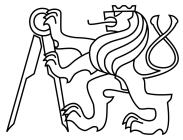
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Propojení chytré televize s chytrým telefonem
Student:	Lukáš Hromadník
Vedoucí:	Ing. Dominik Veselý
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Prozkoumejte možnosti spojení chytré televize AppleTV s mobilní aplikací na platformě iOS / Android. Aplikace na televizi by měla představovat second screen experience, tedy měla by reagovat na dění v mobilní aplikaci, přes kterou budete celé spojení ovládat. Zaměřte se na všechny možnosti tohoto připojení, včetně situací kdy k televizi připojíme 1 nebo více mobilních aplikací. Poznatky využijte k implementaci nejvhodnější technologie do prototypu aplikací pro iOS / Android a AppleTV, na kterých možnosti propojení předvedete. Technologie a funkčnost aplikací konzultujte s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 1. února 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Propojení chytré televize s chytrým telefonem

Lukáš Hromadník

Vedoucí práce: Ing. Dominik Veselý

17. května 2016

Poděkování

Rád bych na tomto místě poděkoval svému vedoucímu Ing. Dominiku Veselému za jeho odborné konzultace a věcné připomínky při tvorbě této práce. Dále bych rád poděkoval své rodině a přítelkyni za podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 17. května 2016 .....

Odkaz na tuto práci

Hromadník, Lukáš. *Propojení chytré televize s chytrým telefonem.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Cílem práce je prozkoumání možností propojení chytré televize Apple TV a chytrého telefonu. Dalším cílem aplikace je implementace prototypu, který bude jednu vybranou technologii z průzkumu implementovat. Prototyp aplikace využívá nízkoúrovňové knihovny pro práci se sítí, pomocí které vytváří spojení a komunikuje s ostatními zařízeními. Výsledek práce slouží jako výchozí bod vývoje budoucích aplikací využívajících propojení chytrého telefonu a Apple TV. V příloze lze najít datové médium se zdrojovými kódy.

Klíčová slova propojení zařízení, Apple TV, chytrý telefon, CocoaAsyncSocket, Bonjour

Abstract

The goal of the thesis is to analyze options how to connect Apple TV with a smartphone. Another goal is to implement a prototype of an application in which one chosen technology from analysis will be used. Application prototype is using low-level networking library. This library is used for communication with other devices. The result of the thesis is used as a basis for future development for similar applications. As an attachment to this thesis is a storage medium which contains source codes.

Keywords device connection, Apple TV, smartphone, CocoaAsyncSocket, Bonjour

Obsah

Seznam obrázků

Úvod

Chytrá televize Apple TV se ve své poslední generaci dočkala vylepšení, které dovoluje vývojářům tvořit aplikace přímo pro toto zařízení. Jedná se však o novou platformu, jejíž možnosti zatím nejsou prozkoumány a koncovému uživateli tak nemusí být dodán ten správný zážitek z jejího užívání.

Zadáním práce je prozkoumat dostupné možnosti propojení chytré televize Apple TV s chytrým telefonem. Díky tomuto spojení může vzniknout prostředí, ve kterém spolu interaguje více zařízení současně a Apple TV tak neslouží pouze jako zobrazovací zařízení. Součástí zadání je i implementace prototypu aplikace, pomocí kterého bude vybraná vhodná technologie demonstrována.

Hlavní motivací autora pro realizaci této práce je seznámení se s technologiemi, které nejsou při tvorbě běžných aplikací možné použít z důvodu jejich malého využití.

Analytická část práce popisuje jednotlivé možnosti propojení zařízení od malých lokálních počítačových sítí až po možnosti propojení zařízení pomocí služeb připojených do sítě Internet, která se mohou nacházet stovky kilometrů daleko. Analytická část popisuje i možnost získávání informací o zařízeních a službách v okolí Apple TV pomocí protokolu Bonjour.

Návrhová část srovnává jednotlivé technologie z hlediska jejich možného využití a porovnává jejich klady a zápory. V návrhové části jsou také zobrazeny snímky z prototypu aplikace, která nejvhodnější vybranou technologii implementuje.

Implementační část se věnuje ukázce základních metod a postupů, jak vytvořit spojení mezi dvěma zařízeními, jak jednotlivá zařízení vyhledat a jak mezi nimi posílat data.

Součástí práce jsou ukázky kódu a snímky obrazovek z prototypu aplikace.

Kapitola 1

Cíl práce

Cílem práce je prozkoumání možností, jak propojit chytrou televizi Apple TV s chytrým telefonem, kde aplikace spuštěná na Apple TV bude zobrazovat informace a pomocí chytrého telefonu se bude s aplikací interagovat. Události z chytrého telefonu budou přenášeny do Apple TV, která je bude zobrazovat, popř. na ně reagovat.

Výsledek průzkumu bude realizován funkčním prototypem, který bude vycházet ze získaných informací. Prototyp bude rozdělen na dvě části. První část je klientská aplikace běžící na chytrém telefonu. Druhá část bude aplikace běžící na Apple TV, která bude zpracovávat příchozí informace. Realizovaný prototyp aplikace, který bude spuštěn na chytrém telefonu, bude implementovat obecné postupy tak, aby bylo možné tyto postupy použít i na ostatních mobilních operačních systémech.

Kapitola 2

Rešerše

2.1 Apple TV

Společnost Apple představila novou generaci přístroje Apple TV na konci září roku 2015 [6]. Kromě vylepšení přístroje z hlediska hardware připravila i zásadní novinku v podobě možnosti vytváření aplikací pro tuto platformu. Do této generace byla Apple TV pouze přístroj, který se připojil k zobrazovacímu zařízení (např. televizoru) a pomocí něj mohl uživatel prohlížet předem připravený obsah nebo mohl přehrávat svá média z domácí knihovny. S novou generací mohou vývojáři připravit své vlastní aplikace, které jsou nativně spuštěny na Apple TV [7].

Podíváme-li se na možnosti propojení nové Apple TV s ostatními produkty společnosti Apple, zjistíme, že k tomuto přístroji lze připojit pouze jeden ovladač, který je součástí balení, a další dva herní ovladače. K Apple TV můžeme připojit i zařízení s operačním systémem iOS. Ta však mohou fungovat pouze jako náhradní ovládání celé Apple TV. Nemohou být nativně využita jako další ovládací prvek v aplikacích, kde by např. jednotlivá zařízení mohla sloužit jako ovladače postav ve hře pro více hráčů [8].

V současné chvíli tedy neexistuje žádné funkční rozhraní připravené přímo od společnosti Apple, které by toto propojení umožňovalo.

2.2 Základní pojmy

Z důvodu rozsáhlejší rešeršní části, která se zabývá síťovými technologiemi, je potřeba na začátku definovat základní pojmy.

