

Implementace protokolu SOAP pro mikropočítače

Implementation of the SOAP Protocol for Microcontrollers

Robert Sedláček

Bakalářská práce
2013



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2012/2013

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Robert SEDLÁČEK**
Osobní číslo: **A10058**
Studijní program: **B3902 Inženýrská informatika**
Studijní obor: **Informační a řídicí technologie**
Forma studia: **prezenční**

Téma práce: **Implementace protokolu SOAP pro mikropočítače**

Zásady pro vypracování:

1. Popište protokol SOAP a jeho využití.
2. Analyzujte možnosti implementace tohoto protokolu pro mikropočítače.
3. Navrhněte způsob komunikace mezi mikropočítačem a osobním počítačem pomocí tohoto protokolu.
4. Navrhněte a implementujte strukturu programové knihovny implementující rozhraní SOAP na zvolený mikropočítač.

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. PINKER, Jiří. Mikroprocesory a mikropočítače. Praha: BEN – technická literatura, 2004. ISBN 80-730-0110-1.
2. HRBÁČEK, J. Komunikace mikrokontroléru s okolím. 1. vyd. Praha: BEN – technická literatura, 1999, 156 s. ISBN 80-860-5642-2.
3. MANN, Burkhard. C pro mikrokontroléry: ANSI-C, kompilátory C, spojovací programy – linkery, práce s ATMEL AVR a MSC-51, příklady programování v jazyce C, nástroje pro programování, tipy a triky. Praha: BEN, 2003. ISBN 80-730-0077-6.
4. ARM Ltd and ARM Germany GmbH. Getting Started: Building Applications with RL-ARM [online]. 2009. [cit. 2013-01-28]. Dostupné z: http://www.keil.com/product/brochures/rl-arm_gs.pdf
5. W3C. SOAP protocol [online]. 2013 [cit. 2013-01-28]. Dostupné z: <http://www.w3.org/TR/soap12-part1/>

Vedoucí bakalářské práce:

Ing. Jan Dolinay, Ph.D.

Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce:

24. února 2013

Termín odevzdání bakalářské práce:

14. června 2013

Ve Zlíně dne 24. února 2013

prof. Ing. Vladimír Vašek, CSc.
děkan



prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Cílem bakalářské práce je zjistit současné možnosti implementace protokolu SOAP a navrhnout případné řešení spolu s implementací pro zvolený mikropočítač. Tato práce je rozdělena do dvou hlavních částí. První část je věnována popisu protokolu HTTP a jazyka XML, na kterých je SOAP postaven spolu s analýzou možností mikropočítačů. Druhá část je pak věnována návrhu programové knihovny a její implementací pro zvolený vývojový kit.

Klíčová slova: Mikropočítač, SOAP

ABSTRACT

The aim of this work is to identify the current options of implementation of SOAP protocol and suggest possible solutions along with implementation for the selected microcontroller. This work is divided into two main parts. The first part is devoted to the description of http protocol and XML language, on which is the SOAP built with an analysis of the options of microcontroller. The second part is devoted to design of software library and its implementation for the selected development kit.

Keywords: Microcontroller, SOAP

Zde bych chtěl poděkovat především své rodině, za její podporu při studiu a také vedoucímu mé bakalářské práce panu Ing. Janu Dolinayovi , Ph.D. za jeho připomínky a vstřícné jednání.

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	9
I TEORETICKÁ ČÁST	10
1 SOAP	11
1.1 POPIS HTTP	11
1.1.1 HTTP požadavek.....	12
1.1.2 HTTP odpověď	12
1.2 JAZYK XML	14
1.2.1 Základy XML	14
1.2.2 Kódování a znakové sady XML dokumentu	15
1.2.3 Jmenné prostory	16
1.2.4 Parsování XML	17
1.3 POPIS A VYUŽITÍ SOAP	18
1.3.1 Historie a alternativy SOAP	18
1.3.2 Struktura a syntaxe zprávy	19
1.3.3 Kódování dat	20
1.3.4 Transportní mechanismy	21
1.3.5 Jazyk WSDL	22
1.3.6 UDDI.....	24
2 MIKROPOČÍTAČE A ROZHRANÍ.....	25
2.1 ROZHRANÍ ETHERNET	25
2.2 PARAMETRY MIKROPOČÍTAČŮ	27
3 ANALÝZA MOŽNOSTI IMPLEMENTACE.....	28
3.1 MOŽNOST IMPLEMENTACE PRO 8MI NEBO 16TI BITOVÝ MIKROPOČÍTAČ SPOLU S POUŽITÍM OBVODU W5100	28
3.2 MOŽNOST IMPLEMENTACE V PŘÍPADĚ POUŽITÍ 32BITOVÉHO MIKROPOČÍTAČE	29
3.3 KOMUNIKAČNÍ ROZHRANÍ PRO SOAP	30
3.4 EXISTUJÍCÍ ŘEŠENÍ	30
4 NÁVRH ZPŮSOBU KOMUNIKACE MEZI MIKROPOČÍTAČEM A OSOBNÍM POČÍTAČEM.....	31
II PRAKTICKÁ ČÁST	32
5 VÝBĚR A POPIS VÝVOJOVÉHO KITU.....	33
5.1 POPIS VÝVOJOVÉHO KITU	33
5.2 POPIS VÝVOJOVÉHO PROSTŘEDÍ KEIL μ VISION 4.....	34
6 NÁVRH STRUKTURY KNIHOVNY.....	35
6.1 POPIS A VYUŽITÍ ROZHRANÍ KNIHOVNY TCPNET PRO ÚČELY SOAP	35
6.1.1 Popis odesílání HTTP zpráv pomocí knihovny TCPnet	35
6.1.2 Příjem http zpráv pomocí knihovny TCPnet.....	36

6.2	NÁVRH KNIHOVNY	37
6.2.1	Zpracování příchozí zprávy	37
6.2.2	Vytvoření odchozí zprávy	39
7	IMPLEMENTACE KNIHOVNY A JEDNODUCHÝ PŘÍKLAD KOMUNIKACE	41
7.1	POPIS ZDROJOVÉHO KÓDU PARSERU	42
7.2	VYTVOŘENÍ ODCHOZÍ ZPRÁVY	42
7.3	UKÁZKA A POPIS PHP KÓDU	43
	ZÁVĚR	45
	ZÁVĚR V ANGLIČTINĚ	46
	SEZNAM POUŽITÉ LITERATURY	48
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	50
	SEZNAM OBRÁZKŮ	51
	SEZNAM TABULEK	52
	SEZNAM PŘÍLOH	53

ÚVOD

V dnešní době je rychlá komunikace nezbytností v mnoha oblastech a je tedy třeba mít k dispozici efektivní způsob komunikace. Stávající spotřební elektronika nedisponuje příliš velkými možnostmi vzájemné spolupráce. Zařízení většinou používají proprietární a „neohebný“ komunikační protokol.

Ideální by bylo, pokud by všechna zařízení napříč různými třídami, mohla spolu komunikovat bez technických překážek. Mohli bychom tak například zjišťovat stav osvětlení, ventilace, baterií, ohřevu užitkové vody nízkoenergetické stavby pomocí jednoho zařízení a jednotlivé subsystémy by mohly snadno komunikovat mezi sebou. V případě použití fotovoltaické elektrárny v kombinaci s požadavkem na praní prádla v pračce by bylo možné směřovat pouze na slunečné dny, přičemž by se předpověď počasí načítala z meteorologických středisek přístupných přes celosvětovou síť - internet. Je tedy vhodné zvolit takový komunikační protokol, který umožní interoperabilitu a zároveň je třeba dbát na implementovatelnost, bezpečnost, rychlost generování a zpracování požadavků, přenosovou rychlost a v neposlední řadě také na energetickou náročnost, která přímo souvisí s výpočetním výkonem potřebným ke zpracování požadavků.

V dnešní době dochází k nárůstu použití webových služeb, mezi které lze SOAP zařadit. Velké weby, jako například vyhledávač Bing nebo aukční web Aukro, mají implementované API, s kterým lze komunikovat pomocí SOAP. Můžeme tak třeba na Aukru vystavovat předměty nebo stahovat informace o objednávkách, aniž bychom museli nějakým složitým způsobem parsovat html stránku nebo otevírat prohlížeč. Tato práce se bude zabývat právě tímto protokolem, který slouží k výměně zpráv založených na XML a možnostmi implementace pro zvolený mikropočítač.

I. TEORETICKÁ ČÁST

1 SOAP

SOAP je protokol pro posílání XML zpráv a je základem webových služeb. Aplikace si mezi sebou mohou posílat dotazy a odpovědi velmi snadno, protože komunikace je založena na platformě nezávislých standardech, především HTTP a XML. [1]

Celá infrastruktura webových služeb je postavena na třech základních technologiích:

- SOAP – protokol používaný ke komunikaci
- WSDL (Web Services Description Language) – standardní formát pro popis rozhraní webové služby
- UDDI (Universal Description, Discovery and Integration) – standardní mechanismus umožňující registraci a vyhledávání webových služeb

Pro co nejlepší pochopení SOAP je třeba nejprve uvést základy Popisu HTTP a Jazyku XML. Znalost těchto dvou technologií je stěžejní pro získání představy o nárocích na mikropočítač a také na implementaci.

1.1 Popis HTTP

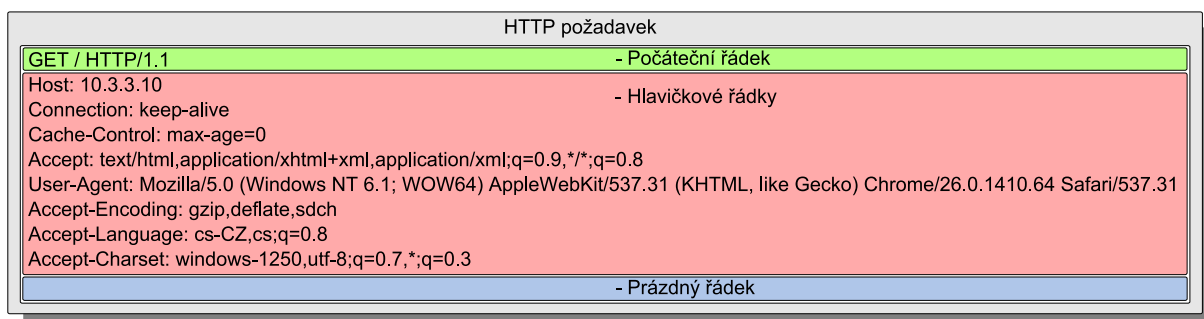
HTTP (Hypertext Transfer Protocol) slouží pro posílání dat v celosvětové síti, přičemž se může jednat o obrázkové soubory, HTML soubory, dotazy nebo cokoli jiného. HTTP využívá TCP/IP soketů. Komunikace se tedy sestává z klienta a serveru. Běžně webový server „naslouchá“ na portu 80 a jako HTTP klient obvykle slouží webový prohlížeč, který otevírá TCP spojení a zasílá skrze něj *požadavky* na server. [2]

Formát *požadavku* je velmi podobný formátu *odpovědi*. Oba typy se sestávají z:

- Počátečního řádku
- Hlavičkových řádků
- Prázdného řádku
- Dat (nepovinné)

1.1.1 HTTP požadavek

Ukázka *požadavku* získaného pomocí analyzátoru síťové komunikace (Wireshark) a jeho rozdělení.



Obrázek 1.1 Ukázka HTTP požadavku

Počáteční řádek se sestává ze tří částí, oddělených mezerami.

