VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

XMPP API PRO WEBOVÉ APLIKACE

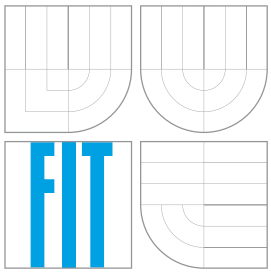
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ VAISAR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

XMPP API PRO WEBOVÉ APLIKACE

XMPP API FOR WEB APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ VAISAR
AUTHOR

VEDOUCÍ PRÁCE — Ing. MAREK SCHMIDT
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce se zabývá návrhem a implementací API umožňující tvorbu XMPP aplikací v ECMAScriptu. Její součástí je implementace navrženého API formou ukázkového pluginu pro XMPP klienta Jabbim. Součástí práce je též implementace hry Dáma vytvořená v ECMAScriptu s využitím navrženého API. Na závěr jsou představena možná rozšíření API v budoucnosti.

Abstract

This thesis deals with design and implementation of an API, that allows creating XMPP applications using ECMAScript. Thesis includes implementation of API as a plugin for Jabbim XMPP client. Part of the thesis is an ECMAScript implementation of board game Draughts using designed API. Finally possibilities of future extensions are discussed.

Klíčová slova

XMPP, Jabber, ECMAScript, JavaScript, WebKit, Návrh API, instant messaging

Keywords

XMPP, Jabber, ECMAScript, JavaScript, WebKit, API design, instant messaging

Citace

Tomáš Vaisar: XMPP API pro webové aplikace, bakalářská práce, Brno, FIT VUT v Brně, 2011

XMPP API pro webové aplikace

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana inženýra Marka Schmidta.

.....
Tomáš Vaisar
18. května 2011

Poděkování

Rád bych poděkoval Ing. Marku Schmidtovi za vedení této práce.

Obsah

Kapitola 1

Úvod

Cílem této práce je navrhnout aplikační programovací rozhraní (API), které bude umožňovat jednoduchou tvorbu aplikací postavených na XHTML a ECMAScriptu, které spolu budou komunikovat pomocí otevřeného komunikačního prokotolu XMPP. Toto API bude umožňovat, že aplikace budou přenosné mezi různými internetovými komunikátory[1]. Pro každý komunikátor bude nutno vytvořit zásuvný modul[2], který bude dané API implementovat a umožní tak běh těchto aplikací v daném komunikátoru. Samotné aplikace však již nebude nutno jakkoliv modifikovat.

Kapitola 2 diskutuje aktuální stav v oblasti spolupráce protokolu XMPP a ECMAScriptu. Zkoumá existující knihovny umožňující současné použití těchto dvou technologií a projekty, které na tomto principu pracují.

V kapitole 3 jsou probírány použité technologie, především komunikační protokol XMPP, skriptovací jazyk JavaScript a jádro pro tvorbu webových prohlížečů WebKit.

Kapitola 4 pojednává o návrhu aplikačního programovacícho rozhraní, které budou budoucí aplikace využívat. Zabývá se především návrhem formátu XML zpráv, pomocí kterých spolu budou aplikace komunikovat.

V kapitole 5 je probírána výsledná implementace navrženého systému. Obsahuje též popis funkcionality, kterou výsledné API poskytuje.

V rámci vývoje byly vytvořeny i testovací aplikace, které demonstrující funkcionalitu API. Tyto aplikace jsou detailněji rozebírány v kapitole 6.

Závěrečná kapitola 7 rozebírá možnosti budoucích rozšíření současné implementace.

[1] Aplikace umožňující textovou (či hlasovou nebo obrazovou) komunikaci prostřednictvím Internetu
[2] Doplněk aplikace rozšiřující její funkcionalitu

Kapitola 2

Charakteristika současného stavu

Tato kapitole se zabývá charakteristikou aktuálního stavu v oblasti spolupráce protokolu XMPP a ECMAScriptu. V současné době existuje několik projektů, které zprostředkovávají komunikaci dvou a více uživatelů prostřednictvím protokolu XMPP v prostředí webového prohlížeče. Mezi nejznámější projekty patří Meebo[1], Jappix[2], Google Talk[3] a Facebook Chat[4]. Ve všech případech dostane uživatel k dispozici rozhraní, které napodobuje prostředí klasického internetového komunikátoru.

Tyto služby však nenabízí možnost vytvořit na jejich základě vlastní aplikaci. K tomuto účelu je vytvořeno několik knihoven pro JavaScript. Ty umožňují vytvoření webových aplikací, které komunikují prostřednictvím XMPP. Patří mezi ně například knihovna JSJaC[5], Strophe[6], xmpp4js[7] nebo dojox.xmpp[8].

Tyto knihovny v sobě buď implementují funkcionalitu celého XMPP klienta, nebo ve skutečnosti odesílají zprávy na server, který zprostředovává XMPP komunikaci za ně. První přístup je nevhodný z pohledu náročnosti na zpracování XMPP komunikace v rámci JavaScriptu. Druhý přístup může být nevhodný z hlediska bezpečnosti, protože server přeposílající zprávy si může tyto zprávy ukládat nebo s nimi jinak manipulovat (záleží na implementaci knihovny).

U většiny z těchto knihoven bohužel chybí použitelná dokumentace.

Tato práce řeší jiný přístup. Místo přinášení XMPP do webu, přináší web do XMPP. V praxi si pod tím lze přestavit, že bude vytvořen zásuvný modul, který umožní v prostředí XMPP komunikátoru spustit webovou aplikaci, která bude rozšířena o možnost komunikace pomocí tohoto protokolu. Lze tak využít již existující funcionality komunikátoru. Tím odpadá nutnost implementovat celou logiku XMPP komunikace a lze se soustředit na tvorbu samotné aplikace.

[1] Meebo – http://www.meebo.org/
[2] Jappix – http://www.jappix.com/
[3] Google Talk – http://www.google.com/talk/
[4] Facebook Chat – http://www.facebook.com/
[5] JSJaC – http://blog.jwchat.org/jsjac/
[6] Strophe – http://strophe.im/
[7] xmpp4js – http://xmpp4js.sourceforge.net/
[8] dojox.xmpp – http://dojotoolkit.org/reference-guide/dojox/xmpp.html

Kapitola 3

Použité technologie

Tato kapitola se zabývá technologiemi, které jsou v této práci využity. Jedná se především o popis komunikačního protokolu XMPP, jazyka JavaScript a jeho formátu pro serializaci dat – JSON. V závěru je popsán WebKit – open-source jádro pro webové prohlížeč, které bude sloužit k zobrazování a běhu aplikací využívajících navrhované API.

3.1 XMPP

XMPP neboli Extensible Messaging and Presence Protocol (do češtiny přeložitelné jako „Rozšiřitelný protokol pro zasílání zpráv a informací o přítomnosti u počítače“) je otevřený standardizovaný protokol pro zasílání zpráv (anglicky *instant messaging*), textovou komunikaci více uživatelů zároveň a v rámci této komunikace například také pro přenos zvuku a videa.

Vznikl v roce 1999 ve snaze vytvořit otevřenou technologii pro textovou komunikaci v rámci Internetu, která by poskytovala alternativu k existujícím uzavřeným protokolům (jako například ICQ nebo MSN). K těmto uzavřeným protokolům totiž neexistuje žádná oficiální specifikace a proto je velice obtížné vytvořit program, který pomocí takového protokolu komunikuje. V případě ICQ je dokonce vytvoření takového programu porušením licenčních podmínek a může být trestně stíháno.

Pro XMPP se běžně se používá název **Jabber**, nicméně se jedná o registrovanou obchodní značku firmy Jabber Inc., tudíž v oficiálních dokumentech (například na stránkách poskytovatelů XMPP služeb) se toto označení příliš často nevyskytuje.

3.1.1 Hlavní přednosti

Hlavními přednostmi XMPP oproti jiným protokolům (např. proti protokolu OSCAR, který využívá služba ICQ) jsou:

- **otevřenost** – Na protokol se nevztahuje žádná komerční licence a je ho tedy možné používat k libovolným účelům a dokonce ho dle vlastních potřeb modifikovat.
- **standardizovanost** – Organizace IETF[1] prohlásila protokol za standard a ten byl popsán v RFC 3920[2] a RFC 3921[3].

[1] IETF (Internet Engeneering Task Force) – http://www.ietf.org/
[2] RFC 3920 – http://www.ietf.org/rfc/rfc3920.txt
[3] RFC 3921 – http://www.ietf.org/rfc/rfc3921.txt

- **bezpečnost** – Protokol umožňuje uzavřít síť vůči jiným sítím, díky čemuž mohou uživatelé komunikovat např. pouze v rámci jedné domény (`frantisek@firma.cz` může komunikovat s `josef@firma.cz`, ale ne s `antonin@jinafirma.cz`). Navíc do samotného jádra protokolu je zaneseno šifrování pomocí technologie SASL[4] a kryptografického protokolu TLS[5].

- **decentralizovanost** – Podobně jako například u e-mailu neexistuje žádný jediný řídící server, ke kterému by se museli všichni uživatelé přihlašovat. Každý si může na svůj počítač nainstalovat XMPP server a dát ho k dispozici uživatelům. V kombinaci s možností uzavřít síť vůči ostatním sítím je tedy XMPP takřka ideálním kandidátem například pro interní komunikační řešení v rámci firem.

- **rozšiřitelnost** – Pokud chce někdo využít protokol v rámci svého projektu, ale nedostačuje mu existující funkcionalita, lze komunikaci obohatit o zprávy vlastního typu. Je čistě na klientské aplikaci, jak se zprávami naloží (jak je zpracuje). Server ani protokol je nijak neomezují.

Mezi další výhody patří například to, že uživatel může být přihlášen na více místech najednou. Kam uživateli posílat zprávy pak server rozhoduje podle toho, u kterého připojení má uživatel nastavenou vyšší prioritu[6]. Protokol také podporuje vzdálené ovládání připojených komunikátorů pomocí technologie XML-RPC[7] (toto vzdálené ovládání však musí být v cílovém klientovi explicitně povoleno), čehož lze využít například k přesměrování dosud nepřečtených zpráv do jiného komunikátoru.