2.2.1 Program

Počítačový program je posloupnost instrukcí definujících chování procesu. Jeden počítačový program může být spuštěn jako více procesů, kde každý proces pracuje s jinými daty [9].

2.2.2 Proces

Proces je instance počítačového programu. Je spuštěn v operační paměti počítače, kde si alokuje prostředky, potřebné pro dokončení svého běhu [10].

2.2.3 Paket

Paket označuje blok dat přenášený v rámci počítačové sítě. Skládá se z řídích dat (metadat), které slouží počítačové síti k doručení paketu (např. adresa zdroje a cíle, kontrolní součet), a uživatelských dat [11]. Uživatelská data z paketů se v cíli skládají do větších celků, ze kterých se skládá původní zpráva. Jednotlivé pakety však mohou dorazit v různém pořadí, o správnou synchronizaci se starají dva protokoly v transportní vrstvě.

2.2.4 TCP protokol

TCP protokol se stará o to, aby původní odeslaná zpráva byla na obou koncích síťového spojení stejná. Jeho hlavním úkolem je doručit pakety ve správném pořadí a v případě výskytu chyby (např. chybějící paket) ji vyřešit [12]. Tento protokol se využívá u spojení, kde je důležitá korektnost získaných dat.

2.2.5 UDP protokol

Oproti TCP protokolu se UDP protokol nestará o správnou synchronizaci, ani o opravu chyb, které vzniknou během spojení. Jeho úkolem je dopravit pakety z jedné strany na druhou co nejrychleji. Používá se především tam, kde je důraz na rychlost a není důležité, zda jsou pakety ve správném pořadí. Hlavní využití je především v oblasti hlasových a obrazových komunikací přes síť v reálném čase (např. hovor pomocí programu Skype) nebo při živém vysílání.

2.2.6 Internet Protocol

Internet Protocol specifikuje formát, jak vypadají pakety, a jakým způsobem mezi sebou komunikují počítače v počítačové síti. Většina počítačových sítí kombinuje tento protokol s transportními protokoly TCP a UDP, díky čemu lze vytvořit spojení mezi počítači [13]. V současné době se používají dvě verze Internet Protocolu:

- **IPv4**: V současnosti nejrozšířenější standard. Adresa je v této verzi tvořena 32-bitovým číslem. S rozvojem Internetu však možný rozsah adresy v této verzi přestal stačit a nyní je plně vyčerpán.
- **IPv6**: Poslední revize Internet Protocolu. Byla vytvořena za účelem vylepšit a nahradit současnou verzi IPv4.

2.2.7 IP adresa

IP adresa je jednoznačné označení síťového rozhraní počítače v počítačové síti [14]. Tohoto označení se využívá při vytváření spojení mezi počítači a při transportu dat mezi nimi. IP adresa je například obsažena v metadatech paketu, kde určuje počáteční a koncový počítač. Díky této adrese je možné v jednotlivých síťových uzlech paket správně přepojovat.

2.2.8 Port

Port je číslo, které určuje, jaký proces bude odpovídat při dotazu na danou IP adresu při komunikaci pomocí TCP nebo UDP protokolu. Nachází se za IP adresou, od které je oddělen dvojtečkou. Port je v hlavičce paketu definován jako 16-bitové číslo, jedno síťové rozhraní tak může obsluhovat až 65 536 portů. Prvních 1 024 portů je rezervováno pro systémové protokoly [15].

2.2.9 Socket

Socket je jeden koncový bod obousměrného komunikačního spojení mezi dvěma programy, které běží v počítačové síti. Pro propojení dvou zařízení tedy potřebujeme dva sockety, každý pro jedno zařízení. Pomocí socketů se data buď posílají nebo přijímají. Každý socket má svoji adresu a ta se skládá z IP adresy a portu. Tato adresa jednoznačně daný socket identifikuje [16].

2.3 Komunikace mezi počítači

Při vytváření aplikace využívající komunikace mezi počítači, je zapotřebí nejdříve rozhodnout, v jakém postavení se vůči sobě budou jednotlivé počítače nacházet.

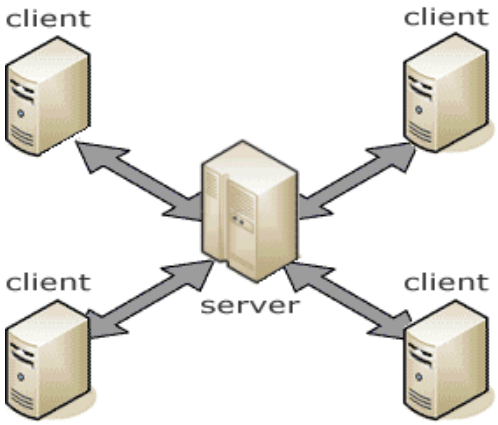
2.3.1 Klient-server

Klient-server je architektura počítačové sítě (obrázek 2.1), která dělí připojená zařízení v síti do dvou rolí:

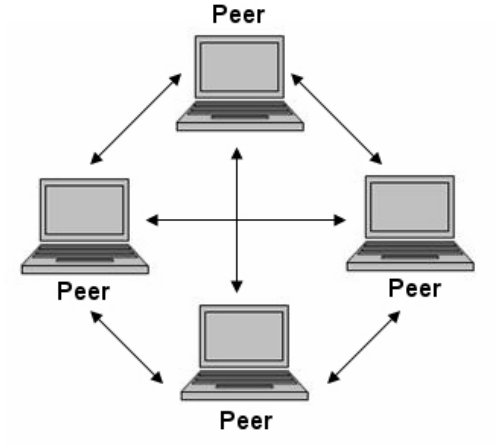
- **server** je zařízení, které s sdílí své prostředky s připojenými zařízeními,
- **klient** je zařízení, které se připojí k serveru za účelem využití jeho prostředků.

Každé zařízení může být server nebo klient anebo může mít obě role najednou.

Hezkým příkladem architektury klient-server jsou webové stránky. Chceme-li se z našeho počítače podívat na nějakou internetovou stránku, zadáme její adresu do prohlížeče. Tím se z nás stal klient, jelikož chceme po serveru, aby nám předal prostředky, pomocí kterých dokážeme zobrazit danou webovou stránku. Pokud dané stránka dovoluje nahrání souboru, ustanoví se role



Obrázek 2.1: Architektura Klient-server [1]



Obrázek 2.2: Architektura Peer-to-peer [2]

opačným způsobem. Webový prohlížeč, resp. počítač, na kterém běží, se stane serverem, který poskytuje data a webová stránka, resp. server, který ji spravuje, se stane klientem, protože daná data požaduje.

