich počtu a přičte jedničku. Tímto získá pořadové číslo nového potomka, kterého může vytvořit. Jinak vytvoří s pořadovým číslem jedna. Nyní předáme referenci na nového potomka. Začne ukládat data a načítat hodnotu. V této části se smaže zásobník a přejde na stav *Value*, kde dojde ke zvýšení počtu otevřených prvků. Jestli ve stavu *Item* načte znak (}), tak přejde na rodiče, pokud může a sníží počet otevřených prvků. Načte-li znak (}) ve stavu *Value*, tak právě načetl hodnotu a má dojít k uzavření potomka. Program přestane ukládat data, přejde do stavu *Item* a nastaví hodnotu, kterou načetl. Jeli to možné, přejde na rodiče a dojde ke snížení

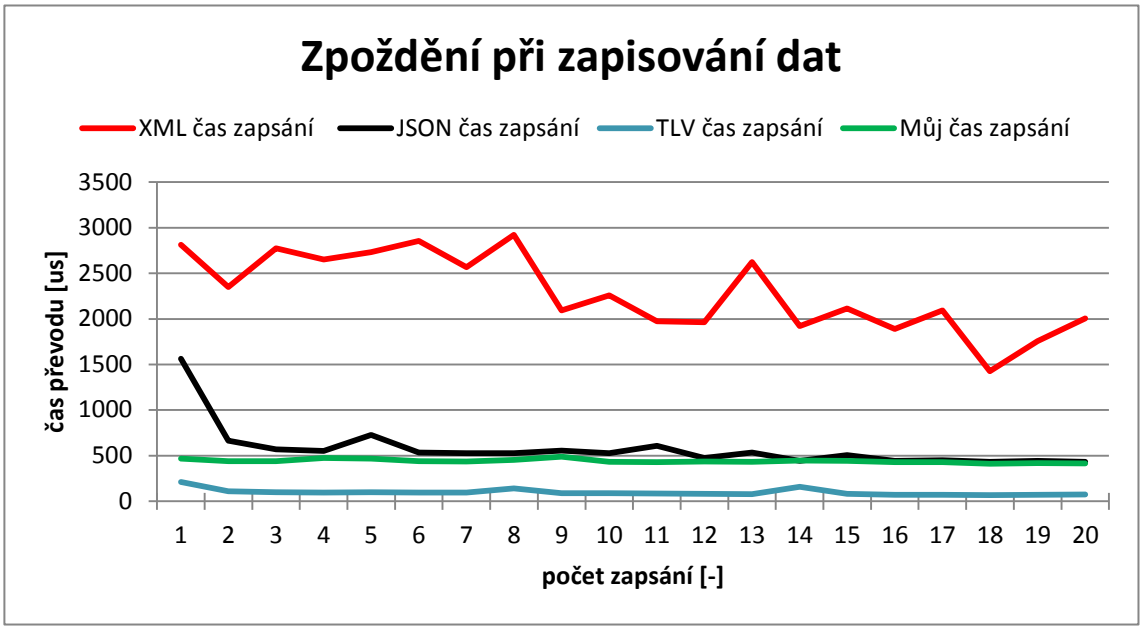
počtu otevřených prvků. Ovšem pokud načetl znak ({), znamená to další vnořený prvek a bude ho ukládat do dat. V tomto případě zjistí, jestli už prvek obsahuje potomky, načte jejich počet, přičte k tomuto počtu číslo jedna a dostane pořadové číslo nového potomka, který může vytvořit. Poté předá referenci na nového potomka a dojde ke zvýšení počtu otevřených prvků. Skončí-li proces, načte konečný čas a následně vypočítá celkový čas. Opět jsou tu zachytávané případné výjimky, a pokud došlo k nějaké chybě, nebo byl soubor poškozen/nekompletní, tak přechází na nejvyšší úroveň a vrátí kořenový prvek. Taktéž obsahuje metody nastavující výstupní soubor a vracející výstupní soubor a poslední metodu vracející celkový čas serializace a deserializace toho druhu formátu.

