DIPLOMOVÁ PRÁCE

Vývoj nástrojů architektury orientované na služby pro vestavné řídicí systémy

Development of Service Oriented Architecture tools for embedded control systems

Plzeň, 2014

Autor: **Bc. Jiří Faist**

Vedoucí práce: **Ing. Pavel Balda, Ph.D.**

Prohlášení

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

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

V Plzni dne _____ _____
podpis

Abstrakt

Cílem práce je osvětlit přínos a možnosti nasazení webových služeb společně s vytvořením webové služby pro přístup k datům z řídicího systému *Rex*. V první kapitole se nachází rozbor technologie webových služeb se zaměřením na protokol SOAP a jeho doplňky. Druhá kapitola se věnuje popisu vývoje webových služeb ve dvou dostupných technologiích, které se pro tento úkol obvykle používají. Třetí kapitola se pak zaměřuje na technologie pro vývoj zařízení typu DPWS, které byly dále použity v kapitole čtvrté pro vývoj služby sloužící jako most mezi systémem *Rex* a klienty komunikujícími protokolem SOAP. Na závěr byla provedena série testů pro ověření výkonu aplikace a vhodnosti nasazení ve vestavných systémech.

Klíčová slova: webová služba, soap, dpws, wsdl, uddi, řídicí systém rex, ws-discovery, ws-eventing

Abstract

The goal of this work is to explain the benefits and the use case of web services together with the creation of a web service for simple access to data from the *Rex* control system. In the first chapter there is an analysis of web service technologies with focus on the SOAP protocol. Second chapter is dedicated to the description of the development of the web services with use of two available frameworks, that are commonly used for this task. Third chapter is focused on the technologies for development of the DPWS devices. These technologies were than used to implement a web service that serves as a bridge between the system *Rex* and clients communicating with the SOAP protocol. A series of tests was performed and there is a discussion about the performance of the service and suitability for embedded systems.

Key words: web service, soap, dpws, wsdl, uddi, rex control system, ws-discovery, ws-eventing

Obsah

Seznam obrázků

Seznam tabulek

Seznam textových ukázek

Úvod

Práce si klade za cíl prostudovat principy a technologie webových služeb a následně tyto nabyté znalosti využít pro implementaci aplikace, která bude zprostředkovávat data z řídicího systému *Rex* přes rozhraní webových služeb. Za webovou službu lze v širším slova smyslu považovat jakoukoliv službu, jejíž rozhraní je přístupné prostřednictvím internetu.

Důvodem zavedení webových služeb byla nutnost vytvoření protokolů jednoduché komunikace mezi aplikacemi v heterogenním prostředí. Toho je dosaženo použitím již existujících standardů, které nejsou závislé na platformě, především jazyka XML (*EXtensible Markup Language* – jazyk určený pro popis a přenos dat) a protokolu HTTP (*Hypertext Transfer Protocol* – síťový protokol pro výměnu hypertextových dokumentů). Práce je zaměřena na webové služby protokolu SOAP (*Simple Object Access Protocol* – protokol pro výměnu zpráv mezi aplikacemi založený na XML) a jeho rozšíření.

V první kapitole je čtenář seznámen s principy a se základními technologiemi webových služeb. Pro webové služby je také velice důležité, aby byly dobře vyhledatelné a aby bylo dobře popsané rozhraní, přes které lze se službou komunikovat. Proto je v této části také nastíněna typická struktura dokumentů v jazyce WSDL (*Web Services Description Language* – jazyk pro popis rozhraní webové služby) a také zde najdeme rozbor dvou rozdílných přístupů k vyhledávání webových služeb.

Druhá kapitola se zaměřuje na dva frameworky(softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci softwarových aplikací), které se používají pro vytvoření webových služeb typu SOAP. Jejich funkčnost je představena na příkladu jednoduché webové služby. Vzhledem k tomu, že cílenou platformou pro výslednou službu pro systém *Rex* jsou vestavné systémy (jednoúčelové systémy, ve kterých je řídicí počítač zcela zabudován do zařízení, které ovládá), které mají obvykle limitované prostředky co se týče procesorové síly i operační paměti, je výběr frameworků omezen na ty, které jsou implementovány v jazyce `C`.

Ve třetí kapitole se od implementace jednoduchých služeb přesuneme ke službám,

které splňují specifikaci DPWS (*Devices Profile for Web Services* – specifikace opírající se o protokol SOAP definující způsob komunikace a interakce s obvykle jednoduchými zařízeními). Jsou zde opět představeny dva frameworky v jazyce `C`, které se pro tento úkol používají.

V poslední kapitole se pak nachází popis vytvořené služby pro řídicí systém *Rex*, jejího rozhraní a vnitřní architektury . Pro výslednou službu bylo provedeno několik zátěžových testů, jejichž výsledky se nachází rovněž v této kapitole.

Kapitola 1

Technologie Webových služeb

1.1 SOA, Webové služby

Webová služba je jakákoliv služba, která je dostupná prostřednictvím internetu, používá standardizovaný formát zpráv (typicky ve formě XML) a není nijak vázána na určitý operační systém či programovací jazyk [Cerami(2002)]. U webových služeb se také typicky klade důraz na dvě vlastnosti. Za prvé, každá nově zveřejněná služba by měla být sebe popisující. Měla by využívat nějaký mechanismus (obvykle s využitím XML), který uživateli popíše jakou funkci daná webová služba zprostředkovává a přes jaké rozhraní s ní může komunikovat. Za druhé, webová služba by měla být jednoduše přístupná a snadno objevitelná.

Nabízí se otázka, proč vytvářet webové služby a jaký přínos webové služby nabízejí. Typické komunikační schéma uplatňující se na webu zahrnuje nějakou osobu, která odesílá požadavek na server, který jí obratem posílá svou odpověď v podobě dokumentu v jazyce HTML (*HyperText Markup Language* – značkovací jazyk pro vytváření webových stránek). Lidský faktor v tomto případě hraje zásadní roli a proto se pro tento model používá termín *human-centric web* (viz obr. 1.1). Naproti tomu existuje model webu označovaný termínem *application-centric web*, ve kterém roli uživatele mohou převzít aplikace a tedy komunikace mezi aplikacemi může probíhat stejně snadno jako mezi uživatelem a serverem (viz. obr. 1.2). A zde přicházejí ke slovu webové služby.

Webová služba může být chápána jako aplikace vytvořená za účelem integrace do dalších aplikací. V některých případech je dokonce možné tuto integraci automatizovat za podmínky, že služba je jednoduše objevitelná, sebe popisující a drží se obvyklých standardů.

Architektura webových služeb

Systém webových služeb lze rozdělit do několika vrstev, přičemž každé z nich náleží určitá množina protokolů.

- **Transportní protokol** zabezpečuje přenos zpráv mezi aplikacemi. (HTTP, SMTP[1], FTP [2])
- **Protokol zpráv** určuje způsob zaznamenání dat typicky v XML formátu. (XML-RPC[3], SOAP, RESTful[4])
- **Protokol pro popis služby** je odpovědný za popis rozhraní služby. (WSDL)
- **Protokol zveřejnění webové služby** je protokol služby, která zajišťuje, aby nová webová služba byla snadno objevitelná a přístupná. (UDDI [5], WS-Discovery [6])

1.2 Protokol SOAP

SOAP [W3C(2000)] je protokol definující tvar zpráv předávaných mezi aplikacemi založený na XML. Zprávy ve formátu SOAP přenášené typicky přes HTML protokol jsou základem webových služeb. SOAP zpráva je obyčejný XML dokument, skládající se z následujících prvků[W3S(2013)]:

- `Envelope` (obálka) – kořenový (*root*) element, který identifikuje XML dokument jako zprávu SOAP
- `Header` (hlavička) – element obsahující některé řídicí parametry, které určují, jak má příjemce zprávu zpracovávat, hlavička je nepovinná, typicky obsahuje informace specifické pro danou aplikaci. Předdefinované atributy:

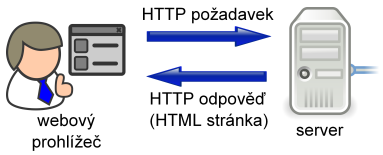
[1] *Simple Mail Transfer Protocol* – internetový protokol určený pro přenos zpráv elektronické pošty
[2] *File Transfer Protocol* – internetový protokol určený pro přenos souborů mezi počítači po počítačové sítě
[3] XML-*Remote procedure call* – je protokol pro provádění vzdáleného volání procedur
[4] *Representational state transfer services* – je protokol webových služeb implementujících architekturu, které se říká REST
[5] *Universal Description, Discovery and Integration* – centralizovaný mechanismus pro zveřejňování a vyhledávání webových služeb
[6] decentralizovaný mechanismus pro zveřejňování a vyhledávání webových služeb

  - `mustUnderstand` – určuje, zda příjemce je povinen zpracovat se zprávou i hlavičku (musí jí rozumět)
  - `actor` – zpráva může cestovat od odesílatele k příjemci přes více koncových bodů a tento atribut umožňuje směřovat hlavičku právě některému z nich

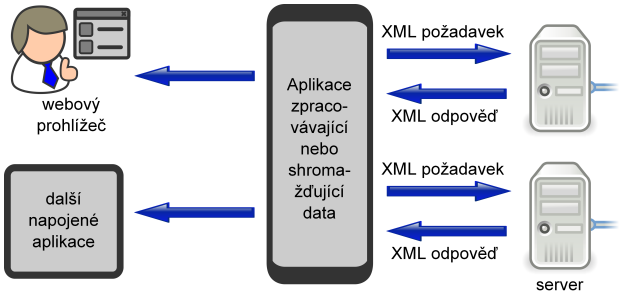
- `Body` (tělo) – element obsahující přenášená data
- `Fault` (porucha) – element s chybovými a stavovými informacemi, který se skládá z elementů:
  - `<faultcode>` – kód identifikující druh chyby
  - `<faultstring>` – textový popis chyby
  - `<faultactor>` – původce chyby
  - `<detail>` – bližší informace o chybě specifické pro danou aplikaci

SOAP zpráva navíc musí využívat jmenné prostory `SOAP Envelope namespace` a `SOAP Encoding namespace`. Princip jmenných prostorů je součástí XML specifikace a zajišťuje jedinečnost a snadné rozlišení elementů a atributů v XML dokumentu.

Každý z elementů SOAP zprávy může obsahovat atribut `encodingStyle`, který definuje použité datové typy v tomto elementu a elementech v něm obsažených. Ukázka kostry SOAP zprávy se nachází na obrázku 1.3.



Obrázek 1.1: *human-centric* web



Obrázek 1.2: *application-centric* web

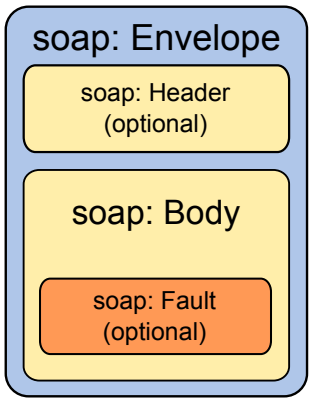
```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Header>
    ...
    </soap:Header>

    <soap:Body>
    ...
        <soap:Fault>
        ...
        </soap:Fault>
    </soap:Body>

</soap:Envelope>
```

(a) Skeleton SOAP zprávy



(b) Struktura SOAP zprávy

Obrázek 1.3: Schéma SOAP zpráv

V současné době na oblibě získávají webové služby typu RESTful. Zjednodušeně by se dalo říci, že největší rozdíl mezi nimi je ten, že zatímco SOAP používá formát XML zpráv s poměrně přesně daným formátem, který je případně rozšiřován dalšími specifikacemi závislými na protokolu SOAP, u webových služeb typu RESTful není formát přenášených dat pevně stanoven žádnou specifikací, využívá se zde krom formátu XML také formát JSON (*JavaScript Object Notation* – jednoduchý formát zápisu dat založený na způsobu deklarace objektů v programovacím jazyce JavaScript).

Tato práce se zabývá pouze webovými službami typu SOAP a proto pokud se v následujících kapitolách bude hovořit o webových službách, jsou tím myšleny pouze webové služby typu SOAP.

1.3 Jazyk WSDL

Jazyk WSDL (*Web Services Description Language*) slouží pro popis webových služeb a jejich rozhraní [W3C(2001)]. Je to jazyk vzniklý z jazyka XML, dokument v jazyce WSDL začíná kořenovým elementem `<definitions>` ze jmenného prostoru

`http://schemas.xmlsoap.org/wsdl/` [Skonnard(2003)]. WSDL také specifikuje umístění služby. Jazyk WSDL používá následující elementy:

- `<types>` – element, který obsahuje definice datových typů podle specifikace XML Schema používaných webovou službou
- `<message>` – definuje zprávu jejím jménem a datovými typy, které obsahuje
- `<portType>` – definuje skupinu operací, každá operace je množinou vstupních a výstupních zpráv
- `<binding>` – definuje konkrétní protokol a formát dat pro konkrétní `portType`
- `<service>` – definuje kolekci portů (koncových bodů), které jsou spojeny s protokolem a formátem dat přes element `binding`

V textových ukázkách 1, 2, 3, 4 a 5 jsou příklady vyjmuté z jednoduchého ukázkového WSDL dokumentu pro webovou službu se základními matematickými operacemi. Z příkladu jsou pro jednoduchost vynechány všechny části týkající se operací jiných než sčítání. V příkladu 1 najdeme nejprve definice jmenných prostorů a poté definice `<types>` datových typů `MathInput` a `MathOutput`. Vstupní typ je složený ze dvou elementů typu `double`, výstupní obsahuje jeden element stejného typu. Dále jsou zde definované elementy `Add` resp. `AddResponse` typu `MathInput` resp. `MathOutput`. Tyto elementy jsou dále použity v příkladu 2, kde jsou definované zprávy `AddMessage` a `AddResponseMessage`.

V příkladu 3 je definice rozhraní `portType` pojmenované jako `MathInterface` s operací sčítání `Add`. Tato operace obsahuje vstupní zprávu `AddMessage` a výstupní zprávu `AddResponseMessage` definované výše. V příkladu 4 se pak definuje `binding` pojmenované `MathSoapHttpBinding` pro rozhraní `MathInterface`. Zde se volí atribut `styl`, který definuje, jakým způsobem se překládá `binding` do SOAP zprávy. Za transportní protokol je zvolen HTTP. Dále se zde nastavuje, zda data přenášená ve zprávě jsou srozumitelná čistě ve formě, v jaké jsou přenášena (`literal`), nebo zda existují nějaká pravidla mimo WSDL soubor, která určují jak nějaká data byla serializována do XML a jak se mají deserializovat z XML (`encoded`). V poslední části WSDL souboru se nachází definice služby s množinou portů spárovanými s danými `bindings`. Port je dále definovaný svou adresou. Na úplném konci je ukončen element `<definitions>`.

Textová ukázka 1 WSDL – definice jmenných prostorů a datových typů

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

  ...

  targetNamespace="http://example.org/math/">
   <types>
     <xs:schema targetNamespace="http://example.org/math/types/"
       xmlns="http://example.org/math/types/"
       elementFormDefault="unqualified" attributeFormDefault="unqualified">
         <xs:complexType name="MathInput">
            <xs:sequence>
               <xs:element name="x" type="xs:double"/>
               <xs:element name="y" type="xs:double"/>
            </xs:sequence>
         </xs:complexType>
         <xs:complexType name="MathOutput">
            <xs:sequence>
               <xs:element name="result" type="xs:double"/>
            </xs:sequence>
         </xs:complexType>
         <xs:element name="Add" type="MathInput"/>
         <xs:element name="AddResponse" type="MathOutput"/>

        ...

     </xs:schema>
   </types>
```

Textová ukázka 2 WSDL – definice zpráv

```
   <message name="AddMessage">
      <part name="parameters" element="ns:Add"/>
   </message>
   <message name="AddResponseMessage">
      <part name="parameters" element="ns:AddResponse"/>
   </message>

   ...
```

Textová ukázka 3 WSDL – definice rozhraní `portType`

```
   <portType name="MathInterface">
      <operation name="Add">
         <input message="y:AddMessage"/>
         <output message="y:AddResponseMessage"/>
      </operation>

  ...

   </portType>
```

Textová ukázka 4 WSDL – definice navázání `binding`

```
   <binding name="MathSoapHttpBinding" type="y:MathInterface">
      <soap:binding style="document"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="Add">
         <soap:operation soapAction="http://example.org/math/#Add"/>
         <input>
            <soap:body use="literal"/>
         </input>
         <output>
            <soap:body use="literal"/>
         </output>
      </operation>

  ...