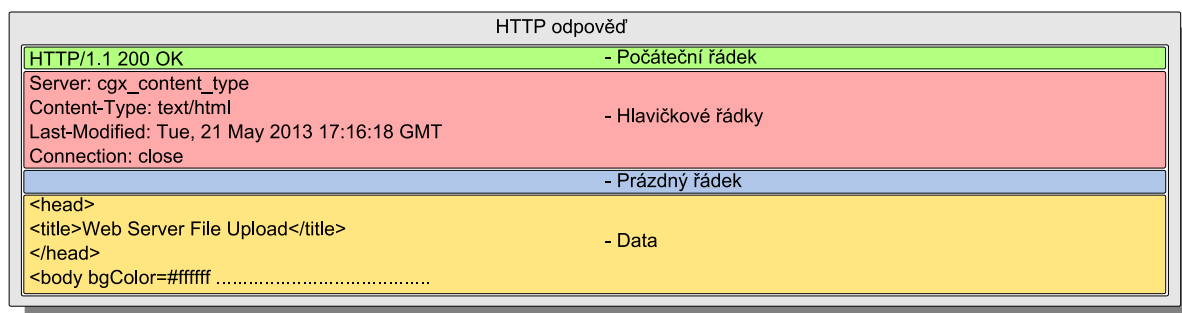
- Název metody
 - POST – umožňuje předávat data v těle *požadavku*
 - GET – umožňuje předávat pouze data, která jsou URL-encoded
 - HEAD – stejné jako GET, ale server vrátí pouze hlavičku *odpovědi*
- Cesta požadovaného zdroje – například „/cesta/index.html“
- Verze HTTP protokolu – vždy ve formátu „HTTP/x.x“

Hlavičková část nese různé informace (kódování, jazyk, ...) a ve verzi HTTP/1.0 definuje až 16 různých hlavičkových řádků, přičemž není žádný povinný, ve verzi HTTP/1.1 definuje 46 různých hlavičkových řádků, z toho jeden povinně.

Jediný hlavičkový řádek povinný ve verzi HTTP/1.1 je „Host: “ a udává název/doménu dotazovaného webového serveru. Toto je nutné uvádět, protože na jednom stroji s jednou unikátní ip adresou může být více webových domén.

1.1.2 HTTP odpověď

Ukázka *odpovědi* získané pomocí analyzátoru síťové komunikace (Wireshark) a její rozdělení.



Obrázek 1.2 Ukázka HTTP odpovědi

Počáteční řádek *odpovědi* se také sestává ze tří částí:

- Verze protokolu HTTP
- Stavového kódu odpovědi – snadné čtení pro počítač
- Anglické fráze popisující stavový kód – snadné čtení pro člověka

Stavový kód se sestává z třímístného celého čísla. Hlavní kategorie jsou [3]:

- **1xx** – pouze informační zpráva
- **2xx** – úspěch operace
- **3xx** – přesměrování klienta na jinou URL
- **4xx** – indikuje chybu na straně klienta
- **5xx** – indikuje chybu na straně serveru

Ukázka nejčastěji se vyskytujících stavových kódů:

200 OK – Požadavek uspěl a výsledek je vrácen v těle zprávy

404 Not Found – Požadovaný zdroj není k dispozici

500 Server Error – Došlo k chybě na straně serveru (špatná syntaxe v těle skriptu,...)

Za počátečním řádkem následují opět hlavičkové, po kterých jsou přenášena data. V ukázce HTTP *odpovědi* je vidět část přenášeného HTML souboru (viz. *Obrázek 1.2*).

1.2 Jazyk XML

XML je definované konsorciem W3C jako formát pro přenos obecných dokumentů a dat a je zkratkou pro eXtensible Markup Language, tj. rozšiřitelný značkovací jazyk. Návrh XML vychází ze staršího a obecnějšího standardu SGML (Standard Generalized Markup Language). Poznamenejme, že ze standardu SGML vycházel i formát dokumentů HTML (Hyper-Text Markup Language). Sada značek formátu HTML je pevná a slouží k vyjádření toho, jak má daný dokument vypadat. Naproti tomu v XML sada značek pevná není, ale může být definována pro různé sady dokumentů odlišně. Definice sady značek může být součástí definice XML dokumentu, specifikována odkazem, nebo dohodnuta předem. [4]

Hned od samého počátku také dbá na potřeby jiných jazyků, než je angličtina. Jako znaková sada se používá ISO 10646 (32bitová znaková sada, která dokáže pojmout všechny znaky dnes používaných jazyků). [5]

1.2.1 Základy XML

Každý XML dokument se skládá z elementů, které jsou do sebe navzájem vnořené. Elementy se v textu vyznačují pomocí tzv. tagů. Většině elementů odpovídají dva tagy, počáteční a ukončovací.

```
<para>obsah_elementu</para>
```

Ukázka obsahuje jeden element para. Jeho obsah je vyznačen pomocí tagů `<para>` (počáteční tag) a `</para>` (ukončovací tag). Názvy tagů se zapisují mezi znaky '`<`' a '`>`'. Ukončovací tag má před svým názvem ještě znak '/', aby se snadno odlišil od počátečního. Některé elementy nemusejí mít žádný obsah.

```
<br></br>
```

Vhodnější je však použití ještě jedné varianty tagu, která říká, že element nemá žádný obsah. Za jméno elementu v počátečním tagu se uvede znak '/'. Ukončovací tag se pak už nepoužije:

```
<para/>
```

Každý XML dokument musí obsahovat pro všechny počáteční tagy odpovídající ukončovací tag, nebo musí být počáteční tag zapsán jako element s prázdným obsahem a nesmí také docházet ke křížení tagů.

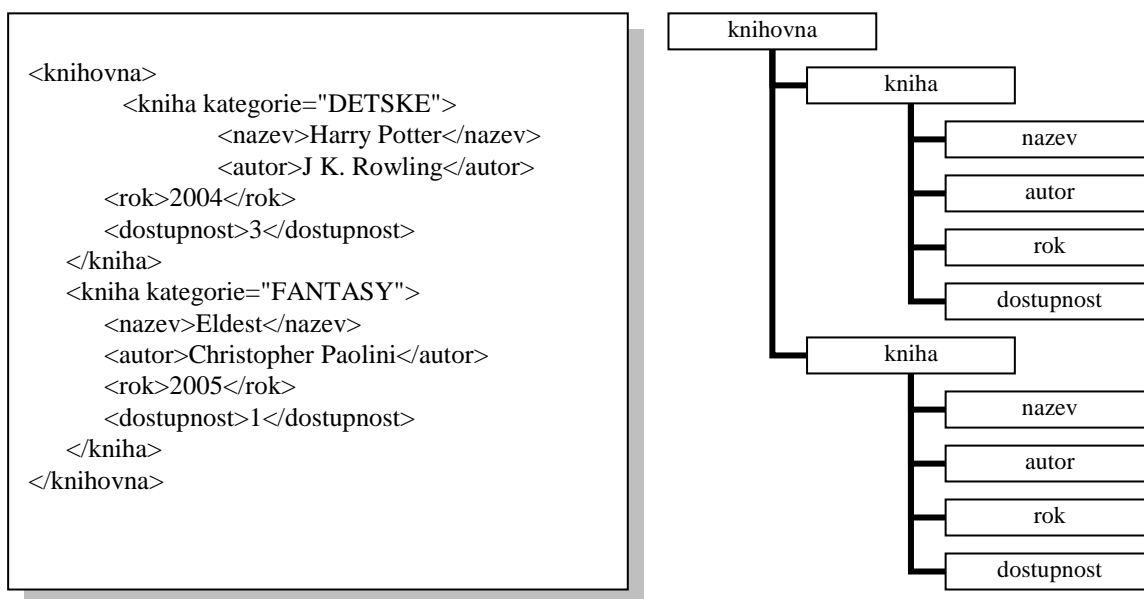
U každého počátečního tagu lze použít ještě atributy.

```
<para atribut="hodnota_atributu">hodnota_elementu </para>
```

Atributy se obvykle používají k upřesnění významu elementu. Hodnota atributu musí být vždy uzavřena do uvozovek nebo do apostrofů. U jednoho tagu lze použít více atributů najednou, musí však být odděleny mezerou.

Vzhledem k tomu, že se znaky ‘<’ a ‘>’ používají pro oddělení tagů od okolního textu, není možné tyto znaky zapsat do dokumentu jen tak. Pro jejich zápis musíme použít tzv. znakové entity. Pro zápis znaku ‘<’ je určena entita < a pro ‘>’ to je > [5]

Každý XML dokument musí být obsažen v jednom elementu, na následujícím obrázku je ukázka správně strukturovaného XML dokumentu.



Obrázek 1.3 Ukázka XML dokumentu a její stromová struktura

1.2.2 Kódování a znakové sady XML dokumentu

Mezi nejstarší a nejznámější znakové sady patří ASCII. Tato znaková sada je 7bitová a obsahuje znaky s kódy 0 až 127. Kromě písmen anglické abecedy, číslic a dalších znaků obsahuje ASCII i některé řídicí znaky. Potřebám angličtiny ASCII zcela vyhovuje, avšak

pro ostatní jazyky chybí některá písmena, například znaky s diakritikou. Z tohoto důvodu začaly vznikat další znakové sady.

Znaková sada definuje, jaké znaky a pod jakou číselnou hodnotou máme k dispozici. Kódování znakové sady určuje, jakým způsobem jsou jednotlivé kódy znaků převedeny na sekvenci bajtů, které znak reprezentují v paměti počítače, v souboru, při přenosu počítačovou sítí apod.

Všechny aplikace, které podporují XML, by měly zvládat práci s kódováními UTF-8 a UTF-16. U každého XML dokumentu lze určit používané kódování. V případě potřeby jiného kódování než UTF-8 nebo UTF-16, je třeba si dát pozor na podporu v cílových aplikacích. [5]

Použité kódování lze určit pomocí parametru encoding, který je součástí XML deklarace

```
<?xml version="1.0" encoding="windows-1250"?>
```

Deklarace musí být uvedena vždy na začátku XML dokumentu.

1.2.3 Jmenné prostory

Jmenné prostory (XML namespaces) nabízejí možnost, jak v jednom dokumentu kombinovat více sad značek (elementy a atributy). Každá sada značek je jednoznačně definována svojí URI adresou. Pokud je zapotřebí v nějakém elementu a jeho podelementech používat elementy patřící do určité sady značek, je třeba si o ně říct pomocí speciálního atributu xmlns.

xmlns:[prefix] = "URI sady značek"

V případě následné potřeby použití nějakého elementu nebo atributu patřícího do dané sady značek, je nutné přidat k jeho označení námi definovaný [prefix]. (viz. *Příklad 1.1*)

```
<root xmlns:html="http://www.w3.org/TR/html4/"  
xmlns:nabytek="http://www.w3schools.com/nabytek">
```

```
<html:tabulka>  
  <html:tr>  
    <html:td>Jablka</html:td>
```



```
<html:td>Hrušky</html:td>
</html:tr>
</html:tabulka>

<nabytek:tabulka>
  <nabytek:nazev>African Coffee Table</nabytek:nazev>
  <nabytek:sirka>80</nabytek:sirka>
  <nabytek:delka>120</nabytek:delka>
</nabytek:tabulka>

</root>
```

Příklad 1.1 Ukázka použití namespaces a prefixu vXML dokumentu

1.2.4 Parsování XML

V XML dokumentu je „zakódována“ spousta informací a je zapotřebí dostat je do patřičných proměnných. Tomu, jak tyto informace získat, se říká parsování. Je velmi výhodné používat parsery v podobě knihoven, protože není potřeba kontrolovat syntaxi dokumentu. Tu automaticky zkontroluje parser, navíc může ověřit i validitu například oproti schématu. Aplikace pak nemusí obsahovat kód pro ošetření chyb ve zpracovávaných datech. Existují dvě standardizované rozhraní parserů - SAX a DOM. [5]

Rozhraní SAX (Simple API for XML) je založeno na řízení pomocí událostí. V praxi to znamená, že jsou definované funkce, které se volají v momentě, kdy parser narazí na:

- začátek elementu
- konec elementu
- obsah elementu
- začátek nebo konec dokumentu

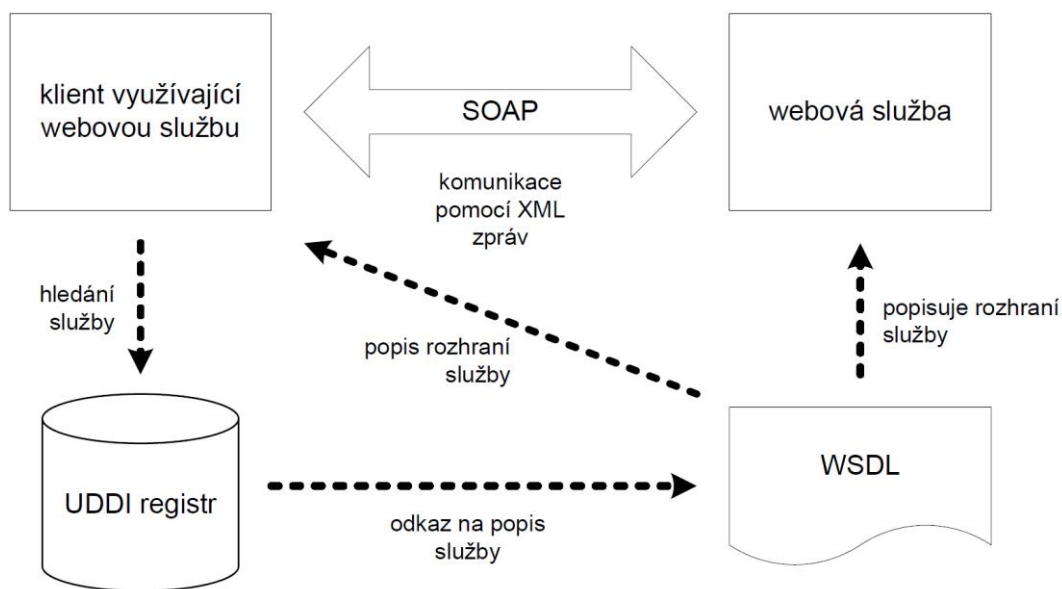
Jednotlivé události jsou tedy volané postupně, jak je čten dokument. Výhodou SAX je oproti DOM velká rychlost a menší paměťová náročnost.