2.3.2 Peer-to-peer

V počítačové síti, která je postavena v architektuře Peer-to-peer (obrázek 2.2), spolu komunikují přímo připojená zařízení (klienti). Je tedy opakem architektury Klient-server, kde zařízení komunikují s centralizovaným serverem a všechna komunikace probíhá přes tento server. V architektuře Peer-to-peer se s pojmem server nesetkáme, avšak v reálném nasazení lze často narazit na server, který zajišťuje prvotní připojení nového zařízení do sítě nebo distribuci identifikátorů (adres) pro komunikaci mezi jednotlivými připojenými klienty.

Příkladem architektury Peer-to-peer je `BitTorrent`, což je komunikační protokol pro sdílení dat po počítačové síti. Tento protokol je velmi využíván pro distribuci většího množství dat. V architektuře Klient-server všichni klienti získávají data od centrálního serveru, naproti tomu v architektuře Peer-to-peer je zajištěna distribuce dat přímo mezi jednotlivými uživateli. Díky tomu není kladen velký nárok na síťové připojení jednotlivých uživatelů, protože data se mohou získat od jakéhokoliv z uživatelů, který je má k dispozici.

2.4 Berkley Sockets (BSD Sockets)

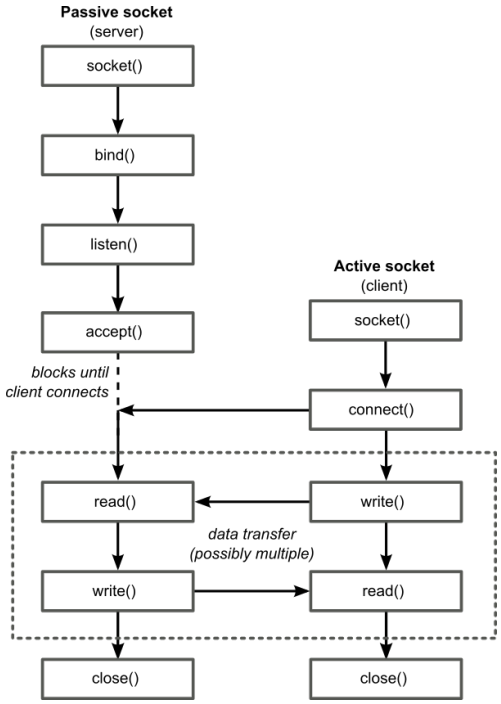
Operační systémy tvOS, iOS a Mac OS X, které společnost Apple dodává do svých zařízení, mají stejné jádro operačního systému. To je tvořeno systémem `Darwin`. `Darwin` je operační systém vycházející z operačního systému `BSD`. `BSD` je odvozen od operačního systému Unix, díky tomu systém `Darwin` a systémy jej využívající obsahují základní funkce pro správu (např. plánování procesů, správa paměti) a utility (např. shell, kompilátory), které jsou pro všechny tyto systémy společné [17]. Díky sdílení jádra napříč operačními systémy lze využít nízkoúrovněné síťové knihovny navržené pro operační systémy založené na Unixu.

BSD Sockets je knihovna, která poskytuje aplikační rozhraní, tzv. API, s jehož pomocí lze programovat aplikace, využívající meziprocesovou komunikaci. Rozhraní poskytované knihovnou je dostatečně obecné, aby bylo možné použít libovolný komunikační protokol pro danou aplikace [18] [19]. Nejrozšířenějším komunikačním protokolem, který se používá při práci s BSD Sockets, je `Internet protocol`.

Při vytváření komunikačního spojení mezi dvěma počítači pomocí knihovny BSD Sockets se využívá architektury Klient-server. Aby mohl klient komunikovat se serverem, je potřeba, aby na serveru bylo vytvořen socket a tomu přiřadit určitou adresu a port. Po přiřazení adresy začne server na této adrese naslouchat a čekat až do doby, než se k němu připojí klient [20]. Na straně klienta je potřeba vytvořit socket a připojit se k serveru. Po úspěšném připojení lze posílat data mezi klientem a serverem (obrázek 2.3) [21].

2.5 Bonjour

Bonjour protokol je společností Apple vytvořená implementace `zeroconf` protokolů [22]. `Zeroconf` (Zero-configuration networking) je soubor technik, pomocí kterých lze automaticky nakonfigurovat IP síť bez nutnosti přidání a nastavování speciálních serverů (např. DHCP, DNS) [23]. Bonjour vytváří architekturu počítačové sítě, která zajišťuje vyhledávání (Device discovery) a publikování služeb, které jsou založeny na TCP/IP, což je soubor komunikačních protokolů [24]. Příkladem služby využívající Bonjour může být síťová



Obrázek 2.3: Vytvoření spojení pomocí knihovny BSD Sockets [3]

tiskárna, která se po připojení do počítačové sítě automaticky objeví uživatelům k užívání bez nutnosti cokoliv nastavovat.

2.5.1 Vyhledávání služeb a zařízení

Pro vyhledávání služeb v rámci počítačové sítě používá Bonjour multicast DNS (mDNS). mDNS slouží stejně jako DNS k převodu doménové jména na IP adresu a zpět. Díky tomu je pro běžného člověka daleko jednodušší pracovat s počítači v síti, protože si nemusí pamatovat adresy, ale stačí si zapamatovat jednoduchý název, který je potom pomocí DNS serveru přeložen na adresu. Samotný DNS systém je stromově orientovaná struktura doménových jmen. Na začátku je jeden kořen (root), jehož potomky jsou další domény. Jednotlivé řády domén jsou od sebe odděleny tečkou. Díky této jmenné konvenci lze domény do sebe zanořovat a tím vytvořit stromovou strukturu dat.

mDNS však nevyužívá DNS serverů pro zjištění IP adresy počítače, ale dotazuje se přímo počítačů v síti pomocí multicast paketu, který je odeslán na speciální IP adresu: 224.0.0.251, UDP port 5353. Odeslání paketu na mutlicastovou adresu znamená, že paket nebude odeslán jednomu konkrétnímu zařízení ale skupině zařízení. Pokud některý z příjemců zná překlad ze zadaného doménového jména na příslušnou IP adresu či naopak, odpoví zdrojovému počítači [23].

Pokud chceme použít Bonjour pouze v rámci lokální sítě, je potřeba nastavit hlavní doménu pro vyhledávání na `local.`. Při dotazu na překlad doménového jména se za toto jméno přidá doména `local.` (např. `steve.local.`) a pro překlad se použije pouze mDNS v rámci lokální sítě. Z principu funkce DNS je potřeba, aby jednotlivá doménová jména byla v rámci sítě unikátní. Pokud tedy nastane situace, že dvě zařízení v rámci síti mají stejné doménové jméno, Bonjour se sám postará o to, aby jedno ze zařízeních bylo přejmenováno.