   </binding>
```

Textová ukázka 5 WSDL – definice služby

```
   <service name="MathService">
      <port name="MathEndpoint" binding="y:MathSoapHttpBinding">
         <soap:address location="http://localhost/math/math.asmx"/>
      </port>
   </service>
</definitions>
```

1.4 UDDI

UDDI (*Universal Description, Discovery and Integration*) [OAS(2004)] je technická specifikace pro popis, vyhledávání a integraci webových služeb [Cerami(2002)]. Jejím cílem je nabídnout firmám mechanismus nejen pro jejich vyhledávání, ale i publikování. Podle této specifikace se dále implementují UDDI registry pro správu dostupných webových služeb. Příkladem může být `jUDDI` implementace v javě pod licencí Apache [Apa(2004)].

UDDI API[7] je protokol založený na výměně zpráv protokolu SOAP určený pro manipulování s UDDI registry [Cerami(2002)]. Rozhraní lze rozdělit na dvě části. Dotazovací rozhraní (*Inquiry API*) je určeno pro vyhledávání v registrech a Publikační rozhraní (*Publisher API*) je určeno pro zveřejňování služeb.

Dotazovací rozhraní lze rozdělit do dvou skupin podobných funkcí a to skupina funkcí `find` pro nalezení záznamu a skupina funkcí `get` pro získání bližších údajů o nalezeném záznamu. Soupis funkcí je uveden níže. Ze jmen funkcí lze snadno odhadnout jejich účel.

V příloze v ukázkových textech 19 a 20 se nachází obsah elementu `<Body>` protokolu SOAP při volání funkce `find_business`. V ukázkových textech 21 a 22 je poté ukázka volání funkce `get_businessDetail`.

`find_xxx` funkce	`get_xxx` funkce
• `find_business`	• `get_businessDetail`
• `find_binding`	• `get_bindingDetail`
• `find_service`	• `get_serviceDetail`
• `find_tModel`	• `get_tModelDetail`

Publikační rozhraní slouží pro uveřejnění nových služeb. Požadavky posílané publikačním rozhraním musí být odeslané na zabezpečený URI (*Uniform Resource Identifier* – textový řetězec s definovanou strukturou pro přesné určení zdroje informací), který musí být odlišný od URI pro dotazovací požadavky. Podobně jako dotazovací API lze

[7] *Application Programming Interface* – je rozhraní, které určuje, jak se má přistupovat k softwarovým komponentám

i zde rozdělit funkce do tří skupin, funkce pro autentizaci, funkce pro uložení záznamu a funkce pro odstranění záznamu.

funkce pro autentizaci	`save_xxx` funkce	`delete_xxx` funkce
• `get_authToken`	• `save_business`	• `delete_business`
• `discard_authToken`	• `save_binding`	• `delete_binding`
	• `save_service`	• `delete_service`
	• `save_tModel`	• `delete_tModel`

Publikování UDDI dat může provádět pouze autorizovaný uživatel. Každá implementace UDDI registru obsahuje vlastní způsob autentizace, ale UDDI API vyžaduje, aby u každé operace `save_xxx` i `delete_xxx` byl přiložen autentizační token, takže před provedením takové operace je nutné nejprve požádat o přidělení autentizačního tokenu funkcí `get_authToken`, která požaduje uvedení uživatelského jména a hesla.

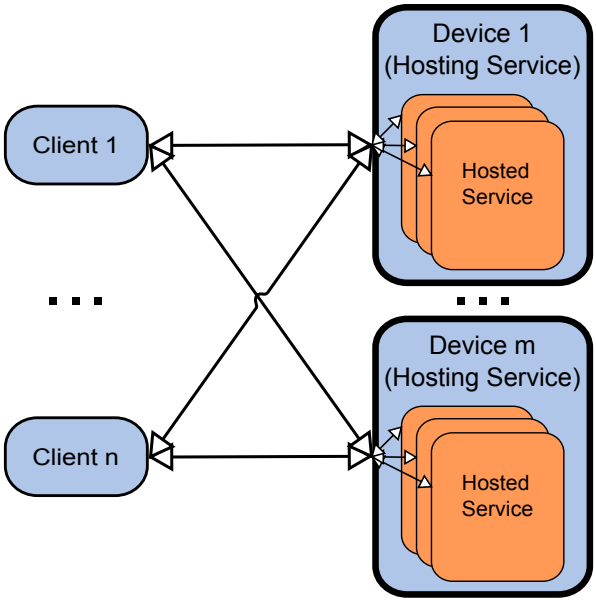
Příklad autentizace je uveden v příloze v textových ukázkách 23 a 24. V ukázce 25 je poté příklad uložení informace o nové společnosti s použitím získaného autentizačního tokenu z ukázky 24.

1.5 DPWS – Devices Profile for Web Services

Webové služby obsahují celý balík specifikací, které definují bohatou funkcionalitu a které mohou být kombinovány pro splnění nejrůznějších požadavků. Pro zvýšení interoperability mezi implementacemi webových služeb a různými flexibilními (resp. adaptivními) klienty tento profil definuje základní soubor specifikací webových služeb v těchto oblastech [OAS(2009a)]:

- odesílání zabezpečených zpráv webovým službám a zpět
- dynamické vyhledávání webových služeb – viz sekce 1.5.1
- popis webových služeb – přenos WSDL dokumentu
- mechanismus pro přihlašování a následný příjem asynchronních zpráv generovaných událostmi webové služby – viz sekce 1.5.2

DPWS definuje architekturu, která rozlišuje dva typy služeb: Služba zařízení (*Hosting Service*) a hostované služby (*Hosted Service*) [SOA(2012)]. Služba zařízení hraje zásadní roli při vyhledávání zařízení a při výměně informací o zařízení. Podoba hostovaných služeb je pak plně závislá na dané aplikaci. Hostované služby implementují funkční stránku zařízení a jsou vystavovány službou zařízení (viz obr. 1.4). DPWS se obvykle využívá pro menší zařízení, typickým příkladem může být síťová tiskárna apod.



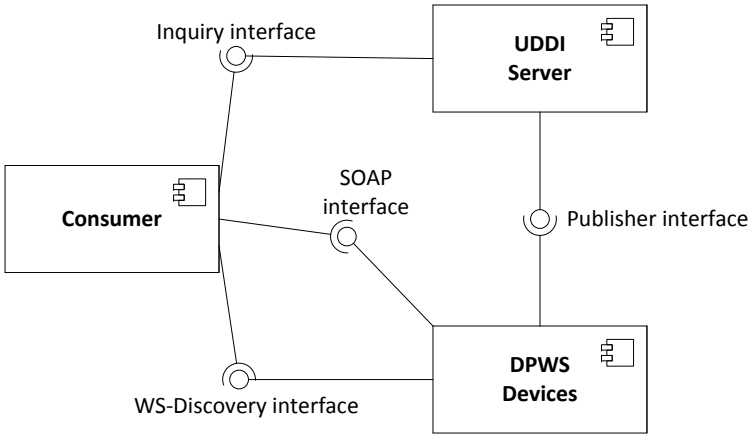
Obrázek 1.4: DPWS – architektura služeb

Na obrázku 1.5 je nakreslen UML[8] Component diagram (diagram pro zobrazení souvislostí mezi jednotlivými softwarovými komponentami) DPWS zařízení a jeho interakce s okolím. Specifikace DPWS určuje, že zařízení musí být vyhledatelné pomocí mechanismu definovaného specifikací *WS-Discovery* (viz sekce 1.5.1) a služby DPWS zařízení jsou volatelné přes SOAP rozhraní. Služby zařízení samozřejmě mohou být také vystavené centralizovaným způsobem nezávisle na DPWS specifikaci a to prostřednictvím UDDI registru.

1.5.1 WS-Discovery

WS-Discovery [OAS(2009b)] je protokol určený primárně pro vyhledání webových služeb a je zároveň jednou z nedílných součástí specifikace DPWS zařízení. Základem je výměna zpráv typu SOAP přes UDP protokol. V tomto případě jsou formy zpráv

[8] *Unified Modeling Language* – grafický jazyk pro návrh a dokumentaci softwarových systémů



Obrázek 1.5: DPWS – UML Component diagram DPWS zařízení a jeho okolí.

dopředu známy a definovány a není tedy potřeba sdílet žádný WSDL soubor, který by službu WS-Discovery popisoval tak, jak je tomu běžné u klasických webových služeb.

Komunikace přes WS-Discovery protokol je znázorněna na obrázku 1.6. Každé DPWS zařízení, které se připojí do sítě musí vyslat multicast zprávu `Hello`, ve které ohlašuje všem poslouchajícím zařízením, že se právě připojilo (viz Textová ukázka 6). Obdobně, pokud se zařízení odpojuje od sítě, odešle multicast zprávu `Bye` (viz Textová ukázka 7).

Klient, který zařízení vyhledává, odešle nezávisle na příchozích nebo odchozích `Hello` nebo `Bye` zprávách zprávu `Probe` opět jako multicast (viz Textová ukázka 8). Všechna připojená DPWS zařízení poslouchají na této multicast adrese a pokud přijmou `Probe` zprávu, pak na ni odpovídají zprávou `ProbeMatch` v případě, že splňují parametry, které jsou v `Probe` zprávě definovány (viz Textová ukázka 9). Těmito parametry klient určuje, jaké zařízení hledá. Po úspěšné výměně `Probe` a `ProbeMatch` klient může použít zprávu `Resolve`, kterou se vyptává přímo po IP adresách, na kterých je dané zařízení k dosažení. Se znalostí cílové IP adresy už může klient volat operace nalezené webové služby.

Specifikace také dovoluje vytvoření prostředníka ("WS-Discovery Proxy"), který při vyhledávání komunikuje místo připojených zařízení a tím šetří provoz v síti.

Textová ukázka 6 WS-Discovery – `Hello` zpráva pro oznámení připojení se k síti

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-
    envelope" xmlns:ns0="https://www.rexcontrols.cz/soa" xmlns:wdp="http
    ://schemas.xmlsoap.org/ws/2006/02/devprof" xmlns:wsa="http://schemas
    .xmlsoap.org/ws/2004/08/addressing" xmlns:wsd="http://schemas.
    xmlsoap.org/ws/2005/04/discovery">
  <SOAP-ENV:Header>
    <wsa:To>urn:schemas-xmlsoap-org:ws:2005:04:discovery</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2005/04/discovery/Hello</
        wsa:Action>
    <wsa:MessageID>urn:uuid:00000066-00e7-4d30-a9fc-000000450037</wsa:
        MessageID>
    <wsd:AppSequence InstanceId="110" MessageNumber="1"/>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <wsd:Hello>
      <wsa:EndpointReference>
        <wsa:Address>urn:uuid:0000268b-0000-1000-8000-d8d38539c9e0</wsa
            :Address>
      </wsa:EndpointReference>
      <wsd:Types>ns0:WSerBridge wdp:Device</wsd:Types>
      <wsd:Scopes/>
      <wsd:XAddrs>http://169.254.169.75:9867/0000268b-0000-1000-8000-
          d8d38539c9e0</wsd:XAddrs>
      <wsd:MetadataVersion>1</wsd:MetadataVersion>
    </wsd:Hello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Textová ukázka 7 WS-Discovery – `Bye` zpráva pro oznámení odchodu ze sítě

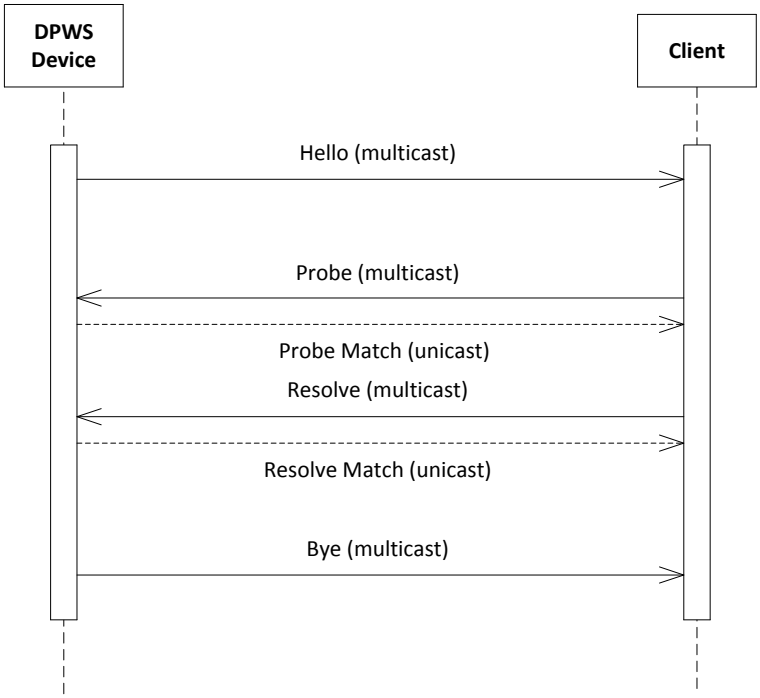
```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-
    envelope" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
    addressing" xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/
    discovery">
  <SOAP-ENV:Header>
    <wsa:To>urn:schemas-xmlsoap-org:ws:2005:04:discovery</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2005/04/discovery/Bye</
        wsa:Action>
    <wsa:MessageID>urn:uuid:0000003b-0026-4856-80c1-0000004600cc</wsa:
        MessageID>
    <wsd:AppSequence InstanceId="110" MessageNumber="3"/>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <wsd:Bye>
      <wsa:EndpointReference>
        <wsa:Address>urn:uuid:0000268b-0000-1000-8000-d8d38539c9e0</wsa
            :Address>
      </wsa:EndpointReference>
    </wsd:Bye>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Textová ukázka 8 WS-Discovery – `Probe` zpráva pro vyhledání zařízení v síti

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-
    envelope" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
    addressing" xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/
    discovery">
  <SOAP-ENV:Header>
    <wsa:To>urn:schemas-xmlsoap-org:ws:2005:04:discovery</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2005/04/discovery/Probe</
        wsa:Action>
    <wsa:MessageID>urn:uuid:000000db-0070-4ac1-ab2a-000000b80050</wsa:
        MessageID>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <wsd:Probe>
      <wsd:Types xmlns:_0="https://www.rexcontrols.cz/soa" xmlns:_1="
          http://schemas.xmlsoap.org/ws/2006/02/devprof">_0:WSerBridge
          _1:Device</wsd:Types>
    </wsd:Probe>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Textová ukázka 9 WS-Discovery – `ProbeMatches` zpráva odpovědi na zprávu `Probe`

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope" xmlns:ns0="https://www.rexcontrols.cz/soa" xmlns:wdp="http://schemas.xmlsoap.org/ws/2006/02/devprof" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/discovery">
  <SOAP-ENV:Header>
    <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2005/04/discovery/ProbeMatches</wsa:Action>
    <wsa:MessageID>urn:uuid:00000017-0075-4153-b494-0000000f0092</wsa:MessageID>
    <wsa:RelatesTo>urn:uuid:000000db-0070-4ac1-ab2a-000000b80050</wsa:RelatesTo>
    <wsd:AppSequence InstanceId="110" MessageNumber="2"/>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <wsd:ProbeMatches>
      <wsd:ProbeMatch>
        <wsa:EndpointReference>
          <wsa:Address>urn:uuid:0000268b-0000-1000-8000-d8d38539c9e0</wsa:Address>
        </wsa:EndpointReference>
        <wsd:Types>ns0:WSerBridge wdp:Device</wsd:Types>
        <wsd:Scopes/>
        <wsd:XAddrs>http://169.254.169.75:9867/0000268b-0000-1000-8000-d8d38539c9e0</wsd:XAddrs>
        <wsd:MetadataVersion>1</wsd:MetadataVersion>
      </wsd:ProbeMatch>
    </wsd:ProbeMatches>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



Obrázek 1.6: WS-Discovery – UML Sequence diagram komunikace.

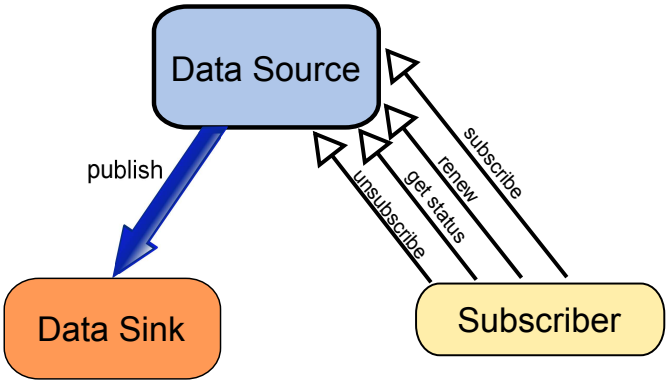
1.5.2 WS-Eventing

WS-Eventing [W3C(2011)] je specifikace popisující protokol dovolující cílové webové službě nebo klientovi ("*event sink*") zaregistrovat u zdrojové webové služby zájem ("*subscription*") o získávání zpráv o událostech ("*notifications*") ze zdrojové služby ("*event source*") [SOA(2012)]. To umožňuje klientovi získávat průběžně informace, aniž by musel neustále odesílat požadavek serveru. Tomuto způsobu komunikace se také někdy říká mechanismus *subscribe/publish*. Klientské *subscription* je vždy časově omezené a klient si ho musí po uplynutí určité doby obnovit.