5 PROVEDENÉ SIMULACE

5.1 Časová náročnost formátů

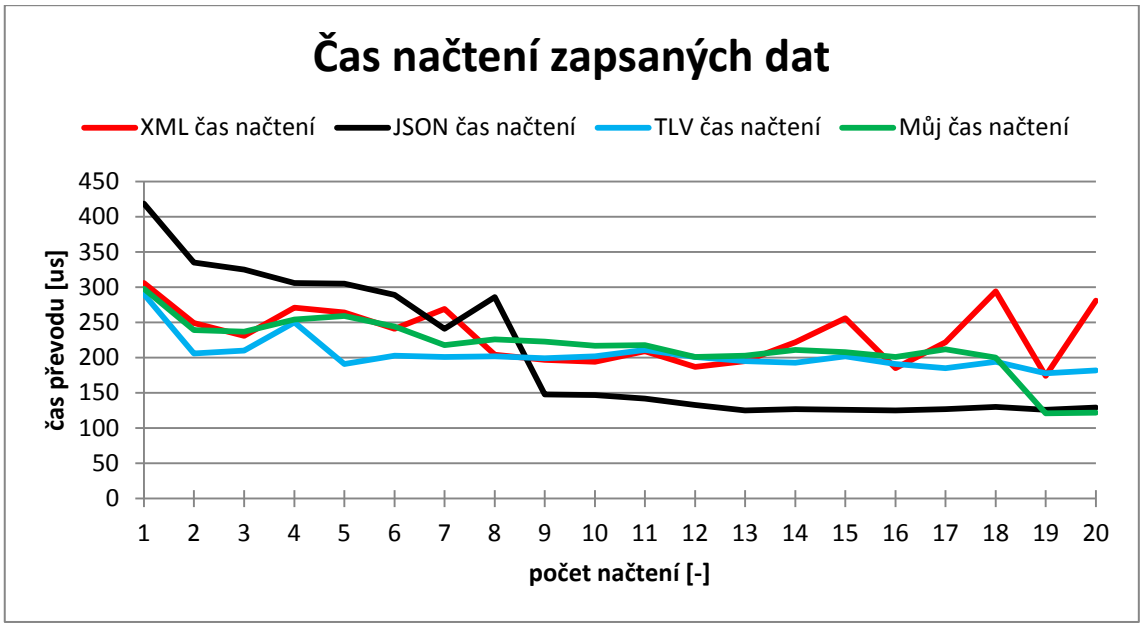
V rámci bakalářské práce byly provedeny simulace některých serializačních formátů. Těchto formátů bylo celkem čtyři. Více by ani nebylo vhodné, vyvinutá aplikace vyžadovala odladění použitých formátů a následné ovládání v aplikaci.