Další vlastností je podpora komunikace více uživatelů najednou (tzv. *MUC – Multi-user conference*), která vypadá obdobně jako komunikace prostřednictvím protokolu IRC. Uživatelé mají k dispozici *místnost*, ve které probíhá jejich hromadná konverzace. Tato místnost může být veřejná (tedy přístupná pro koholiv) nebo chráněná heslem, které je vyžadováno pro vstup do místnosti. Ve výchozím stavu se zprávy odesílají všem uživatelům. Pokud chtějí dva uživatelé *šeptat* (tedy komunikovat spolu izolovaně od ostatních), je mezi nimi vytvořena zvláštní místnost.

3.1.2 JID

Každý uživatel je jednoznačně indentifikován pomocí tzv. JID – *Jabber Identificator* skládajícího se ze tří částí:

- **uživatelské jméno** – přezdívka, či jméno uživatele(unikátní pro daný server)

- **server** – název serveru, na kterém je uživatel zaregistrován (např. `jabbim.cz`)

- **resource** – identifikátor prostředku, pomocí něhož uživatel připojen (nejčastěji název počítače nebo komunikačního programu, který uživatel aktuálně používá)

Právě díky tomu, že je uživatelovo spojení se serverem identifikováno i prostředkem, pomocí kterého je přihlášen, je možné, aby byl jeden uživatel přihlášen najednou z více

[4] Simple Authentication and Security Layer – http://tools.ietf.org/html/rfc4422

[5] The Transport Layer Security (TLS) Protocol – http://tools.ietf.org/html/rfc5246

[6] Priorita – číselná hodnota určující preferované spojení, zprávy jsou ve výchozím chování doručovány spojení s vyšší prioritou

[7] XML-RPC – vzdálené volání procedur pomocí technologie XML skrz protokol HTTP. Více viz http://www.xmlrpc.com/.

míst. Pro server jsou `incik@jabbim.cz/jabbim` a `incik@jabbim.cz/htcdesire` dvě různá spojení jednoho uživatele. Tato spojení mezi sebou dokonce mohou vést normální konverzaci.

Pokud je při komunikaci v JID uveden *resource* a uživatel je přihlášen z více míst, je možné komunikovat s tímto spojením bez ohledu na to, zda má uživatel u jiného spojení nastavenou vyšší prioritu. Pokud není *resource* uveden, je zpráva doručena spojení s nejvyšší priotitou.

3.1.3 Popis XMPP zpráv

Komunikace probíhá formou výměny zpráv ve formátu XML, které jsou snadno čitelné nejen pro počítač, ale i pro lidi.

```
<message xmlns='jabber:client' from='incik@jabbim.cz/jabbim'
    xml:lang='en' to='johntestovic@jabbim.cz/jabbim' type='chat'
    id='H_329'>
    <body>ahoj!</body>
</message>
```

Obrázek 3.1: Ukázková zpráva typu `message`

Kromě zpráv typu `message` uvedeného v ukázce 3.1 jsou dále využívány zprávy typu `IQ` (viz ukázka 3.2). Ty jsou určeny primárně pro komunikaci typu *dotaz – odpověď*. Typ této zprávy je identifikován pomocí atributu `type`, který může nabývat následujících hodnot:

1. **set** – používá se při zasílání dotazů, které vyžadují nějaké parametry
2. **get** – používá se u dotazů, které nevyžadují žádné parametry
3. **result** – odpověď zasílaná v případě úspěšného vykonání požadovaného dotazu
4. **error** – odpověď zasílaná, pokud dojde k chybě při zpracování dotazu. Obsahuje původní dotaz a chybové hlášení (kód chyby).

```
<iq xmlns='jabber:client' type='result' id='bind_1' >
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
        <jid>incik@jabbim.cz/jabbim</jid>
    </bind>
</iq>
```

Obrázek 3.2: Ukázková zpráva typu `IQ`

Každá XMPP zpráva dále povinně obsahuje následující atributy:

- **xmlns** – určuje jmenný prostor, do kterého zpráva patří (ty slouží k určení toho, pro jakou službu se daná zpráva používá a ke kontrole, zda je zaslaná zpráva validní)
- **type** – určuje typ zprávy, na základě kterého aplikace příjemce určuje. Kromě `chat`, využívaného pro konverzace, lze použít například typ `message`, které se chovájí podobně jako e-mailové zprávy (konkrétní chování se liší dle používaného komunikátoru).
- **id** – každá zaslaná zpráva má svůj jednoznačný číselný identifikátor, díky kterému je možné například zjistit, která zpráva je reakcí na kterou

Pro zprávy typu `message` jsou také povinné parametry:

- **from** – určuje, kým byla zpráva odeslána
- **to** – určuje, komu je zpráva určena

Další atributy a elementy zprávy jsou volitelné. Zpráva může obsahovat text, data ve formě obrázku nebo informaci o tom, že druhý uživatel právě píše zprávu.

3.1.4 XMPP transporty

Zajímavou vlastností XMPP je možnost komunikovat se sítěmi používajícími jiné protokoly (např. ICQ) pomocí takzvaných transportů. Narozdíl od komunikátorů, který dokáží komunikovat v rámci více protokolů zároveň, fungují transporty na úrovni serveru. Uživatel se přihlásí k bráně, která je připojená například k ICQ serveru. Uživatel poté pomocí standardního XMPP komunikuje s bránou a ta komunikuje s ICQ serverem pomocí jeho protokolu (OSCAR).

Registrace k bráně je svazaná s účtem uživatele a ne s jeho komunikátorem. Tudíž kdekoliv se uživatel přihlásí svým XMPP účtem, může mít k dispozici své ICQ či MSN kontakty, své e-maily, zprávy ze sociální sítě Twitter apod.

3.1.5 Současné využití

Protokol XMPP je ověřený díky nasazení v aplikacích, které využívají denně miliony lidí. V současné době jej využívá společnost Google pro svoji aplikaci Google Talk. Ta umožňuje primárně komunikaci prostřednictvím XMPP v rámci webového prohlížeče. Ten kromě zasílání zpráv podporuje i přenos hlasu a videa.

Další aplikace, která v nedávné době začala XMPP využívat, je sociální síť Facebook resp. její služba *Facebook chat* [10]. Tato komunikační síť je bohužel uzavřena vůči ostatním sítím, což v praxi znamená, že člověk může pro Facebook chat používat svůj oblíbený XMPP komunikátor a psát si tedy se všemi přáteli v rámci sítě Facebook, ale nemůže si psát například s uživatelem `johndoe@jabber.org`. Pokud by Facebook se svými více než 500 miliony uživatelů [4] svoji síť otevřel vůči ostatním sítím, stalo byl se XMPP nejrozšířejnějším komunikačním prostředkem v rámci Internetu (vyjma e-mailu).

3.2 ECMAScript

V roce 1995 vyvinul Brendan Eich ve firmě Netscape programovací jazyk Mocha. Cílem bylo umožnit lidem, kteří nejsou tak sběhlí v programování (cíleno především na webdesignéry), oživit své webové stránky pomocí jednoduchých animací a dát jim možnost alespoň částečně ovlivňovat funkcionalitu – například chování webových formulářů. Alternativou v té době byla například možnost tvořit tzv. Java Applety – programy napsané v jazyce Java, které běží v rámci internetového prohlížeče jako prvek stránky. Tyto applety však byly velice náročné na výkon počítače, navíc vyžadovaly znalost programovacího jazyka Java.

Během následujícího půl roku byl jazyk Mocha přejmenován na LiveScript a později po oznámení spolupráce mezi firmami Netscape a Sun Microsystems byl přejmenován na JavaScript (přestože nemá s programovacím jazykem Java nic společného). První oficiální implementace JavaScriptu byla v internetovém prohlížeči Netscape Navigator 2.0. V roce

1996 se objevila jeho implementace i v prohlížeči Internet Explorer 3.0 od společnosti Microsoft – jejich verze se jmenovala JScript. Nicméně vzhledem k odlišnému přístupu k DOM[8] spolu nebyly tyto dvě verze zcela kompatibilní.

Proto se téhož roku (1996) organizace ECMA rozhodla JavaScript standardizovat. Výsledným produktem byl jazyk **ECMA-262**. Tento název je však pro běžnou praxi poněkud nepraktický, proto se jazyk začal označovat názvem **ECMAScript**. Ani tento název se však v praxi příliš neujal – sám Brendan Eich o něm prohlásil, že zní jako jméno kožní choroby [8]. Běžně se tedy používá název původní označení JavaScript, který i já budu ve zbytku práce používat.

3.2.1 Vlastnosti

JavaScript je dynamicky typovaný plně objektově orientovaný skriptovací jazyk. Na rozdíl od jazyků jako je Java nebo C++ však není JavaScript založen na třídách, ale na prototypech. Prototyp je předpis, který dává (podobně jako třída) vlastnosti objektu, který je podle něho vytvořen. Na rozdíl od třídy má ale například tu výhodu, že pokud změníme za běhu programu prototyp, změní se zároveň s tím i všechny objekty, které podle něj byly vytvořeny.

V souvislosti s tím je třeba zmínit, že v JavaScriptu je vše objekt. Řetězec, pole, funkce (viz dále) a dokonce i číslo je objekt. Má tedy atributy, ke kterým lze přistupovat a metody, které lze volat[2]. Konkrétně u čísla však nelze volit zcela klasický přístup pro zápis volání metod, protože kód v ukázce 3.3 způsobí syntaktickou chybu. Pokud za číslem následuje tečka, interpret předpokládá, že se jedná o desetinné číslo, a proto když místo čísla narazí na znak `t`, zahlásí chybu. Jak toto chování obejít a zavolat metodu `toString()` nad číslem `42` je vidět v ukázce 3.4.

```
42.toString();
```

Obrázek 3.3: Příklad nesprávného volání metody `toString()` nad číslem `42`

```
42..toString(); // je též možná varianta 42.0.toString()
42 .toString(); // také regulérní zápis (povšimněte si mezery)
(42).toString(); // patrně nejčitelnější zápis
```

Obrázek 3.4: Ukázka způsobů, jak zavolat metodu `toString()`na číslem

JavaScript se dá označit za *multiparadigmatický* jazyk – tedy jazyk umožňující různý přístup k zápisu programů a řešení problémů.

Primárně užívaný přístup je imperativní, tedy způsob zápisu, na jaký jsme zvyklí z programovacích jazyku jako je C++ nebo Java. JavaScript si ale propůjčuje i některé vlastnosti **funcionálních** jazyků. Díky tomu, že vše, i funkce, je objekt, tak je i funkci možno přiřadit do proměnné nebo ji vracet jako návratovou hodnotu jiné funkce.