Koncový bod, který posílá požadavek pro vytvoření *subscription*, se nazývá "*subscriber*". Často je *subscriber* a *event sink* jeden koncový bod, ale není to podmínkou. Funkčnost, kterou zajišťuje tato specifikace je vyobrazena na obrázku 1.7. V textové ukázce 10 je ukázka zprávy, kterou se klient přihlašuje k odběru notifikací. Za zmínku stojí element `wse:Expires`, ve kterém klient definuje požadovanou trvanlivost svého *subscription*.

Textová ukázka 10 WS-Eventing – zpráva přihlášení se k odběru notifikací

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-
    envelope" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
    addressing" xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/
    eventing" xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.
    xsd">
  <SOAP-ENV:Header>
    <wsa:To>http://169.254.169.75:9867/BridgeServices</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/08/eventing/
        Subscribe</wsa:Action>
    <wsa:MessageID>urn:uuid:00000006-003f-4c69-bfa2-0000007100f4</wsa:
        MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/
          role/anonymous</wsa:Address>
    </wsa:ReplyTo>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <wse:Subscribe>
      <wse:EndTo>
        <wsa:Address>http://169.254.169.75:8834/00000044-00bf-4ee8-96b3
            -0000004d00bb</wsa:Address>
      </wse:EndTo>
      <wse:Delivery Mode="http://schemas.xmlsoap.org/ws/2004/08/
          eventing/DeliveryModes/Push">
        <wse:NotifyTo>
          <wsa:Address>http://169.254.169.75:8834/00000044-00bf-4ee8-96
              b3-0000004d00bb</wsa:Address>
        </wse:NotifyTo>
      </wse:Delivery>
      <wse:Expires>PT1H</wse:Expires>
    </wse:Subscribe>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



Obrázek 1.7: WS-Eventing – mechanismus *subscribe/publish*

Kapitola 2

Vývoj Webových služeb

Webové služby typu SOAP jsou obvykle kombinací několika z mnoha specifikací, které protokol SOAP doplňují. Navíc zpracování (parsování[1] zpráv XML) je také velice komplikovaná úloha. Proto se při vývoji aplikací s rozhraním webových služeb využívá téměř vždy nějaký soubor podpůrných prostředků (framework), který tuto úlohu zjednodušuje a pomáhá udržet interoperabilitu mezi aplikacemi.

V zásadě existují dva typy vývojových nástrojů pro webové služby:

- nástroje založené na generování zdrojového kódu pro parsování, serializaci[2] odchozích zpráv a deserializaci příchozích zpráv (např. framework *gSOAP*)
- nástroje, které zajišťují serializaci a deserializaci bez nutnosti generovat specifický kód, parsování zprávy ale musí provést vývojář sám (např. framework *Axis2/C*)

2.1 gSOAP

Nástroj *gSOAP* zajišťuje automatické mapování SOAP a XML dat na struktury jazyka C a C++. Mapování je založené na kompilační technologii. Nástroj je určen pro zjednodušení vývoje webových služeb *gSOAP* pro aplikace v jazyce C a C++ automatickým generováním zdrojového kódu a pokročilých metod mapování [van Engelen(2013)].

[1] parsování neboli syntaktická analýza je takový proces, při kterém se zkoumá posloupnost formálních prvků s cílem určit jejich gramatickou strukturu vůči předem dané formální gramatice. viz *Slovník pojmů*: parsování

[2] serializace je proces, při kterém se převádí libovolně složitý objekt do své sériové(sekvenční) podoby. Složité datové struktury se tak z paměti počítače převedou na posloupnost bitů, která se pak může například přenést po síti

Díky tomu se uživatel nemusí starat o zpracovávání specifických detailů WSDL souborů, SOAP zpráv a XML formátu přenášených dat a jediné, co musí uživatel vyvíjet, je samotná logika aplikace. Mapování do C a C++ navíc zajišťuje typovou bezpečnost příchozích i odchozích XML dat.
Samotný systém *gSOAP* se skládá ze dvou částí:

- `wsdl2h` – zpracovává WSDL soubor a vytváří speciálně strukturovaný hlavičkový soubor, který je dále zpracováván nástrojem `soapcpp2`
- `soapcpp2` – zpracovává hlavičkový soubor a generuje stub rutiny (funkce, kterou používá klient pro vzdálené volání funkce na serveru, více viz *Slovník pojmů:* stub rutina) pro klientskou aplikaci a skelety funkcí, které obstarávají serializaci a deserializaci na straně webové služby.

Oba tyto nástroje jsou zkompilované a dostupné pro operační systémy Windows, Linux a Mac OS. Zdrojový kód generovaný aplikací *gSOAP* je na všech platformách totožný a je tedy možné tento kód transportovat a zkompilovat ho přímo na cílové platformě.

2.1.1 Příklad aplikace – Kalkulačka

Jako příklad funkce systému *gSOAP* je dále uvedeno vytvoření jednoduché webové služby, která obsahuje funkce pro jednoduché matematické operace typu sčítání, odčítání, násobení a dělení. V první části je vytvořena samotná webová služba a ve druhé je vytvořen klient, který webovou službu využívá.

2.1.1.1 Webová služba

gSOAP umožňuje vytvoření webové služby v podobě CGI aplikace (*Common Gateway Interface* – skript generující dynamické WWW stránky, je to jakýkoliv na straně serveru spustitelný soubor, který po spuštění zapíše na výstup HTTP hlavičku a poté obvykle obsah v HTML) a nebo jako samostatnou službu s využitím jednoduchého vestavěného HTTP serveru.

Oba způsoby se liší pouze v implementaci funkce `main()` (viz Textová ukázka 26). Zde bude předvedena druhá možnost.

Postup vytvoření služby

1. Vytvoříme hlavičkový soubor reprezentující službu. Můžeme si ho buď nechat vygenerovat programem `wsdl2h`, pokud je WSDL soubor dostupný a nebo ho můžeme napsat sami. V textové ukázce 11 je uveden hlavičkový soubor pro naši službu webové kalkulačky.

   V komentářích hlavičkového souboru se mohou nacházet parametry, které upravují některé vlastnosti protokolu SOAP:

   - //gsoap <ns> service name: <WSDLserviceName> <documentationText>
   - //gsoap <ns> service style: [rpc|document]
   - //gsoap <ns> service encoding: [literal|encoded]
   - //gsoap <ns> service namespace: <WSDLnamespaceURI>
   - //gsoap <ns> service location: <WSDLserviceAddressLocationURI>
   - //gsoap <ns> service method-style: <methodName> [rpc|document]
   - //gsoap <ns> service method-encoding: <methodName> [literal|encoded]
   - //gsoap <ns> service method-action: <methodName> <actionString>
   - //gsoap <ns> service method-documentation: <methodName> <documentation>

   V našem případě je využit komentář obsahující jmenný prostor webové služby. Ke jménu funkce se přidává prefix `ns__`, který označuje, že tato metoda je součásti jmenného prostoru webové služby. U definice funkce je zvykem, že všechny argumenty jsou považovány za vstupní, kromě posledního, který je naopak výstupní.

   Pokud má funkce vracet více parametrů, pak je nutné definovat strukturu, která bude parametry obsahovat a uvést jako poslední parametr tuto strukturu. Hlavičkový soubor pojmenujeme například `calc.h`.

2. Dalším krokem je vygenerovat z hlavičkového souboru kostru budoucí webové služby. To se provede spuštěním kompilátoru příkazem:
   `> soapcpp2 calc.h`
   Skelety funkcí serveru `add`, `sub`, `mul`, `div` pro obsluhu požadavků klienta se jmenují `soap_serve_ns_add`, `soap_serve_ns_sub`, `soap_serve_ns_mul` a `soap_serve_ns_div` a jsou zapsány ve vygenerovaném souboru `soapServer.cpp`.

3. Nyní už jen stačí napsat obsah funkcí, které má služba obsahovat (viz Textová ukázka 12). Jako první parametr mají tyto funkce navíc odkaz na kontext prostředí frameworku. Musíme také vytvořit funkci `main()`, kde se přijímají klientské požadavky a ty se vygenerovanou funkcí `soap_serve()` rozdělují k příslušným obslužným stub rutinám (viz Textová ukázka v příloze 26). Této funkci se obvykle říká *dispatching function*.

4. Na závěr je nutné server zkompilovat překladačem pro jazyk C (resp. C++) například příkazem:
   ```
   > g++ -o Server server.cpp soapC.cpp soapServer.cpp stdsoap2.cpp
   ```

Textová ukázka 11 *gSOAP* – příklad hlavičkového souboru

```
//gsoap ns service namespace: urn:Calc
int ns__add(double a, double b, double &result);
int ns__sub(double a, double b, double &result);
int ns__mul(double a, double b, double &result);
int ns__div(double a, double b, double &result);
```

Textová ukázka 12 *gSOAP* – implementace funkcí webové služby

```
int ns__add(struct soap *soap, double a, double b, double &result)
{
  result = a + b;
  return SOAP_OK;
}
...
int ns__div(struct soap *soap, double a, double b, double &result)
{
  if (b != 0)
  {
    result = a/b;
    return SOAP_OK;
  }
  else
  {
    return soap_sender_fault(soap, "Jmenovatel nesmi byt rovny nule.",
       "Deleni nulou.");
  }
}
```

2.1.1.2 Klient

Sestavení klienta je velmi podobné sestavení serveru a mnoho věcí lze využít. Stub rutiny pro vzdálené volání funkcí na serveru se rovněž generují aplikací programu `soapcpp2` na hlavičkový soubor (takže první dva kroky postupu jsou stejné a není nutné je opakovat).

Postup vytvoření klienta Kroky 1. a 2. jsou stejné jako v případě vytváření služby.

3. Po vygenerování stub rutin je nutné napsat obsah klientské aplikace, která bude tyto stub rutiny využívat. Příklad takové jednoduché aplikace je obsažen v Textové ukázce v příloze 27. Stub rutiny mají tvar `soap_call_ns__<jménofunkce>` a navíc oproti původní definici funkcí obsahují na prvních místech parametry `struct soap *soap, char *URL, char *action`, tedy odkaz na kontext prostření frameworku *gSOAP*, cílovou URL (`NULL` znamená default) a hodnotu položky `SOAPAction`, která je obsažena v hlavičce HTTP protokolu a která obsahuje URI popisující účel SOAP zprávy. `SOAPAction` je nepovinná hlavička a může být použita servery a firewally jako filtr HTTP požadavků.

4. V posledním kroku je nutné klientskou aplikaci opět zkompilovat například příkazem:
```
> g++ -o Client client.cpp soapC.cpp soapClient.cpp stdsoap2.cpp
```

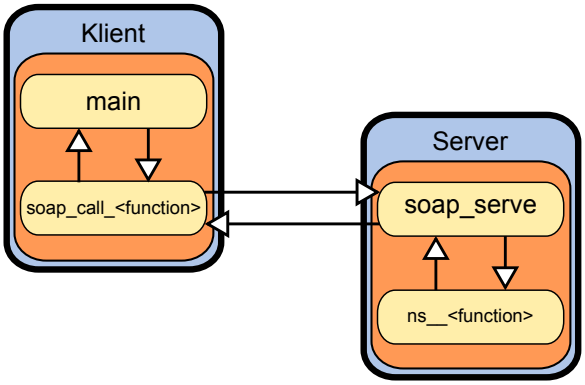
2.1.2 Shrnutí, vlastnosti

Na obrázku 2.1 je znázorněna interakce klienta a serveru vytvořených nástrojem *gSOAP*. Klient jednoduše ve své funkci `main`() zavolá stub rutinu s danými parametry, která způsobí vytvoření a odeslání zprávy, server přijme zprávu a zavolá na ni funkci `soap_serve`, která už se sama postará o zavolání patřičné funkce webové služby a poté odešle odpověď zpět klientovi.

Díky tomu, že stub funkce vznikají přesně na míru pro každou webovou službu podle hlavičkového souboru kompilací nástrojem `soapcpp2`, mohou aplikace vzniklé za pomocí *gSOAP* dosahovat velmi efektivního parsování příchozích zpráv a vygenerované aplikace jsou vhodné i pro vestavné systémy (viz *Slovník pojmů:embedded system*).

Pro rozšíření schopností nástroje *gSOAP* lze využít možnosti zavádění pluginů (neboli zásuvný modul je software, který nepracuje samostatně, ale jako doplňkový modul jiné

aplikace a rozšiřuje tak její funkčnost) při spuštění služby. Příkladem takového pluginu může být například vlastní plugin pro logování příchozích a odchozích zpráv nebo plugin pro autentizaci HTTP protokolu.



Obrázek 2.1: *gSOAP* architektura

2.2 Apache Axis2/C

Apache Axis2/C je softwarový engine (jádro počítačového programu, které je zodpovědné za běh programu, ale je zřetelně oddělené od periferních částí programu jako je například uživatelské rozhraní) pro tvorbu a správu webových služeb. Existují dvě implementace frameworku, v jazyce *C* a v jazyce *Java*. Podobně jako u architektury *gSOAP* i zde je hlavním cílem umožnit uživateli tvorbu webových služeb bez nutnosti zvládání specifických detailů protokolu webových služeb a XML formátu dat. Způsob, jakým tohoto cíle dosahuje *Axis2/C*, je přeci jen značně odlišný.

Jádrem *Axis2/C* je technologie *Apache Axiom*. Axiom je knihovna pro tvorbu stromové reprezentace datového modelu XML dokumentu s podporou opožděného vyhodnocování konkrétních uzlů [Apa(2012)]. To znamená, že příchozí zpráva ve formátu XML se převádí do spojové stromové struktury a že obsah jednotlivých uzlů tohoto stromu je vytvářen až v té době, kdy je opravdu zapotřebí, což způsobuje, že výsledná aplikace je méně náročná na potřebnou paměť. Tento způsob parsování bývá také někdy označován jako *pull parsování*. Protikladem je takzvané *push parsování*, kdy je celá zpráva zpracována najednou a tedy celá její reprezentace je v jeden okamžik kompletně obsažena v paměti, což může zapříčinit vyšší paměťové nároky.

Repozitář Framework *Axis2/C* předpokládá, že mu je přidělena složka, která slouží jako repozitář pro ukládání jeho konfiguračních souborů, knihoven webových služeb a zásuvných modulů. Tato složka obsahuje:

- `lib` – složka pro knihovny potřebné pro spuštění jádra systému
- `modules` (nepovinné) – složka pro zásuvné moduly
- `services` (nepovinné) – složka pro uložení webových služeb
- `axis2.xml` – konfigurační soubor frameworku

Repozitář může být sdílený pro klienty a webové služby a nebo mohou existovat dva nezávislé repozitáře.

2.2.1 Příklad aplikace – Kalkulačka

Jako příklad implementace webové služby a klienta jsem opět zvolil příklad jednoduché kalkulačky jako v případě nástroje *gSOAP*.

2.2.1.1 Webová služba

Framework *Axis2/C* předpokládá, že webové služby jsou uloženy v repozitáři ve složce `services`, kde každá služba má vlastní složku pojmenovanou stejným jménem jako služba samotná. Této složce se nachází všechny potřebné soubory pro danou službu.

Service API Každá webová služba v enginu *Axis2/C* musí implementovat rozhraní definované v souboru `axis2_svc_skeleton.h`. Samotná implementace je již ponechána na vývojáři webové služby.
Rozhraní `axis2_svc_skeleton` obsahuje funkce:

- `init` – provádí inicializaci služby během přijetí prvního požadavku na službu
- `invoke` – je funkce, která je zavolána při přijetí uživatelského požadavku
- `on_fault` – je funkce, která je zavolána, pokud engine detekuje chybu
- `free` – funkce pro uvolnění paměti po webové službě

Krom těchto funkcí musí ještě každá služba implementovat dvě další funkce, pro vytvoření instance služby a pro odstranění instance služby:

- `axis2_get_instance` – funkce pro vytvoření instance služby (zde se provádí alokace paměti)
- `axis2_remove_instance` – funkce pro odstranění instance služby (zde se provádí uvolňování paměti)

Funkce jako první parametr přijímají ukazatel na strukturu `axis2_svc_skeleton_t` definující vlastní službu. Dalším parametrem funkcí je ukazatel na prostředí enginu typu `axutil_env_t`. U funkcí `invoke` a `on_fault` je navíc přítomen ukazatel na uzel reprezentující kořen stromu přijaté zprávy XML.

Obě tyto funkce mají jako návratovou hodnotu opět ukazatel na kořen stromu ale tentokrát odchozí zprávy. Návratovou hodnotou ostatních funkcí je `integer`, který označuje zda funkce v pořádku skončila, nebo zda došlo při vykonávání k nějaké chybě. Funkce `invoke` má navíc jako vstupní parametr ukazatel na kontext přijaté zprávy typu `axis2_msg_ctx_t`.