Následující shrnutí se týká simulací pro časovou náročnost serializace a deserializace jednotlivých formátů. Pro porovnání byly provedeny simulace pro všechny čtyři druhy formátů. Vyvinutá aplikace, na které se simulace prováděly, potvrdila fakta o časové náročnosti serializačního formátu XML, který se jeví jako nejvíce náročný z hlediska zápisu dat do formátu XML. Tento formát je i z hlediska časové prodlevy náročnější při načítání dat z formátu, než u jiných formátů serializace. Naopak tomu byl formát TLV, který z hlediska časové náročnosti byl ve všech ohledech nejméně náročný. TLV mělo nejkratší dobu zápisu dat do serializačního formátu i zpětná deserializace měla nejkratší časovou prodlevu. Je to i logické, poněvadž formát XML obsahuje velmi mnoho textových, párových tagů a to vše je velmi náročné převádět a zpětně obnovit do původního stavu. Navíc TLV formát je velmi elegantně upraven pro tyto účely, a proto není divu, že z těchto simulací na časovou náročnost vyšel jako jeden z nejlepších formátů. Průměrného výsledku dosáhl formát typu JSON. Tento formát serializace měl mnohem lepší časovou závislost, než je tomu u podobného formátu XML, formáty jsou velmi sobě podobné. Formát JSON měl v průběhu simulace zhruba čtvrtinovou časovou náročnost jako formát XML u zápisu dat do souboru. Při porovnání načítání dat ze souboru byl už velmi výkonný a byl téměř shodný s výsledkem formátu TLV. I tento formát serializace JSON byl rychlejší v načtených datech oproti formátu XML. Konečnému porovnání časové náročnosti serializace dat byl podroben analýze formát mého navržení. Snaha toho formátu byla, aby byla zachována velmi dobrá čitelnost kódu serializace a snadná orientace v datech. Pro tyto podmínky musel být formát zároveň časově nenáročný při překladu a znovu načtení dat ze souboru. Doba, za kterou aplikace převede serializaci dat mého formátu, je v těchto výsledcích druhá nejlepší. Rozdíl zápisu dat do souboru je celkem znatelný vůči dvěma textovým formátům serializace, s kterými byl porovnán. Jsou myšleny formáty XML a JSON. Ovšem jednoduchost serializace formátu TLV nebyla překonána. Časová závislost se s tímto formátem nemůže rovnat, jelikož TLV je formátem číselným a je zde snazší překlad serializace. V případě deserializace je navržený kód formátu průměrný a dosahuje ve výsledcích středních hodnot z hlediska časové náročnosti. Všechny zhodnocené formáty lze vidět na Obrázek 5.1 Časová závislost zapsaných dat, kde jsou zobrazeny časové závislosti serializace s porovnáním všech čtyřech formátů. Algoritmus programu je postupně provedl s cílem prozkoumat časovou závislost serializačních formátů podle zadaných parametrů. Jako optimální hodnota pro opakování simulace byl stanoven počet opakování jednotlivého formátu serializace na dvacet opakování. Toto nastavení je pouze optimální na těchto uměle vytvořených podmínkách. V reálných sítí při přenosu mohou být optimální podmínky odlišné.



Obrázek 5.1 Časová závislost zapsaných dat

Další graf znázorňuje průběh časové závislosti načítání dat ze souboru. Obrázek 5.1 Časová závislost zapsaných dat. Deserializace a provedení časové závislosti bylo provedeno u všech formátů. V grafu jsou porovnány hodnoty dosažené při simulaci deserializace ve vyvinuté aplikaci.



Obrázek 5.2 Časová závislost načtených dat

5.2 Datová náročnost formátů

Výstupní soubory jednotlivých formátů jsou pozorovány i z hlediska datové náročnosti. V tomto případě byly použity náhodné vstupní hodnoty databáze pro účely simulace. Avšak tyto zvolené vstupní hodnoty byly stejné po celou dobu simulace u daných formátů. Vlastní zpracování bylo provedeno vždy jednotlivým formátem a poté byla data uložena do výstupního souboru. Velikosti souboru byly zjištěny integrovaným průzkumníkem ve Windows Seven. Vstupní soubor měl velikost **541** bajtů.

Velikost výstupního souboru serializace formátu XML pro zvolené vstupní hodnoty dosahovala velikosti **1007** bajtů.