2.5.2 Pojmenovávání služeb

Jelikož Bonjour dokáže vyhledávat zařízení a zároveň služby, vznikla zde potřeba oddělit názvy pro tyto dvě skupiny. Pro názvy služeb se tak vytvořila konvence s použitím podtržítek před slovy. Formát pro názvy služeb vypadá takto: `_ServiceType._TransportProtocolName.` [25].

- **ServiceType** určuje, o jakou službu se jedná. Může se jedna např. o `http`, `ftp` nebo `printer`. Pokud je potřeba, lze zvolit libovolný název.

- **TransportProtocolName** je označení transportního protokolu, který se stará o přenos dat mezi propojenými zařízeními. Standardně se používá `tcp` nebo `udp`.

Pokud by byl vytvořen HTTP server, jehož úkolem je přenášet data pomocí protokolu TCP, název služby tohoto serveru by byl `_http._tcp.`. Každý název služby je ještě doplněn o instanční název služby, což je libovolný řetězec, pomocí kterého se uživateli blíže specifikuje, o kterou službu se jedná.

Příklad nechť je počítač, na kterém je spuštěna sdílená hudební knihovna. Název služby pro tento počítač by byl například `_music._tcp.`. Pokud by však v síti bylo více počítačů se sdílenou hudební knihovnu, nelze je jednoznačně identifikovat. Lze proto přiřadit každé ze služeb instanční jméno. Instanční jméno může být název počítače, na kterém je služba spuštěna. Díky tomu uživatel může jasně rozpoznat, ke které službě právě přistupuje.

Implementace protokolu Bonjour neobsahuje funkce na propojení jednotlivých zařízení či vnitřní logiku daného zařízení. Poskytuje pouze rozhraní pro vyhledávání a publikování zařízení či služeb v rámci počítačové sítě.

2.6 Server

Pokud se pohled na problém změní z problému realizovaném v rámci lokální sítě na problém realizovaný pomocí počítačů připojených do sítě Internet, můžeme využít dalších možností, které nám tento pohled dovoluje.

Server je počítač připojený do sítě Internet, ke kterému máme přístup a který disponuje dostatečně velkou výpočetní silou pro zvládnutí velkého množství operací v krátkém časovém intervalu. Servery se ve velké části případů používají jako hlavní část aplikací, která disponuje daty, které chceme nějakým způsobem zpracovat, popř. je zobrazit. Díky tomu lze docílit centralizace dat a programátoři v rámci svých aplikací přistupují k datům na jedno místo.

Realizace propojení dvou zařízení pomocí serveru s sebou přináší i určitá úskalí a rizika. K vytvoření spojení mezi dvěma zařízeními pomocí serveru je potřeba vymyslet způsob, kterým informujeme jedno zařízení o tom druhém. Příkladem je webová aplikace zobrazující aktuální sportovní výsledky. Ve chvíli, kdy dojdě ke změně dat na serveru, aplikace musí na tuto změnu patřičným způsobem zareagovat. Ke změně však může dojít kdykoliv a webová aplikace nemůže vědět kdy.

2.6.1 Pravidelné dotazování

Zisk změněných či nových dat lze uskutečnit pomocí pravidelného dotazování, kdy se aplikace v pravidelných intervalech dotazuje serveru, zda-li došlo ke změně dat. Na každý tento požadavek server odpoví příslušnými daty nebo prázdnou odpovědí indikující, že ke změně nedošlo. Toto řešení ale není ideální. Zbytečné dotazování na server zvyšuje zatížení serveru, které může vézt až k zahlcení a možnému zhroucení [26].

2.6.2 HTTP Long Polling

HTTP Long Polling je technika, při které se klientská aplikace dotazuje serveru na nová data, přičemž ale si server požadavek „podrží“ až do chvíle, kdy se nová data objeví. Ve chvíli, kdy na požadavek odpoví a klientská aplikace změny zpracuje, opět se vytvoří požadavek na nová data a tento cyklus se opakuje. Díky tomu se eliminují zbytečné dotazy a sníží se vytížení serveru [27].

2.6.3 Perzistentní spojení

Perzistentní spojení mezi klientskou aplikací a serverem lze vytvořit stejně, jako při propojení dvou zařízení v rámci lokální sítě. Pro navázání komunikace se využívá socketů, pomocí kterých lze vytvořit obou směrné komunikační spojení. Tím lze umožnit serveru posílat data přímo do klientské aplikace bez toho, aniž by bylo nutné se z klientské aplikace vyslat požadavek na server.

2.6.4 Push notifikace

Perzistentního spojení se serverem využívají push notifikace. Push notifikace slouží k odeslání určitých dat ze serveru přímo do mobilního zařízení, které na tuto push notifikaci zareaguje. Každá push notifikace má předem definovanou

strukturu dat, díky které systém zareaguje na push notifikaci vždy stejně a jednotlivé modifikace chování lze změnit pouze pomocí změny hodnoty jednotlivých parametrů. Jelikož aplikací, které využívají push notifikace, může být v jednom zařízení více, je pro tuto komunikaci vytvořeno jedno perzistentní IP spojení se serverem, který spravuje push notifikace.

2.7 Webová real-time databáze

Webové real-time databáze je označení pro databáze, které spojují klasické databáze a systémy reálného času. Přívlastek webová odkazuje na použití především u webových aplikací. Jejich možnosti lze ale také přenést do mobilních aplikací. Jedna z předností webových real-timových databází je, že v současné době existuje velké množství těchto databází, které fungují jako služba. Tedy je nad nimi vytvořeno rozhraní pro komunikaci. Komunikace s databází probíhá často pomocí obousměrného spojené (podobné jako push notifikace), avšak spojení je vytvořeno jedno pro danou aplikaci. Díky obousměrnému spojení je možné jak do databáze zapisovat, tak z ní číst, přičemž změny v databázi jsou propogovány asynchronně. Aplikace tak nemusí vytvářet nové požadavky na databázi, ale pouze „poslouchá“ na určité adrese, až z databáze přijdou nová či změněná data.

Struktura ukládaných dat je často odlišná od klasických relačních databází. Lze se setkat se strukturou ve formátu JSON, která se používá u serverových aplikací komunikujících pomocí REST API. Odlišná struktura uložených dat umožňuje optimalizovat úložiště tak, aby jednotlivé činnosti při práci s uloženými daty byly co nejrychlejší.

2.8 Bluetooth

Bluetooth je technologie pro bezdrátový přenos dat na určitou vzdálenost. Tato technologie vznikla za účelem vytvořit propojení mezi dvěma a více zařízeními bez použití kabelu. O vývoj technologie se stará skupina Bluetooth Special Interest Group (SIG), která řídí vývoj technologie a současně dohlíží na dodržování standardů u výrobků, které Bluetooth používají.