Postup vytvoření služby

1. V prvním kroku je nutné implementovat funkce vykonávající operace, pro které je webová služba určena a funkci `invoke()`, která bude tyto funkce volat na základě obsahu přijaté zprávy. Zatímco u nástroje *gSOAP* byla tato operace velice snadná, zde už to tak jednoduché není. V systému *gSOAP* se naše funkce volala již s parametry získanými parsováním z přijaté zprávy. Zde je funkci `invoke()` předán odkaz na uzel stromové struktury vytvořené knihovnou Axiom a funkce `invoke()` se sama musí postarat o vyčtení údajů z tohoto stromu. Knihovna Axiom ovšem obsahuje mnoho funkcí, které tuto úlohu velice usnadňují, na vývojáři ale přesto zůstává, aby ošetřil všechny chybové stavy. V Textové ukázce 13 je příklad implementace funkce `invoke()`, která ze stromu přijaté zprávy přečte typ operace a danou operaci poté zavolá. Příklad operace lze najít v Textové ukázce 14. I přesto, že je z ukázky odstraněno veškeré ošetřování chyb, je funkce stále velice dlouhá, což je zapříčiněno extrahováním dat ze stromu přijaté zprávy. K tomu jsou používány funkce knihovny Axiom jako například `axiom_get_data_element` a další.
2. Nyní je nutné vytvořit zbylé funkce definované v `axis2_svc_skeleton`. V našem případě není nutné provádět žádnou inicializaci a tak stačí v těle funkce `init()` pouze vrátit hodnotu `AXIS2_SUCCESS`. Funkci `on_fault()` také není nutné imple-

mentovat. V ukázkovém textu v příloze 28 je ukázka jednoduché funkce `free()`, kde se pouze uvolňuje paměť.

3. Dále je třeba vytvořit funkce pro vytvoření a uvolnění instance služby. V Textové ukázce v příloze 29 je ukázka implementace funkce pro vytvoření nové instance `math_create`, kde se pro novou službu alokuje paměť a poté se jí předá ukazatel na strukturu `axis2_svc_skeleton_ops_t` obsahující funkce implementující rozhraní služby. V Textové ukázce v příloze 30 je ukázka uvolnění instance služby. Ve funkci se volá makro `AXIS2_SVC_SKELETON_FREE`, které pouze volá funkci `free()` služby předané jako parametr.

4. Posledním krokem je napsat pro novou službu soubor `services.xml`, který slouží jako jednoduchý konfigurační soubor pro zavedení služby do enginu. Ukázka obsahu souboru je v Textové ukázce 15.

5. Nyní již stačí webovou službu přeložit jako sdílenou knihovnu a slinkovat ji s knihovnami enginu *Axis2/C*. Na platformě Windows lze použít například následující příkazy:

```
> cl.exe /nologo /D "WIN32" /D "AXIS2_DECLARE_EXPORT" /D "_WINDOWS"
     /D "_MBCS" *.C /I.\..\include /c
> link.exe /nologo *.obj /LIBPATH:.\..\lib axiom.lib axutil.lib axis2_engine.l
     axis2_parser.lib /DLL /OUT:math.dll
```

Pro spuštění webové služby lze využít připravený jednoduchý webový server umístěný v repozitáři na místě `/bin/axis2_http_server`, který při spuštění zavede všechny webové služby ze složky `services` a spustí je.

Textová ukázka 13 *Axis2/C* – příklad implementace funkce `invoke()`

```
axiom_node_t *AXIS2_CALL math_invoke
(axis2_svc_skeleton_t * svc_skeleton, const axutil_env_t * env,
  axiom_node_t * node, axis2_msg_ctx_t * msg_ctx)
{
  if (node)
  {
    if (axiom_node_get_node_type(node, env) == AXIOM_ELEMENT)
    {
      axiom_element_t *element = NULL;
      element =
      (axiom_element_t *) axiom_node_get_data_element(node, env);
      if (element)
      {
        axis2_char_t *op_name =
        axiom_element_get_localname(element, env);
        if (op_name)
        {
          if (axutil_strcmp(op_name, "add") == 0)
            return axis2_math_add(env, node);
          ...
          if (axutil_strcmp(op_name, "div") == 0)
            return axis2_math_div(env, node);
        }
      }
    }
  }
  printf("Math service ERROR: invalid OM parameters in request\n");
  /** return a SOAP fault here */
  return node;
}
```

Textová ukázka 14 *Axis2/C* – příklad implementace sčítací funkce `add()`

```
axiom_node_t * axis2_math_add(const axutil_env_t * env, axiom_node_t *
    node)
{
  .
  .
  .
  param1_node = axiom_node_get_first_child(node, env);
  param1_text_node = axiom_node_get_first_child(param1_node, env);
  if (axiom_node_get_node_type(param1_text_node, env) == AXIOM_TEXT)
  {
    axiom_text_t *text =(axiom_text_t *) axiom_node_get_data_element(
        param1_text_node, env);
    if (text && axiom_text_get_value(text, env))
      param1_str = (axis2_char_t *) axiom_text_get_value(text, env);
  }
  param2_node = axiom_node_get_next_sibling(param1_node, env);
  param2_text_node = axiom_node_get_first_child(param2_node, env);
  if (axiom_node_get_node_type(param2_text_node, env) == AXIOM_TEXT)
  {
    axiom_text_t *text =(axiom_text_t *) axiom_node_get_data_element(
        param2_text_node, env);
    if (text && axiom_text_get_value(text, env))
      param2_str = (axis2_char_t *) axiom_text_get_value(text, env);
  }
  if (param1_str && param2_str)
  {
    long int result = 0;
    axis2_char_t result_str[255];
    axiom_element_t *ele1 = NULL; axiom_node_t *node1 = NULL, *node2 =
        NULL;
    axiom_namespace_t *ns1 = NULL;
    axiom_text_t *text1 = NULL;
    param1 = strtol(param1_str, NULL, 10);
    param2 = strtol(param2_str, NULL, 10);
    result = param1 + param2; /* Operation 'add' is only here! */
    sprintf(result_str, "%ld", result);
    ns1 = axiom_namespace_create(env, "http://axis2/test/namespace1", "
        ns1");
    ele1 = axiom_element_create(env, NULL, "result", ns1, &node1);
    text1 = axiom_text_create(env, node1, result_str, &node2);

    return node1;
  }
}
```

Textová ukázka 15 *Axis2/C* – příklad konfiguračního souboru webové služby

```
<service name="math">
  <parameter name="ServiceClass" locked="xsd:false">math</parameter>
   <description>
    WebService - Calculator
   </description>
  <operation name="add">
      <!--messageReceiver class="axis2_receivers" /-->
  </operation>
...
  </operation>
  <operation name="div">
      <!--messageReceiver class="axis2_receivers" /-->
  </operation>
</service>
```

2.2.1.2 Klient

I v případě klientské aplikace je situace o něco složitější než při použití nástroje *gSOAP*, kde stačilo jednoduše zavolat danou stub rutinu s příslušnými parametry. Zde je před odesláním nutné zprávu "ručně" sestavit s pomocí knihovny Axiom. V Textové ukázce 16 je příklad funkce pro vygenerování objektového modelu (`object model - om`) požadavku klienta.

Client API Rozhraní určené pro klientské aplikace ve frameworku *Axis2/C* je definované jako struktura `axis2_svc_client`. Toto je základní rozhraní, ale s distribucí *Axis2/C* jsou dodávána i další rozhraní, která fungují jako wrapper (funkce nebo knihovna funkcí, která obaluje původní funkci nebo knihovnu funkcí, *viz Slovník pojmů:* wrapper) původního rozhraní.

Funkce rozhraní:

- `axis2_svc_client_fire_and_forget` – je funkce, která odešle požadavek webové službě, ale neočekává žádnou odpověď, takže se klient nedozví ani o případné chybě
- `axis2_svc_client_send_robust` – je funkce, která odešle požadavek webové službě, neočekává žádnou odpověď, ale informuje klienta, pokud při zpracování požadavku na serveru dojde k nějaké chybě
- `axis2_svc_client_send_receive` – je funkce, která odešle požadavek webové službě a přijme odpověď, kterou vrací zpět v podobě spojové struktury. Funkce je blokující, blokuje klienta, dokud není přijata odpověď.
- `axis2_svc_client_send_receive_non_blocking` – je funkce, která odešle požadavek webové službě a přijme odpověď, kterou vrací zpět v podobě spojové struktury. Funkce ale není blokující a po přijetí odpovědi spustí callback funkci, kterou uživatel předává v posledním parametru.

Textová ukázka 16 *Axis2/C* – příklad vytvoření požadavku klienta

```
axiom_node_t *
build_om_programatically( const axutil_env_t * env, const axis2_char_t
    * operation, const axis2_char_t * param1, const axis2_char_t *
    param2)
{
    axiom_node_t *math_om_node = NULL;
    axiom_element_t *math_om_ele = NULL;
    axiom_node_t *text_om_node = NULL;
    axiom_element_t *text_om_ele = NULL;
    axiom_namespace_t *ns1 = NULL;

    axiom_xml_writer_t *xml_writer = NULL;
    axiom_output_t *om_output = NULL;
    axis2_char_t *buffer = NULL;

    ns1 =
     axiom_namespace_create(env, "http://ws.apache.org/axis2/services/
        math", "ns1");

    math_om_ele =
     axiom_element_create(env, NULL, operation, ns1, &math_om_node);

    text_om_ele =
     axiom_element_create(env, math_om_node, "param1", NULL, &
        text_om_node);
    axiom_element_set_text(text_om_ele, env, param1, text_om_node);

    text_om_ele =
     axiom_element_create(env, math_om_node, "param2", NULL, &
        text_om_node);
    axiom_element_set_text(text_om_ele, env, param2, text_om_node);

    xml_writer =
     axiom_xml_writer_create_for_memory(env, NULL, AXIS2_FALSE,
        AXIS2_FALSE, AXIS2_XML_PARSER_TYPE_BUFFER);
    om_output = axiom_output_create(env, xml_writer);

    axiom_node_serialize(math_om_node, env, om_output);
    buffer = (axis2_char_t *) axiom_xml_writer_get_xml(xml_writer, env)
        ;
    AXIS2_LOG_DEBUG(env->log, AXIS2_LOG_SI, "\nSending OM node in XML :
        %s \n", buffer);
    if (om_output)
    {
        axiom_output_free(om_output, env);
        om_output = NULL;
    }
    return math_om_node;
}
```

2.2.2 Shrnutí, vlastnosti

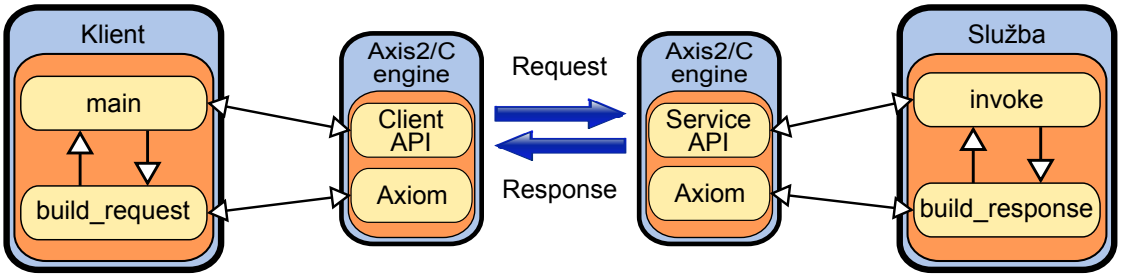
Na obrázku 2.2 je přiblížen princip komunikace klienta a webové služby prostřednictvím enginu *Axis2/C*. Klient ve své funkci `main()` obvykle zavolá nějakou funkci, která se za pomocí knihovny Axiom postará o vytvoření obsahu zprávy, a poté prostřednictvím Klientského API odešle požadavek webové službě. Engine na straně serveru poté zavolá funkci `invoke()`, která požadavky podle obsahu roztřídí a zavolá další funkci, která se dále postará o zpracování požadavku a případně pomocí knihovny Axiom sestaví obsah odpovědi, kterou engine odešle zpět.

Jako server lze použít již připravený jednoduchý `http` server dodávaný s distribucí frameworku, nebo lze *Axis2/C* využívat s Apache HTTP serverem zavedením modulu `mod_axis2` anebo lze *Axis2/C* použít i s Microsoft IIS Servery.

Zajímavou vlastností frameworku je také možnost vytváření a zavádění modulů, které přidávají další dodatečné funkce. Typickým příkladem může být zavedení modulu pro vlastní specifické logování příchozích zpráv. Modul lze také zavést v globálním kontextu pro všechny příchozí respektive odchozí zprávy nebo jen pro určitou webovou službu.

V současné době existují tři projekty, které implementují různé doplňující specifikace v podobě přídavných modulů pro *Axis2/C*:

- Apache Rampart/C – *WS-Security* – zaručuje zabezpečení zpráv
- Apache Sandesha2/C – *WS-ReliableMessaging* – zajišťuje bezpečné doručení zpráv a potvrzování přijetí zpráv
- Apache Savan/C – *WS-Eventing* – subscribe/publish mechanismus komunikace (viz sekce1.5.2)



Obrázek 2.2: *Axis2/C* architektura

2.3 gSOAP vs. Axis2/C

Obě tyto technologie si kladou za cíl zjednodušit tvorbu webových služeb. I přes stejný cíl, způsob, jakým toho dosahují, je dosti odlišný. Nedílnou součástí vývoje webových služeb je parsování zpráv ve formátu XML. Nástroj *gSOAP* pro každou webovou službu generuje specifický parser, který je poté značně výkonný a je tedy vhodný i pro nasazení ve vestavných systémech.

O parsování v *Axis2/C* se stará přímo pro tento framework vytvořená knihovna *Axiom*, která převádí XML zprávu do stromové struktury, přičemž jednotlivé uzly stromu se vyhodnocují až v době, kdy jsou vyžadovány. Díky tomu by aplikace měla mít nižší paměťové nároky, což je pro vestavěná zařízení také velmi důležitá vlastnost.

Dalším zásadním rozdílem je množství kódu, který musí vývojář napsat, aby vytvořil jednoduchou webovou službu a popřípadě klienta. Zatímco v případě nástroje *gSOAP* musí developer napsat téměř pouze obsah funkcí, které jsou přímo definované webovou službou, a o vše ostatní se postará samotný framework, u *Axisu* je množství potřebného kódu mnohonásobně větší.

Pokud vývojář potřebuje rychle a snadno vytvořit webovou službu, která se po vytvoření už nebude příliš upravovat, pak je *gSOAP* jasnou volbou. To, že vše funguje automaticky, ale také znamená, že se předpokládá, že se vývojář bude starat jen o implementaci logiky služeb a do ničeho jiného nebude „strkat nos“.

Naopak u frameworku *Axis2/C* je toho na vývojáři vyžadováno daleko více, protože sám musí vést parsování příchozí zprávy a sám musí poskládat odchozí zprávy to vše za pomoci Axiom knihovny. Ovšem v tomto případě má uživatel veškerou funkcionalitu webové služby ve svých rukou a sám se může rozhodnout na jaké vstupy bude jakým způsobem reagovat. Z toho ale pramení riziko, že webová služba může nakonec částečně vykonávat jinou službu, než by podle WSDL souboru vykonávat měla. *gSOAP* provádí mapování typů v XML zprávě na datové typy v jazyce C a zajišťuje tak typovou bezpečnost a úzké provázání s WSDL souborem. O to se *Axis2/C* nepostará a typová kontrola je čistě ponechána na vývojáři.

Pro *gSOAP* také hraje to, jak umí manipulovat s WSDL soubory. Nejen to, že umí z WSDL souboru vygenerovat hlavičkový soubor, podle kterého se dále generují stub procedury, ale umí i při kompilaci služby zpětně z hlavičkového souboru vygenerovat WSDL soubor s popisem webové služby v případě, že bychom WSDL soubor apriori neměli a skelety funkcí bychom vygenerovali z ručně psaného hlavičkového souboru. Pro *Axis2/C* existuje program `wsdl2c`, který také umí generovat skelety funkcí. Bohužel ale

tento program vyžaduje prostředí *Java*.

Nástroj *gSOAP* se zaměřuje na každou službu zvlášť, kdežto engine *Axis2/C* je postavený pro spravování libovolného množství webových služeb. Názorným příkladem může být třeba možnost zavedení nových rozšíření. V případě *Axisu* lze nové pluginy jednoduše aplikovat na všechny webové služby bez jakýchkoliv úprav implementace těchto služeb, kdežto u nástroje *gSOAP* by se do zdrojového kódu každé webové služby musel přidat kód registrující plugin a každá webová služba by musela být překompilována.