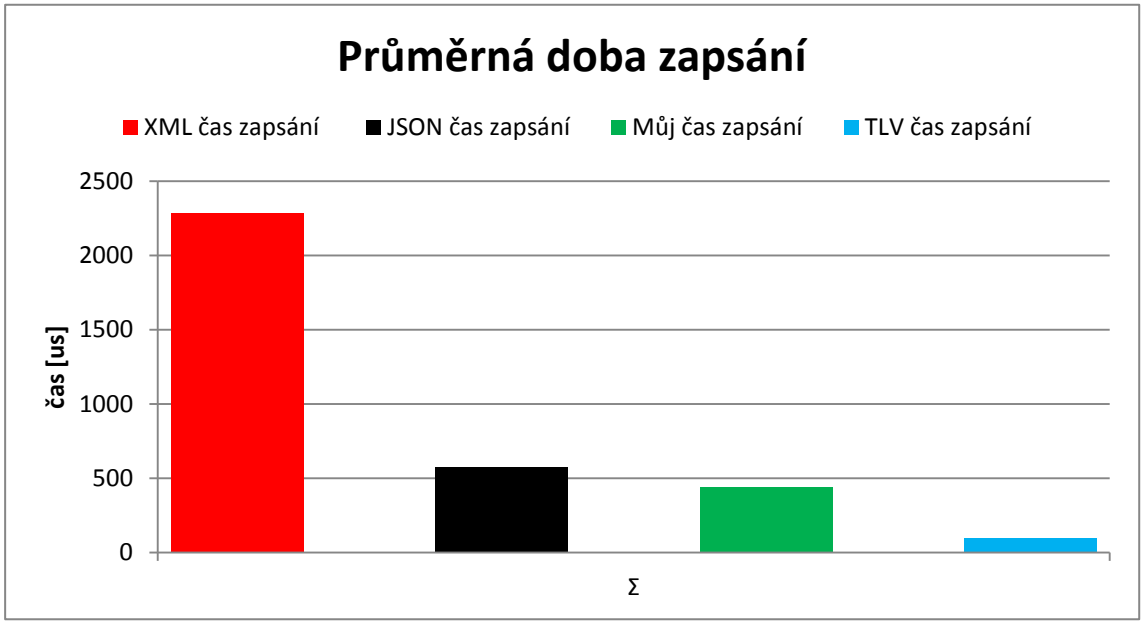
Velikost u formátu JSON výstupního souboru byla **648** bajtů.

Formát TLV zabíral pouze velikost **283** bajtů ve výstupním souboru.

Navržený formát zabíral se stejnými vstupními daty pouhých **222** bajtů.