2.8.1 Párování

Pro vytvoření spojení mezi dvěma zařízeními je potřeba dostat zařízení na dostatečnou vzdálenost a spárovat je navzájem. Párování zařízení probíhá od verze 2.1 zabezpečeně bez nutného zásahu od uživatele. Nejprve si zařízení vytvoří společný veřejný klíč, pomocí kterého se při spojení autentizují. Uživatel může jednotlivé klíče z důvodu bezpečnosti v nastavení zařízení smazat.

2.8.2 Spojení

Před započetím přenosu dat je zapotřebí zařízení spárovat a vytvořit pomocí nich síť - `piconet`. `Piconet` je malá ad-hoc počítačová síť, kde jednotlivá zařízení jsou propojena technologií Bluetooth. Přenos dat i komunikace mezi zařízeními probíhá pomocí komunikačního modelu Master / slave. V tomto modelu jedno zařízení (`master`) přebírá řízení komunikace nad jedním nebo více zařízeními (`slave`). Zařízení si mohou role prohazovat dle potřeby komunikace, vždy však bude v rámci `piconet`u pouze jeden `master`. Příkladem tohoto modelu je připojení sluchátek k multimediálnímu přehrávači. Při prvotním spojení sluchátka převezmou roli `master`, jelikož chtějí komunikovat s přehrávačem, a započnou tak komunikaci. Po úspěšném spojení se role prohodí – roli `master` převezme přehrávač, protože začne přehrávat obsah do sluchátek.

V sítě `piconet` může jedno zařízení komunikovat až se 7 dalšími zařízeními. Toto omezení je dáno velikostí adresového prostoru, který má 3 bity. Adresa s hodnotou 0 se navíc používá pro `broadcast` (adresa, pomocí které se odešle zpráva všem zařízením připojeným do sítě). Pokud se více sítí `piconet` propojí, vzikne síť `scatternet`. Síť `scatternet` však není přímo technologií Bluetooth podporována. Možnost vytvoření tohoto spojení musí být implementovaná v rámci jednotlivých zařízení.

2.8.3 Přenos dat

Bluetooth používá frekvenční pásmo 2.4 GHz pro přenos dat a komunikaci. Aby mohla komunikace správně fungovat, všechna zařízení v síti používají systémové hodiny zařízení, které má roli `master`. Data se mezi zařízeními přenáší pomocí `slotů`. Jeden `slot` je vytvořen dvěma `tiky` systémových hodin. Data jsou podobně jako u TCP / UDP protokolu přenášena pomocí paketů. Pakety mohou mít velikost jednoho, tří nebo pěti slotů. Správna synchronizace vysílání a přijmu dat je zajištěna rozložením slotů pro jednotlivé role. `Master` vysílá data v sudých slotech a přijímá v lichých. U role `slave` je to přesně naopak, tedy v sudých slotech přijímá a v lichých vysílá.

Kapitola 3

Návrh

3.1 Výběr technologie

3.1.1 Technologie v lokální síti

Pro řešení problému propojení dvou a více zařízení v rámci lokální počítače sítě jsou nejvhodnější tyto diskutované technologie:

- knihovna BSD Sockets,
- Bluetooth.

Knihovna BSD Sockets představuje standardní rozhraní pro komunikaci více zařízení v počítačové síti. Knihovna je implementací práce se sockety pro operační systémy založené na Unixu. Pro využití knihovny je potřeba pouze síťové rozhraní, pomocí kterého bude komunikace probíhat. Jelikož je jádro operačních systémů od společnosti Apple pro všechna zařízení stejné - tvoří ho operační systém `Darwin`, lze tuto knihovnu použít na všech zařízení. Mobilní operační systém Android je také založený na operačním systému Unix, knihovnu lze tedy použít i na této platformě. Pro platformu Windows lze využít specifické knihovny pro práci se sockety vytvořenou výhradně pro operační systémy Windows.

Druhou možností je využití technologie Bluetooth. Bluetooth se nachází v zařízeních, jejichž komunikace probíhá v rámci malé vzdálenost. Příkladem využití jsou:

- sluchátka pro bezdrátový poslech,
- dálkové ovladače (klávesnice, myš),
- iBeacon.

Použití technologie Bluetooth pro spojení dvou a více zařízení se v porovnání se sockety jeví jako méně vhodné. Bluetooth funguje pouze na určitou vzdálenost a dalším omezením Bluetoothu může být počet zařízení, které můžou mezi sebou aktivně komunikovat – pro Bluetooth je to jedno `master` zařízení a 7 aktivních zařízeních v roli `slave`. Bluetooth se jeví jako vhodná technologie v případě, že by se propojovala zařízení, kde jedno z nich nemá funkční síťové rozhraní nebo nemůže být do počítačové sítě připojeno. Pro realizaci propojení po lokální počítačové síti je tedy vhodnější použití knihovny pro práci se sockety.

3.2 Technologie v síti Internet

Pro realizace propojení dvou zařízení v rámci síti Internet lze použít následující technologie:

- server,
- webovou real-time databázi.

Webová real-time databáze v porovnání se serverem přináší spoustu výhod. Využitím real-time databáze, která je obsahem již existující služby, lze získat plně funkční rozhraní pro ukládání a získávání dat z jednoho místa. Navíc zařízení využívající tuto službu získá obousměrné spojení s databází. O data se tedy nemusí žádat, ale budou asynchronně zasílána na základě předem definovaných událostí (např. změna dat, nová data). Použití serveru jako zařízení pro vytvoření spojení je řešení méně vhodné. Je-li server implementován jako zařízení komunikující pomocí TCP protokolu, lze využít obousměrného komunikačního spojení pomocí socketů. Toto řešení však lze uskutečnit napřímo v lokální síti a spojení přes server nepřináší žádné výhody. Vytvoření spojení toutu cestou tedy není vhodné. Druhou možnou implementací serveru je možnost použití klasického HTTP serveru s API. Zařízení budou se serverem komunikovat přes toho rozhraní. Možnosti posílání dat ze serveru je diskutovaná v kapitole 2.6. Pokud je potřeba pouze centrálně shromažďovat data, nad kterými není potřeba provádět žádné další operace, použití webové real-time databáze se jeví jako lepší volba.

Obě tyto možnosti ale mají jednu nevýhodu. Pokud by mělo propojení fungovat bez připojení k síti Internet, využití těchto technologií nebude možné.

3.3 Shrnutí

Použití webové real-time databáze a vytvoření komunikace po lokální síti se navzájem nevylučuje. Pokud bude propojení vytvořeno pomocí modelu klient-server, kde klient bude chytrý telefon a server bude Apple TV, lze využít spojení mezi klientem a serverem v rámci lokální sítě pomocí knihovny pro

práci se sockety, a jednotlivé servery mohou sdílet mezi sebou data pomocí webové real-time databáze. Toto řešení se zdá z řady výše zmíněných důvodů jako nejlepší možné.