V případě rozhraní DOM (Document Object Model) je dokument reprezentován jako stromová struktura. Přičemž nabízí funkce, pomocí kterých lze přidávat, mazat a měnit jednotlivé uzly. V dokumentu se lze pohybovat dle potřeby na rozdíl od SAX, kde se musí dokument procházet od začátku ke konci. DOM je tedy vhodné pro aplikace vyžadující náročnější práci s dokumenty (editory atp.). [5]

1.3 Popis a využití SOAP

SOAP umožňuje zasílání XML zpráv mezi dvěma aplikacemi, přičemž se jedná o jednosměrný přenos informace od odesílatele k příjemci, ale díky kombinování několika zpráv můžeme pomocí SOAPu snadno implementovat běžné komunikační scénáře. Nejčastěji se SOAP používá jako náhrada vzdáleného volání procedur (RPC), tedy v modelu požadavek/odpověď. Jedna aplikace pošle v XML zprávě požadavek jiné aplikaci, ta požadavek obslouží a výsledek zašle jako druhou zprávu zpět původnímu iniciátorovi komunikace. V tomto případě bývá webová služba vyvolána webovým serverem, který čeká na požadavky klientů a v okamžiku, kdy přes HTTP přijde SOAP zpráva, spustí webovou službu a předá jí požadavek. Výsledek služby je pak předán zpět klientovi jako odpověď. [1]

Využití HTTP je mimo jiné velmi výhodné také z důvodu, že různé sítě s různě nastaveným firewallem není třeba přenastavovat, neboť protokol HTTP na portu 80 bývá v drtivé většině neblokovaný.



Obrázek 1.4 Vztah tří základních technologií (SOAP, WSDL, UDDI) [1]

1.3.1 Historie a alternativy SOAP

SOAP původně navrhli Dave Winer, Don Box, Bob Atkinson a Mohsen Al-Ghosein v roce 1998 za podpory firmy Microsoft. Jeho první verze vznikla v roce 1999 a navazovala na o

rok mladší a méně flexibilní XML-RPC. Během roku 2000 byla představena vylepšená verze 1.1 a v současné době je k dispozici SOAP verze 1.2. Tato práce se bude věnovat poslední verzi 1.2. Dnes je SOAP specifikace držena XML skupinou tvořící internetové protokoly z W3C konsorcia.

Protože dochází k rozmachu webových služeb, existují i alternativy k tomuto protokolu. Jedná se například o protokol Atom, na který přešel například v nedávné době vyhledávač Google a zrušil podporu SOAP. Jako další protokoly lze zmínit Cisco Etch nebo REST.

1.3.2 Struktura a syntaxe zprávy

Mezi důležitá pravidla syntaxe patří, že zpráva musí být kódována použitím XML, využívat jmenné prostory SOAP Envelope a SOAP Encoding a při tom nesmí obsahovat DTD a instrukce pro zpracování. [6]

Kostra zprávy obsahuje kořenový element Envelope, uvnitř kterého jsou uzavřeny dva elementy Header (hlavička) a Body (tělo). Hlavička je přitom nepovinná a používá se pro přenos pomocných informací pro zpracování zprávy – například identifikaci uživatele, autentizační informace (jméno, heslo) apod. O to nejdůležitější se stará tělo zprávy, v němž se přenášejí informace identifikující volanou službu a předávané parametry, resp. návratové hodnoty služby. SOAP používá jmenné prostory pro identifikaci jednotlivých částí XML zprávy. Obálka, hlavičky a tělo zprávy patří do jmenného prostoru <http://www.w3.org/2001/12/soap-envelope>. [5]

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
```

```
</soap:Body>
</soap:Envelope>
```

Příklad 1.2 Ukázka kostry SOAP zprávy [6]

1.3.3 Kódování dat

V SOAP zprávě lze ještě definovat způsob kódování přenášených dat do XML. Se SOAP lze použít libovolný způsob serializace, standard SOAPu jeden rovnou definuje. Standardní serializace je schopná do XML převést graf obsahující otypované objekty. V praxi se nejčastěji používají skalární datové typy (čísla, řetězce, atd.) které definují XML schémata.

Obecně platí, že se hodnoty ukládají vždy jako obsah elementu (jedna hodnota do jednoho elementu). Datové typy mohou být definovány buď v externím XML schématu nebo přímo v XML zprávě. Druhý způsob je jednodušší a více používaný. [5]

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <clovek>
      <jméno xsi:type="xsd:string">Jan Novak</jméno>
      <bydliste xsi:type="xsd:string">Zlín</bydliste>
      <vek xsi:type="xsd:int">26</vek>
    </clovek>
  </soap:Body>
</soap:Envelope>
```

Příklad 1.3 Ukázka použití kódování v SOAP zprávě

Je však nutné, aby prefixy xsi a xsd byly svázány se jmennými prostory:

<http://www.w3.org/2001/XMLSchema-instance> a <http://www.w3.org/2001/XMLSchema>.

1.3.4 Transportní mechanismy

SOAP zpráva se zasílá v těle HTTP *požadavku* nebo *odpovědi*. Používá se přitom metoda POST, která dovoluje posílat data v těle HTTP zprávy. V hlavičce HTTP zprávy je však potřeba uvést ještě typ obsahu: *application/soap+xml*. Jednotlivé implementace webových služeb podporují i další přenosové mechanismy, patří mezi ně například protokol SMTP určený k odesílání e-mailů.

Ukázka jednoduchého HTTP *požadavku* se SOAP zprávou převzatého z w3schools.com:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

Příklad 1.4 Ukázka HTTP požadavku se SOAP zprávou [7]

Ukázka jednoduché HTTP *odpovědi* se SOAP zprávou převzaté z w3schools.com:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Příklad 1.5 Ukázka HTTP odpovědi se SOAP zprávou [7]

1.3.5 Jazyk WSDL

Jazyk WSDL slouží k popisu síťových služeb jako množiny koncových bodů zpracovávajících zprávy. Operace a zprávy jsou popisovány na abstraktní úrovni a teprve poté jsou svázány s konkrétním síťovým protokolem a datovým formátem. To umožňuje snadné vytvoření popisu rozhraní, které nabízí jednu službu několika způsoby. V praxi WSDL nejčastěji popisuje služby, které si posílají zprávy pomocí formátu SOAP a protokolu HTTP. WSDL vzniklo na základě potřeby sjednocení jazyka používaného pro popis rozhraní webových služeb. [5]

WSDL popisy jsou poskytovány ve formě XML dokumentu. Ten se skládá z následujících základních elementů [9]:

- Types – obsahuje definici datových struktur, nejčastěji se využívá XML schémat
- Message – abstrakt, psané definice dat, která mají být předána
- Operation – abstraktní popis operací podporovaných službou
- Port Type – sada operací
- Binding – slouží k navázání určitého portu na konkrétní protokol
- Port – jeden koncový bod definovaný jako kombinace binding a síťové adresy
- Service – kolekce koncových bodů

Ukázka WSDL dokumentu převzatého z tutorialspoint.com:

```
<definitions name="HelloService"
```

```
targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<message name="SayHelloRequest">
  <part name="firstName" type="xsd:string"/>
</message>
<message name="SayHelloResponse">
  <part name="greeting" type="xsd:string"/>
</message>

<portType name="Hello_PortType">
  <operation name="sayHello">
    <input message="tns:SayHelloRequest"/>
    <output message="tns:SayHelloResponse"/>
  </operation>
</portType>

<binding name="Hello_Binding" type="tns:Hello_PortType">
<soap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="sayHello">
  <soap:operation soapAction="sayHello"/>
  <input>
    <soap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:examples:helloservice"
      use="encoded"/>
  </input>
  <output>
    <soap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:examples:helloservice"
      use="encoded"/>
  </output>
</operation>
</binding>

<service name="Hello_Service">
  <documentation>WSDL File for HelloService</documentation>
```

```
<port binding="tns:Hello_Binding" name="Hello_Port">
  <soap:address
    location="http://www.examples.com/SayHello/">
  </port>
</service>
</definitions>
```

Příklad 1.6 Ukázka WSDL dokumentu [8]

V praxi se WSDL používá velmi často. Vzhledem k tomu, že jeho ruční vytváření je mírně pracnější, častěji se k jeho tvorbě používají generátory (v případě, že jsou k dispozici).

1.3.6 UDDI

UDDI nabízí mechanismy pro registrování, kategorizování a vyhledávání webových služeb. Jeho funkce si lze představit jako velký adresář, který obsahuje informace o subjektech a jimi poskytovaných službách. Samotný registr pracuje rovněž jako webová služba a komunikace s ním tedy opět probíhá pomocí SOAP.

UDDI registr obsahuje následující čtyři druhy entit:

- podnikatelské entity (firmy) – business entity (název, kontakt, popis, atd.).
- služby – business service (seznamy služeb, které firma poskytuje)
- šablony vazeb – binding template (popisují, jak a kde je možné se službou komunikovat)
- typy služeb – service typ

Typická práce s UDDI probíhá tak, že vývojář prohledá registr a najde si služby, které potřebuje. Získá pro ně popis WSDL a může je začít rovnou používat. UDDI však nemusí obsahovat jen WSDL, ale lze do něj ukládat popisy v libovolných formátech. [5]

2 MIKROPOČÍTAČE A ROZHRAŇÍ

Mikropočítač můžeme funkčně rozdělit na několik základních částí. Je to procesor, paměť programu, paměť dat a periferní obvody. Vždy jsou však přítomny alespoň vstupní a výstupní obvody, které zprostředkovávají spojení počítače s okolím. Zvláště v případě využití v automatizaci jsou mikropočítače vybaveny spoustou různých periférií, jako jsou převodníky, čítače a časovače, výkonové výstupy atd. [10]

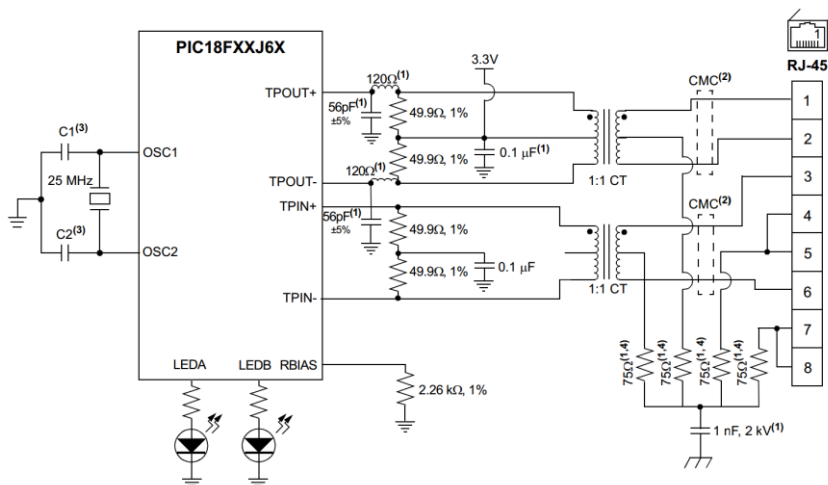
Podrobný popis mikropočítačů a rozhraní není předmětem této práce, proto v této části bude uvedeno pouze rozhraní ethernet, nutné pro použití protokolu SOAP a přehled některých méně či více vhodných procesorů se zaměřením na jejich parametry, ovlivňující možnou implementovatelnost SOAP.