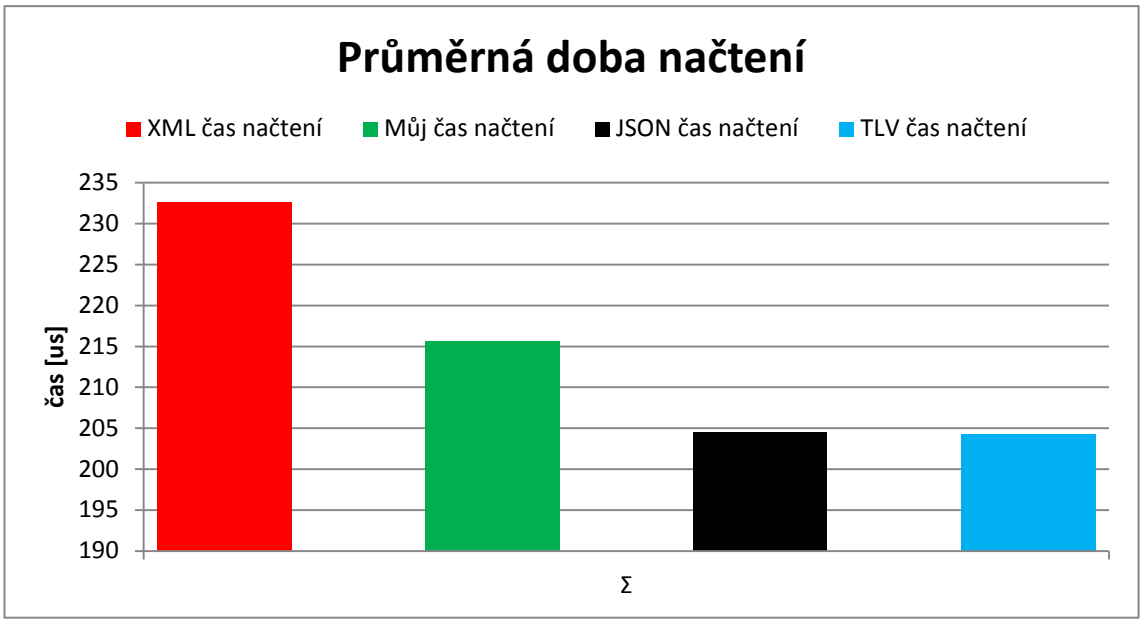
5.3 Hodnocení formátů

Simulované časové závislosti všech formátů jsou pro lepší představu zobrazeny na Obrázek 5.3 Průměrná doba zápisu dat jednotlivých formátů. V grafu je jasně zobrazeno, který formát je pro zapsání dat vhodnější. Jedná se pouze o vhodnější z hlediska časové náročnosti. Formát XML je díky své větší náročnosti na serializaci výrazně pomalejší oproti zbylým formátům v tomto srovnání. Průměrnými hodnotami zapsání dat jsou formáty JSON a mé vlastní navržení formátu. Nejlépe vyhodnocený formát je TLV, který má časovou náročnost menší než 100µs.



Obrázek 5.3 Průměrná doba zápisu dat jednotlivých formátů

V druhé části analýzy projektu byly porovnány časy načítání dat z výstupního souboru. Formát XML opět zaujímá první pozici v grafu. Nicméně doba deserializace formátu XML již není tak drastická, jako tomu bylo u serializace, kde rozdíl byl velmi patrný v časovém měřítku. V průměrné časové závislosti je i navržený formát. Rozdíl načítání dat ze souboru je uspokojivý vzhledem k jeho náročnosti a obsažené struktuře formátu. Vynikající výsledek mají formáty JSON a TLV. Jejich hodnoty jsou téměř identické, liší se pouze dvěma desetinami ku prospěchu TLV. V časové závislosti v převodu kódu je jednoznačně nevýhodnější formát TLV jak po stránce serializace, tak po jeho druhé stránce deserializace. Výsledky časové náročnosti formátů jsou vidět v grafu. Obrázek 5.4 Průměrná doba načtení dat jednotlivých formátů.



Obrázek 5.4 Průměrná doba načtení dat jednotlivých formátů

Další cílem bylo porovnat datovou náročnost jednotlivých formátů. Každý formát po serializaci dat byl uložen do výstupního souboru a ten byl prozkoumán a byla zjištěna jeho velikost. V této datové analýze je nejvhodnější mnou navržený formát. Jeho velikost souboru nepřesáhla přes 222 bajtů a je tedy nejmenším souborem ze všech formátů. Druhý skončil formát TLV, který má o 61 bajtů větší velikost s vybranými vstupními daty. Znatelně se již liší velikosti formátů JSON a XML. Formát JSON je menší oproti XML, které zabírá mnohem více velikosti prostoru souboru. XML je tak obsáhlé z důvodu svých párovacích tagů, které serializačnímu formátu přidají o mnoho více bajtů.

Při celkovém hodnocení formátů může hodnocení pro formát serializace XML dopadnout velmi negativně, tedy nekvalitním hodnocením. Po stránkách časové náročnosti a velikosti výstupního souboru byl v těchto parametrech podstatně horší. Zaujímá pomyslné poslední místo v této analýze. Ovšem za těmi nechvalně vyplývajícími hodnotami je jeho výhoda velmi dobrá orientace ve výstupním souboru. Nedělá problém najít požadovanou hodnotu i méně zdatnému uživateli, jelikož je vše

uzavřeno v tagách. Proto se dokáže v tomhle výstupním souboru velmi rychle zorientovat a najít v případě potřeby požadované informace. V tomto je značná výhoda formátu XML.

Dalším formátem je JSON. Tento formát je z hlediska časové náročnosti podstatně rychlejší a v tomto ohledu výhodnější. Hodnoty zápisu dat a načítání dat z výstupního souboru jsou průměrné až docela dostačující. Vzhledem k velikosti výstupního souboru se řadí formát JSON k pomyslné třetí příčce výhodného formátu k serializaci dat. Jeho výhoda je opět výborná orientace ve výstupním souboru. Jelikož je to textový formát, jsou hodnoty vypsány přesně a člověk se v nich velmi rychle orientuje v případě nutné potřeby.