Shrnuto a podtrženo, pro snadný a rychlý vývoj jednoduché samostatné webové služby je nástroj *gSOAP* bezkonkurenční, ale pokud chce mít vývojář větší kontrolu nad výsledným kódem a nebo pokud vytváří systém o větším množství webových služeb, pak je pro něj *Axis2/C* lepší volbou.

Kapitola 3

Vývoj zařízení typu DPWS

V kapitole 2 jsme se zabývali vývoji jednoduchých webových služeb a uvedli jsme dva nástroje, které se k tomu obvykle používají a celý proces značně zjednodušují. Problém ovšem nastává, pokud se v požadavcích na funkcionalitu webové služby objeví požadavek na splnění některé z dalších specifikací, které rozšiřují protokol SOAP. V takovém případě často narazíme na problém, že frameworky pro práci s webovými službami s touto specifikací nepočítali, a může se dokonce stát, že z architektury frameworku vyplyne, že tuto funkcionalitu nelze ani jednoduše doimplementovat.

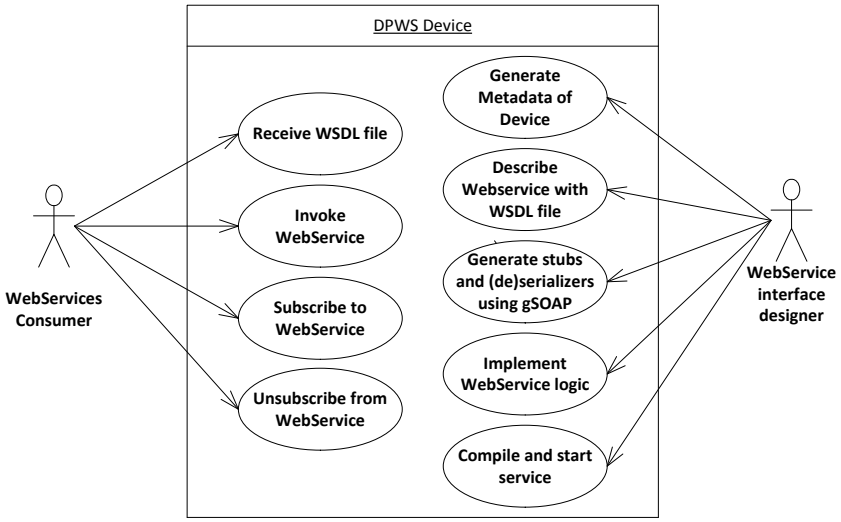
Jakýmsi balíkem těchto rozšiřujících specifikací je specifikace DPWS (viz sekce 1.5), která patří mezi obvyklé požadavky při implementaci webových služeb pro vestavná zařízení.

Bohužel pro implementaci DPWS se ukazuje *Axis2/C* jako velmi nevhodný. Má sice v sobě modul pro asynchronní komunikaci podle *WS-Eventing* (viz sekce 1.5.2), ale problém by nastal při pokusu o implementaci mechanismu vyhledávání zařízení *WS-Discovery* (viz sekce 1.5.1). *Axis2/C* totiž postrádá možnost komunikace prostřednictvím protokolu UDP, v repertoáru má pouze protokol HTTP a čistý protokol TCP. Existuje sice projekt implementující i UDP protokol pro *Axis2/C*, ale jeho vývoj je již zastavený a produkt je se současnou verzí nekompatibilní.

Jako velmi vhodným se naopak jeví framework *gSOAP*, který dokonce obsahuje základní implementaci specifikace *WS-Discovery*. Na druhou stranu ale postrádá implementaci specifikace *WS-Eventing*.

Axis2/C i *gSOAP* obsahují jednu ze dvou základních specifikací, které profil DPWS obsahuje. Přesto pro *Axis2/C* neexistuje žádný projekt pro vytvoření frameworku pro DPWS zařízení založeném na jeho jádře, zatímco pro *gSOAP* existují dokonce dva. Jedná se o frameworky *WS4D-gSOAP* a *DPWSCore*.

Postup vytvoření zařízení DPWS je u obou frameworků podobný a je znázorněn společně se základními operacemi klienta na UML Use-Case diagramu na obrázku 3.1.



Obrázek 3.1: DPWS – Use-Case diagram

3.1 WS4D-gSOAP

Jak už název napovídá, *WS4D-gSOAP* [Uni(2010)] je tedy framework pro implementaci specifikace DPWS založený na frameworku *gSOAP*.

Na obrázku 3.2 je vyobrazen diagram tříd, který přibližuje softwarovou architekturu frameworku pro implementaci DPWS zařízení. Třída `WebService` je jediná třída, kterou musí vývojář vytvořit ručně. V této třídě musí inicializovat struktury knihoven frameworku `struct soap` a `struct dpws`. Tato třída také musí obsahovat funkci `main`, kde se ve smyčce provádí příjem požadavků a odesílání odpovědí. Nejdůležitější částí je vytvoření funkcí, které implementují logiku operací webové služby. Skelety těchto funkcí jsou definovány ve třídě `svcStub`, která společně s třídami `svc.nsmap`, `svc_metadata` a `svc_wsdl` vznikla vygenerováním z WSDL souboru. Pro implementaci těchto funkcí platí stejná pravidla, jako při implementaci funkcí ve frameworku *gSOAP* (viz textová ukázka 12).

Třída `svc.nsmap` obsahuje strukturu, která popisuje mapování jmenných prostorů

v XML zprávách přijímaných resp. odchozích zpráv. Třída `svc_wsdl` obsahuje jedinou funkci pro přiřazení cesty WSDL souboru k dané službě.

Třída `svc_metadata` je určená pro práci s metadaty (strukturovaná data o datech) daného zařízení. Pokud jsou ale metadata vytvořena ještě před generováním skeletonů tříd, pak není nutné tuto třídu využívat.

Nejdůležitějšími třídami jsou bezesporu třídy `stdsoap2` a `stddpws`. Třída `stdsoap2` je původní třídou nástroje *gSOAP*, který se stará o generování skeletů tříd. Krom toho se také stará o vytváření a udržování HTTP a TCP spojení.

Třída `stddpws` je třídou nástavby nad *gSOAP* framewokem a stará se o věci, které jsou specifické přímo pro DPWS zařízení. Mezi ně patří zejména komunikace protokolem *WS-Discovery* a dále pak asynchronní komunikace *WS-Eventing*.

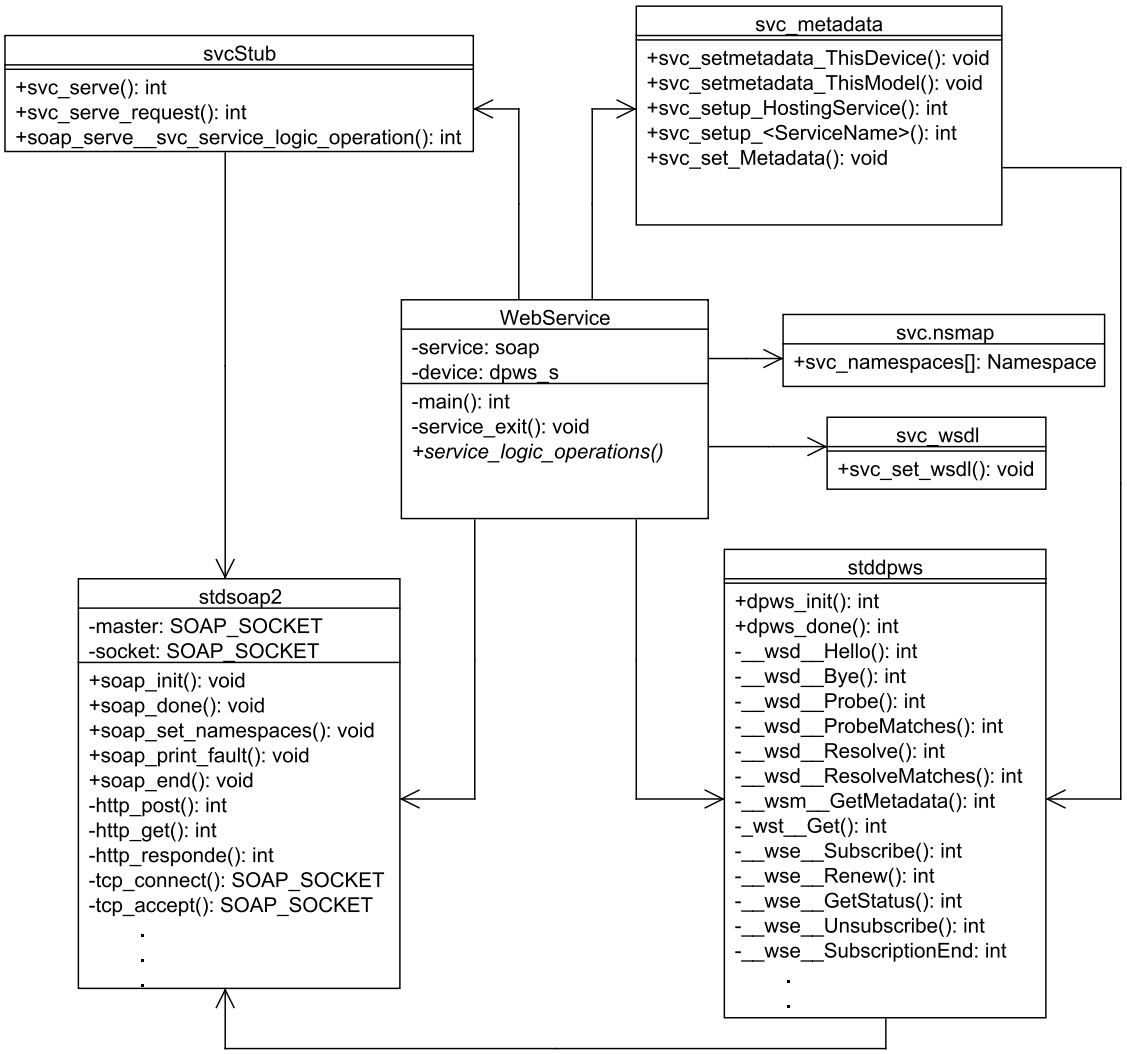
3.2 DPWSCore

DPWSCore [SOA(2012)] je další framework pro implementaci DPWS zařízení postavený na základech frameworku *gSOAP*. Vývoj služby probíhá v klasickém případě obdobným způsobem jako u frameworku *WS4D-gSOAP*. Na obrázku 3.3 se nachází UML Class diagram popisující vnitřní strukturu programu webové služby.

Vývojář opět nejprve vytvoří WSDL dokument popisující službu a nechá si vygenerovat serializaci a deserializaci zpráv a skelety operací služby v souboru `svcStub`. Tyto funkce využívají funkce ze souboru `dc_Runtime`, kde se nachází jen a pouze funkce pro použití ve vygenerovaném zdrojovém kódu a vývojář by je neměl nikde sám používat. V souboru `dc_Dpws` se poté nachází funkce implementující profil zařízení DPWS jako funkce specifikací *WS-Discovery* a *WS-Eventing*. Tyto funkce využívají funkce z knihovny nástroje *gSOAP* `stdsoap2`.

K dovednostem tohoto frameworku navíc patří možnost využít interní webový server k hostování libovolných webových stránek. Je pochopitelné, že tyto webové stránky se budou zpravidla týkat daného zařízení a může to být například jednoduchá webová stránka popisující funkčnost zařízení. Tuto vlastnost zajišťují funkce ze souboru `dc_DpwsRequest`.

Tento framework navíc disponuje naprosto unikátní vlastností, která se nazývá *generic stub & skeleton*. Jedná se o alternativní mechanismus serializace a deserializace příchozích a odchozích zpráv.



Obrázek 3.2: *WS4D-gSOAP* – UML Class diagram služby

Generické *stub* funkce umožňují zavolat webovou službu bez nutnosti generování kódu pro serializaci a deserializaci z WSDL dokumentu. K dispozici jsou dvě funkce:

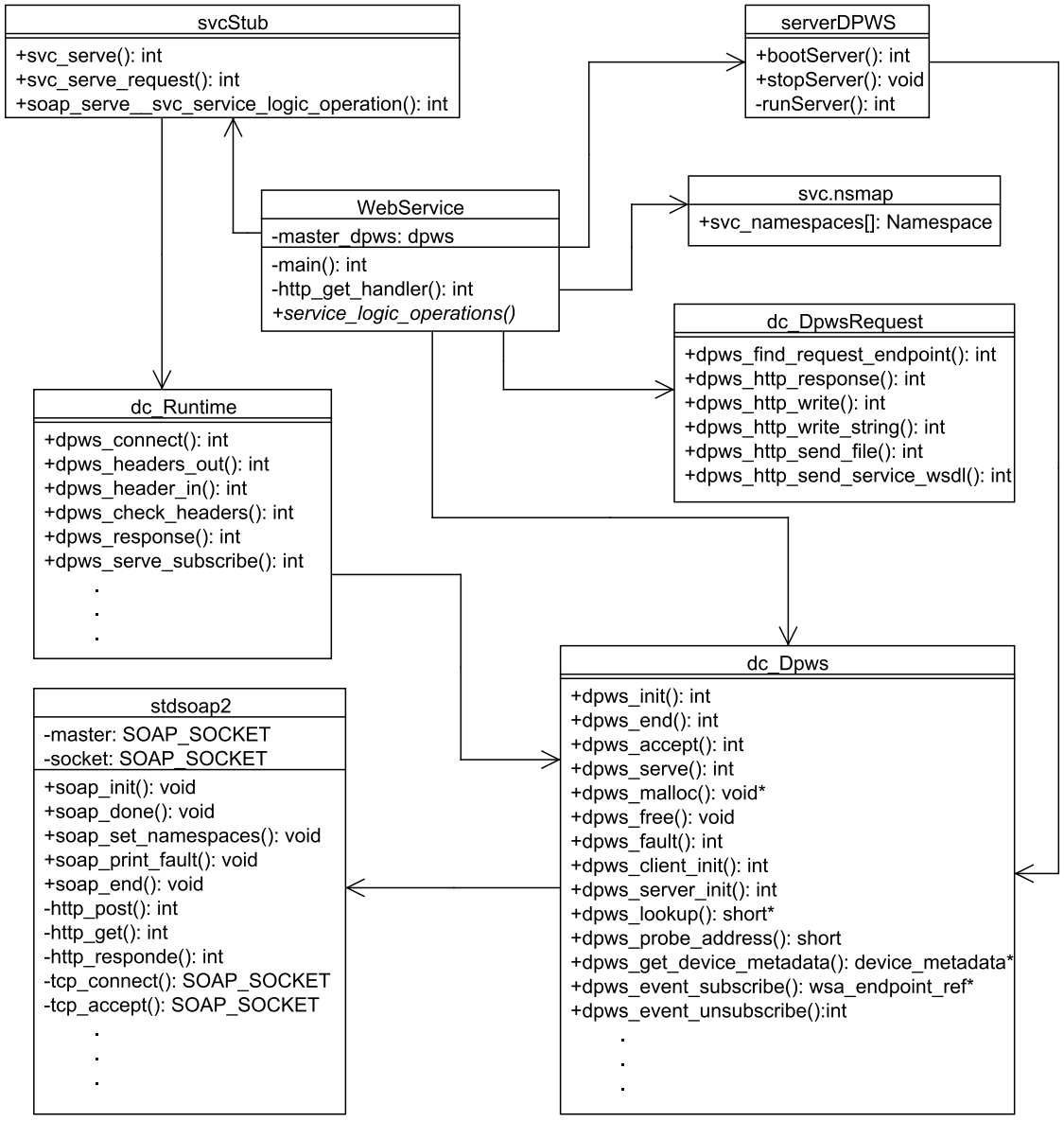
- `dpws_send` – pro jednosměrnou komunikaci směrem od klienta ke službě
- `dpws_call` – pro blokující volání s čekáním na odpověď od webové služby

Generický *skeleton* umožňuje vytvořit si vlastní funkci pro dispatching[1] příchozích zpráv. K tomuto účelu slouží funkce:

- `dpws_process_request`

[1]rozdělování příchozích zpráv podle obsahu jejich hlavičky k příslušným funkcím

Struktura webové služby, která používá generický skeleton je znázorněna na UML Class diagramu na obr. 3.4. Pro zpracování příchozích zpráv a pro vytváření odchozích zpráv slouží takzvaný EPX parser, přes jehož rozhraní lze jednoduše získat z jakéhokoliv XML dokumentu data a nebo vytvořit nový XML dokument. Příklad vytvoření zprávy je v ukázce 18.