2.1 Rozhraní ethernet

Jedná se o nejvíce rozšířenou a instalovanou síťovou technologii na světě. Rozšiřuje se už od počátku 80.tých let a vztahuje se k ní standard IEEE 802.3, který specifikuje komunikační rychlosti. V případě mikropočítačů se jedná nejčastěji o rychlosti 10 Mbps nebo 100 Mbps (často označované jako 10/100 Ethernet). Kombinace ethernetu a mikropočítače dovoluje různým aplikacím vysokou přenosovou rychlost a možnost zpřístupnění v rámci celého internetu bez nutnosti použití dalšího počítače. [11]

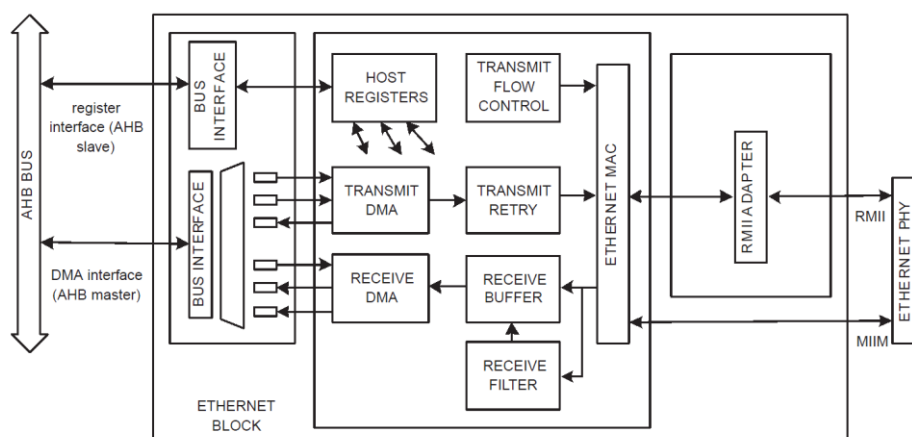
V současné době existuje několik způsobů připojení mikropočítače do sítě. Dále budou uvedeny nejčastěji používané varianty.

Na trhu jsou dostupné modely, které mají již potřebné periferie přímo v pouzdře. V případě připojení do metalické sítě je pak potřeba externě připojit již jen oddělovací transformátor. Představiteli tohoto typu jsou například procesory PIC18FxxJ6xx firmy Microchip.



Obrázek 2.1 Zapojení procesoru z řady PIC18FxxJ6xx do metalické sítě [12]

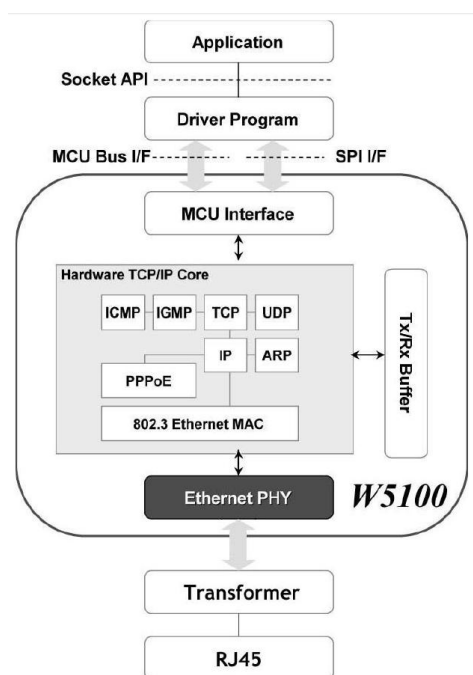
Dále existují modely s pouze částečnou implementací rozhraní. K tomuto typu je pak nutno připojit PHY (Ethernet Physical Layer) integrovaný obvod, který implementuje nejnižší (fyzickou) vrstvu ISO/OSI modelu. Toto řešení může být výhodné v případech, kdy nám nevyhovuje metalická síť a chtěli bychom zařízení připojit například do optické nebo bezdrátové sítě. Představiteli tohoto typu jsou například procesory řady LPC1700 firmy NXP s jádrem ARM Cortex M3.



Obrázek 2.2 Blokové schéma zapojení procesoru z řady LPC1700 firmy NXP [13]

Poslední možností je, že mikropočítač neobsahuje žádné periferie určené pro práci s rozhraním ethernet a je třeba použít specializovaných obvodů, s kterými lze komunikovat

například pomocí rozhraní SPI nebo USART. Toto řešení nemusí být vždy úplně vhodné z hlediska rychlosti přenášených dat nebo ceny. Na druhou stranu u těchto obvodů bývá implementován TCP/IP zásobník a základní práce s rodinou protokolů TCP/IP (např. UDP, TCP, ICMP, ARP, apod.). Z toho tedy plyne značná úspora času při vývoji programu a nižší nároky na výkon procesoru. Mezi tyto obvody patří například Wiznet W5100.



Obrázek 2.3 Struktura obvodu W5100 [14]

2.2 Parametry mikropočítačů

Na trhu je v dnešní době rozmanitá nabídka procesorů různých výrobců. Jednotlivé mikropočítače lze rozdělit například podle:

- architektury (RISC, CISC)
- délky slova (4,8,16,32 nebo 64bitové)
- velikosti a typu paměti určené pro samotný program
- velikosti operační paměti určené pro data programu
- taktovací frekvence, dostupných periférií, pouzder, pracovních teplot apod.

3 ANALÝZA MOŽNOSTI IMPLEMENTACE

Vzhledem k vyšší náročnosti práce s rozhraním ethernet a s tím spojenými nároky na výpočetní výkon a znalosti programátora, je důležité zvolit vhodný mikropočítač. Měl by být dostatečně výkonný a také nám nabídnout co nejvíce usnadňujících funkcí, popřípadě softwarovou podporu ze strany výrobce.

Jednou z nejvíce omezujících podmínek pravděpodobně bude velikost paměti dat a programu. Například v případě 8mi a 16ti bitových mikropočítačů bývá velikost paměti pro data v řádu desítek kB a pro program se pohybuje kolem stovek kB. Obvyklá paměťová náročnost zásobníku je však několik set kB programové paměti a několik stovek kB paměti dat. Existují však implementace zásobníku, které zabírají pouze několik kB paměti programu a velikost v paměti dat lze snížit až na hodnotu stovek Bajtů. [15]

3.1 Možnost implementace pro 8mi nebo 16ti bitový mikropočítač spolu s použitím obvodu W5100

Jako jedna z nejjednodušších možností se nabízí použití prakticky libovolného levného mikropočítače s rozhraním SPI spolu s obvodem Wiznet W5100. V tomto případě je TCP/IP zásobník (spolu s rodinou TCP/IP protokolů) implementován v obvodu W5100 a tím odpadá problém s implementací zásobníku a protokolů v mikropočítači.

Při zvolení této možnosti tedy zůstanou téměř všechny paměťové prostředky dostupné pro účely daného programu a práci se SOAP zprávami. Je třeba však počítat s určitými nároky na paměť dat, do které se musí uložit SOAP zpráva určená ke zpracování. SOAP zprávy však nemají omezenou délku a mohou bez problému dosahovat velikosti desítek kB. Po přijetí zprávy je třeba použít parser a vytvořit SOAP odpověď. Mikropočítač se navíc musí starat i o jiné funkce, je tedy velmi velká pravděpodobnost nutnosti použití operačního systému reálného času, který však s sebou přináší další nároky na paměť.

Shrnutím těchto nároků, lze říci, že za určitých omezujících podmínek by bylo možné realizovat výměnu SOAP zpráv avšak s omezenou funkčností. Proto se touto možností dále tato práce zabývat nebude.

3.2 Možnost implementace v případě použití 32bitového mikropočítače

V dnešní době jsou na trhu dostupné výkonné 32bitové mikropočítače za částky srovnatelné s 8mi nebo 16ti bitovými. Tyto procesory poskytují dostatek výkonu potřebného pro běh operačního systému reálného času spolu s řízením síťové komunikace. Navíc jejich výrobci poskytují odladěné knihovny nebo zdrojové kódy usnadňujících vývoj. Také často nabízejí výkonné periferie, jako například 10/100 Mbps Ethernet. Tyto typy procesorů dodávají například firmy Atmel (řada AVR32) nebo Microchip (PIC32). [16]

V současné době jsou však nejprogresivnější obvody založené na architektuře ARM Cortex-M3 a v blízké budoucnosti se zřejmě stanou nejpoužívanější ARM architekturou. Především je tomu tak v důsledku vyšší produktivity, nižší složitosti programovacího modelu, lépe propracovaným systémem přerušení a v neposlední řadě i díky své nízké ceně. [17]

Pro účely této práce bude popsána možnost implementace pro velice oblíbený mikropočítač LPC1768 firmy NXP. K tomuto mikropočítači je dostupných mnoho zdrojových kódů, které usnadňují a především urychlují vývoj. Především se jedná o operační systémy reálného času s dostupnými knihovnami pro síťovou komunikaci. Mezi nejznámější patří freeRTOS, μ C/OS-III nebo RTX RTOS firmy Keil. Vzhledem k tomu, že se jedná o architekturu ARM (která vlastní dceřinou společností Keil), bude k analýze použit právě operační systém firmy Keil spolu s knihovnami pro síťovou komunikaci (TCPnet).

Následující tabulka uvádí přehled paměťových nároků služeb, které jsou předpokladem pro správnou funkci protokolu SOAP.

	Nároky na paměť dat (RAM)	Nároky na paměť programu
Dostupná paměť LPC1768	64 kB	512 kB
Implementace ethernet	19,4 kB	8,2 kB
Správa UDP + TCP soketů	0,2 kB	4,9 kB
HTTP server	0,3 kB	6,6 kB
RTX jádro	428 B	4 kB
RTX nároky na jednu úlohu	52B + velikost zásobníku	-

Tabulka 3.1 Přehled vybraných paměťových nároků na procesor LPC1768 [18]

Z předcházející tabulky lze snadno odečíst, že paměť dat je zaplněná téměř z poloviny a nároky na paměť programu jsou téměř zanedbatelné. V tabulce však nejsou uvedeny

nároky na parsování zpráv a generování nových. Mimo jiné část knihovny, která se stará o HTTP server, již obsahuje částečnou podporu pro SOAP. Rozsah této podpory a její použití bude uvedeno dále v praktické části této práce.

Možnosti implementace pro tento typ mikropočítače jsou tedy rozsáhlejší než v předchozích případech. Nelze však stále počítat s plnou funkcionalitou, jako například na osobním počítači nebo serveru. Zvláště pokud si uvědomíme, že délka SOAP zprávy není omezená. Při zpracování zpráv je navíc velmi vhodné použít parser. Jak již bylo řečeno v kapitole 1.2.4, existují dva typy parserů SAX a DOM. V případě tohoto mikropočítače se budeme muset omezit nejspíše pouze na typ SAX. Oproti DOM má podstatně nižší nároky na paměť a je rychlý.

Nejvhodnější možností by byl mikropočítač s pamětí dat v řádu MB. [19]

Takových však na trhu příliš nenajdeme. Je tedy třeba zavést jistá omezení, například na již zmíněnou velikost SOAP zpráv.