Dalším prvkem známým především z funkcionálních jazyků jsou tzv. **vnořené funkce** (anglicky *nested functions*). Jak již název zlehka napovídá, jedná se o funkce, které jsou definovány uvnitř jiných funkcí. Tyto pak mohou být volány pouze v rámci funkcí, ve kterých jsou definovány a určitým způsobem tak nahrazují privátní metody. Vnořená funkce

[8]DOM (Document Object Model) tedy *objektový model dokumentu* je objektová reprezentace struktury XML dokumentu.

má navíc k dispozici všechno, co bylo k dispozici vnější funkci před jejím zavoláním i poté, co vnější funkce ukončí svojí činnost.

Velkou devizou JavaScriptu jsou **anonymní funkce**. Ty (jak již název napovídá) nemají název a proto je nelze volat z jiných částí kódu než tam, kde jsou definovány. V javascriptových aplikacích funkce velmi běžně nevrací hodnotu, ale požaduje jako jeden z parametrů název funkce, které předá výsledek své činnosti a zavolá ji – tzv. *callback*[9]. Protože se však ve skutečnosti nepředává název funkce, ale reference na ni, tak je možné místo této reference přímo zapsat anonymní funkci, která získaný výsledek zpracuje. Tímto způsobem se například zapisuje valná většina kódu využívajícího populární javascriptové knihovny **jQuery**[10].

Jednoduchý kód bez anonymní funkce v ukázce 3.5 lze tedy snadno převést na funkčně ekvivalentní kód 3.6.

```
function resultHandler(result) { alert(result); }
...
doSomething(35.42, 18.5, resultHandler);
```

Obrázek 3.5: *Callback* předaný referencí na funkci

```
doSomething(35.42, 18.5, function(result) { alert(result); });
```

Obrázek 3.6: *Callback* tvořený anonymní funkcí

Při použití tohoto přístupu odpadá potřeba definovat funkce, které se použijí jen jednou. Naopak pokud se později zjistí, že je potřeba onu anonymní funkci volat znovu z jiného místa, lze ji lehce upravit na pojmenovanou funkci a tu poté volat (viz ukázky 3.5 a 3.6).

Zajímavostí jsou tzv. **samovolající se funkce** (anglicky *self-invoking functions*). Jedná se o funkce, jejichž definice je ihned následována operátorem volání funkce - `()`. Takovýto zápis způsobí, že funkce je vykonána ihned poté, co interpret přečte její definici. Pokud do své aplikace načítáte externí javascriptovou knihovnu (například jQuery), pak je pravděpodobné, že bude definována přibližně tak, jak je uvedeno v ukázce 3.7.

```
(function (window, undefined) {
    if (window.mojeTrida === undefined) {
    ...
    }
})(this);
```

Obrázek 3.7: Ukázka definice *self-invoking* funkce

Takovýto zápis má dvě výhody. Zaprvé bude knihovna připravena k použití hned po načtení souboru, ve kterém je definována. Zadruhé si povšimněte, že ač je definice funkce uvedena se dvěma parametry, předáváme jí jen jeden. Díky tomu, že parametr `undefined` nemá zadanou hodnotu, je tato hodnota taktéž *undefined* (obdoba `null` v C/C++). Pokud uvnitř této funkce použijeme slovo `undefined`, nebudeme pracovat přímo s javascriptovou hodnotou *undefined*, ale s proměnnou `undefined`, jejíž hodnota je *undefined*. Zní to jako

[9] Callback – kód (nebo refence na něj), který se předává jako parametr jinému kódu.
[10] jQuery – http://jquery.com/

zbytečnost, ale z hlediska zabezpečení správné funkcionality kódu je tento přístup velmi vhodný. Pokud by totiž někdo ve zdrojovém kódu před načtením knihovny náhodou použil zápis uvedený v ukázce 3.8, došlo by s velmi vysokou pravděpodobností k nesprávnému chování. Protože však v lokálním kontextu (tedy uvnitř funkce) překrývají lokální proměnné ty, které jsou definovány na vyšších úrovních, je náš kód „v bezpečí“.

```
var undefined = true;
```

Obrázek 3.8: Ukázková zpráva typu `message`

3.2.2 JSON

Tento jednoduchý textový formát, navržený pro výměnu dat mezi různými systémy, je součástí JavaScriptu, která byla standardizovaná ve třetím vydání standardu ECMA-262 v roce 1999 [7].

JSON podporuje datové typy *řetězec*, *číslo*, *pole*, *objekt*, *pravda*, *nepravda* a `null` – tedy prázdná hodnota. Složitější datové struktury v něm existují pouze dvě. Již zmíněný *objekt* a *seznam*. Objekt je reprezentován jako kolekce dvojic typu klíč – hodnota. Seznam je kolekce prvků oddělených čárkou, uzavřených do hranatých závorek.

```
{
    pokusnyObjekt: {
        id: 1,
        nazev: 'Pokusný objekt',
        pole: [1,2,3,4],
        data: {
            command: 'unset',
            arguments: null
        },
        notifikovat: true,
        logovat: false,
        potomek: null
    }
}
```

Obrázek 3.9: Příklad reprezentace objektu v JSON

JSON je formát natolik univerzální, že se v současné době používá nejen v JavaScriptu, ale i v dalších programovacích jazycích, kde nahrazuje další častou formu serializace objektů, a sice XML. Tento formát, který měl být původně čitelný pro lidi (*human-readable*) se totiž pomalu stává zbytečně složitým [6].

Aby bylo možno ověřit, zda je XML validní a bylo ho možné správně deserializovat, je třeba pro něj definovat XSD[11]. To u JSONu neplatí. Ten je čistě textový a je na cílové aplikaci, jak s daným objektem naloží.

Jak by vypadal ekvivalent ukázky 3.9 je vidět na ukázce 3.10. Lze namítnout, že existuje kratší varianta zápisu (viz ukázka 3.11), nicméně ta je na tom s čitelností v některých případech ještě o něco hůře.

[11] XML Schema Definition (definice schématu XML) – Dokument v jazyce XML, který popisuje strukturu XML objektů

```
<pokusnyObjekt>
    <id>1</id>
    <nazev>Pokusný objekt</nazev>
    <pole>
        <item>1</item>
        <item>2</item>
        <item>3</item>
        <item>4</item>
    </pole>
    <data>
        <dataObject>
            <command>unset</command>
            <arguments />
        </dataObject>
    </data>
    <notifikovat>true</notifikovat>
    <logovat>false</logovat>
    <potomek />
</pokusnyObjekt>
```

Obrázek 3.10: Ukázková zpráva typu `message`

```
<pokusnyObjekt id='1' nazev='Pokusný objekt' notifikovat='true'
    logovat='false'>
    <pole>
        <item>1</item>
        <item>2</item>
        <item>3</item>
        <item>4</item>
    </pole>
    <data>
        <dataObject command='unset' />
    </data>
</pokusnyObjekt>
```

Obrázek 3.11: Ukázková zpráva typu `message`

Co se datové velikosti týče, tak je JSON ve většině případů úspornější bez ohledu na formu zápisu XML.

3.3 WebKit

WebKit je open-source jádro pro webové prohlížeče. Byl vytvořen firmou Apple Inc. na základě jader KHTML a KJS, což byla původně jádra (vykreslovací, respektive javascriptové) pro prohlížeč Konqueror, patřící do projektu KDE[12]. Firma využila tyto technologie jako základ pro svůj prohlížeč Safari. V roce 2005 jej uvolnila jako open-source projekt[9].

[12]KDE - K Desktop Environment - populární grafické uživatelské rozhraní pro operační systémy unixového typu (především GNU/Linux a FreeBSD)

Patrně největší popularity dosáhl WebKit poté, co jej použila firma Google pro svůj webový prohlížeč Google Chrome[13]. Navíc v současné době jeho implementaci nalezneme i zařízeních používající operační systém Android, MeeGo a Symbian.

Součástí WebKitu (kterou ocení zejména vývojáři webových stránek) je integrovaný nástroj *Web Inspector*. Ten je obdobou velice populárního zásuvného modulu FireBug pro prohlížeč Mozilla Firefox. Mezi jeho hlavní schopnosti patří:

- **živé zobrazování DOMu webové stránky** – pokud dojde ve stránce ke změně (přesun prvku v rámci DOMu nebo jen změna části textu), ve Web Inspektoru se tato změna ihned promítne
- **vestavěný nástroj pro ladění JavaScriptu** – umožňuje krokovat skripty a provádět změny v javascriptovém kódu bez nutnosti znovu načítat stránku
- **sledování sítě** – umožňuje sledovat odesílané/přijímané HTTP hlavičky a analyzovat jejich obsah

K dalšími výhodám pak například patří podpora HTML5 a CSS3, což jsou standardy, které jsou stále ještě ve fázi schvalování. Díky HTML5 je možné kupříkladu přehrávat videa ze serveru YouTube bez nutnosti mít nainstalovaný Adobe Flash. YouTube totiž nabízí možnost přepnout přehrávání do HTML5 módu, kdy se místo Flashe využívá tagu `<video>` a o vykreslování videa se stará samotný prohlížeč. Prozatím (v květnu 2011) se však jedná o experimentální funcionalitu[3].

Zajímavostí je také podpora technologie WebGL. Ta je rozšířením možností JavaScriptu o možnost vykreslovat interaktivní 3D grafiku přímo v okně prohlížeče za využití výkonu grafické karty.