Vlastní navržený formát je podle výsledků druhý nejlepší v serializaci dat. Jeho časové prodlevy v zakódování dat do výstupního souboru a posléze načtení těchto dat jsou taktéž průměrné hodnoty. Časové údaje jsou naměřeny větší, kvůli dobré orientaci a jednoduchosti ve výstupním souboru. Jelikož výstupní soubor je natolik zmenšen, dělá z něho velmi dobrou volbu pro přenos dat, který nezabírá příliš mnoho kapacitních prostředků.

Posledním formátem simulovaným v aplikaci je formát serializace TLV. Tento formát má velmi slibné hodnoty v zapisování hodnot do výstupního souboru. V časové náročnosti se tento formát TLV projevil jako jeden z nejvhodnější. Časy zapisování dosahovaly hodnot pod úrovní 100µs. Výsledek analýzy je proto v tomto ohledu velmi uspokojivý. Doba načítání formátu byla také jedna z nejlepších. Tento časový údaj byl sice o malý kousek rychlejší, než načítání dat pomocí formátu JSON, ale přesto je nejrychlejší. Datová náročnost formátu TLV je také velmi žádoucí. Velikost souboru dosahovala 283 bajtů. Tato velikost byla v porovnání s ostatními formáty druhá nejmenší. Patrnou nevýhodou může být fakt, že v této struktuře se již člověk dobře nevyzná. Orientace v této struktuře je znatelně komplikovanější.

6 ZÁVĚR

V této práci byly uvedeny určité typy datových formátů serializace pro výměnu informací, u kterých byly zhodnoceny jejich vlastnosti a výhody či nevýhody použití. Bylo podáno vysvětlení pojmů serializace a deserializace dat. Rozbor a vysvětlení nejzákladnějších druhů serializace, typy serializací. V krátkosti prozkoumány přehledy podporovaných programovacích jazyků a jejich rozbor v podpoře pro serializaci datových formátů.

Stěžejním výsledkem bakalářské práce je vyvinutá simulační aplikace, která se nazývá DataCoding – Kódování dat a formátů. Tato aplikace umožňuje simulovat tři nejzákladnější typy formátu serializace pro výměnu informací. Aplikace slouží k serializaci známých formátů typu XML, JSON a TLV. Dále slouží i pro navržený formát serializace. Dokáže všechny vyjmenované formáty zpětně převést, neboli deserializovat. DataCoding umožňuje pohodlné ovládání, vyhodnocení jednotlivých formátů a simulaci určitého typu serializace. Aplikace je schopna načíst databázi naplněnou datovými informacemi. Je schopna ukládat data jednotlivých simulovaných formátů do výstupního souboru pro další zpracování. Ve výstupním souboru je možnost odhadnout náročnost struktury jednotlivých datových formátů. Dále je možné zkontrolovat si data po serializaci zpětným načtením dat ze souboru. Všechny tyto možnosti jsou funkční pro každý typ formátu. Ve výstupním okně aplikace zobrazuje čas, za který byl proces proveden. Tento čas se vypisuje jak při serializaci dat, tak také při deserializaci dat. Z časových údajů je možné posoudit přesný časový úsek pro danou operaci určitého typu formátu.

Simulace jednotlivých datových formátů serializace potvrdily některé předpoklady na přenos informací po síti. Schopnost dosáhnout co nejlepšího výsledku ve vlastním navrženém formátu byla následně z části potvrzena na simulačním modelu. Z vycházejících měření časové náročnosti formátů a datové náročnosti souborů je patrné, které formáty jsou vyhodnoceny jako nejlepší pro výměnu dat po síti. V bakalářské práci byla sledována časová závislost a datová náročnost jako jeden ze stěžejních bodů této práce.

Formát typu TLV není v aplikaci zcela správně odladěn. Tento typ formátu má být číselný. V aplikaci je tento formát veden částečně jako textový. Mohou být výsledky dosažené simulací tímto programem zkreslené a neudávat tak pravdivé hodnoty měření a simulace kódu.