3.3 Komunikační rozhraní pro SOAP

SOAP používá výhradně rozhraní ethernet. Teoreticky by však šlo zasílat SOAP zprávy i přes jiné rozhraní než ethernet, například RS-232 nebo RS-485. RS-232 umožňuje přenos dat pouze mezi dvěma zařízeními. RS-485 naproti tomu umožňuje sběrnicové připojení více stanic. Bylo by tedy třeba vymyslet nějaké rozšíření SOAP protokolu, které by řešilo přístup k této sběrnici a adresování jednotlivých stanic. Protože SOAP neříká nic o délce dat, museli bychom ji předem nadefinovat. SOAP také nedefinuje žádnou kontrolu dat – například CRC. Se zavedením vlastních omezení a rozšíření lze tedy SOAP zprávy přenášet prakticky přes jakékoliv rozhraní.

3.4 Existující řešení

Jediné řešení, které se podařilo najít, dodává česká firma SOLARControls s.r.o. a jedná se o zařízení WATTrouter M. Nepodporuje však přímo komunikaci pomocí protokolu SOAP ale pouze výměnu jednoduchých XML zpráv pomocí HTTP.

4 NÁVRH ZPŮSOBU KOMUNIKACE MEZI MIKROPOČÍTAČEM A OSOBNÍM POČÍTAČEM

SOAP bývá nejčastěji spojován s komunikačním protokolem HTTP, z tohoto důvodu bude tato práce předpokládat tento protokol. Proto je nutné, aby byl mikropočítač vybaven rozhraním ethernet. Mikropočítač a osobní počítač musí mít vhodně nastavenou ip adresu, masku sítě a v případě, že leží v různých podsítích i výchozí bránu.

Dalším předpokladem je vhodné nastavení síťového firewallu tak, aby nedošlo k blokaci portu, na kterém je dostupný HTTP server a tedy i SOAP. Obvykle se však jedná o port 80 a není třeba dalšího nastavování.

Pro účely této práce bude osobní počítač vybaven síťovou kartou a propojen přímo pomocí metalického kabelu se síťovým rozhraním, které je součástí mikropočítače. Je tedy nutné nastavit oběma zařízením rozdílné ip adresy z jedné podsítě.

Mezi oběma zařízeními poté může začít probíhat výměna SOAP zpráv.

II. PRAKTICKÁ ČÁST

5 VÝBĚR A POPIS VÝVOJOVÉHO KITU

Na trhu existuje nepřeberné množství vývojových kitů různých výrobců. Pro účely této práce bylo třeba vybírat pouze z těch, které disponují rozhraním ethernet a dostatečně výkonným mikropočítačem. Vzhledem k rozšířenosti mikropočítače LPC1768 jej lze nalézt v mnoha vývojových kitech různých výrobců (značkových i neznačkových). Většinou bývají už osazeny ethernetem, USB porty, dotykovým LCD, sběrnici CAN, RS232 apod.

Byl zvolen levnější, ale plně vyhovující vývojový kit. Tento kit dodává čínská společnost PowerMCU. Daní za nízkou cenu jsou v mém případě ukázkové zdrojové kódy, dodávané s komentáři v čínštině.

5.1 Popis vývojového kitu



Obrázek 5.1 Pohled na zvolený vývojový kit [20]

Vývojový kit obsahuje:

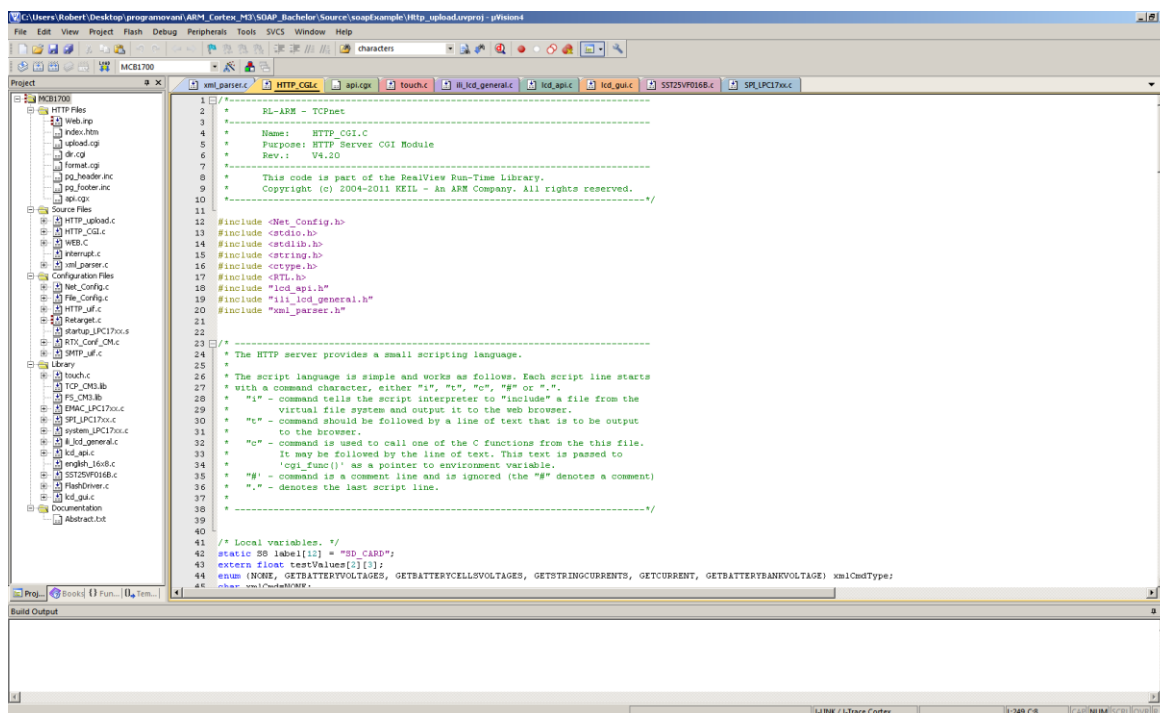
- 2x RS232 sériové rozhraní
- 2x sběrnice komunikační sběrnice CAN
- RS485 sériové rozhraní
- RJ45-10/100M Ethernet (Ethernet PHY: DP83848).
- analogový výstup, integrovaný malý reproduktor
- analogové vstupy, potenciometr
- dotykový LCD displej

- USB2.0 s podporou hosta
- SD/MMC slot pro paměťovou kartu (SPI)
- I2C rozhraní, s připojenou 2Kbit EEPROM paměti
- SPI rozhraní s připojenou 16Mbit Flash paměti
- 4 tlačítka
- 8 LED diod
- 5ti směrový joystick
- JTAG programovací a ladící rozhraní
- Integrovaný JLINK emulátor podporující ladění a simulace
- Možnost zvolit externí napájení nebo napájení z USB

Velkou výhodou je integrovaný JLINK emulátor, který umožňuje programování a ladění. Není tedy potřeba dokupovat žádné další zařízení.

5.2 Popis vývojového prostředí Keil μ Vision 4

Toto prostředí vyvíjí firma Keil a umožňuje pohodlný vývoj aplikací a správu projektů. Obsahuje také nástroje pro ladění a simulace. Lze jej použít při tvorbě programů pro velkou řadu typů mikropočítačů (založených na ARM, 8051, apod.)



Obrázek 5.2 Pohled na vývojové prostředí Keil μ Vision 4

6 NÁVRH STRUKTURY KNIHOVNY

Popis a vývoj celé knihovny, která by implementovala pro SOAP nezbytné ovládání rozhraní ethernet a HTTP, by byl velmi pracný a zdoluhavý. Proto existují již hotové knihovny, které usnadňují a především zrychlují vývoj. V našem případě se bude jednat o knihovnu TCPnet, vyvíjenou firmou Keil.

6.1 Popis a využití rozhraní knihovny TCPnet pro účely SOAP

Tato knihovna se sestává z ovladačů pro síťové rozhraní a konfiguračního souboru. Přídavná aplikační vrstva umožňuje použití služeb jako například SMTP, TELNET, FTP nebo HTTP. Samozřejmě obsahuje základní funkce pro práci s TCP a UDP. Navíc umožňuje použití HTML stránek a dalších souborů uložených jako pole v jazyce C nebo na SD kartě. [21]

6.1.1 Popis odesílání HTTP zpráv pomocí knihovny TCPnet

V případě, že je třeba dynamické změny dat v odesílaných http zprávách, TCPnet nabízí jednoduchý CGI skriptovací jazyk pomocí kterého lze snadno měnit obsah HTML dokumentu určeného k odeslání. Skriptovací jazyk obsahuje čtyři základní příkazy:

Příkaz	Popis
i	Vložení dalšího HTML souboru
t	Znaky následující za tímto znakem jsou čistý text
c	Tento řádek představuje příkazovou část a bude zavolána funkce <i>cgi_func()</i>
.	Tečka musí být vždy na posledním řádku
#	Křížek uvozuje komentář

Tabulka 6.1 Přehled CGI příkazů

Tyto příkazy musí být vždy uvedeny na začátku řádku. V příkladu 5.1 jsou první dva řádky uvozeny příkazem 't' který značí, že následující text nebude žádným způsobem upravován. Řádek, který začíná znakem c, bude předán funkci *cgi_func()*. Předaný řádek začíná uživatelsky definovaným znakem, v tomto případě 'a'. Tato funkce tedy musí obsahovat kód, který daný řádek rozparsuje a poté naformátuje zbytek řádku. K naformátování slouží funkce *sprintf()*, která má jako jeden z parametrů právě předávaný řádek. Je tedy nutné, aby v předávaném řádku figurovaly speciální sekvence ('%s', '%d', apod.). Ukázka funkce *cgi_func()* je v příkladu 5.2.

```

t    <html><head><title>Ahoj svete</title></head></html><body>
t    <h2 align=center> Ukazka </h2>
c a    <p> %s </p>
t    </body>
t    </html>

```

Příklad 6.1 Příklad html dokumentu s dynamicky se měnícím obsahem

```

U16 cgi_func(U8 *env, U8 *buf, U16 buflen, U32 *pcgi){
    switch(env[0]){
        case 'a':
            len=sprintf((S8 *) buf, (const S8 *) &env[2], "text");
            break;
    }
    return ((U16) len);
}

```

Příklad 6.2 Příklad funkce cgi_func()

Funkci *cgi_func()* lze samozřejmě použít i pro úpravu a přenos jiných formátů než jen html. Pokud se jedná o SOAP zprávu, je nezbytné změnit v HTTP hlavičce pole *Content-Type*. K tomuto účelu slouží funkce *cgx_content_type()*, která vrací ukazatel na řetězec (viz. příklad 6.3). [21]

```

U8 *cgx_content_type (void) {
    return " application/soap+xml; charset=utf-8\r\n";
}

```

Příklad 6.3 Příklad funkce cgx_content_type()

6.1.2 Příjem http zpráv pomocí knihovny TCPnet

V případě přijetí POST http požadavku, je volána funkce *cgi_process_data()*, které je předán ukazatel na začátek datové části spolu s délkou dat a číselný kód, který označuje typ dat. Typy číselných kódů jsou uvedeny v tabulce *Tabulka 6.2*. Hlavička funkce *cgi_process_data()* je uvedena v příkladu 6.4.

Kód	Typ dat
0	Data předávaná pouze v URL
1	Jméno nahrávaného souboru na server
2	Data nahrávaného souboru
3	Konec nahrávání souboru
4	XML data
5	XML data s předpokladem, že budou následovat další

Tabulka 6.2 Přehled významů kódu dat

Existuje i funkce pro zpracování GET požadavku, ta však pro účely SOAP komunikace není zapotřebí a proto bude její popis vynechán.

```
void cgi_process_data (U8 code, U8 *dat, U16 len) {
    switch (code) {
        case 0:
            /* Url encoded form data received. */
            break;
        case 1:
            ...
    }
}
```

Příklad 6.4 Část funkce *cgx_content_data()*

6.2 Návrh knihovny

Vzhledem k existenci funkcí velmi usnadňující práci, nejsou nároky na knihovnu příliš velké. Kvůli dodržení co nejnižších nároků na paměť dat, se bude knihovna prakticky sestávat pouze z SAX parseru.