3.3.1 Rozšíření JavaScriptu

Pokud tvoříme webový prohlížeč na základě WebKitu nebo vytváříme aplikaci, která ho využívá, máme k dispozici možnost rozšířit klientský JavaScript o vlastní objekty. Toho lze využít například k implementaci netriviálních matematických výpočtů, jejichž provádění by v JavaScriptu trvalo nepříjmeně dlouho. Tento objekt je vytvořen v rámci původní aplikace, ve které pracujeme s WebKitem, je tedy definován v programovacím jazyce, ve kterém je tato aplikace vytvářena (tedy v C++, Pythonu, ...).

```
class NaseTrida(QtCore.QObject):
    def __init__(self):
        QtCore.QObject.__init__(self)

    @QtCore.pyqtSlot(result=str)
    def pozdrav(self):
        return 'Dobry den!'
```

Obrázek 3.12: Definice třídy, jejíž instance může být připojena do JavaScriptu

Použitím zápisu uvedného v ukázce 3.12 dostaneme k dispozici třídu, jejíž instanci lze zaregistrovat do JavaScriptu (respektive do javascriptového objektu `window`). Tato *registraci* se dá provést pomocí metody `addToJavaScriptWindowObject`, která jako argumenty

[13]Google ve svém prohlížeči Google Chrome jako javascriptové jádro používá vlastní řešení nazvané v8.

přijímá instanci námi definované třídy a textový řetězec určující, pod jakým jménem bude objekt z JavaScriptu dostupný – viz ukázka 3.13.

```
mainFrame.addToJavaScriptWindowObject('naseTrida', NaseTrida())
```

Obrázek 3.13: Registrace vlastní třídy do JavaScriptu

Takto do JavaScriptu přidaný objekt je poté v dokumentu (naší webové aplikaci) přístupný přes `window.naseTrida` a lze s ním pracovat jako s každý jiným objektem.

```
alert(naseTrida.pozdrav()); // zobrazí hlášku s~textem 'Dobry den!'
```

Obrázek 3.14: Ukázka práce s třídou, o kterou byl JavaScript programově rozšířen

Jak je z ukázky 3.14 patrné, lze k našemu objektu a jeho metodám přistupovat i pomocí kratšího zápisu `naseTrida`, respektive `naseTrida.pozdrav()`, protože objekt `window` se v JavaScriptu uvažuje jako implicitní jmenný prostor.

Kapitola 4

Návrh

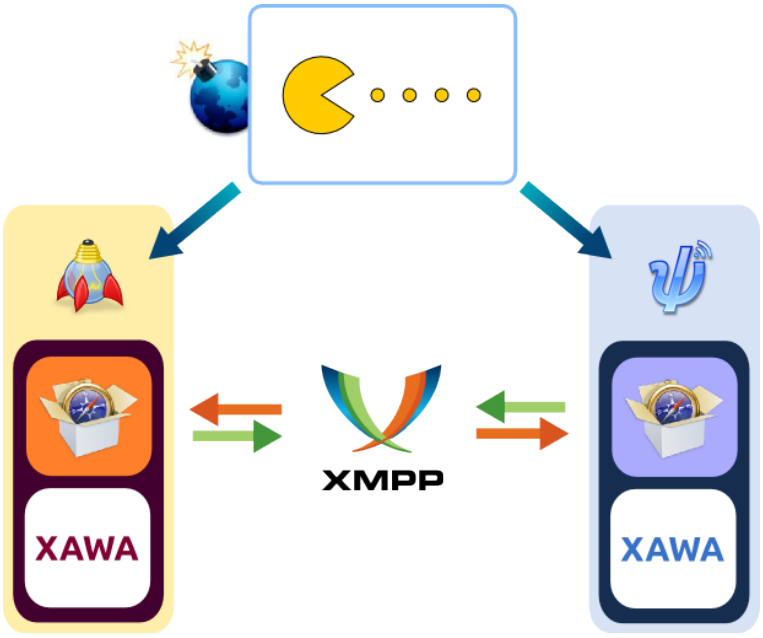
V této kapitole je probíran návrh způsobu komunikace aplikací využívajících navrhované API a princip fungování zásuvného modulu, který toto API implementuje.

API je zkratka pro Aplikační programovací rozhraní (Application Programming Interface). Jedná se o rozhraní, které definuje, jak pracovat s určitou kolekcí funkcí.

Jak již bylo zmíněno v úvodu, API má umožňovat tvorbu zásuvných modulů pro internetové komunikátory. Navrhovaný princip fungování celého systému je vidět na obrázku 4.1. Pomocí těchto zásuvných modulů poté bude možno spouštět aplikace, které budou toto API využívat (v běžném prohlížeči taková aplikace nebude fungovat, protože ten nebude mít k dispozici XMPP funkce, které API implementuje).

Díky tomu, že vytvářené zásuvné moduly pro komunikátory implementují stejné API (funkce jsou stejně pojmenované a mají ekvivalentní chování), nemusí pro každý komunikátor existovat zvláštní verze aplikace.

V dalším textu budou používat označení **XAWA** (XMPP API for Web Applications).



Obrázek 4.1: Schematický návrh fungování celého systému

4.1 Umístění aplikací

V současné době se nepočítá s tím, že by aplikace byly součástí zásuvného modulu, který implementuje XAWA. Počítá se s umístením na lokálním, či vzdáleném webovém serveru, odkud si ji všichni uživatelé načítají. Toto řešení má tu výhodu, že v případě potřeby aktualizace aplikací stačí novou verzi nahrát jen na jedno místo a uživatelé si ji stáhnou při prvním následujícím přístupu na stránku. Odpadá tedy nutnost složité distribuce mezi uživatele.

4.2 Uživatelské rozhraní

Uživatelské rozhraní bude tvořeno oknem, které bude obsahovat webový prohlížeč. Funkcionalita tohoto prohlížeče bude rozšířena o podporu XAWA. Díky tomu bude umožněno zobrazení a běh webových aplikací, které toto API využívají. Prohlížeč si bude udržovací pomocí metody `addToJavaScriptWindowObject` udržovat referenci na hlavní bod programu, díky němuž bude moci využívat již existující funkcionalitu (posílání/příjem zpráv, přístup k informacím o uživatelích v seznamu kontaktů, apod.).

4.3 Průběh komunikace

Celý systém by měl fungovat následovně:

1. Uživatel A spustí ve svém internetovém komunikátoru aplikaci využívající XAWA.
2. Po spuštění aplikace zašle uživateli B pozvánku k připojení se do této aplikace.
3. Uživatel přijme pozvánku a otevře se mu okno s aplikací, do které byl pozván.
4. Začíná komunikace.
5. Uživatelé pracují s aplikací, ta odesílá zprávy (textové, či datové) druhému uživateli a jeho instance aplikace reaguje na příchozí zprávy.
6. Po ukončení práce s aplikací jedním z uživatelů, je druhému uživateli odeslána zpráva, informující ho, že první uživatel skončil.

4.4 Reakce na přijaté zprávy

Po přijetí zprávy je prohlížečem (komponentou zásuvného modulu, která se stará o zobrazování a běh cílové aplikace) vyvolána událost, na kterou programátor může navázat vlastní funkcionalitu.

4.5 Formát XML zpráv

Jak již bylo zmíněno, protokol XMPP je rozšiřitelný a je ho možno obohatit o vlastní typy zpráv, či modifikovat existující. Této vlastnosti je využito i v této práci. Pro komunikaci bude API používat modifkované zprávy typu `message` a především zprávy typu `IQ`.

4.5.1 Zasílání pozvánek

Pokud chce uživatel A komunikovat s uživatelem B pomocí aplikace využívající XAWA, je nutno o tom uživatele B informovat prostřednictvím pozvánky, která je odeslána z oné aplikace. Taková pozvánka ve formě IQ zprávy (viz ukázka 4.2) v sobě poté obsahuje:

1. Kdo koho zve
2. Název a URL aplikace, k jejímuž používání je uživatel zván
3. Konfigurační objekt aplikace (viz oddíl 5.3)

```
<iq from='incik@jabbim.cz/jabbim' to='johntestovic@jabbim.cz/jabbim'
    type='set'>
    <query xmlns='http://xawa.vaisar.cz'>
        <session appName='Pokusná aplikace'
            appUrl='http://xawa.vaisar.cz/apps/pokusna'>
            <invite />
            <configuration>
                {__window : { width: 640, height: 480 }}
            </configuration>
        </session>
    </query>
</iq>
```

Obrázek 4.2: Pozvánka do aplikace

Uživatel A poté čeká na odpověď, na kterou má uživatel B určitý předem definovaný čas. Pokud uživatel B v zadaném časovém intervalu neodpoví, aplikace pokračuje jako by uživatel B poznání odmítl.

4.5.2 Přijímání/odmítání pozvánek

Je-li uživateli B doručena pozvánka do aplikace (objeví se dialogové okno s informací o tom, kdo uživatele zve do které aplikace), může ji buď přijmout nebo odmítnout. Pokud pozvánku přijme, otevře se mu okno s aplikací a po jejím načtení se uživateli A odešle zpráva, že je připraven ke komunikaci (ukázka 4.3). Pokud pozvání odmítne, pouze se odešle zpráva o tom, že pozvání odmítl (ukázka 4.4).

```
<iq from='incik@jabbim.cz/jabbim' to='johntestovic@jabbim.cz/jabbim'
    type='result'>
    <query xmlns='http://xawa.vaisar.cz'>
        <session appName='Pokusná aplikace'
            appUrl='http://xawa.vaisar.cz/apps/pokusna'>
            <accept />
        </session>
    </query>
</iq>
```

Obrázek 4.3: Zpráva o přijmutí pozvání do aplikace

```
<iq from='incik@jabbim.cz/jabbim' to='johntestovic@jabbim.cz/jabbim'
    type='result'>
    <query xmlns='http://xawa.vaisar.cz'>
        <session appName='Pokusná aplikace'
            appUrl='http://xawa.vaisar.cz/apps/pokusna'>
            <refuse />
        </session>
    </query>
</iq>
```

Obrázek 4.4: Zpráva o odmítnutí pozvání do aplikace

4.5.3 Výměna zpráv

Pro možnost vedení rozhovoru přímo v rámci kontextu aplikace je zavedena možnost zasílání textových zpráv. Jsou k dispozici dvě metody zasílání:

1. zasílání klasických zpráv (např. pokud chce autor aplikace, aby se zprávy ukládaly do historie konverzací)
2. zasílání zpráv ve formátu, který zpracuje jen XAWA – v historii konverzací se tedy tyto zprávy neobjeví

Klasická zpráva zobrazená v ukázce 4.5 je mezi ostatními zprávami identifikována díky použitému předmětu zprávy `xawa_message`. Zpráva s takovýmto předmětem je zachycena zásuvným modulem, který implementuje XAWA a nezobrazí se mezi běžnými přijatými zprávami – je však uložena do historie konverzací.

```
<message xmlns='jabber:client' from='incik@jabbim.cz/jabbim'
    xml:lang='en' to='johntestovic@jabbim.cz/jabbim' type='chat'
    id='H_329'>
    <body>ahoj!</body>
    <subject>xawa_message</subject>
</message>
```

Obrázek 4.5: Klasická zpráva s předmětem `xawa_message`

Pokud autor aplikace chce, aby přijaté zprávy byly zpracovávané výhradně zásuvným modulem, který implementuje XAWA, využije druhou metodu zasílání, a sice upravenou zprávu typu IQ (viz ukázka 4.6).