3.4 Knihovna pro síťovou komunikaci

Nejvhodnějším a nejefektivnějším řešení komunikace po lokální síti bude využití knihoven pro práci se sockety. Prototyp aplikace bude spuštěn na zařízení od společnosti Apple, proto bude důraz při výběru správné knihovny kladen především na kompatibilitu s touto platformou.

3.4.1 CFNetwork

Samotný Apple vytvořil vlastní knihovnu CFNetwork, která obaluje a rozšiřuje systémovou knihovnu BSD Sockets diskutovanou výše. Díky tomu mají programátoři k dispozici rozhraní, pomocí kterého mohou programovat aplikace využívající nízko úrovňové síťové nástroje (např. sockety).

Tato knihovna však není příliš uživatelsky přívětivá z hlediska objektového návrhu. Jelikož vychází a obaluje knihovnu BSD Sockets, její funkce jsou často nuceny používat datové typy a programové návrhy jazyky C, ve kterém je knihovna BSD Sockets napsána.

3.4.2 CocoaAsyncSocket

Tato knihovna slouží pro asynchronní práci se sockety a práci s nimi. CocoaAsyncSocket je knihovna, která obaluje systémovou knihovnu CFNetwork, avšak používá datové typy, které jsou pro programovací jazyky Swift a Objective-C více vhodné.

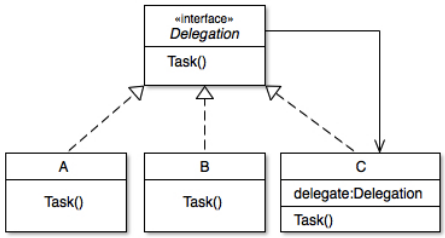
3.4.2.1 Návrhový vzor Delegation

Knihovna využívá pro asynchronní volání návrhového vzoru Delegation. Tento návrhový vzor využívá možnosti přesměrování jednotlivých podpožadavků na jiné objekty. Díky tomu lze přesunout část funkcionality na různé objekty i jiným způsobem než děděním.

Návrhový vzor Delegation je implementován na řadě míst přímo v systémech iOS i tvOS. Využívají se především v místech, kdy objekty delegují funkcionalitu na jiné objekty, se kterými by dle architektury aplikace neměly komunikovat napřímo (viz 3.6.1).

3.5 Vyhledávání zařízení

Pro vyhledání zařízení na síti byla zvolena technologie Bonjour. Bonjour, který byl diskutován v kapitole 2, poskytuje rozhraní, jak vyhledávat v síti zařízení



Obrázek 3.1: Návrhový vzor Delegation [4]

či služby. Zařízení si definuje svoji službu pomocí názvu a pak ji publikuje na síť. Ostatní připojená zařízení, která vyhledávají danou službu, se o tomto zařízení dozví, a pokud chtějí, mohou s ním navázat kontakt. Po navázání kontaktu je vyhledávajícímu zařízení předána IP adresa s portem, na kterém se nachází vyhledané zařízení.

Jelikož je Bonjour implementací sady technik nazývajících se zeroconf, existuje v současné době mnoho řešení, které zeroconf implementují na ostatních platformách. Díky tomu lze zaručit možnost v budoucnu připojit i zařízení, která využívají jiné operační systémy, než ty od společnosti Apple.

3.6 Aplikace

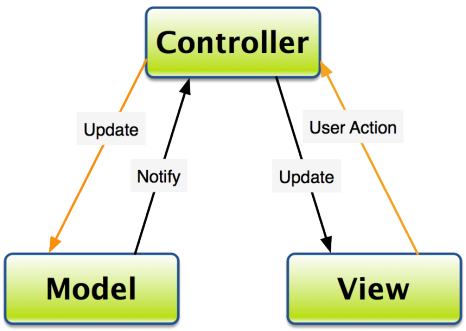
Pro prototyp aplikace demonstrující vybranou metodu propojení chytré televize Apple TV s chytrým telefonem, byla jako zástupce chytrého telefonu vybrán iPhone. Tento telefon je stejně jako Apple TV produktem společnosti Apple.

Operační systém iOS, který je součástí mobilních telefonů společnosti Apple, je velice podobný tomu, který je součástí Apple TV. Oba tyto operační systémy mají mnoho společných funkcí a vlastností a odlišují se pouze v detailech.

Prototyp aplikace má demonstrovat propojení mezi dvěma zařízeními. Z tohoto důvodu bude součástí prototypu návrh dvou nezávislých částí, které se budou navzájem tvářit jako klient-server architektura, kde jedna část bude spuštěna na Apple TV a druhá část bude spuštěna na chytrém telefonu.

3.6.1 Model-View-Controller

Samotná aplikace, ať už pro iOS, či tvOS, je tvořena v architektuře Model-View-Controller (MVC). Tato architektura je specifická tím, že dělí aplikaci do tří samostatných logických celků tak, aby jednotlivé celky šlo upravovat samostatně a jednotlivé změny v těchto částech měly co nejmenější dopad na části ostatní [28].



Obrázek 3.2: Architektura Model-View-Controller [5]

3.6.1.1 Model

Modelové objekty zajišťují přístup k datům aplikace, popř. manipulaci s nimi. Jednotlivé modely mohou obsahovat další modely v sobě vnořené, se kterými mohou komunikovat. Modelové objekty lze přenést do reálného života. Modelovým objektem může být např. článek nebo auto.

3.6.1.2 View

View, nebo-li pohled, je objekt, který uživatel reálné vidí. S těmito objekty může dále interagovat a tyto akce se přes controller předávají dále do modelu. Pohledy také čekají na změny v modelech a na tyto změny adekvátně reagují (např. změna barvy). Přestože pohledy reagují na změnu v modelech, nejsou s nimi napřímo spojeny – je to jeden za základních prvků architektury.

3.6.1.3 Controller

Controller je hlavní řídící jednotka aplikace. Jednotlivé pohledy přes controller komunikují s modely. Na druhé straně, pokud v modelovém objektu vznikne nějaká změna, je povinnost controlleru upozornit o této změně příslušné pohledy.

Díky tomuto rozdělení jednotlivých částí, se logika pro každou kategorii vyskytuje pouze na přiděleném místě a zvyšuje se tím přehlednost kódu. V rámci prototypu aplikace se však uchýlím k modernější verzi architektury Model-View-Controller, která ji rozšiřuje o další vrstvu a ještě více zvyšuje přehlednost kódu. Tato architektura se nazývá Model-View-ViewModel.