6.2.1 Zpracování příchozí zprávy

V případě přijetí SOAP zprávy je volána funkce *cgi_process_data()* s číselným kódem 4 (v případě krátkých dat) nebo 5 (pro dlouhá data, v tomto případě bude tato funkce volána několikrát po sobě). Zpráva je uložena přímo v paměti. Prakticky jí tedy lze hned předat parseru.

Využijeme tedy znalosti principu SAX parseru (ke každému elementu jsou volány různé funkce). Samotný parser se bude sestávat z jediné funkce s dvěma parametry, prvním parametrem bude ukazatel na přijatá data a druhým parametrem bude ukazatel na strukturu s ukazateli na funkce. Tato struktura bude ukazovat na funkce, které budou volány

v případě začátku dokumentu, konce dokumentu, začátku elementu, konce elementu a začátku obsahu elementu. Funkcím budou navíc předávány příslušné parametry jako například jméno elementu a atributy elementu. Navrhovaná struktura s ukazateli na jednotlivé funkce je uvedena v příkladu 6.5.

```
typedef struct{
    void (*startDocument) ();
    void (*startElement)(char *, char **, char**,U8);
    void (*characters)(char *);
    void (*endElement)(char *);
    void (*endDocument) ();
} eventFunct;
```

Příklad 6.5 Struktura s ukazateli na jednotlivé funkce

Popis funkcí:

startDocument(), endDocument():

- Tyto funkce se volají pouze před začátkem parsování a na jeho konci. Těmito funkcím nejsou předávány žádné parametry.

*characters(char *name):*

- Tato funkce je volána v případě, že parser narazí na obsah elementu a jako parametr se jí předává ukazatel na místo začátku

*startElement(char *name, char **paramName, char **param, U8 n):*

- Tato funkce je volána v případě, že parser narazí na začátek elementu. Jejími parametry jsou ukazatel na začátek názvu elementu, pole ukazatelů na jednotlivé názvy parametrů elementu, další pole ukazatelů na hodnoty parametrů elementu a celkový počet parametrů

endElement():

- Tato funkce je volána v případě, že parser narazí na konec elementu. Jejím parametrem je pouze ukazatel na název elementu

Díky použití struktury s ukazateli na jednotlivé funkce, lze použít parser s různými sadami funkcí. Pouze se parsovací funkci předá ukazatel na námi vytvořenou strukturu (a tedy sadu funkcí). Tato struktura je definována jako uživatelský datový typ.

Každý řetězec znaků musí být ukončen znakem ‘\0’ to znamená, že pokud budeme předávat jednotlivým funkcím ukazatele na řetězce, je vhodné zajistit na jejich konci ukončovací znak. Nabízí se dvě možnosti, jednou z nich je dynamicky alokovat vždy potřebnou část paměti, zavolat příslušnou funkci a po jejím provedení zase paměť uvolnit. V případě počátečního tagu by se musel spočítat počet parametrů a délka všech řetězců, poté provést alokaci a daný tag projít znovu kvůli naplnění nově alokovaného místa. Druhou možností je upravovat původní dokument. Pokud se zamyslíme, není potřeba funkcím předávat znaky typu ‘<’ ‘>’ ‘=’ atp. Tyto lze nahradit přímo v dokumentu ukončovacím znakem. Například tag by poté měl poslední znak ‘>’ nahrazen ukončovacím znakem.

Obě řešení mají určité výhody a nevýhody. První možnost je pomalejší a má vyšší nároky na paměť, na druhou stranu dokument zůstane beze změny. Hodí se tedy například pro případy, kdy bychom jej chtěli procházet vícekrát. Druhá možnost má nižší nároky na paměť a zpracování je rychlejší. Dojde však ke změnám v dokumentu.

Velmi vhodné je před spuštěním parseru otestovat, jestli má dokument správnou syntaxi a nedochází ke křížení značek. Testování může fungovat na principu zásobníku, kdy se v případě počátečního tagu uloží název elementu do zásobníku. V případě, že narazí na další počáteční tag jiného elementu, tento se také uloží do zásobníku. Jakmile však narazí na ukončovací tag, porovná se s vrcholem zásobníku. Pokud budou oba řetězce stejné, je vše bez problémů a tento řetězec bude ze zásobníku vyjmut. Na konci musí zůstat zásobník prázdný.

6.2.2 Vytvoření odchozí zprávy

Pro účely vytvoření odchozí zprávy bude použito možností, které nabízí knihovna TCPnet, bude tedy vytvořena následující kostra odchozí zprávy dostupná pod názvem *api.cgx*:

```
t <?xml version="1.0" encoding="UTF-8"?>
t <env:Envelope
t   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
```

```
t    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
t    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
t    <env:Body>
c a
t    </env:Body>
t </env:Envelope>
t
. End of script must be closed with period.
```

Příklad 6.6 Kostra dokument, jejíž část se bude programově upravovat

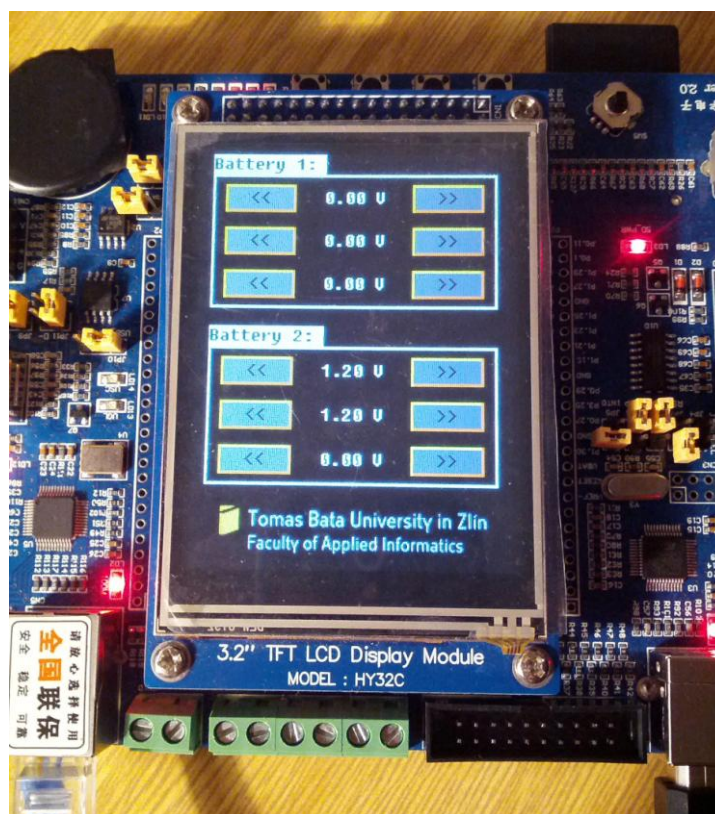
V příkladu 4.3 jsou vytvořeny všechny důležité elementy nezbytné pro SOAP zprávu a u kterých je předpoklad, že nebudou potřeba žádným způsobem měnit. Tato část dokumentu bude tedy součástí všech odchozích zpráv. V našem případě se bude programově měnit pouze obsah elementu `<env:Body>`.

V případě, že odchozí zpráva má reagovat na příchozí, lze obsah těla měnit přímo na základě elementů v těle přijaté zprávy. A to například tak, že pokud parser narazí na nějaký elementu v příchozí zprávě, může se na jeho základě ihned přidat jiný element do těla odchozí zprávy.

7 IMPLEMENTACE KNIHOVNY A JEDNODUCHÝ PŘÍKLAD KOMUNIKACE

Vzhledem k tomu, že není předpoklad pro pozdější zpracování příchozí zprávy a z důvodu využití co nejméně paměti, bude naprogramován jednoduchý parser, který přímo upravuje předávaná příchozí data. V parseru nebyla implementována kontrola syntaxe příchozí zprávy. Funkce (začátek elementu atd.) budou obsahovat kód pro přímé vytváření elementů, které se poté dosadí do kostry dostupné v *api.cgx*.

Funkčnost implementace bude předvedena na příkladu s bateriemi, které zálohují dům. Tyto baterie jsou hlídány mikropočítačem, který průběžně sleduje jejich kapacitu a využití jednotlivých článků. Pro naše účely budeme uvažovat pouze dvě baterie, každá složená ze tří článků. Pomocí SOAP se bude možné dotázat pouze na jejich napětí. Jednotlivá napětí budou zobrazena na displeji vývojového kitu a pomocí dotykového ovládání umožněna jejich změna (viz. obrázek *Obrázek 7.1 Ukázka rozhraní simulujícího napětí článků*). Dotazy budou odesílány z osobního počítače, přičemž bude použito jazyka PHP.



Obrázek 7.1 Ukázka rozhraní simulujícího napětí článků

7.1 Popis zdrojového kódu parseru

Hlavička parsovací funkce je následující:

```
void parseXML(eventFunct *events, U8 *dat)
```

Zdrojový kód parsovací funkce je uveden v příloze P I.

Na začátku parsování je volána funkce *startDocument()* a poté celá funkce běží v cyklu, dokud není testovaný znak na n-té pozici roven ukončovacímu znaku. Celý dokument se tedy projíždí znak po znaku. Jakmile narazí na znak '<', jedná se o počáteční tag. Poté se najde zakončovací znak tagu '>' a nahradí se znakem '\0'. V případě, že hned za počátečním znakem '<' následuje znak '/', jedná se o ukončovací tag a je volána funkce *endElement()* z předávané struktury. Parametrem funkce *endElement()* je ukazatel na místo, které následuje ihned za znaky '</'.

Pokud za znakem '<' následuje pouze text, jedná se o počáteční tag. V tomto případě se v celém tagu spočítá počet všech parametrů a alokují se dvě pole ukazatelů s příslušnou délkou. Poté je tag projit znak po znaku znova a příslušná pole ukazatelů jsou naplněna ukazateli na začátky názvů parametrů a začátky hodnot parametrů. Nakonec jsou v celém tagu nahrazeny znaky za jednotlivými řetězci znakem '\0' (jedná se o znaky '>', mezera, uvozovka nebo rovná se) a nakonec je volána funkce *startElement()*. Po provedení této funkce dojde k uvolnění paměti.

V případě obsahu elementu je situace podobná. Najde se poslední znak počátečního tagu '>' a testuje se, zda se za ním nachází začátek posledního tagu, začínajícího znakem '<'. Pokud ne a následuje text, zavolá se funkce *characters()*, které je předán parametr na počáteční znak řetězce. Poslední znak je obvykle '<' a je nahrazen ukončovacím znakem, proto je nutná pomocná proměnná, která informuje o této situaci. Všechny funkce jsou volané v takovém sledu, aby nedošlo k přeskočení žádného elementu, či jeho obsahu.

Na konci parsování je volána funkce *endDocument()*.

7.2 Vytvoření odchozí zprávy

Na začátku je alokováno místo v paměti určené pro novou zprávu. V průběhu tvorby je kontrolována délka zprávy a v případě nutnosti je provedena realokace. Vytvoření obsahu

odchozí zprávy zajišťují přímo funkce, volané z parseru. Například pokud příchozí zpráva obsahuje element `<getBatteryVoltage id="1"/>`, sečtou se všechna dílčí napětí článků a do odchozí zprávy se přidá:

```
<battery id="YY"><voltage xsi:type="xsd:float">XX </voltage></battery>
```

kde XX je výsledné sečtené napětí a YY je id baterie.

Tato část alokované paměti je poté „vlozena“ do těla kostry dokumentu umístěné v `api.cgx`. Po odeslání zprávy je alokovaná paměť uvolněna. Všechny funkce, na které je odkazováno ze struktury `eventFunc` jsou v příloze P II.

7.3 Ukázka a popis PHP kódu

Webová stránka, je vytvořena za účelem otestování SOAP komunikace. Tato stránka je rozdělena do tří částí. V první části je prováděn výběr příkazu, na základě kterého dojde k vytvoření odchozí SOAP zprávy. Po výběru příkazu a stisknutí tlačítka „Proved“ Dojde k odeslání SOAP zprávy a následná zpráva s odpovědí je poté zpracována a zobrazena v druhé části. V třetí části jsou zobrazeny XML data odchozí a přijaté SOAP zprávy, spolu se strukturovaným výpisem proměnné s výsledkem.