Aby bylo pravidlo „dotaz – odpověď“ dodrženo i při tomto odesílání textových a datových zpráv, odešle druhá strana po přijetí zprávy jednoduchou zprávu zobrazenou v ukázce 4.7, která oznamuje, že byla původní zpráva v pořádku doručena.

4.5.4 Výměna dat

Data budou zasílána jako textový řetězec reprezentující objekt ve formátu JSON. Na základě zasílání těchto dat, lze měnit stav instance aplikace uživatele B a tím reflektovat změny, které provedl uživatel A.

Jak již bylo zmíněno dříve, původně bylo v plánu k zasílání zpráv využívat zpráv typu `message`. Tyto zprávy by se od normálních lišily vyplněným předmětem `xawa_data` (viz

```
<iq from='incik@jabbim.cz/jabbim' to='johntestovic@jabbim.cz/jabbim'
    type='set'>
    <query xmlns='http://xawa.vaisar.cz'>
        <xawaMessage>
            <body>ahoj!</body>
        </xawaMessage>
    </query>
</iq>
```

Obrázek 4.6: Zpráva typu `IQ` obsahující textou zprávu

```
<iq from='incik@jabbim.cz/jabbim' to='johntestovic@jabbim.cz/jabbim'
    type='set'>
    <query xmlns='http://xawa.vaisar.cz'>
        <ack />
    </query>
</iq>
```

Obrázek 4.7: Zpráva potvrzující přijetí textové zprávy či dat

ukázka 4.8). Taková zpráva by byla identifikována, zpracována zásuvným modulem a nepropadala by mezi ostatní přijaté zprávy.

```
<message xmlns='jabber:client' from='incik@jabbim.cz/jabbim'
    xml:lang='en' to='johntestovic@jabbim.cz/jabbim' type='chat'
    id='H_332'>
    <body>{'player':[{'playerid':'player2'}]}</body>
    <subject>xawa_data</subject>
</message>
```

Obrázek 4.8: Zpráva typu `message` obsahující data ve formátu JSON

Nicméně fakt, že takováto zpráva byla uložena do historie konverzací (do té se ukládají všechny zprávy typu `message`), vedl k tomu, že se pro datové zprávy začal též používat vlastní formát na základě IQ zpráv (viz ukázka 4.9).

Po přijetí této zprávy je opět z důvodu dodržení pravidla „dotaz – odpověď“ odesílateli odeslána zpráva IQ zpráva obsahující `<ack />`.

4.5.5 Zaslání informace o ukončení práce s aplikací

Pokud se jeden z uživatelů rozhodne ukončit práci s aplikací, je třeba o tom, druhého uživatele informovat. Standardně tedy pokud uživatel A zavře okno aplikace, je uživateli B odeslána zpráva, že opustil relaci (viz ukázka 4.10). Je ale k dispozici také možnost poslat tuto informaci druhému uživateli manuálně (například po kliknutí na tlačítko).

```
<iq from='incik@jabbim.cz/jabbim' to='johntestovic@jabbim.cz/jabbim'
    type='set'>
    <query xmlns='http://xawa.vaisar.cz'>
        <xawaData>
            <data>{'player':[{'playerid':'player2'}]}</data>
        </xawaData>
    </query>
</iq>
```

Obrázek 4.9: Zpráva typu `IQ` obsahující data ve formátu JSON

```
<iq from='incik@jabbim.cz/jabbim' to='johntestovic@jabbim.cz/jabbim'
    type='set'>
    <query xmlns='http://xawa.vaisar.cz'>
        <session appName='Pokusná aplikace'
            appUrl='http://xawa.vaisar.cz/apps/pokusna'>
            <leave />
        </session>
    </query>
</iq>
```

Obrázek 4.10: Zpráva oznamující ukončení relace

Kapitola 5

Implementace

V této kapitole je probírána výsledná implementace aplikačního rozhraní pro JavaScript. Dále se kapitola věnuje implementaci navrženého aplikačního programovacího rozhraní v zásuvném modulu pro komunikátor Jabbim. Protože byla během vývoje vytvořena i javascriptová knihovna zapouzdřující některé operace, bude bude se jí tato kapitola též věnovat.

Několik termínů, které budu dále v textu používat:

- **prohlížeč** – komponenta, která zajišťuje běh programované aplikace v rámci zásuvného modulu
- **programovaná/cílová aplikace** – webová aplikace využívající XAWA běžící v rámci implementovaného zásuvného modulu

Implementace prošla několika fázemi. V první fázi se počítalo s přístupem, kdy programovaná aplikace v pravidelných intervalech kontrolovala, zda se nezměnila hodnota proměnné, do které zásuvný modul ukládal přijatou zprávu či data. Tento způsob označuji dále v textu pojemem *staré API* nebo *starý způsob*. S použitím tohoto způsobu bylo napsáno několik prvních testovacích aplikací. Ukázalo se však, že neustálá periodická kontrola stavu je nepraktická z hlediska nepřehlednosti zdrojového kódu. Protože aplikace periodicky prováděla volání kontrolní funkce, nebyl uspokojivý ani její výkon. Pokud by navíc taková aplikace měla být provozována například na mobilním telefonu, tak by pravděpodobně díky neustálé aktivitě procesoru brzy vyčerpala baterii zařízení.

Tento způsob implementace byl tedy nahrazen jiným přístupem[1]. Při přijetí zprávy či dat zásuvným modulem vyvolána událost, na kterou lze navázat vlastní funkcionalitu – tedy vytvořit obsluhu této události[2].

5.1 Události

Programátor vytvářející v JavaScriptu aplikaci využívající XAWA dostává k dispozici následující sadu událostí, na které může navázat vlastní funkcionalitu:

onApplicationReady – je vyvolána jakmile je aplikace kompletně načtena (včetně externích javascriptových skriptů a obrázků). Lze ji využít například k zaslaní informace o připravenosti aplikace druhé straně (pro bezproblémové zahájení komunikace).

[1] Nicméně z důvodu zpětné kompatibility s dříve napsanými aplikacemi je původní způsob stále implementován. Jeho používání však není doporučeno.

[2] Označení „událost“ je v tomto kontextu poněkud nepřesné. Ve skutečnosti totiž dochází k volání funkce, jejíž chování programátor nahradí vlastní funkcí.

onInvitationAccept – je vyvolána u uživatele A, pokud uživatel A odešle pozvánku do aplikace uživateli B a uživatel B ji *přijme*

onInvitationRefuse – je vyvolána u uživatele A, pokud uživatel A odešle pozvánku do aplikace uživateli B a uživatel B ji *odmítne*

onMessageReceived – je vyvolána při přijetí textové zprávy, jako parametr jí je předán obsah přijaté zprávy

onDataReceived – je vyvolána při přijetí datové zprávy, jako parametr jí jsou předána přijatá data

onSessionLeave – je vyvolána u uživatele A, pokud uživatel B ukončí relaci

Pokud tedy dojde například k přijetí textové zprávy určené pro námi programovanou aplikace, dojde k vyvolání události `onMessageReceived` (resp. se začne provádět funkce téhož jména), které je jako parametr předán obsah přijaté zprávy. Tuto funkci je třeba v aplikaci definovat, jinak dojde k vyvolání výjimky. K validnímu chování stačí například následující zápis v ukázce 5.1.

```
function onMessageReceived(message) {
    alert(message);
}
```

Obrázek 5.1: Definice chování aplikace při události `onMessageReceived`

5.2 Metody

Hlavní součástí API jsou ale metody třídy `xawa`, které v JavaScriptu zprostředkovávají XMPP funkcionalitu. Všechny níže uvedené metody lze využít v programované aplikaci javascriptovým voláním `xawa.názevMetody()`.

init – tato funkce nemá žádnou specifickou funkcionalitu, je k dispozici pro testování, zda je k dispozici prostředí podporující XAWA

getXawaVersion – vrací textový řetězec, který obsahuje číslo verze API, které zásuvný modul implementuje

getRecipient – vrací textový řetezec, který obsahuje JID příjemce zpráv (tj. uživatele, se kterým komunikujeme)

getSender – vrací textový řetězec, který obsahuje JID odesílatele zpráv (tj. naše)

sendInvite – odešle pozvánku do aplikace. Tato metoda přijímá dva argumenty

- JID uživatele, kterému se má pozvánka odeslat
- konfigurační objekt aplikace, obsahující informace o aplikaci (viz oddíl 5.3)

sendMessage – odešle zprávu uživateli, se kterým je navázána komunikace. Přijímá jeden parametr, a sice text zprávy, která se má odeslat

sendClassicMessage – podobně jako sendMessage, ale zpráva je odeslána jako klasická XMPP `message`

sendData – odešle druhému uživateli JSON objekt, který je funkci předán jako parametr

leave – odešle druhému uživateli informaci o ukončení relace

Následující funkce jsou k dispozici z důvodu zpětné kompatibility se starší verzí API a jejich používání se již příliš nedoporučuje.

sendDataInLegacyMode – odešle druhému uživateli JSON objekt (který přijímá jako parametr) ve formátu klasické XMPP message – viz oddíl 4.5.4

isMessageUnread – vrací logickou hodnotu (*true*/*false*), zda již byla poslední přijatá textová zpráva aplikací přečtena

isDataUnread – vrací logickou hodnotu (*true*/*false*), zda již byla aplikací přečtena poslední přijatá data

getMessage – vrací obsah poslední přijaté zprávy

getData – vrací poslední přijatá data

markMessageAsRead – označí přijatou textou zprávu za přečtenou

markDataAsRead – označí přijatá data za přečtená

Pozn.: Při implementaci tohoto API pro jiné komunikátory je z důvodu kompatibility třeba dodržet stejné pojmenování metod, jejich rozhraní (vstupní parametry) a chování – tedy zajistit stejné reakce stejné vstupy (parametry) a výstupy (reakce na zadané parametry).