Obrázek 3.3: *DPWSCore* – UML Class diagram služby

Textová ukázka 17 *DPWSCore* – dispatching příchozích zpráv pomocí generického skeletonu

```
int generic_serve_request(struct dpws *dpws)
{
  char* action = dpws->action ? dpws->action : "";
  if (dpws_is_subscribe_action(dpws, action))
  {
    return dpws_serve_subscribe(dpws);
  }
  if (!strcmp(action, "https://www.rexcontrols.cz/soa/BridgeServices/
     ReadItems"))
  {
    return dpws_process_request(dpws,"https://www.rexcontrols.cz/soa/
       BridgeServices/ReadItemsResponse", NULL, NULL, 0,
       ReadItems_Request);
  }
  if (!strcmp(action, "https://www.rexcontrols.cz/soa/BridgeServices/
     WriteItems"))
  {
    return dpws_process_request(dpws,"https://www.rexcontrols.cz/soa/
       BridgeServices/WriteItemsResponse", NULL, NULL, 0,
       WriteItems_Request);
  }
  if (!strcmp(action, "https://www.rexcontrols.cz/soa/BridgeServices/
     ReadItemsAsync"))
  {
    return dpws_process_request(dpws, NULL, NULL, NULL, 0,
       ReadItemsAsync_Request);
  }
  return dpws_dpws2soap(dpws)->error = SOAP_NO_METHOD;
}
```

Textová ukázka 18 *DPWSCore* – příklad vytvoření zprávy serializací funkcemi knihovny EPX

```
int WriteItems_Response(void *serializer_context, void *user_data)
{
  char* reply = (char*)user_data;
  int err = EPX_OK;
  if (epx_start_document(serializer_context, NULL, EPX_OPT_INDENT))
  {
    printf("Could not initialize serialization (%d)\n",
       epx_get_serializer_error(serializer_context));
    exit(1);
  }
  if (epx_start_element(serializer_context, "https://www.rexcontrols.cz
     /soa", "reply")
    || epx_define_prefix(serializer_context, "ns", "https://www.
       rexcontrols.cz/soa")
    || epx_put_characters(serializer_context, reply)
    || epx_end_element(serializer_context, "https://www.rexcontrols.cz/
       soa", "reply")
    || epx_end_document(serializer_context))
  {
    err = epx_get_serializer_error(serializer_context);
    printf("Serialization error (%d)\n", err);
  }
  free(reply);
  return err;
}
```

3.3 WS4D-gSOAP vs. DPWSCore

Oba dva frameworky pro vývoj DPWS zařízení jsou si v případě vývoje generováním kódu vcelku podobné. U obou stačí vytvořit WSDL soubor, metadata a pak napsat kód operací webové služby.

Ve většině případů nebude vývojář požadovat nic jiného než základní funkčnost podle specifikace DPWS a tu mu bez problémů poskytnou oba frameworky. Na druhou stranu, pokud vývojář hledá nějaké pokročilejší funkce, najde je spíše u frameworku *DPWSCore*.

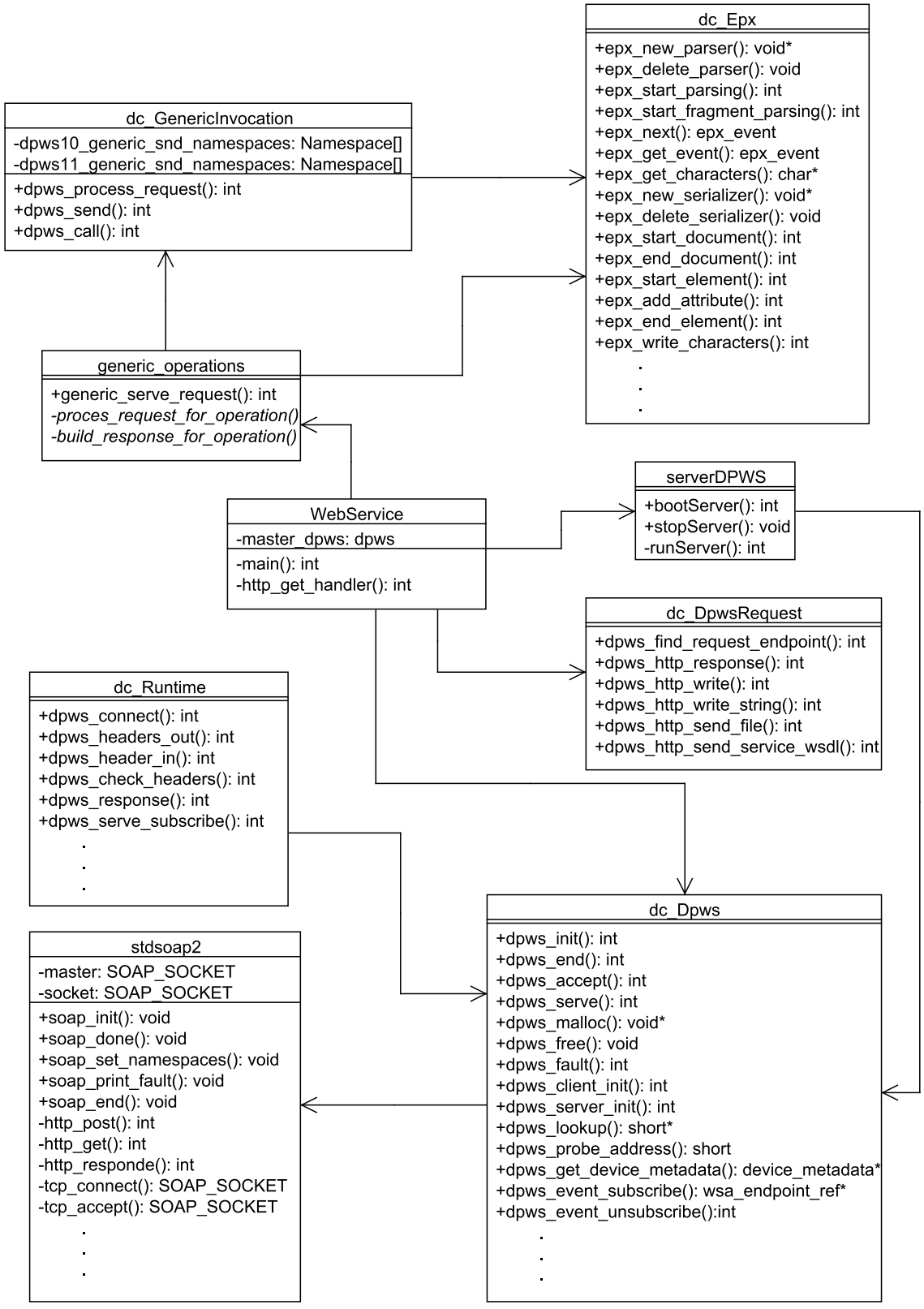
Velmi užitečnou funkcí navíc je přítomnost generických stub funkcí a skeletonu. Díky tomu již vývojář není odkázán pouze na kód, který mu vygeneroval nástroj *gSOAP* a tím se mu dostává větší volnosti podobně jako tomu je ve frameworku *Axis2/C* (viz sekce 2.2). Další z výhod *DPWSCore* je přístup k internímu webovému serveru.

Oba frameworky jsou dostupné pro operační systémy typu *Windows* i *Linux* a díky tomu, že jádro tvoří nástroj *gSOAP*, jsou i oba vhodné pro nasazení ve vestavných systémech.

Bohužel ani jeden z frameworků nepodporuje aktuální verzi nástroje *gSOAP* 2.8.17. *DPWSCore* je postaven na verzi 2.7.6 a *WS4D-gSOAP* na verzi 2.7.13. Přestože je *DPWSCore* vytvořen na starší verzi, jeho aktuální vývoj se zdá být stále aktivní, kdežto vývoj *WS4D-gSOAP* se zřejmě již zastavil. U frameworku *WS4D-gSOAP* navíc tato neaktuálnost s sebou přináší ten problém, že jím používaná verze *gSOAP* není kvůli chybě přeložitelná na operačních systémech založených na operačním systému *Debian*(což je problém při kompilaci pro Raspberry Pi viz sekce 4.2).

U frameworku *WS4D-gSOAP* byly navíc pozorovány neoprávněné přístupy do paměti při ukončení programu služby.

Na druhou stranu služby i klientské aplikace obou frameworků jsou bez problémů navzájem kompatibilní.



Obrázek 3.4: *DPWSCore* – UML Class diagram služby s generickým parsováním zpráv

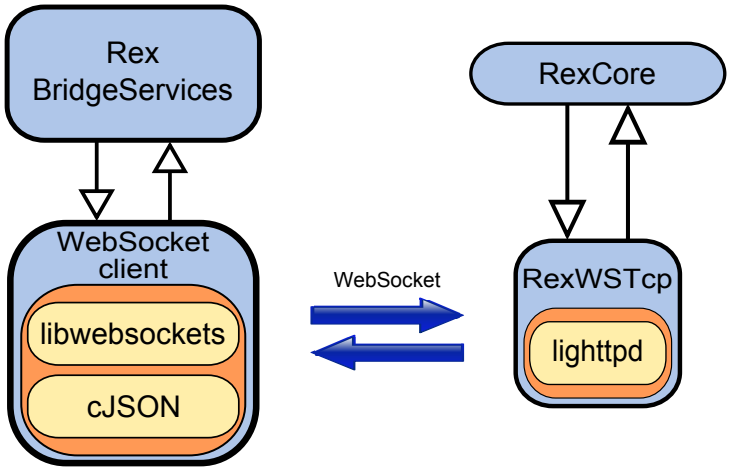
Kapitola 4

REX BridgeServices

Jedním z klíčových cílů této práce je vyvinout aplikaci, která bude sloužit jako pojící most mezi řídícím systémem *Rex* (řídicí systém a nástroj pro návrh a realizaci komplexních algoritmů automatického řízení) [REX(2014)] a klienty komunikujícími přes webové služby. Odtud plyne název aplikace – *BridgeServices*. Bylo rozhodnuto, že budou použity webové služby typu SOAP. Mezi frameworky, které se zdály být vhodné pro vývoj, patřil zejména *Axis2/C* a *gSOAP*.

Ovšem vzhledem k rozsáhlosti doprovodných specifikací, které rozšiřují a doplňují protokol SOAP bylo nutné si zvolit omezený počet těchto specifikací, které bude aplikace splňovat. Po úvaze bylo rozhodnuto, že webová služba bude podporovat standard DPWS, který se jeví být komplexním balíkem specifikací, které plně postačují pro většinu běžných použití. Zde přišly ke slovu frameworky postavené na nástroji *gSOAP*, tedy *WS4D-gSOAP* a *DPWSCore*. Aplikace byla implementována v obou těchto produktech.

Na obrázku 4.1 je znázorněna struktura architektury aplikace *BridgeServices*. Implementace komponenty *Rex BridgeServices* je závislá na zvoleném frameworku. Tato komponenta využívá klienta protokolu WebSocket (protokol pro plně duplexní komunikaci přes jediné TCP spojení) [IET(2011)] pro komunikaci se serverem téhož protokolu systému *Rex* – *RexWSTcp*, který dále komunikuje přímo s jádrem systému *Rex*. Klient používá knihovnu *libwebsockets* [libwebsockets()], která je populární zejména pro svou jednoduchost a efektivitu díky implementaci čistě v jazyce C. Zprávy, přenášené přes protokol WebSocket jsou ve formátu JSON, a proto klient navíc používá knihovnu *cJSON* [cjson()] pro serializaci a deserializaci příchozích zpráv.



Obrázek 4.1: *Rex BridgeServices* – schéma architektury

4.1 Rozhraní

V příloze v ukázkových textech 31, 32, 33 a 34 se nachází WSDL dokument, který popisuje službu *BridgeServices* a ze kterého byl generován kód nástrojem *wsdl2h* (viz sekce 2.1).[1]

Služba obsahuje tyto operace:

- `ReadItems` – vstupně-výstupní operace pro čtení dat
- `WriteItems` – vstupně-výstupní operace pro zápis dat
- `ReadItemsAsync` – vstupní operace, která způsobuje vyvolání události čtení dat podle *WS-Eventing*
- `ReadItemsAsyncCallback` – výstupní operace, která funguje jako callback při asynchronním čtení dat podle *WS-Eventing*

Do rozhraní webové služby byly krom běžných operací čtení a zápisu přidány operace pro komunikaci typu *subscribe/publish* specifikace *WS-Eventing*. Jejich činnost spočívá v tom, že klient zaregistrovaným k odběru novinek zasláním zprávy *subscribe* při invokaci operace `ReadItemsAsynch` získá odpověď o přečtených datech prostřednictvím výstupní operace `ReadItemsAsyncCallback`. Společně s ním tuto zprávu obdrží i všichni další zaregistrovaní klienti. Tato operace byla přidána hlavně z důvodu ověření této funkce v jednotlivých implementacích.

[1] první verzi WSDL dokumentu vytvořil Ing. Ondřej Severa

4.2 Testování výkonu

Služba *BridgeServices* byla implementována v obou představených systémech pro tvorbu DPWS zařízení. Pro porovnání výkonu mi byla poskytnuta totožná služba implementovaná v jazyce *Java* ve frameworku *JMEDS*[2] (*JMEDS* – Java Multi Edition DPWS Stack) [MAT(2012)]. Tato služba by dokonce měla být oproti zde vyvíjené ve výhodě, neboť se systémem *Rex* komunikuje přímo, kdežto naše služba komunikuje přes prostředníka v podobě WebSocket serveru.

Výkon frameworků byl testován na operacích `ReadItems` a `WriteItems`. Operace byla vždy provedena $10000x$ a poté byl spočten průměrný čas mezi odesláním požadavku a přijetím odpovědi na straně klienta.

V tabulce 4.1 jsou výsledky měření pro oba frameworky založené na nástroji *gSOAP*. Pro framework *DPWSCore* byl měřen rovněž výkon při implementaci pomocí generického skeletonu. Měření bylo provedeno na notebooku s operačním systémem Windows 8 a s dvoujádrovým procesorem *i5-430UM*. Klient pro benchmarking byl vytvořen pomocí frameworku *WS4D-gSOAP*. Z výsledků plyne, že oba frameworky jsou si co se rychlosti týče víceméně rovnocenné a to i v případě použití generického skeletonu s "manuálním"parsováním pomocí EPX parseru.

V tabulkách 4.2 a 4.3 jsou výsledky měření rychlosti frameworku *DPWSCore* a *JMEDS*. V první z tabulek jsou výsledky pro Windows (sestava viz předešlý odstavec). Ve druhé z tabulek se nachází výsledky pro Raspberry Pi [Ras(2014)], což je miniaturní jednodeskový počítač o velikosti kreditní karty s procesorem architektury *ARM*, který byl použit kvůli přibližně shodným vlastnostem jako se vyskytují u počítačů vestavných systémů. Operační systém pro Raspberry Pi má linuxové jádro a je postavený na systému *Debian*. Z výsledků je patrné, že *DPWSCore* je přibližně $2,5x$ rychlejší než *JMEDS* na obou platformách. Pro testování byl nyní použit klient, postavený na oblíbené knihovně v jazyce C *libcurl* [libcurl()]. Klient posílá jednoduchý HTTP POST požadavek se SOAP zprávou a přijímá odpověď. Na straně klienta se neprovádí žádná serializace ani deserializace zpráv a proto jsou výsledky lepší, než v tabulce 4.1.

Na platformě Raspberry Pi byl navíc proveden test měřící využití operační paměti utilitou `memusg` [Shin(2010)]. V tabulce 4.4 jsou uvedeny nejvyšší hodnoty paměti, kterou pro běh programu alokoval operační systém. Z výsledku vyplývá, že framework *DPWSCore* potřebuje ke svému běhu $10x$ méně paměti, než framework *JMEDS*.

[2] službu implementoval Ing. Ondřej Severa

	ReadItems	WriteItems
DPWSCore	8.07 ms	8.00 ms
DPWSCore generic	8.02 ms	8.48 ms
WS4D-gSOAP	8.12 ms	8.36 ms

Tabulka 4.1: Rychlost frameworků založených na nástroji *gSOAP*.

	ReadItems	WriteItems
DPWSCore	5.14 ms	5.09 ms
JMEDS	14.61 ms	14.71 ms

Tabulka 4.2: Porovnání rychlosti *DPWSCore* a *JMEDS* na Windows PC.

	ReadItems	WriteItems
DPWSCore	9.12 ms	8.99 ms
JMEDS	22.30 ms	25.07 ms

Tabulka 4.3: Porovnání rychlosti *DPWSCore* a *JMEDS* na Raspberry Pi.

	ReadItems	WriteItems
DPWSCore	2764 kB	2652 kB
JMEDS	27600 kB	27680 kB

Tabulka 4.4: Spotřeba paměti *DPWSCore* a *JMEDS* na Raspberry Pi.

Závěr

Důvodem vzniku požadavku na rozšíření řídícího systému *Rex* o komunikační rozhraní typu webových služeb je účast vývojového týmu katedry kybernetiky na Fakultě Aplikovaných Věd na projektu evropské unie jménem *eScop*[3], jehož cílem je vytvoření softwarové architektury pro implementaci automatizovaných výrobních informačních systémů[4] pro plánování produkce spojenou s řízením v reálném čase. Architektura je rozdělena do několika vrstev, přičemž ta nejnižší, hardwarová vrstva, se sestává z vestavných systémů komunikujících s okolím prostřednictvím webových služeb.