LITERATURA

[1] BRUSILOVSKY, P.; KOBSA, A.; NEJDL, W. *The Adaptive Web Methods and Strategies of Web Personalization* [online]. 2007. Dostupné z WWW: <http://www.springerlink.com/content/x646782t122p/#section=372718&page=1>.

[2] DENARO, G.; MARIANI, L. *Towards Testing and Analysis of Systems that Use Serialization* [online]. 2005. Dostupné z WWW: <http://www.sciencedirect.com/science/article/pii/S1571066104052855>.

[3] ENGLANDER, Robert. Java and SOAP. O'Reilly Media, 2002. 276s. ISBN 978-0596001759.

[4] TROELSEN, A.; AGARWAL, V. *Pro VB 2010 and the .NET 4 Platform* [online]. 2010. Dostupné z WWW: <http://www.springerlink.com/content/978-1-4302-2985-8/#section=839142&page=1>.

[5] IMRE, G. et al. *A Novel Cost Model of XML Serialization* [online]. 2010. Dostupné z WWW: <http://www.sciencedirect.com/science/article/pii/S1571066110000113>.

[6] HARMS, D. et al. *Začínáme programovat v jazyce Python*. 2.: Computer Press, 2008. 456 s. ISBN 8025121615.

[7] ECKEL, B. *Myslíme v jazyku C++: knihovna programátora* [online]. 2006. Dostupné z WWW: <http://books.google.com/books?id=fSk99oy_mKcC&printsec=frontcover&hl=cs#v=onepage&q&f=false>.

[8] KOSEK, J. *XML a PHP* [online]. 2009. Dostupné z WWW: <http://books.google.com/books?id=8h_GzeBWatkC&printsec=frontcover&hl=cs#v=onepage&q&f=false>.

[9] MCARTHUR, K. *Pro PHP Patterns, Frameworks, Testing and More* [online]. 2008. Dostupné z WWW: <http://www.springerlink.com/content/978-1-59059-819-1/#section=211142&page=1&locus=0>.

[10] KOSEK, Jiří. XML pro každého. Grada Publishing, 2000. 164s. ISBN 80-7169-860-1.

[11] SOMA, R. *A Model-Based Framework for Developing and Deploying Data Aggregation Services* [online]. 2006. Dostupné z WWW: <http://www.springerlink.com/content/978-3-540-68147-2/#section=534115&page=1&locus=0>.

[12] BROWN, S. *Dynamic Apache with Ajax and JSON* [online]. 2006. Dostupné z WWW: <http://books.google.com/books?id=XZZ9QjbUPsMC&printsec=frontcover&hl=cs#v=onepage&q&f=false>.

[13] BOOKS LLC. *Data Serialization Formats: Xml, Soap, S-Expression, Abstract Syntax Notation One, Serialization, Yaml, Json, Lightweight Markup Language*. General Books LLC, 2010. 186 s. ISBN 1157460399.

[14] BLACK, Uyless. *MPLS and label switching networks*. Prentice Hall, 2007. 236 s. ISBN 0130158232.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

.NET	dotNET
µs	Jednotka času
AJAX	Asynchronous JavaScript and XML
Bajt	V anglickém znění byte, jednotka množství dat
BASE64	Datový formát zobrazující binární data pomocí tisknutelných znaků ASCII
C	Programovací jazyk
C++	Objektově orientovaný programovací jazyk
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
double	Reálné číslo se zvýšenou přesností
GUI	Graphical user interface
char	Character – znaková proměnná
INI	Textové soubory uložené v kódování UTF-8
int	Integer – celočíselná proměnná
JAR	Java Archive
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
MSDN	Microsoft Developer Network
PHP	Personal Home Page – skriptovací programovací jazyk
SDK	Software development kit
SOAP	Simple Object Access Protocol
SWT	Standard Widget Toolkit
TLV	Type-length-value
UTF-8	UCS Transformation Format
XML	Extensible Markup Language
XSD	XML Schema Definition
YAML	YAML Ain't Markup Language