5.3 Konfigurační objekt

V rámci programované aplikace je nutno definovat objekt, který obsahuje její konfiguraci (viz ukázka 5.2). Jedná se zejména o název aplikace, URL adresu, na které se daná aplikace nachází, rozměry okna, v němž je aplikace spuštěna apod. Tento objekt je poté v rámci pozvánky do aplikace zaslán druhému uživateli. Na jeho straně jsou tyto informace využity k tomu, aby se jeho zásuvný modul správně nastavil (např. aby uživatel viděl skutečně celou hrací plochu).

Povinnými položkami, které tento objekt musí obsahovat jsou:

- **appName** – určuje jméno aplikace
- **appUrl** – určuje URL, odkud má být aplikace po přijetí pozvánky načtena a spuštěna
- **__window** – zapouzdřuje nastavení rozměrů okna, musí obsahovat další dva parametry, a sice:
  - *width* – určuje šířku okna aplikace (v pixelech)
  - *height* – určuje výšku okna aplikace (v pixelech)

Konfigurační objekt lze doplnit o libovolné další parametry a s těmi poté v aplikaci pracovat.

```
var XAWA_APP_CONFIG = {
    appName : 'Draughts - Simple board game',
    appUrl : 'http://localhost/xawa/draughts/'',
    __window : {
        width : 900,
        height : 760
    }
};
```

Obrázek 5.2: Příklad konfiguračního objektu

5.4 Zásuvný modul pro Jabbim

Jabbim je český open-source XMPP komunikátor napsaný ve skriptovacím jazyce Python s využitím frameworku[3] PyQt (Qt[4] verzi pro programovací jazyk Python). Díky tomu je Jabbim možné spustit na běžných operačních systémech pro osobní počítače – během vývoje bylo Jabbim se zásuvným modulem testováno na MS Windows (XP/7), GNU/Linux (Ubuntu 10.10 x64 a Kubuntu 11.04 x86) a na Mac OS X 10.6.7.

5.4.1 Implementace zásuvného modulu

V rámci vývoje API bylo nutno zavést vlastní rozšíření protokolu XMPP. Toto rozšíření je identifikováno použitím jmenného prostoru `http://xawa.vaisar.cz`. Všechny zprávy, které jsou identifikovány tímto jmenným prostorem jsou poté zpracovány implementovaným zásuvným modulem.

Dle protokolu XMPP posílá komunikátor během procesu přihlášování serveru seznam tzv. *capabilities*, tedy seznam toho, s čím umí komunikátor v rámci protokolu XMPP pracovat[1]. Do tohoto seznamu je třeba přidat vytvořené rozšíření. Díky tomu je pak možné detekovat v seznamu kontaktů uživatele, kteří používají zásuvný modul implementující XAWA.

Jen u těchto uživatelů umožníme komunikaci využívající toto API. U uživatelů, jejichž komunikátor XAWA podporuje je totiž kontextové menu[5] rozšířeno o položku „Open XAWA window“ (viz obrázek A.2). Po kliknutí na tuto položku je otevřeno okno zásuvného modulu a je nastaven *kontext*, v němž bude probíhat následující komunikace – kontakt, na který bylo kliknuto je pro tuto relaci nastaven jako *příjemce*.

Samotný zásuvný modul je tvořen dvěma třídami – **Plugin** a **xawa**. Třída `Plugin` je potomkem třídy `PluginBase`, která v Jabbim slouží jako základ pro tvorbu zásuvných modulů a předepisuje některé metody, které je třeba implementovat, aby mohl být zásuvný modul správně zaveden do programu a bylo možno s ním pracovat. Třída `Plugin` zajišťuje veškerou komunikaci s jádrem komunikátoru a umožňuje využívat funkcionalitu, kterou disponuje samotný komunikátor. Díky tomu může přijímat a odesílat zprávy, přistupovat k informacím o uživatelích v seznamu kontaktů, apod. Tato třída se také stará o zobrazení okna zásuvného modulu, které obsahuje prohlížeč. Ten se stará o samotné zobrazení a běh cílových aplikací.

Druhou třídou, která tvoří zásuvný modul, je třída `xawa`. Ta je potomkem třídy `QObject`

[3]Framework neboli „vývojová platforma“ je sada knihoven a nástrojů, které ulehčují vývoj software.
[4]Qt – http://qt.nokia.com/products/
[5]nabídka zobrazující se po kliknutí pravým tlačítkem na kontakt v seznamu kontaktů

a tvoří klíčového prostředníka mezi JavaScriptem a XMPP klientem. Její instance je pomocí metody `addToJavaScriptWindowObject` přidána do objektu `window` cílové aplikace (viz oddíl 3.3.1). Jak již bylo zmíněno v návrhu (oddíl 4.2), především si však udržuje referenci na instanci třídy `Plugin`, která může pracovat se zbytkem programu Jabbim. Díky tomu je možné v JavaScriptu přístupnit již zmíněnou existující funkcionalitu.

Aby byl zásuvný modul schopen zpracovávat zprávy, které jsou pro něj určené, je nutno se přihlásit k jejich odběru. To provedeno pomocí metody `addObserver` (kterou poskytuje knihovna `twisted`[6]), výrazu zapsaného v jazyce XPath[7] a reference na *obslužnou metodu*. Tato metoda je při přijetí určeného typu zprávy vyvolána a jako paremetr jí je předána zachycená XML zpráva.

```
x = '/iq[@type='set']/query[@xmlns='http://xawa.vaisar.cz']/session/invite'
self.main.client.xmlstream.addObserver(x, self.onInvite, priority = 1)
```

Obrázek 5.3: Zachycení zprávy obsahující pozvání do aplikace (viz oddíl 4.5.2)

Pokud komunikátor obdrží zprávu uvednou v ukázce 5.3, zavolá metodu `self.onInvite` a jako argument jí předá přijatou XML zprávu. Z této zprávy jsou načteny informace o odesílateli a cílové aplikaci. Poté je vyvoláno dialogové okno, pomocí kterého je příjemce dotázán, zda chce pozvání přijmout, či odmítnout. V závislosti na jeho rozhodnutí je poté původnímu odesílateli odeslána odpověď (viz oddíl 4.5.2).

Pokud komunikátor zachytí zprávu, která má v prohlížeči vyvolat událost, je tato vyvolána pomocí metody `evaluateJavaScript`. Tato metoda je součástí WebKitu a umožňuje v prohlížeči vykonat javascriptový kód, který jí je předán jako parametr.

```
mainFrame.evaluateJavaScript('onInvitationAccept(); null')
```

Obrázek 5.4: Ukázka implementace vyvolání události `onInvitationAccept`

Příkaz `null`, kterým končí vykonávaný JavaScript, je v ukázce 5.4 z důvodu výkonu. Metoda `evaluateJavaScript` totiž jako svou návratovou hodnotu vrací výsledek posledního vykonaného příkazu. To může v některých případech trvat nepříjemně dlouho. Použitím `null` jako posledního příkazu je dosaženo toho, že je hodnota navrácena okamžitě.

Pokud není pro takto vyvolanou událost v cílové aplikaci vytvořen posluchač (metoda, která definuje chování při této události), je vyvolána výjimka `ReferenceError`.

5.5 XAWAlib

Při vytváření webových aplikací nad XMPP je nutno provádět stále stejné úkony, např. zaslat pozvánku a zareagovat na její přijetí/odmítnutí. Rozhodl jsem se tedy, že pro tyto účely vytvořím jednoduchou javascriptovou knihovnu, která budete tyto běžná operace zastřešovat a zajišťovat tak vyšší úroveň abstrakce. Díky tomu se bude moci programátor více soustředit na vývoj samotné aplikace a nebude nucen řešit „nízkoúrovňové“ záležitosti typu incializace prostředí nebo vytvoření posluchače události, kterou vyvolává příchozí zpráva.

Kromě tohoto zapouzdření umožňuje XAWAlib také zápis formou, která lépe koresponduje se stylem zápisu kódu využívajícího jQuery. To vede nejen k lepší vizuální kompakt-

[6] Ta v Jabbim tvoří jádro XMPP komunikace
[7] XPath – jazyk určený pro adresování částí XML dokumentu

nosti kódu, ale zároveň programátor zvyklý s jQuery pracovat bude rychle schopen používat i XAWAlib.

Hlavní část knihovny je tvořena jedinou sebe sama volající anonymní funkcí. Po vykonání této funkce je původní objekt `window.xawa` nahrazen novým a nelze tedy využívat původní volání. Tento přístup byl zvolen zejména kvůli tomu, aby stejně pojmenované metody mohly být rozšířeny o nové parametry. Při použití knihovny se navíc znepřístupní metody, které se využívaly ve starší verzi API a jejich používání již není doporučeno (tedy např. `xawa.isDataUnread`).

5.5.1 Funkcionalita

Následuje popis jednotlivých metod, jejichž používání XAWAlib umožňuje. Povinné parametry metod jsou označeny **tučně**, nepovinné *kurzívou*.