Pro řešení projektu byl nejprve zvolen protokol SOAP a později bylo blíže určeno, že zařízení by měla odpovídat specifikaci DPWS. Vzhledem k aplikaci výsledného produktu na platformách vestavných systémů, byly pro implementaci zvoleny frameworky v jazyce `C`. Přestože oba frameworky dosahují shodných výsledků při testování rychlosti, framework *DPWSCore* je jednoznačně lepší volbou zejména díky větší stabilitě, jednoduchosti implementace požadované funkcionality a dále kvůli pokročilým funkcím v podobě možnosti využití vnitřního serveru k vlastním účelům a možnosti použití generického skeletonu a stub funkce. Bohužel ani tento framework není bez chyby a zejména použití generického skeletonu je kvůli chybám doposud pro produkci nepoužitelné. Je zde ale stále naděje, že tyto chyby budou opraveny v následující vydané verzi, neboť vývoj ještě není plně zastaven. Na druhou stranu při vývoji běžným způsobem s generováním skeletonů funkcí k žádnému problému nedošlo. Vnitřní server byl využit pro implementaci jednoduché webové stránky s klientem v jazyce Javascript, který umožňuje zavolat operace webové služby.

Výsledná služba *REX BridgeServices* byla testována a výsledky porovnány se stejnou službou implementovanou v jazyce Java ve frameworku *JMEDS*. Z výsledků je jasně patrné, že nasazení služby vzniklé ve frameworku *DPWSCore* je pro vestavné systémy podstatně vhodnější ať už kvůli vyšší rychlosti nebo nižším paměťovým nárokům.

[3]Embedded systems Service-based Control for Open manufacturing and Process automation
[4]MES - Manufacturng Execution Systems

Práce dle mého názoru splňuje požadavky, které na ni byly kladeny. Aplikace by se do budoucna měla dočkat vylepšení v podobě komunikace se systémem *Rex* přímo po nativním protokolu namísto prostředníka v podobě WebSocket serveru, čímž by se komunikace měla ještě urychlit.

Slovník pojmů

API je rozhraní, které určuje, jak se má přistupovat k softwarovým komponentám.

Axiom je knihovna pro tvorbu stromové reprezentace datového modelu XML dokumentu.

CGI je skript generující dynamické WWW stránky, je to jakýkoliv na straně serveru spustitelný soubor, který po spuštění zapíše na výstup HTTP hlavičku a poté obvykle obsah v HTML, skript je spouštěn HTTP serverem.

DPWS je specifikace opírající se o protokol SOAP definující způsob komunikace a interakce s obvykle jednoduchými zařízeními typu tiskáren.

embedded system je jednoúčelový systém, ve kterém je řídicí počítač zcela zabudován do zařízení, které ovládá.

framework je softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci softwarových aplikací. Může obsahovat podpůrné programy, knihovny a další.

FTP je internetový protokol určený pro přenos souborů mezi počítači po počítačové sítě.

HTML je značkovací jazyk pro vytváření WWW stránek.

HTTP je síťový protokol pro výměnu hypertextových dokumentů ve formátu HTTP.

JSON je jednoduchý formát zápisu dat založený na způsobu deklarace objektů v programovacím jazyce JavaScript.

metadata jsou strukturovaná data o rozsáhlejších datech.

parsování neboli syntaktická analýza je takový proces, při kterém se zkoumá posloupnost formálních prvků s cílem určit jejich gramatickou strukturu vůči předem dané formální gramatice. Parsováním se převádí vstupní text do datové struktury(obvykle stromu), která usnadňuje další zpracování a přitom zachovává hierarchii vstupních dat.

plugin neboli zásuvný modul je software, který nepracuje samostatně, ale jako doplňkový modul jiné aplikace a rozšiřuje tak její funkčnost.

REST je styl softwarové architektury obvykle používaný pro popis webových architektur.

RESTful je protokol webových služeb implementujících REST architekturu.

serializace je obecně takový proces, při kterém se převádí libovolně složitý objekt do své sériové(sekvenční) podoby. Složité datové struktury se tak z paměti počítače převedou na posloupnost bitů, která se pak může například přenést po síti a ze které lze deserializací získat původní objekt.

SMTP je internetový protokol určený pro přenos zpráv elektronické pošty.

SOAP je protokol pro výměnu zpráv mezi aplikacemi založený na XML.

softwarový engine je jádro počítačového programu, které je zodpovědné za běh programu, ale je zřetelně oddělené od periferních částí programu jako je například uživatelské rozhraní apod..

stub rutina je funkce, kterou používá klient pro vzdálené volání funkce na serveru. Tato funkce provede serializaci (*viz. Slovník pojmů:* serializace) vstupních dat, postará se o odeslání dat serveru, kde se typicky zavolá další stub rutina, která provede deserializaci dat a tyto data předá požadované funkci na serveru a poté odešle výstup funkce zpět stub rutině klienta.

TCP je internetový protokol transportní vrstvy udržující spojení mezi dvěma body sítě a zaručující doručení zpráv v daném pořadí.

UDDI je mechanismus pro zveřejňování a vyhledávání webových služeb založený na XML.

UDP je internetový nespojový protokol transportní vrstvy určený pro přenos datagramů.

UML je grafický jazyk pro vizualizaci, návrh a dokumentaci softwarových systémů.

URI je textový řetězec s definovanou strukturou pro přesné určení zdroje informací.

URL je URI pro popis umístění, každá URL je URI, ale ne každá URI je URL.

webová služba je jakákoliv služba dostupná přes internet často využívající zprávy ve formátu XML.

wrapper je funkce nebo knihovna funkcí, která obaluje původní funkci nebo knihovnu funkcí. Wrapper funkce tedy provádí jen minimální výpočetní operace a volá v sobě funkci původní. Tímto mechanismem lze ke knihovnám vytvořit nové uživatelsky příjemnější rozhraní..

WSDL je protokol pro popis veřejného rozhraní webové služby, zapisuje se v XML formátu.

WWW je označení pro aplikace internetového protokolu HTTP.

XML je rozšířený značkovací jazyk, podobný HTML, určený pro přenos dat.

XML Schema je jazyk pro definování datových struktur založený na XML.

XML-RPC je protokol pro provádění vzdáleného volání procedur, využívá XML.

Zkratky

API Application Programming Interface.

Axiom Axis Object Model.

CGI Common Gateway Interface.

DPWS Devices Profile for Web Services.

FTP Simple Mail Transfer Protocol.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

JSON JavaScript Object Notation.

REST Representational state transfer.

SMTP Simple Mail Transfer Protocol.

SOAP Simple Object Access Protocol.

TCP Transmission Control Protocol.

UDDI Universal Description, Discovery and Integration.

UDP Devices Profile for Web Services.

UML Unified Modeling Language.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

WSDL Web Services Description Language.

WWW World Wide Web.

XML EXtensible Markup Language.

XML-RPC XML-Remote procedure call.

Literatura

[Apa(2004)] *Apache License 2.0*. Apache Software Foundation, 2004. Dostupné z: `http://www.apache.org/licenses/`.

[Apa(2012)] *Axiom*. The Apache Software Foundation, 2012. Dostupné z: `http://ws.apache.org/axiom/`.

[Cerami(2002)] CERAMI, E. *Web Services Essentials*. O'Reilly, 2002.

[cjson()] cjson. *cJSON*, 2009. Dostupné z: `http://cjson.sourceforge.net/`.

[IET(2011)] *The WebSocket Protocol*. IETF (Internet Engineering Task Force), 2011. Dostupné z: `http://tools.ietf.org/html/rfc6455`.

[libcurl()] libcurl. *libcurl*, 2010. Dostupné z: `http://curl.haxx.se/libcurl/`.

[libwebsockets()] libwebsockets. *libwebsockets*, 2010. Dostupné z: `http://libwebsockets.org/trac/libwebsockets`.

[MAT(2012)] *JMEDS (Java Multi Edition DPWS Stack)*. MATERNA Information & Communications and TU Dortmund, Dpt. of Computer Science, 2012. Dostupné z: `http://sourceforge.net/projects/ws4d-javame/`.

[OAS(2009a)] *Devices Profile for Web Services Version 1.1*. OASIS (Organization for the Advancement of Structured Information Standards), 2009a. Dostupné z: `http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf`.

[OAS(2004)] *UDDI Version 3.0.2*. OASIS (Organization for the Advancement of Structured Information Standards), 2004. Dostupné z: `http://uddi.org/pubs/uddi_v3.htm`.

[OAS(2009b)] *Web Services Dynamic Discovery (WS-Discovery) Version 1.1*. OASIS (Organization for the Advancement of Structured Information Standards),

2009b. Dostupné z: `http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html`.

[Ras(2014)] *Raspberry Pi Documentation*. Raspberry Pi Foundation, 2014. Dostupné z: `https://github.com/raspberrypi/documentation`.

[REX(2014)] *Funkcni bloky systemu REX – Referencni prirucka*. REX Controls s.r.o, 2014. Dostupné z: `www.rexcontrols.cz/media/DOC/CZECH/BRef_CZ.pdf`.

[Shin(2010)] SHIN, J. *memusg*, 2010. Dostupné z: `https://gist.github.com/netj/526585`.

[Skonnard(2003)] SKONNARD, A. *Understanding WSDL*. Northface University, 2003. Dostupné z: `http://msdn.microsoft.com/en-us/library/ms996486.aspx`.

[SOA(2012)] *DPWS Core version 2.4 - User Guide*. SOA4D (Service-Oriented Architecture for Devices), 2012. Dostupné z: `https://forge.soa4d.org/docman/view.php/8/45/DPWSCore+User+Guide.pdf`.

[Uni(2010)] *WS4D-gSOAP 0.8*. Universitat Rostock, Fakultat fur Informatik und Elektrotechnik, 2010. Dostupné z: `http://trac.e-technik.uni-rostock.de/projects/ws4d-gsoap/chrome/site/ws4d-gsoap-refman-0.8.pdf`.

[van Engelen(2013)] ENGELEN, R. *gSOAP 2.8.15 User Guide*. GENIVIA INC, 2013. Dostupné z: `http://www.cs.fsu.edu/~engelen/soapdoc2.html`.

[W3C(2000)] *Simple Object Access Protocol (SOAP) 1.1*. W3C (World Wide Web Consortium), 2000. Dostupné z: `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`.

[W3C(2001)] *Web Services Description Language (WSDL) 1.1*. W3C (World Wide Web Consortium), 2001. Dostupné z: `http://www.w3.org/TR/wsdl`.

[W3C(2011)] *Web Services Eventing (WS-Eventing)*. W3C (World Wide Web Consortium), 2011. Dostupné z: `http://www.w3.org/TR/ws-eventing/`.

[W3S(2013)] *SOAP Tutorial*. W3Schools, 2013. Dostupné z: `http://www.w3schools.com/soap/`.

Příloha A

Přiložené ukázky

Textová ukázka 19 UDDI Dotazovací API – příklad volání funkce `find_business`

```
<find_business generic="1.0" xmlns="urn:uddi-org:api">
  <name>Business Name</name>
</find_business>
```

Textová ukázka 20 UDDI Dotazovací API – odpověď na volání funkce `find_business`

```
<businessList generic="1.0" operator="Microsoft Corporation"
  truncated="false" xmlns="urn:uddi-org:api">
  <businessInfos>
    <businessInfo businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64">
      <name>XMethods</name>
      <description xml:lang="en">Business description</description>
      <serviceInfos>
        <serviceInfo
        serviceKey="d5b180a0-4342-11d5-bd6c-002035229c64"
        businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64">
        <name>Service name</name>
        </serviceInfo>
        ...
      </serviceInfos>
    </businessInfo>
  </businessInfos>
</businessList>
```

Textová ukázka 21 UDDI Dotazovací API – příklad volání funkce `get_businessDetail`

```
<get_businessDetail generic="1.0" xmlns="urn:uddi-org:api">
  <businessKey>ba744ed0-3aaf-11d5-80dc-002035229c64</businessKey>
</get_businessDetail>
```

Textová ukázka 22 UDDI Dotazovací API – odpověď na volání funkce `get_businessDetail`

```
<businessDetail generic="1.0" operator="Microsoft Corporation"
truncated="false" xmlns="urn:uddi-org:api">
  <businessEntity
  businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64"
  operator="www.ibm.com/services/uddi" authorizedName="0100001QS1">
  <discoveryURLs>
    <discoveryURL useType="businessEntity">
    http://www.ibm.com/services/uddi/uddiget?
    businessKey=ba744ed0-3aaf-11d5-80dc-002035229c64
    </discoveryURL>
  </discoveryURLs>
  <name>Business name</name>
  <description xml:lang="en">Business description</description>
  <contacts>
    <contact useType="Founder">
    <description xml:lang="en" />
    <personName>František Dobrota</personName>
    <phone useType="Founder" />
    <email useType="Founder">fdobrota@example.net</email>
    <address>
      <addressLine>rodák z blízké vesnice</addressLine>
      ...
    </address>
  </contact>
  </contacts>
  <businessServices>
    ...
  </businessServices>
  </businessEntity>
</businessDetail>
```

Textová ukázka 23 UDDI – požadavek na autentizaci klienta

```
<get_authToken generic="1.0" xmlns="urn:uddi-org:api"
  userID="boss@example.com" cred="theBoss">
</get_authToken>
```

Textová ukázka 24 UDDI – odpověď na požadavek autentizace klienta

```
<authToken generic="1.0" xmlns="urn:uddi-org:api"
operator="http://uddi.microsoft.com">
  <authInfo>1BAAAAAAAHmykB2ylo*pV*pnrFoS4a*IblrgZSlpa
  jYC853wq9HIfsbaozLxYpG2Bo;1AAAAAAAAAKZi8QkOJ8BJfnMc
  *HeQCtOTHvu3TDkPEogcauDpvtHyxQGczEE0cj9bdl7C48RRyrK
  H7ReaF8OHivQEMltSEhgD8RNhmt0rHFdZoWkANFe*uSLmab4VvA
  FKLHFouvDh3MJ*9VK9YML14dg
</authInfo>
</authToken>
```

Textová ukázka 25 UDDI – uložení `businessEntity`

```
<save_business generic="1.0" xmlns="urn:uddi-org:api">
  <authInfo>1BAAAAAAAHmykB2ylo*pV*pnrFoS4a*IblrgZSlpa
  jYC853wq9HIfsbaozLxYpG2Bo;1AAAAAAAAAKZi8QkOJ8BJfnMc
  *HeQCtOTHvu3TDkPEogcauDpvtHyxQGczEE0cj9bdl7C48RRyrK
  H7ReaF8OHivQEMltSEhgD8RNhmt0rHFdZoWkANFe*uSLmab4VvA
  FKLHFouvDh3MJ*9VK9YML14dg
  </authInfo>
  <businessEntity
    businessKey="03754729-3d3c-48e0-854a-1c1fd576ca5b">
    <name>Business Name</name>
    <description xml:lang="en">
      Business description
    </description>
  </businessEntity>
</save_business>
```

Textová ukázka 26 *gSOAP*– implementace `main` funkce serveru

```
int main()
{
  struct soap soap;
  int m, s; // master and slave sockets
  soap_init(&soap);
  m = soap_bind(&soap, "localhost", 18083, 100);
  if (m < 0)
    soap_print_fault(&soap, stderr);
  else
  {
    fprintf(stderr, "Socket connection successful: master socket = %d\n
       ", m);
    for (int i = 1; ; i++)
    {
      s = soap_accept(&soap);
      if (s < 0)
      {
        soap_print_fault(&soap, stderr);
        break;
      }
      fprintf(stderr, "%d: accepted connection from IP=%d.%d.%d.%d
         socket=%d", i,
        (soap.ip >> 24)&0xFF, (soap.ip >> 16)&0xFF, (soap.ip >> 8)&0xFF
           , soap.ip&0xFF, s);
      if (soap_serve(&soap) != SOAP_OK) // process RPC request
        soap_print_fault(&soap, stderr); // print error
      fprintf(stderr, "request served\n");
      soap_destroy(&soap); // clean up class instances
      soap_end(&soap); // clean up everything and close socket
    }
  }
  soap_done(&soap); // close master socket and detach context
}
```