3.6.2 Model-View-ViewModel

Model-View-ViewModel je architektura založená na architektuře MVC. Při použití této architektury se přidává mezi vrstvu Controller a View další logická jednotka nazývaná ViewModel. ViewModel přebírá z controlleru logickou část

kódu a na Controlleru zůstává rozmístění jednotlivých pohledů a reakce na systémové notifikace. Vrstva ViewModelu vznikla za účelem oddělit návrh (rozmístění pohledů) od logiky pohledů [29].

Architektura MVVM je často doplněná o prvky reaktivního a funkcionálního programování. Tyto dvě metody programování společně tvoří silný celek, který je při správném použití velice silný. Reaktivní programování se zaměřuje na zpracování datového toku, kde kód aplikace reaguje na změny, které nastavenou v průběhu toku. Funkcionální programování k tomu přidává deklarativní programovací paradigma, které specifikuje, co se má s danou změnou stát. Opakem deklarativního programování je programování imperativní. Hlavním rozdílem mezi těmito dvěma druhy je, že deklarativní programování říká, co se má udělat, kde na druhé straně imperativní programování říká, jak se to má udělat [30]. Při použití těchto programovacích paradigmat v reálném projektu, lze velice jednoduše vytvořit lehce čitelný a přehledný kód, který by byl při použití standardních technik složitý a těžce čitelný. Příkladem může být jednoduchá aplikace, která vykoná definovanou operaci poté, co uživatel klikne na tlačítko. Tato akce vykoná změnu, která je zachycena a s využitím reaktivního programování a pomocí pár funkcí správně a jednoduše zpracována.

3.7 Programovací jazyk

Pro implementaci prototypů aplikací byl zvolen programovací jazyk Swift. Ten představila společnost Apple v roce 2014 jako nástupce do té doby používaného jazyka Objective-C. Swift je moderní programovací jazyk, který v sobě ukrývá řadu paradigmat z ostatních programovacích jazyků. Obsahuje prvky objektového a funkcionálního programování, je staticky typovaný a kompilovaný. Inspirací pro tvůrce byly např. jazyky Scala, Ruby ale i Objective-C. Swift byl navržen pro tvorbu aplikací pro produkty od společnosti Apple a pro aplikace běžící na operačním systému Linux. Na konci roku 2015 byl Swift uvolněn jako open-source. Díky tomu se v poslední době začíná Swift objevovat i na jiných platformách, než na které byl na začátku předurčen.

3.8 Návrh uživatelského rozhraní

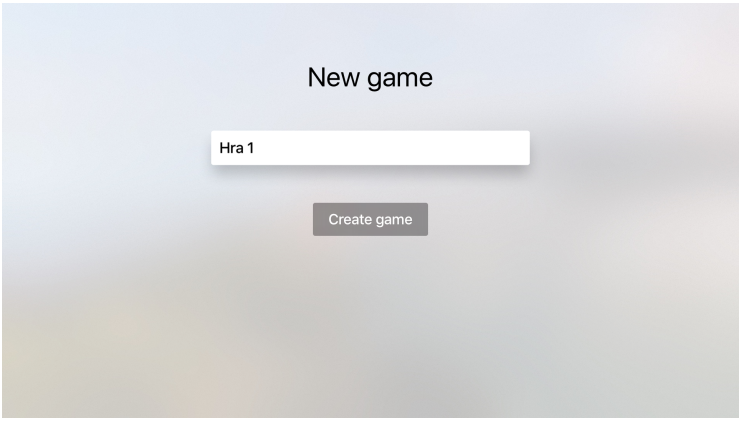
Pro implementaci spojení byla jako ukázková aplikace zvolena kvízová hra. Prototyp aplikace se bude sestávat ze dvou částí.

3.8.1 Serverová část

První částí aplikace je serverová část, která obstárává samotnou hru.

3.8.1.1 Založení hry

Hra začne jejím založením. Obrazovka založení hry (obrázek 3.3) se skládá ze dvou prvků: textového pole a tlačítka „Create game“. Po vyplnění názvu hry a kliknutí na tlačítko se hra vytvoří a je dostupná pro klientskou část aplikace.



Obrázek 3.3: Ukázka serverové aplikace – založení hry

3.8.1.2 Zobrazení otázky

Po úspěšném vytvoření hry a připojení jednotlivých hráčů se zobrazí obrazovka s otázkou (obrázek 3.4). Na této obrazovce je zobrazena zadaná otázka, její možné odpovědi, připojení uživatelé a jejich současné skóre ve hře. Každá otázka je časově omezená. Časový interval pro odpovědi je vidět pod textem otázky. Hráč, který již na zadanou otázku odpověděl, je indikován pomocí šedého čtyřúhelníku vedle svého bodového odohodnocení.

Obrázek 3.4: Ukázka serverové aplikace – zadání otázky

3.8.1.3 Vyhodnocení otázky

Po uplynutí časového intervalu server automaticky vyhodnotí odeslané odpovědi a hráčům přiřadí adekvátně body. Každá správně odpověď přidá danému hráči jeden bod, za špatnou odpověď hráč nic nezíská ale ani nic neztratí. Obrazovka s vyhodnocenou otázkou (obrázek 3.5) se od obrazovky se zadnou otázkou (obrázek 3.4) liší pouze v zobrazení správné odpovědi (špatné odpovědi jsou ztmaveny) a u jednotlivých hráčů je zobrazena jejich odpověď. Skóre hráčů je již upraveno v závislosti na odesláné odpovědi. Hráč se správnou je označen tak, že řádek s jeho jménem má zelené pozadí, hráč se špatnou odpovědí je označen pomocí červeného pozadí.

Po uplynutí časového intervalu 5 sekund se načte nová otázka a obrazovka se vrací zpět na Zobrazení otázky.

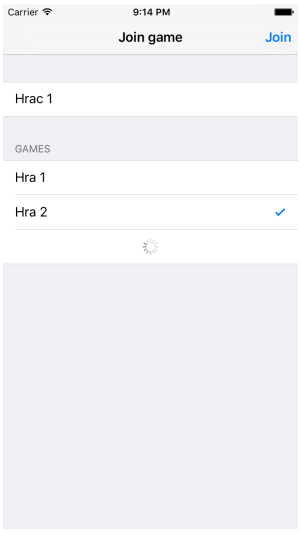
Obrázek 3.5: Ukázka serverové aplikace – vyhodnocení otázky

3.8.2 Klientská část

Druhá část aplikace je klientská část, která poběží na chytrém telefonu. Úkolem této části je vytvoření a připojení nového uživatele a odesílání uživatelem zvolených odpovědí do serverové části aplikace. Klientská část se skládá ze dvou obrazovek.