_init – Tato metoda za programátora testuje, zda je k dispozici prostředí XAWA. Je automaticky volána ihned po inicializaci knihovny a není ji tedy nutno volat explicitně. Pokud vykonávání této metody selže, je vyvolána výjimka a ukončí se další běh programu – pokud není k dispozici XAWA, nemá smysl pokračovat. Toto chování bylo zavedeno, aby nebylo možné aplikace spouštět z běžného internetového prohlížeče.

sendMessage – Tato metoda slouží k zasílání textových zpráv. Zpráva je zaslána *příjemci* (uživateli, který je pro aktuální relaci nastaven jako příjemce zpráv).
Parametry :

- **message** – textový řetězec reprezentující odesílanou zprávu
- *options* – konfigurační objekt
  - sendClassicMessage – pokud má tento atribut hodnotu `true`, jsou zprávy odesílány jako klasické XMPP zprávy typu `message` označené předmětem `xawa_message`

Ukázka použití: viz ukázka 5.5

```
xawa.sendMessage('ahoj!', { sendClassicMessage : true } );
```

Obrázek 5.5: Ukázka odeslání zprávy typu `message`

sendData – Tato metoda je určena pro posílání dat ve formě JSON objektů. Obdobně jako u `sendMessage` je zpráva odeslána *příjemci*.
Parametry :

- **json_data** – odesílaný JSON objekt
- *options* – konfigurační objekt
  - sendDataInLegacyMode – pokud má tento atribut hodnotu `true`, jsou data odesílána „starou metodou“ – tedy jako klasické XMPP zprávy označené předmětem `xawa_data`

invite – Tato metoda slouží k zasílání pozvánek a definici chování při jejich přijetí/odmítnutí.
Parametry :

- **jid** – JID uživatele, kterému má být pozvánka odeslána

- **appConfig** – konfigurační objekt aplikace
- *callbacks* – konfigurační objekt definující funkce, které mají být vyvolány na základě reakce uživatele, jemuž je pozvánka zaslána
  - onAccept – určuje funkci vyvolanou při přijetí pozvánky
  - onRefuse – určuje funkci vyvolanou při odmítnutí pozvánky

Ukázka použití: viz ukázka 5.6

```
xawa.invite(xawa.recipient, XAWA_APP_CONFIG, {
    onAccept: function() { alert('Let the game begin!'); },
    onRefuse: function() {
        alert(xawa.recipient + 'doesn't want to play.');
    }
});
```

Obrázek 5.6: Příklad použití metody `xawa.invite`

leave – Metoda, která odešle *příjemci* informaci o ukončení relace

recipient – Tento atribut obsahuje JID aktuálního *příjemce*

sender – Tento atribut obsahuje JID *odesílatele*

version – Tento atribut obsahuje číslo aktuální verze implementovaného API

5.5.2 Zachycování událostí

Knihovna též rozšiřuje aplikace o základní zachycování událostí, které prohlížeč vyvolává na základě příchozích dat a zpráv. Pokud programátor nenadefinuje vlastní chování při zachycení těchto událostí, zobrazí se přehledné varování obsahující informaci o tom, která událost byla zachycena a jaká data byla obdržena. Pokud chce programátor na tyto události reagovat, je třeba toto výchozí chování překrýt vlastním. K tomu postačuje jednoduchý zápis ve formě uvedené v ukázce 5.7.

```
onMessageReceived = function(message) {
    alert(message);
};
```

Obrázek 5.7: Příklad registrace posluchače události `onMessageReceived`

5.5.3 Použití v aplikacích

Knihovnu lze do aplikace připojit jako běžný externí javascriptový soubor (viz ukázka 5.8). Díky tomu, že je implementována jako sebevolající se funkce, je k dispozici ihned po načtení souboru (viz oddíl 3.2.1).

```
<script src='xawalib.js' type='text/javascript'></script>
```

Obrázek 5.8: Připojení knihovny do aplikace

Kapitola 6

Testovací aplikace

V této části se budu věnovat aplikacím, které byly během návrhu API a implementace zásuvného modulu pro Jabbim vytvořeny. Jedná se dvě jednoduché aplikace, na kterých byla testována správná funkčnost navrženého systému. Především se ale budu věnovat plhohodnotné aplikaci, která byla nad API vytvořena, a sice implementaci deskové hry Dáma.

6.1 Jednoduché testovací aplikace

Tyto aplikace byly vytvořeny nad prvotní verzí API. Využívaly tedy pro zasílání zpráv klasických XMPP zpráv typu `message` a aktivně se dotazovaly na příchozí data. Během vývoje se ukázalo, že tento přístup není vhodný ani pro takto jednoduché aplikace. To vedlo k přehodnocení implementace a vytvoření současné verze, která pracuje s událostmi.

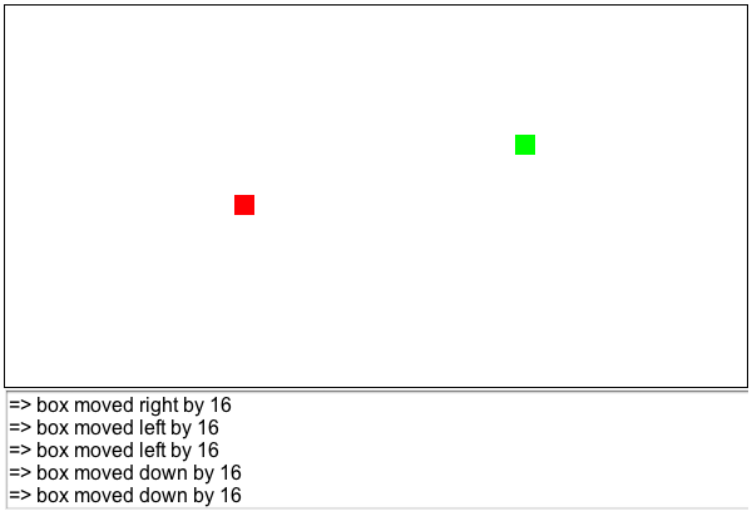
Jednoduchý chat

Tato aplikace testovala zasílání textových zpráv. Po zadání textu do textového políčka a kliknutí na tlačítko je zpráva odeslána a zobrazena v *okně konverzace*. Příchozí zprávy určené pro aplikaci využívající XAWA lze též vidět v *okně konverzace* (viz obrázek 6.1).

Obrázek 6.1: Jednoduchý chat

Jednoduchá aplikace posílající data v JSONu

Tato aplikace funguje jako jednoduchá hra pro dva hráče. Dva hráči mají k dispozici barevné čtverce, každý jeden. Pomocí kurzorových šipek ovládájí svůj čtverec a snaží se dohnat jeden druhého (viz obrázek 6.2).



Obrázek 6.2: Jednoduchá hra využívající zasílání dat ve formátu JSON

6.2 Dáma

Dáma je jednoduchá desková hra pro dva hráče. Hraje se na čtvercové desce o rozměru 8x8 polí (střídají se světlá a tmavá pole). Hráči posouvají střídavě bílé a černé kameny po diagonálách tvořených černými poli a to pouze vpřed a jen o jedno pole.

Pokud se některý z kamenů dostane na některé z tmavých polí v poslední řadě u soupeře, mění se v dámu. Dáma se může pohybovat o libovolný počet polí vpřed i vzad, ale opět pouze po diagonálách.

Pro detailnější pravidla viz Pravidla české dámy České federace dámy[5].

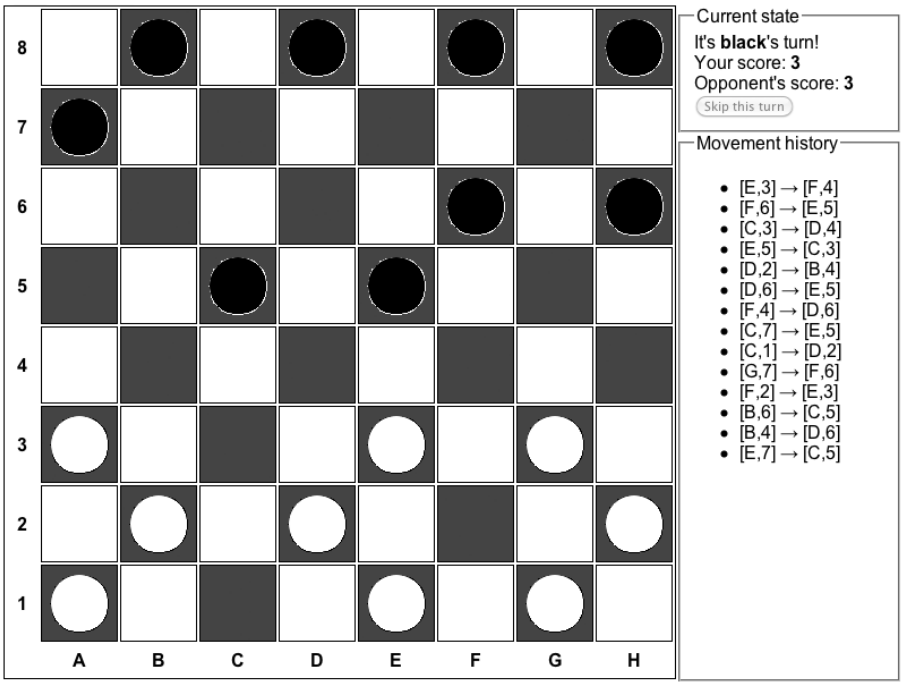
6.2.1 Průběh hry

V následujícím textu budou používány termíny *odesílatel* či *uživatel A* – uživatel, který inicioval hru, a *příjemce* či *uživatel B* – uživatel, který byl do hry přizván.

Při spuštění hry, jejíž rozhraní je možno vidět na obrázku 6.3, je odesílateli zobrazena nabídka, ve které lze kliknout na jedinou položku – „Invite user“. Po kliknutí na toto tlačítko je odeslána pozvánka do aplikace příjemci, v jehož kontextu byla hra vytvořena (viz oddíl 5.4.1). Poté aplikace čeká na to, zda příjemce pozvání do hry přijme, či ne.

Pokud uživatel B přijme pozvání, nabídka u uživatele A se zavře a je zahájena hra. Začíná hráč s bílými kameny (tj. uživatel, který inicioval hru). Hráči se poté střídají v tazích kameny – každý tah je kontrolován, zda je v souladu s pravidly.

Pokud jeden z hráčů sebere soupeřův kámen, má automaticky možnost hrát dál – dle pravidel je možné sebrat jedním vlastním kamenem více kamenů soupeřových. Pokud však již nemá kam s kamenem táhnout, musí kliknout na tlačítko „skip this turn“



Obrázek 6.3: Prostředí hry Dáma

(„přeskočit/ukončit tento tah“), neboť ve hře není implementováno prohledávání stavového prostoru a rozhodování, zda je pro daný kámen k dispozici další validní tah. Zde tedy aplikace spoléhá na poctivost účastníků hry, kteří nevyužijí této vlastnosti hry k tahu jiným kamenem.

Hra je ukončena buď vítězstvím jednoho z hráčů (soupeř již nemá žádné kameny) nebo tím, že jeden z účastníků hry klikne na tlačítko „End game“. V takovém případě je druhému uživateli odesláno oznámení, že vyhrál, protože soupeř ukončil hru.

6.2.2 Implementace

Při tvorbě aplikace byl kladen důraz na maximální využití javascriptových knihoven jQuery a jQuery UI. Tyto knihovny umožňují kromě snadného přístupu k prvkům v DOM i snadné začlenění funcionality *drag&drop* (*táhni a pusť*), tedy možnost přesouvání prvků v zobrazované HTML stránce tažením myši.

V maximální míře též byla využita již zmíněná vlastní knihovna XAWAlib, která zapouzdřuje některé netriviální operace.