Pro potřeby otestování funkčnosti jsou implementovány pouze dva typy možných elementů:

- `<getBatteryCellsVoltage id="XX"/>` - slouží ke zjištění dílčích napětí baterie s identifikátorem XX.
- `<getBatteryVoltage id="XX"/>` - slouží ke zjištění napětí baterie s identifikátorem XX. Napětí všech článků tedy budou sečtena.

K vytvoření této stránky bylo využito znalosti html a PHP. Jazyk PHP implementuje rozšíření, které velmi usnadňuje použití SOAP. Obsahuje rozhraní DOM a práce s ním je velmi pohodlná a rychlá. Nejprve se vytvoří pomocí konstruktoru třídy `SoapClient()` instance klienta. Poté se zavolá její metoda `__soapCall()`, která přímo vrací výsledek. Dále následují funkce, které pouze naformátují a zobrazí výsledek.

Náhled testovací stránky je na obrázku *Obrázek 7.2*. Zdrojový kód této stránky je k nahlédnutí v příloze P III.

Ukazkova stranka s jednoduchym testem SOAP

Vyber prikazu:

Vyber baterie:

Ziskana data:

Napeti baterie:
3.85 V
3.95 V

Odeslana SOAP zprava:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" xmlns:ns1="http://10.3.3.10/" xmlns:xsd="
```

Prichazi SOAP zprava:

```
<?xml version="1.0" encoding="UTF-8"?><env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" x
<batteryVoltages xmlns="urn:mojeURI">
<battery id="0">
<voltage xsi:type="xsd:float">3.850000</voltage>
</battery><battery id="1">
<voltage xsi:type="xsd:float">3.950000</voltage>
</battery></batteryVoltages>
</env:Body></env:Envelope>
```

Strukturovany vypis promenne ve ktere je ulozen vysledek dotazu:

```
array
'battery' =>
  array
    0 =>
      object(stdClass) [4]
        public 'voltage' => float 3.85
    1 =>
      object(stdClass) [5]
        public 'voltage' => float 3.95
```

Obrázek 7.2 Ukázka webové stránky určené k otestování komunikace pomocí SOAP

ZÁVĚR

Cílem této bakalářské práce je analýza možnosti implementace protokolu SOAP pro mikropočítače. Nejprve tedy bylo nutné seznámit se s tímto protokolem. Protože však staví na technologiích HTTP a XML, bylo nezbytné seznámit se i s těmito dvěma.

Jelikož HTTP (a tím pádem i SOAP) využívá ke své činnosti rozhraní ethernet, bylo zapotřebí jej zohlednit při výběru mikropočítače. Byla provedena analýza možných řešení připojení mikropočítače do sítě ethernet spolu s analýzou předpokládané hardwarové náročnosti implementace SOAP. Závěrem je volba mikropočítače s 32bitovou délkou slova, protože tyto nabízejí dostatečné hardwarové prostředky.

Dalším bodem byl konkrétní výběr vhodného mikropočítače. Velké oblibě se v dnešní době těší mikropočítače s jádrem ARM Cortex M3, z tohoto důvodu byl vybrán velmi dostupný mikropočítač LPC1768 firmy NXP. Také bylo zjištěno, že jeho pořizovací cena je často nižší než u mnohých méně výkonných 8mi nebo 16ti bitových mikropočítačů. Byl zvolen vývojový kit čínského výrobce PowerMCU. Bohužel nízká cena byla vyvážena polofunkčními zdrojovými kódy s čínskými komentáři. Musel být například upraven ovladač LCD a naprogramována kalibrace dotykové plochy.

V dnešní době je trendem dostat co nejrychleji výrobek na trh, a proto důležité zajistit rychlý vývoj programu. Z tohoto důvodu jsou k tomuto mikropočítači dostupné knihovny a ukázkové kódy pro práci s rozhraním ethernet a protokolem HTTP. Pro účely vývoje byla zvolena knihovna TCPnet firmy Keil, která v případě přijetí SOAP zprávy zavolá naši knihovnu a předá jí celou zprávu. Návrh a implementace knihovny pro SOAP se tedy sestává pouze z parseru. Pro svou nenáročnost byl zvolen parser typu SAX.

Pro otestování byla vytvořena webová stránka na osobním počítači. SOAP požadavky jsou vytvářeny pomocí jednoduchého programu napsaného v jazyce PHP.

Na zmíněném mikropočítači byla bez problémů realizována a odladěna knihovna spolu s příkladem použití. Tuto práci lze využít k získání představy o požadavcích na hardware a složitosti implementace SOAP pro mikropočítače.

Knihovna by se dala rozšířit například o kontrolu správné syntaxe předávaných zpráv, zpracování jmenných prostorů nebo implementaci WSDL. WSDL dokument by však bylo vhodné umístit do externí paměti kvůli své velikosti.

ZÁVĚR V ANGLIČTINĚ

The aim of this work is analysis of options of implementation of SOAP protocol for microcontroller. First of all it was necessary get into the problematic with this protocol and the HTTP and XML technologies.

Because the HTTP (so the SOAP) operates by the Ethernet interface, it was necessary think of it during the selection of the microcontroller. The analysis of possible solutions of connecting the microcomputers to ethernet was made with an analysis of the expected hardware implementation of SOAP. The final choices are microcontrollers with 32-bit word length, because these offer sufficient hardware resources.

Another point of this work was the specific choice of a suitable microcomputer. Very popular is nowadays a microcontroller with core ARM Cortex M3, for this reason a very available microcomputer LPC1768 from NXP Company was chosen. Also, it was found that the price is often lower than the price of many less powerful 8-or 16-bit microcomputers. The development board from Chinese producer PowerMCU was chosen. Unfortunately, the low price was offset by half-working source codes with Chinese comments. The driver LCD had to be modified and the calibration of touch green had to be programmed.

Nowadays the trend is to get the product on the markets fast as possible, and therefore it is important to provide rapid development. For this reason the libraries and sample codes of this microcomputer for working with Ethernet and HTTP are available. For the purpose of development was chosen library TCPnet of Keil Company, which in case of reception of SOAP message calls our library and passes it the whole message. The design and the implementation of library for SOAP consist only of the parser. For its modesty was elected parser type SAX.

For the test was created a website on a personal computer. SOAP requests are generated by using a simple program written in PHP language.

At the mentioned microcontroller was easily realized library with example of use. This work can be used to get information about hardware requirements and complexity of SOAP implementation for microcontrollers.

Library could be extended for example for the control of the corrections of syntax of transmitted messages, processing namespaces or implementation of WSDL. WSDL document would be better to place into the external memory due to its size.

SEZNAM POUŽITÉ LITERATURY

- [1] KOSEK, Jiří. *Intelligentní podpora navigace na WWW s využitím XML*. Praha, 2002. Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Katedra informačního a znalostního inženýrství.
- [2] MARSHALL, James. *HTTP Made Really Easy: A Practical Guide to Writing Clients and Servers*. *HTTP Made Really Easy: A Practical Guide to Writing Clients and Servers* [online]. © 1997-2012, Last modified: December 10, 2012 [cit. 2013-05-22]. Dostupné z: <http://www.jmarshall.com/easy/http/>
- [3] RFC2616. *Hypertext Transfer Protocol – HTTP/1.1*. June 1999. Dostupné z: <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- [4] Mlýnková, Irena, Pokorný, Jaroslav. *XML Technologie*. 1. Vyd. Praha: Garda Publishing, a.s., 2008. 272 s. ISBN 978-80-247-2725-7
- [5] KOSEK, Jiří. *XML pro každého: podrobný průvodce*. 1. Vyd. Praha: Grada, 2000, 163 s. Průvodce (Grada). ISBN 80-716-9860-1.
- [6] SOAP Syntax. *W3schools.com* [online]. © 1999-2013 [cit. 2013-05-24]. Dostupné z: http://www.w3schools.com/soap/soap_syntax.asp
- [7] SOAP Example. *W3schools.com* [online]. © 1999-2013 [cit. 2013-05-24]. Dostupné z: http://www.w3schools.com/soap/soap_example.asp
- [8] Christensen, E. – Curbera, F. – Meredith, G. – Weerawarana, S.: *Web Services Description Language (WSDL) 1.1*. W3C Note. 2001. URL: <http://www.w3.org/TR/wsdl>
- [9] WSDL Document Example. *Tutorialspoint* [online]. © 2013 [cit. 2013-05-24]. Dostupné z: http://www.tutorialspoint.com/wsdl/wsdl_example.htm
- [10] PINKER, Jiří. *Mikroprocesory a mikropočítače*. Praha: BEN – technická literatura, 2004. ISBN 80-730-0110-1.
- [11] Ethernet with NXP Microcontrollers. In: *LPCware: NXP MCU COMMUNITY* [online]. 2012-06-06 [cit. 2013-05-25]. Dostupné z: <http://www.lpcware.com/content/project/nxp-peripherals/ethernet-nxp-microcontrollers>

- [12] MICROCHIP TECHNOLOGY INCORPORATED. *PIC18F97J60 Family Data Sheet: 64/80/100-Pin, High-Performance, 1-Mbit Flash Microcontrollers with Ethernet*. U.S.A., © 2011. ISBN 978-1-61341-069-1. Dostupné z: <http://ww1.microchip.com/downloads/en/DeviceDoc/39762f.pdf>
- [13] NXP B.V. *UM10360: LPC17xx User Manual*. 19 August 2010. Dostupné z: http://www.nxp.com/documents/user_manual/UM10360.pdf
- [14] WIZNET CO., Inc. *W5100 Datasheet*. 2011. Dostupné z: http://www.wiznet.co.kr/UpLoad_Files/ReferenceFiles/W5100_Datasheet_v1.2.4.pdf
- [15] DUNKELS, Adam. *The uIP TCP/IP stack*. [2003]. Dostupné z: <http://www.gaisler.com/doc/net/uip-0.9/doc/html/main.html>
- [16] JANÁSEK, Vojtěch. Praktické zkušenosti s procesory ARM Cortex M3. *Praktické zkušenosti s procesory ARM Cortex M3* [online]. 7. 4. 2009 [cit. 2013-05-26]. Dostupné z: <http://www.hw.cz/soucastky/embedded-systemy/mcu/prakticke-zkusenosti-s-procesory-arm-cortex-m3.html>
- [17] Úvod do architektury Cortex-M3 - díl. 1. *Úvod do architektury Cortex-M3 - díl. 1* [online]. 10. února 2010 [cit. 2013-05-26]. Dostupné z: http://pandatron.cz/?1252&uvod_do_architektury_cortex-m3_-_dil._1
- [18] ARM Development Tools: TCP/IP Networking Suite Memory Requirements. ARM LTD AND ARM GERMANY GMBH. *KEIL: Tools by ARM* [online]. © 2013 [cit. 2013-05-26]. Dostupné z: http://www.keil.com/rl-arm/rl-tcpnet_size.asp
- [19] RL-ARM User's Guide: SOAP Interface. ARM LTD AND ARM GERMANY GMBH. *KEIL: Tools by ARM* [online]. © 2013 [cit. 2013-05-26]. Dostupné z: http://www.keil.com/support/man/docs/rlarm/rlarm_tn_soap_if.htm
- [20] LandTiger Baseboard. In: *Mbed* [online]. 10 Dec 2012 [cit. 2013-05-26]. Dostupné z: <http://mbed.org/users/wim/notebook/landtiger-baseboard/>
- [21] ARM Ltd and ARM Germany GmbH. *Getting Started: Building Applications with RL-ARM* [online]. 2009. [cit. 2013-01-28]. Dostupné z: http://www.keil.com/product/brochures/rl-arm_gs.pdf

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SOAP Simple Object Access Protocol. (pouze do verze 1)

XML Extended Markup Language.

SAX Simple API for XML

DOM Dynamic Object Model

RPC Remote Procedure Call

DTD Document Type Definition

WSDL Web Services Description Language

UDDI Universal Description, Discovery and Integration

HTTP Hypertext Transfer Protocol

HTML Hyper Text Markup Language

URL Uniform Resource Locator

URI Uniform Resource Identifier

SMTP Simple Mail Transfer Protocol

CGI Common Gateway Interface

PHY Ethernet Physical Layer

API Application Programming Interface

SEZNAM OBRÁZKŮ

<i>Obrázek 1.1 Ukázka HTTP požadavku</i>	<i>12</i>
<i>Obrázek 1.2 Ukázka HTTP odpovědi.....</i>	<i>13</i>
<i>Obrázek 1.3 Ukázka XML dokumentu a její stromová struktura.....</i>	<i>15</i>
<i>Obrázek 1.4 Vztah tří základních technologií (SOAP, WSDL, UDDI) [1]</i>	<i>18</i>
<i>Obrázek 2.1 Zapojení procesoru z řadyPIC18FxxJ6xx do metalické sítě [12]</i>	<i>26</i>
<i>Obrázek 2.2 Blokové schéma zapojení procesoru z řady LPC1700 firmy NXP [13]</i>	<i>26</i>
<i>Obrázek 2.3 Struktura obvodu W5100 [14]</i>	<i>27</i>
<i>Obrázek 5.1 Pohled na zvolený vývojový kit [20]</i>	<i>33</i>
<i>Obrázek 5.2 Pohled na vývojové prostředí Keil μVision 4</i>	<i>34</i>
<i>Obrázek 7.1 Ukázka rozhraní simulujícího napětí článků.....</i>	<i>41</i>
<i>Obrázek 7.2 Ukázka webové stránky určené k otestování komunikace pomocí SOAP.....</i>	<i>44</i>

SEZNAM TABULEK

<i>Tabulka 3.1 Přehled vybraných paměťových nároků na procesor LPC1768 [18]</i>	<i>29</i>
<i>Tabulka 6.1 Přehled CGI příkazů</i>	<i>35</i>
<i>Tabulka 6.2 Přehled významů kódu dat</i>	<i>37</i>

SEZNAM PŘÍLOH

P I Zdrojový kód XML parseru

P II Funkce volané z parseru

P III Testovací stránka SOAPU

P IV CD disk s bakalářskou prací a zdrojové kódy

PŘÍLOHA P I: ZDROJOVÝ KÓD XML PARSERU