Textová ukázka 27 *gSOAP*– příklad implementace `main` funkce klienta

```
int main(int argc, char** argv)
{
  struct soap *soap = soap_new();
  double a, b, result;
  if(argc > 3 )
  {
    a = strtod(argv[1], NULL);
    b = strtod(argv[3], NULL);
  }
  else
      return -1;
  switch (*argv[2]) {
  case '+':
    if(soap_call_ns__add(soap, "localhost:18083", NULL, a, b, result) ==
       0)
      printf("%.2f + %.2f = %.2f{\textbackslash}n", a, b, result);
    else{
      printf("error{\textbackslash}n");
      soap_print_fault(soap, stderr);
  }
    break;
  case '-':
    if(soap_call_ns__sub(soap, "localhost:18083", NULL, a, b, result) ==
       0)
      printf("%.2f - %.2f = %.2f\n", a, b, result);
    else
      soap_print_fault(soap, stderr);
    break;

...

  default:
    printf("unknown operation\n");
    break;
  }
  return 0;
}
```

Textová ukázka 28 *Axis2/C*– příklad implementace funkce `free`

```
int AXIS2_CALL math_free(axis2_svc_skeleton_t * svc_skeleton, const
    axutil_env_t * env)
{
  if (svc_skeleton)
  {
    AXIS2_FREE(env->allocator, svc_skeleton);
    svc_skeleton = NULL;
  }
  return AXIS2_SUCCESS;
}
```

Textová ukázka 29 *Axis2/C*– příklad implementace funkce `axis2_get_instance`

```
static const axis2_svc_skeleton_ops_t math_svc_skeleton_ops_var = {
  math_init,
  math_invoke,
  NULL,
  math_free
};

AXIS2_EXTERN axis2_svc_skeleton_t *AXIS2_CALL math_create
( const axutil_env_t * env)
{
  axis2_svc_skeleton_t *svc_skeleton = NULL;
  svc_skeleton = AXIS2_MALLOC(env->allocator, sizeof(
     axis2_svc_skeleton_t));

  svc_skeleton->ops = &math_svc_skeleton_ops_var;

  svc_skeleton->func_array = NULL;

  return svc_skeleton;
}

AXIS2_EXPORT int axis2_get_instance
( struct axis2_svc_skeleton **inst, const axutil_env_t * env)
{
  *inst = math_create(env);
  if (!(*inst))
  {
    return AXIS2_FAILURE;
  }
  return AXIS2_SUCCESS;
}
```

Textová ukázka 30 *Axis2/C*– příklad implementace funkce pro odebrání instance

```
AXIS2_EXPORT int axis2_remove_instance
( axis2_svc_skeleton_t * inst, const axutil_env_t * env)
{
  axis2_status_t status = AXIS2_FAILURE;
  if (inst)
  {
    status = AXIS2_SVC_SKELETON_FREE(inst, env);
  }
  return status;
}
```

Textová ukázka 31 *Rex BridgeServices* – WSDL dokument popisující službu (types)

```
<wsdl:definitions xmlns:tns="https://www.rexcontrols.cz/soa" xmlns:wsdl
   ="http://schemas.xmlsoap.org/wsdl/" xmlns:wsoap12="http://schemas.
   xmlsoap.org/wsdl/soap12/" xmlns:xs="http://www.w3.org/2001/XMLSchema
   " xmlns:s12="http://www.w3.org/2003/05/soap-envelope" xmlns:wsam="
   http://www.w3.org/2007/05/addressing/metadata" xmlns:wse="http://
   schemas.xmlsoap.org/ws/2004/08/eventing" targetNamespace="https://
   www.rexcontrols.cz/soa">
  <wsdl:types>
    <xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       targetNamespace="https://www.rexcontrols.cz/soa"
       elementFormDefault="qualified" attributeFormDefault="unqualified
       ">
      <xs:element name="WriteItemsMessage" type="tns:
         WriteItemsComplexType"/>
      <xs:element name="ReadItemsAsyncResponseMessage" type="tns:
         ReadItemsAsyncResponseComplexType"/>
      <xs:element name="values" type="xs:string"/>
      <xs:element name="cstrings" type="xs:string"/>
      <xs:element name="reply" type="xs:string"/>
      <xs:complexType name="WriteItemsComplexType">
        <xs:sequence>
          <xs:element name="cstrings" type="xs:string"/>
          <xs:element name="values" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="ReadItemsAsyncResponseComplexType">
        <xs:sequence>
          <xs:element name="cstrings" type="xs:string"/>
          <xs:element name="values" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
```

Textová ukázka 32 *Rex BridgeServices* – WSDL dokument popisující službu (messages)

```
<wsdl:message name="WriteItemsMessage">
    <wsdl:part name="parameters" element="tns:WriteItemsMessage"/>
  </wsdl:message>
  <wsdl:message name="WriteItemsResponseMessage">
    <wsdl:part name="parameters" element="tns:reply"/>
  </wsdl:message>
  <wsdl:message name="ReadItemsMessage">
    <wsdl:part name="parameters" element="tns:cstrings"/>
  </wsdl:message>
  <wsdl:message name="ReadItemsResponseMessage">
    <wsdl:part name="parameters" element="tns:values"/>
  </wsdl:message>
  <wsdl:message name="ReadItemsAsyncMessage">
    <wsdl:part name="parameters" element="tns:cstrings"/>
  </wsdl:message>
  <wsdl:message name="ReadItemsAsyncResponseMessage">
    <wsdl:part name="parameters" element="tns:
       ReadItemsAsyncResponseMessage" />
  </wsdl:message>
```

Textová ukázka 33 *Rex BridgeServices* – WSDL dokument popisující službu (port)

```
<wsdl:portType name="BridgeServices" wse:EventSource="true">
    <wsdl:operation name="WriteItems">
      <wsdl:input name="WriteItems" message="tns:WriteItemsMessage"
          wsam:Action="https://www.rexcontrols.cz/soa/BridgeServices/
          WriteItems"/>
      <wsdl:output name="WriteItemsResponse" message="tns:
          WriteItemsResponseMessage" wsam:Action="https://www.
          rexcontrols.cz/soa/BridgeServices/WriteItemsResponse"/>
    </wsdl:operation>
    <wsdl:operation name="ReadItems">
      <wsdl:input name="ReadItems" message="tns:ReadItemsMessage" wsam:
          Action="https://www.rexcontrols.cz/soa/BridgeServices/
          ReadItems"/>
      <wsdl:output name="ReadItemsResponse" message="tns:
          ReadItemsResponseMessage" wsam:Action="https://www.rexcontrols
          .cz/soa/BridgeServices/ReadItemsResponse"/>
    </wsdl:operation>
    <wsdl:operation name="ReadItemsAsync">
      <wsdl:input name="ReadItemsAsync" message="tns:
          ReadItemsAsyncMessage" wsam:Action="https://www.rexcontrols.cz
          /soa/BridgeServices/ReadItemsAsync"/>
    </wsdl:operation>
    <wsdl:operation name="ReadItemsAsyncCallback">
      <wsdl:output message="tns:ReadItemsAsyncResponseMessage" wsam:
          Action="https://www.rexcontrols.cz/soa/BridgeServices/
          ReadItemsAsyncCallback"/>
    </wsdl:operation>
  </wsdl:portType>
```

Textová ukázka 34 *Rex BridgeServices* – WSDL dokument popisující službu (binding a service)

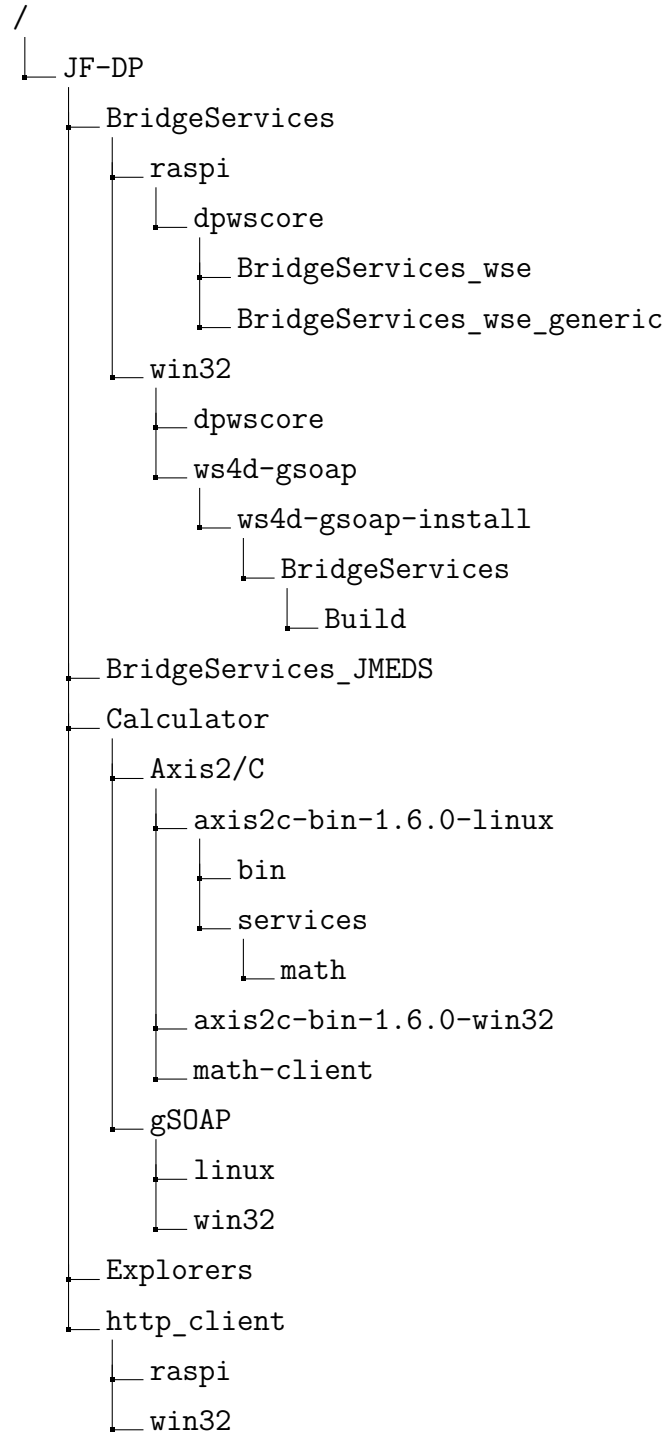
```
<wsdl:binding name="BridgeServicesBinding" type="tns:BridgeServices">
    <wsoap12:binding style="document" transport="http://schemas.xmlsoap
        .org/soap/http"/>
    <wsdl:operation name="WriteItems">
      <wsoap12:operation soapAction="https://www.rexcontrols.cz/soa/
          BridgeServices/WriteItems" soapActionRequired="false"/>
      <wsdl:input>
        <wsoap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <wsoap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="ReadItems">
      <wsoap12:operation soapAction="https://www.rexcontrols.cz/soa/
          BridgeServices/ReadItems" soapActionRequired="false"/>
      <wsdl:input>
        <wsoap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <wsoap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="ReadItemsAsync">
      <wsoap12:operation soapAction="https://www.rexcontrols.cz/soa/
          BridgeServices/ReadItemsAsync" soapActionRequired="false"/>
      <wsdl:input>
        <wsoap12:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="ReadItemsAsyncCallback">
      <wsoap12:operation soapAction="https://www.rexcontrols.cz/soa/
          BridgeServices/ReadItemsAsyncCallback" soapActionRequired="
          false"/>
      <wsdl:output>
        <wsoap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="BridgeService">
    <wsdl:port name="BridgeServicesPort0" binding="tns:
        BridgeServicesBinding">
      <wsoap12:address location="http://127.0.0.1:80/cz/soa/
          BridgeService"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Příloha B

Obsah přiloženého CD

K této práci je přiloženo CD, na kterém jsou uloženy zdrojové kódy. Struktura adresářů je nastíněna na obrázku č. B.1.

- Adresář *BridgeServices* je rozdělen na adresáře *raspi* a *win32* podle platformy, uvnitř se nachází implementace webové služby *BridgeServices* ve frameworku *DPWSCore* a *WS4D-gSOAP*.
- Adresář *BridgeServices_JMEDS* obsahuje implementaci služby v jazyce Java.
- Adresář *Calculator* obsahuje jednak složku s repositáři pro pro linux a windows pro framework *Axis2/C* se službou jednoduché kalkulačky a s jednoduchým klientem této služby, a dále je zde složka s frameworkem *gSOAP* se stejnou službou opět pro windows a linux.
- Adresář *Explorers* obsahuje několik průzkumníků pro vyhledávání zařízení DPWS, většina je implementována v jazyce Java.
- Adresář *http_client* obsahuje jednoduchého klienta pro webovou službu *BridgeServices* pro platformy *raspi* a *win32*.



Obrázek B.1: Struktura adresářů na přiloženém CD.

Příloha C

Uživatelská příručka

C.1 Axis2/C

Služba

Kompilace Linux:

```
gcc -shared -fPIC -olibmath -l$AXIS2C_HOME/include/axis2-1.6.0
-L$AXIS2C_HOME/lib -laxutil -laxis2_axiom -laxis2_parser
 -laxis2_engine -lpthread -laxis2_http_sender
 -laxis2_http_receiver math.c math_skeleton.c
```

Klient

Kompilace Linux:

```
gcc -o math-client math-client.c  -I$AXIS2C_HOME/include/axis2-1.6.0/
-L$AXIS2C_HOME/lib -laxutil -laxis2_axiom -laxis2_parser
-laxis2_engine -lpthread -laxis2_http_sender -laxis2_http_receiver
-ldl -Wl,--rpath -Wl,$AXIS2C_HOME/lib
```

C.2 gSOAP

Služba

Kompilace Linux:

```
g++ -o Server server.cpp soapC.cpp soapServer.cpp stdsoap2.cpp
```

Kompilace Windows:

```
cl /EHsc server.cpp soapC.cpp soapServer.cpp stdsoap2.cpp
```

Klient

Kompilace Linux:

```
g++ -o Client client.cpp soapC.cpp soapClient.cpp stdsoap2.cpp
```

Kompilace Windows:

```
cl /EHsc client.cpp soapC.cpp soapClient.cpp stdsoap2.cpp
```

Spuštění:

```
>client.exe 2 + 3
```

C.3 WS4D-gSOAP

Pro kompilace je nutné *Microsoft Visual Studio 2005*. Soubor řešení je `BridgeService.sln`

Služba

Spuštění:

```
>BridgeService.exe -i 192.168.1.1 -u urn:uuid:‘uuidgen‘
```

Klient

Spuštění:

```
>BSvcs_benchmark_client.exe -i 127.0.0.1 -o w -n 1000
-c mtuner.PIDMA:ti;mtuner.PIDMA:amp -v 4;5.0
```

C.4 DPWSCore

Služba

Kompilace Linux:

```
gcc -I../include -I./src/gen server.c listener.c fileio.c
RexWSoc_client.c cJSON.c wsServerLib.c -Llib -ldcruntime
-lm -lwebsockets -o server -Wl,-rpath=./lib
```

Kompilace Windows:

```
cl -DWIN32 -DDEBUG  /D "_DEBUG" /D "_CONSOLE" /D "DC_USER_CONFIG"
 /D "_UNICODE" /D "UNICODE" /I ..\include /I .\src\gen server.c
  wsServerLib.c listener.c fileio.c RexWSoc_client.c cJSON.c
  /link /LIBPATH:C:\dpwscore\dpwscore-2.5.1\lib dcruntime.lib
  /LIBPATH:c:\dpwscore\dpwscore-2.5.1\BridgeServices\libwebsockets\lib
  websockets.lib
```

Kompilace Windows (generická služba):

```
cl /D "WIN32" /D "_DEBUG" /D "_CONSOLE" /D "DC_USER_CONFIG"
 /D "_UNICODE" /D "UNICODE" /I ..\include /I .\src\gen server.c
  listener.c fileio.c generic_parsing.c RexWSoc_client.c cJSON.c
  /link /LIBPATH:C:\dpwscore\dpwscore-2.5.1\lib dcruntime.lib dcxml.lib
  /LIBPATH:c:\dpwscore\dpwscore-2.5.1\BridgeServices_generic\libwebsockets\lib
  websockets.lib
```

Klient

Kompilace Windows:

```
cl client.c listener.c ws_Client.c .\gen\wsClientLib.c
.\gen\wsHandlerLib.c /Od /I "..\include" /I ".\gen" /D "WIN32"
/D "_DEBUG" /D "_CONSOLE" /D "DC_USER_CONFIG" /D "_UNICODE" /D "UNICODE"
/link /LIBPATH:".\..\lib" iphlpapi.lib ws2_32.lib ssleay32.lib
libeay32.lib xmltoolslib.lib dpwslib.lib gsoaplib.lib common.lib
dcpl.lib al.lib  kernel32.lib user32.lib gdi32.lib winspool.lib
comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib
odbc32.lib odbccp32.lib
```

C.5 http_client

Kompilace Linux:

```
gcc -o http_client -I../include http-post.c -lcurl
```

Kompilace Windows:

```
cl http-post.c /I..\include /link /LIBPATH:..\lib libcurl.lib
```

Spuštění:

```
>http-post.exe -i 127.0.0.1 -p 9867 -o w -n 1000
```