Hned po načtení aplikace se provede **registrace posluchačů událostí** – tedy definice funkcí, které budou obsluhovat prohlížečem vyvolávané události. Je využita událost `onApplicationReady`, po jejímž vyvolání odešle aplikace objekt s informací o tom, že je připravena ke hře.

Dalším definovaným posluchačem je funkce reagující na událost `onSessionLeave`, která je vyvolána, pokud jeden z uživatelů zavře okno aplikace a tím přeruší relaci. V takovém případě je uživateli, který v relaci zůstal oznámeno, že vyhrál.

Nejpodstatnějším posluchačem je funkce navázaná na událost `onDataReceived`. Instance aplikace si totiž daty ve formě JSON objektů vyměnují příkazy, které mění jejich

stav (pokyn k zahájení hry, jednotlivé tahy hráčů, apod.). Tato funkce přijatá data analyzuje a podle jejich obsahu provede potřebný úkon (např. přesune odesílatelův kámen v okně příjemce).

V rámci této výměny dat probíhá ihned po spuštění hry výměna informací o tom, kdo bude ve hře označován jako *hráč 1* a kdo *hráč 2* (*hráč 1* je uživatel, který inicioval hru).

Po dokončení této výměny je zahájena hra. Po tahu odesílatele je příjemci pomocí `xawa.sendData()` odeslán objekt reprezentující provedený tah, následovaný objektem obsahující příkaz pro výměnu hráčů[1]. Na straně příjemce je poté tah proveden a příkaz pro výměnu hráčů vykonán. Nyní je na tahu příjemce.

Hráči se takto střídají dokud jeden z nich nevyhraje nebo jeden z nich neukončí relaci.

[1] Příkaz pro výměnu hráčů není zaslán pokud odesílatel sebral příjemcův kámen a může tedy hrát dál.

Kapitola 7

Závěr

V rámci práce bylo navrženo aplikační programovací rozhraní, které umožňuje prostřednictvím jazyků XHTML a JavaScript snadnou tvorbu aplikací využívajících ke komunikaci protokolu XMPP. Toto rozhraní bylo implementováno v zásuvném modulu pro internetový komunikátor Jabbim a demonstrováno na netriviální aplikaci – hře Dáma.

Požadavek na přenositelnost aplikací mezi různými IM klienty byl též splněn. Pokud bude zásuvný modul pro jiný komunikátor implementovat navržené chování a dodrží stejné rozhraní metod pro komunikaci, bude v něm možno tyto aplikace spouštět.

Pro další vývoj API je výhodou navrženého řešení možnost rozšířit či optimalizovat formát XMPP zpráv bez nutnosti zásahu do již naprogramovaných aplikací. Stačí upravit pouze zásuvný modul, který implementuje XAWA. Aplikací, které jej využívají se tato změna nijak nedotkne.

7.1 Možnosti rozšíření

Navržené API je velice snadno rozšiřitelné. Například díky nativní podpoře přehrávání videa a audia ve WebKitu pomocí technologií obsažených HTML5, by bylo možné v kombinaci s Jingle[1] tvořit aplikace, které by dokázaly pracovat s obrazem a zvukem.

7.1.1 Rozšíření modulu pro Jabbim

V současné době může být v Jabbim spuštěna pouze jedna instance zásuvného modulu implementujícího XAWA. Všechny přijaté zprávy jsou směrovány právě do této jedné instance. Lze tedy využít možnosti hrát najednou s více hráči jednu hru (např. Člověče, nezlob se), nicméně nelze najednou hrát dvě partie Dámy se dvěma různými uživateli. V plánovaných rozšířeních pro tento modul (a pro rozšíření celého API obecně) je tedy zavedení možnosti spuštění více oddělených instancí pro různé uživatele.

[1] Jingle – rozšíření protokolu XMPP o možnost přímého propojení mezi uživateli (bez účasti serveru), kterého lze využít k přenosu obrazu a zvuku

Literatura

[1] XEP-0115: Entity Capabilities. Technická zpráva, XMPP Standards Foundation, http://xmpp.org/extensions/xep-0115.html, 2008-02-26 [cit. 2011-03-07].

[2] Number - MDC Docs. Technická zpráva, Mozilla Developer Network, https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Number, 2010-09-26 [cit. 2011-05-01].

[3] Přehrávač videí YouTube HTML5 [online]. Technická zpráva, Google, Inc., http://www.youtube.com/html5, [cit. 2011-05-05].

[4] Statistics [online]. Technická zpráva, Facebook, http://www.facebook.com/press/info.php?statistics, [cit. 2011-05-05].

[5] Pravidla české dámy České federace dámy. Technická zpráva, Česká federace dámy, http://www.damweb.cz/pravidla/cdfull.html , [cit. 2011-05-11].

[6] Crockford, D.: JSON: The Fat-Free Alternative to XML [online]. http://www.json.org/fatfree.html, 2006-12-06 [cit. 2011-04-21].

[7] Crockford, D.: Introducing JSON [online]. http://www.json.org, [cit. 2011-05-05].

[8] Krill, P.: JavaScript creator ponders past, future [online]. http://www.infoworld.com/d/developer-world/javascript-creator-ponders-past-future-704, 2008-06-23 [cit. 2011-04-14].

[9] Molkentin, D.: Apple Opens WebKit CVS and Bug Database [online]. http://dot.kde.org/2005/06/07/apple-opens-webkit-cvs-and-bug-database, 2005-07-05 [cit. 2011-04-13].

[10] Reiss, D.: Facebook Chat Now Available Everywhere [online]. http://www.facebook.com/blog.php?post=297991732130, 2010-10-02 [cit. 2011-04-12].

Dodatek A

Používání modulu pro Jabbim

A.1 Instalace

A.1.1 Použití instalačního skriptu

Pro uživatele linuxových distribucí vycházejících z distribuce Ubuntu je k dispozici instalační skript `install_jabbim.sh`. Ten provede následující kroky:

1. Z repozitářů Ubuntu nainstaluje kompletní prostředí pro běh komunikátoru Jabbim (tedy všech potřebných knihoven)
2. Prostřednictvím `svn`[1] stáhne nejnovější verzi komunikátoru
3. Z webu Github[2] stáhne nejnovější verzi XAWA
4. Nakopíruje zásuvný modul do složky s komunikátorem

Nyní je Jabbim s podporou XAWA připraveno ke spuštění pomocí příkazů uvedených v ukázce A.1.

```
$ cd jabbim
$ ./jabbim.sh
```

Obrázek A.1: Posloupnost příkazů pro spuštění programu Jabbim

Testováno na Ubuntu 10.10 x64 a Kubuntu 11.04 x86.

A.1.2 Ruční instalace

V případě potřeby ruční instalace je potřeba provést následující kroky:

1. Stáhnout a nainstalovat Jabbim[3].
2. Stáhnout z `https://github.com/incik/XAWA/zipball/master/` nejnovější verzi XAWA.
3. Stažený archív rozbalit a obsah složky `plugins/jabbim` zkopírovat do složky `cesta-k-jabbbim/plugins`

[1] SVN neboli *Subversion* je systém pro správu a verzování software, velmi vhodný při týmovém vývoji
[2] Github - `http://github.com`
[3] Intrukce pro instalaci jsou k dispozici na `http://dev.jabbim.cz/jabbim/`

A.1.3 Aktivace modulu

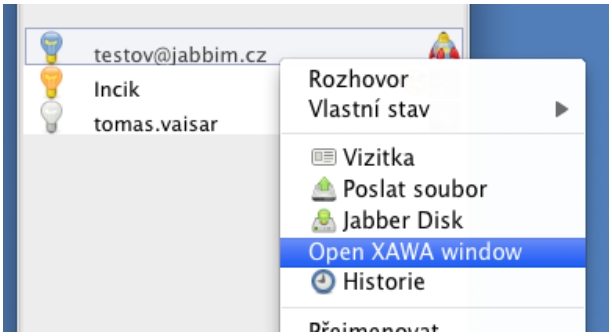
Jak po instalaci pomocí skriptu, tak po ruční instalaci modulu do adresáře `plugins` je třeba modul ještě aktivovat v nastavení programu. Je třeba provést následující posloupnost kroků:

1. V hlavním menu programu oveřit nabídku **Akce** a vybrat položku **Nastavení**.
2. V levé nabídce Vybrat kategorii **Rozšíření**.
3. V pravé nabídce v kategorii *Různé* zaškrtnout **XAWA Plugin**.
4. Kliknout na tlačítko **Uložit**

Nyní je třeba se odhlásit a znovu přihlásit. XAWA plugin bude načten a bude připraven k použití.

A.2 Příklad používání modulu

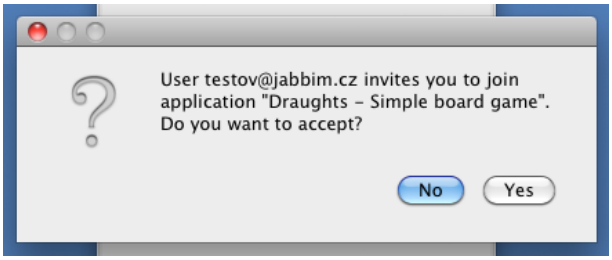
Po kliknutí pravým tlačítkem na uživatele, který má též nainstalovaný XAWA plugin se v kontextovém menu objeví položka „Open XAWA window“ (viz obrázek A.2).



Obrázek A.2: Kontextové menu pro uživatele, který má nainstalovaný XAWA plugin

Po kliknutí na tuto položku je otevřeno hlavní okno pluginu, prozatím prázdné. Po kliknutí na tlačítko **Refresh** je do něj načten *rozcestník* umožňující výběr aplikace.

Je-li uživatel pozván k používání nějaké aplikace, je mu zobrazeno dialogové okno s dotazem, zda chce pozvání příjmout (viz obrázek A.3). Pokud klikne na „Ano“, otevře se okno s aplikací, do které je zván.



Obrázek A.3: Dialogové okno vyvolané při doručení pozvánky do aplikace

Dodatek B

Obsah CD

B.1 Adresářová struktura

Stromové zobrazení adresářové struktury na přiloženém CD.

- **apps** – složka obsahující hru Dáma a všechny vytvořené testovací aplikace
- **docs** – složka obsahující technickou zprávu ve formátu PDF a její zdrojové soubory pro LaTeX
- **plugins** – složka obsahující zásuvný modul pro Jabbim
- README – soubor s instrukcemi pro instalaci zásuvného modulu