```
void parseXML(eventFunct *events, U8 *dat){
    int n,stE,p; U8 nfp,nfp_t; char **parN, **par;
    n=0; p=0;
    events->startDocument();
    while(dat[n]!=0){ //dokud není konec dokumentu, projizdi znak po znaku
        if(dat[n]!=' ' && dat[n]!='>' && dat[n]!='<' && dat[n]!=0x0a){
            stE=n;
            while(dat[n]!='<') n++;
            dat[n]=0;
            events->characters((char *) (dat+stE));
            p=1;
        }
        if(dat[n]=='<' || p==1){
            p=0;
            dat[n]=0;
            if(dat[n+1]=='/'){
                n+=2;
                while(dat[n]==' ') n++;
                stE=n;
                while(dat[n]!='>') n++;
                dat[n]=0;
                events->endElement((char *) (dat+stE));
            }else if(dat[n+1]=='?'){
                while(dat[n]!='>') n++;
                dat[n]=0;
            }else{
                n++;
                while(dat[n]==' ') n++;
                stE=n;
                nfp=0;
                while(dat[n]!='>'){
                    if(dat[n]==' ' && dat[n+1]!=' ' && dat[n+1]!='>' && dat[n+1]!='/'){
                        nfp++;
                        n++;
                        while(dat[n]!='"') n++;
                        n++;
                        while(dat[n]!='"') n++;
                    }
                    n++;
                }
                if(nfp>0){
                    parN=(char**) malloc(nfp*sizeof(char*));
                    par=(char**) malloc(nfp*sizeof(char*));
                    n=stE;
                    nfp_t=0;
                    while(dat[n]!='>'){
```

```

        if(dat[n]==' ' && dat[n+1]!=' ' && dat[n+1]!='>' && dat[n+1]!='/{') {
            dat[n]=0;
            parN[nfp_t]=(char *) (dat+n+1);
            while(dat[n]!='=') n++;
            dat[n]=0;
            while(dat[n]!='"') n++;
            n++;
            par[nfp_t]=(char *) (dat+n);
            while(dat[n]!='"') n++;
            dat[n]=0;
            nfp_t++;
        }
        n++;
    }
    dat[n]=0;
    events->startElement((char *) (dat+stE),parN,par,nfp);
    free(parN);
    free(par);
} else {
    dat[n]=0;
    events->startElement((char *) (dat+stE),NULL,NULL,0);
}
}
}
n++;
}
events->endDocument();
return;
}

```

PŘÍLOHA P II: FUNKCE VOLANÉ Z PARSERU

```
void f_startDocument(){
    //1000Bytes is enough for this example
    newXML_len=1000;
    strlen_newXML=0;
    newXML=(char *)malloc(1000);
}

void f_endDocument(){
}

void f_startElement(char *name, char **paramName, char **param, U8 n){
    int i;
    if(strstr(name,"getBatteryCellsVoltages")!=NULL){
        strlen_newXML += sprintf (newXML+strlen_newXML,"\n<batteryCellsVoltages
xmlns=\"urn:mojeURI\">\n");
        return;
    }

    if(strstr(name,"getBatteryCellsVoltage")!=NULL && n>0){
        for(i=0; i<n; i++){
            if(strstr(paramName[i],"id")!=NULL){
                strlen_newXML += sprintf (newXML+strlen_newXML,<battery
xmlns=\"urn:mojeURI\" id=\"%s\">\n",param[0]);
                for(i=0; i<(sizeof(testValues[0])/sizeof(testValues[0][0])); i++){
                    strlen_newXML += sprintf (newXML+strlen_newXML,<cell id=\"%d\">\n"
                                                                                       "<voltage
xsi:type=\"xsd:float\">%f</voltage>\n"
                    testValues[atoi(param[0])][i]);
                }
                strlen_newXML += sprintf (newXML+strlen_newXML,</battery>\n");
            }
        }
        return;
    }

    if(strstr(name,"getBatteryVoltages")!=NULL){
        strlen_newXML += sprintf (newXML+strlen_newXML,"\n<batteryVoltages
xmlns=\"urn:mojeURI\">\n");
        return;
    }

    if(strstr(name,"getBatteryVoltage")!=NULL && n>0){
        float sumVoltage;
        sumVoltage=0;
        for(i=0; i<(sizeof(testValues[0])/sizeof(testValues[0][0])); i++){
            sumVoltage+=testValues[atoi(param[0])][i];
        }
        strlen_newXML += sprintf (newXML+strlen_newXML,<battery id=\"%s\">\n"
```



```

        "<voltage xsi:type=\"xsd:float\">%f</voltage>\n"
        "</battery>", param[0], sumVoltage);

    return;
}

}

void f_endElement(char *name){
    if(strstr(name, "getBatteryCellsVoltages") != NULL){
        strlen_newXML += sprintf (newXML+strlen_newXML, "</batteryCellsVoltages>\n");
        return;
    }

    if(strstr(name, "getBatteryVoltages") != NULL){
        strlen_newXML += sprintf (newXML+strlen_newXML, "</batteryVoltages>\n");
        return;
    }
}

void f_characters(char *name){
}

//inicializace struktury s funkcemi - vraci ukazatel
eventFunct *initializeParserFunctions(void){
    static eventFunct events;
    events.startDocument=f_startDocument;
    events.endDocument=f_endDocument;
    events.startElement=f_startElement;
    events.startElement=f_startElement;
    events.endElement=f_endElement;
    events.characters=f_characters;
    return &events;
}

```

PŘÍLOHA P III: TESTOVACÍ WEBOVÁ STRÁNKA SOAPU

```
<!DOCTYPE html>

<html>

<body>

<h2> Ukazkova stranka s jednoduchym testem SOAP </h2><br>

<form action="index.php" method="get">

Vyber prikazu:

<select name="opt">

    <option value="cells">Napeti dilcich clanku</option>

    <option value="battery" <?php if(isset($_GET["opt"])) && isset($_GET["batid"]))
if($_GET["opt"]=="battery") echo"selected=\"selected\" " ?>>Napeti baterie</option>

</select><br>

Vyber baterie:

<select name="batid">

    <option value="0">0</option>

    <option value="1" <?php if(isset($_GET["batid"])) && isset($_GET["opt"]))
if($_GET["batid"]=="1") echo"selected=\"selected\" " ?>>1</option>

    <option value="all" <?php if(isset($_GET["batid"])) && isset($_GET["opt"]))
if($_GET["batid"]=="all") echo"selected=\"selected\" " ?>>Obe</option>

</select><br> <input type="submit" value="Proved">

</form><hr>

<?php

$URL = 'http://10.3.3.10/api.cgx';

$client = new SoapClient(null, array(

    'soap_version' => SOAP_1_2,

    'location' => $URL,

    'uri' => "http://10.3.3.10/",

    'trace' => 1,

));

if(isset($_GET["opt"])) && isset($_GET["batid"])){

    if($_GET["opt"]=="cells") $return = $client->__soapCall("getBatteryCellsVoltages",
($_GET["batid"]!="all") ? array(new SoapVar('<getBatteryCellsVoltage
id="'.$_GET["batid"].'" />', XSD_ANYXML)) : array(new SoapVar('<getBatteryCellsVoltage
id="0" />', XSD_ANYXML), new SoapVar('<getBatteryCellsVoltage id="1" />', XSD_ANYXML)));

    if($_GET["opt"]=="battery") $return = $client->__soapCall("getBatteryVoltages",
($_GET["batid"]!="all") ? array(new SoapVar('<getBatteryVoltage id="'.$_GET["batid"].'"
/>', XSD_ANYXML)) : array(new SoapVar('<getBatteryVoltage id="0" />', XSD_ANYXML), new
SoapVar('<getBatteryVoltage id="1" />', XSD_ANYXML)));

}

else $return = $client->__soapCall("getBatteryCellsVoltages", array(new
SoapVar('<getBatteryCellsVoltage id="0" />', XSD_ANYXML)));

?>

<p><b>Ziskana data:</b></p>

<?php

if(isset($_GET["opt"])) && isset($_GET["batid"])){

    // Zobrazeni napeti pouze pro vsechny baterie

    if($_GET["opt"]=="cells" && $_GET["batid"]=="all"){

        echo "Napeti jednotlivych clanku:<br>";

        foreach ($return['battery'] as &$valueN) {
```

```

        foreach ($valueN->cell as &$valueP){
            echo $valueP->voltage . " V <br>";
        }
        echo "<br>";
    }
    unset($valueN); unset($valueP);
}
if($_GET["opt"]=="battery" && $_GET["batid"]=="all"){
    echo "Napeti baterie:<br>";
    foreach ($return['battery'] as &$value) {
        echo $value->voltage . " V<br>";
    }
    unset($value);
}
// Zobrazeni napeti pouze pro jednotlivy baterie
if($_GET["opt"]=="cells" && $_GET["batid"]!="all"){
    echo "Napeti jednotlivych clanku:<br>";
    foreach ($return->cell as &$valueP){
        echo $valueP->voltage . " V <br>";
    }
    unset($valueP);
}
if($_GET["opt"]=="battery" && $_GET["batid"]!="all"){
    echo "Napeti baterie:<br>";
    //foreach ($return['battery'] as &$value) {
        echo $return->voltage . " V<br>";
    //}
    //unset($value);
}
}
else{
    echo "Napeti jednotlivych clanku:<br>";
    foreach ($return->cell as &$valueP){
        echo $valueP->voltage . " V<br>";
    }
    unset($valueP);
}
?>
<br><hr><br>
<?php
echo "<b>Odeslana SOAP zprava:</b><pre>" . htmlentities($client->__getLastRequest()) .
"</pre><br>";
echo "<b>Prichozí SOAP zprava:</b><pre>" . htmlentities($client->__getLastResponse()) .
"</pre><br>";
echo "<a><b>Strukturovaný výpis proměnné ve které je uložen výsledek dotazu:</b></a>\n";
var_dump($return);
?>
</body>
</html>

```