3.8.2.1 Připojení se ke hře

Obrazovka s připojením se ke hře (obrázek 3.6) obsahuje jedno textové pole, do kterého uživatel vloží svou přezdívku. Dále je zde výpis dostupných her v okolí. Výpis se v reálném čase aktualizuje. Posledním prvekem na obrazovce je tlačítko „Join“, pomocí kterého se uživatel připojí k vybrané hře. Vybraná hra je indikovaná modrou „fajfkou“.



Obrázek 3.6: Ukázka klientské aplikace – připojení uživatele

3.8.2.2 Odpovídání na otázku

Obrazovka s odpověďmi (obrázek 3.7) obsahuje 4 odpovědi. Odpovědi jsou popsané a rozlišené barvami. Uživatel odpoví tak, že na vybranou odpověď klikne. Po kliknutí se obrazovka zablokuje až do přijetí dat k následující otázce. Po odeslání odpovědi již vybranou odpověď nelze změnit.

Obrázek 3.7: Ukázka klientské aplikace – zobrazení odpovědí

Kapitola 4

Implementace

4.1 Vytvoření socketu

Aby mohlo být spojení realizováno, je zapotřebí systému říci, na kterém portu bude aplikace komunikovat s ostatními zařízeními. Komunikace probíhá pomocí socketu. Socket je označení pro jeden konec komunikačního kanálu, přes který mezi sebou jednotlivá zařízení komunikují. Bez vytvoření socketu není možné komunikaci navázat.

Pro lepší přehlednost kódu byla pro práci se sockety vybrána knihovna `CocoaAsyncSocket`, která poskytuje rozhrání, pomocí kterého lze jednoduchým způsobem pracovat ve zdrojovém kódu se sockety. Knihovna obaluje nativní systémovou knihovnu `CFNetwork` a přidává k ní různá vylepšení, která umožňují vývojáři používat konstrukty z programovacího jazyka Swift.

Pro vytvoření socketu, je třeba zavolat následující incializační metodu objektu `GCDAsyncSocket`:

```
public init!(delegate aDelegate: AnyObject!,
    delegateQueue dq: dispatch_queue_t!)
```

Parametr

- **`delegate`** je instance objektu, který bude zodpovědný za zpracování dat z metod, které budou pomocí návrhového vzoru Delegation, diskutovaný v kapitole 3.4.2.1, předány k dalšímu zpracování,

- **`delegateQueue`** určuje, na jakém vlákně se delegované metody budou provádět.

Celá knihovna pro práci se sockety pracuje asynchronně, tedy vykonávání programu nečeká na okamžik, kdy prostřednictvím socketu dorazí nějaká data, ale pokračuje dále ve vykonávání, a ve chvíli, kdy data dorazí, patřičně na tuto událost zareaguje.

Samotné vytvoření socketu ale pro jeho správnou práci nestačí. Socket je přímo určen IP adresou a příslušným portem. Proto pro odstartování sledování činnosti na socketu je potřeba zavolat následující metodu:

```
public func acceptOnPort(port: UInt16) throws
```

Tato metoda přebírá pouze jeden parametr, kterým je číslo portu, na kterém bude daný socket vytvořen. V případě, že je jako parametr `port` předáno číslo 0, přiřadí dané číslo portu socketu systém. Od této chvíle socket plně funguje a aplikace může začít reagovat na jednotlivé komunikační události.

4.2 Publikování služby

Pokud se socket správně vytvoří a je mu přidělena IP adresa s portem, lze přistoupit k vytvoření služby, která bude na dané adrese k dispozici ostatním zařízením v síti. Pro vytvoření služby je potřeba zavolat její konstruktor:

```
public init(domain: String, type: String, name:
    String, port: Int32)
```

Parametr:

- **`domain`** určuje název domény, ve které bude služba publikována. Pro publikování služby v rámci lokální počítačové sítě je tomuto parametru předán řetězec `"local."`.
- **`type`** označuje typ služby, která bude publikována. Parametru `type` je předán řetězec dle jmenné konvence z kapitoly 2.5.2.
- **`name`** označuje název služby viditelný pro zařízení, které ji vyhledají. Toto označení slouží především pro uživatele aplikace, který se v případě výskytu více stejných služeb, může rozhodnout pro jednu konkrétní z nich.
- **`port`** je číslo portu, na kterém bude služba poběží. Do tohoto parametru je předáno číslo portu dříve vytvořeného socketu pro zachycení komunikace služby po jejím publikováním pomocí tohoto socketu.

Po vytvoření instance služby je zapotřebí službu publikovat. Publikování se provede pomocí metody:

```
public func publish()
```

Práce se službami (třídou `NSNetService`) je stejně jako knihnovna pro práci se sockety navržená asynchronně. Jednotlivé události, které se v průběhu vytváření a publikování objeví, se při běhu programu zpracovávají dynamicky, tedy ve chvíli, kdy nastanou. Aby se mohlo dosáhnout tohoto návrhu, využívá implementace opět návrhového vzoru `Delegation`. Jako delegáta lze službě přiřadit libovolnou instanci. Podmínkou pro tuto instanci je implementování

metod, které jsou definovány v protokolu `NSNetServiceProtocol`. Tyto metody se volají ve chvíli, kdy pro danou službu vznikne událost, na kterou může přiřazený delegát zareagovat.

Mezi dvě nejdůležitější metody tohoto protokolu patří:

```
optional public func netServiceDidPublish(sender:
    NSNetService)
```

díky které může aplikace zareagovat na stav, kdy služba byla úspěšně publikována na síti a

```
optional public func netService(sender: NSNetService,
    didNotPublish errorDict: [String : NSNumber]).
```

Tato metoda slouží k zjištění důvodu, proř se nepodařilo danou službu publikovat a dává možnost aplikaci na tuto událost zareagovat.

4.3 Nalezení služby

Funkce nalezení služby je velice podobná, jako publikování služby. Prvním rozdílem je, že tuto funkcionalitu přináší třída `NSNetServiceBrowser` a druhým podstatným rozdílem je, že by tato funkce měla být spuštěna v klientské části aplikace, která chce navázat kontakt se serverem, který danou službu publikoval.

Vyhledávání služeb se odstratuje po vykonání následujícího příkazu:

```
func searchForServicesOfType(_ type: String, inDomain
    domainString: String)
```

Parametr:

- **`type`** je stejný jako u `NSNetService` a určuje typ služby, která je vyhledávána.
- **`domainString`** určuje, v jaké doméně se služba bude vyhledávat. Jelikož je vyhledávání prováděno v rámci lokální sítě, je tomuto parametru předán řetězec `"local."`.

Stejně jako třída `NSNetService` i `NSNetServiceBrowser` využívá návrhového vzoru `Delegation` pro asynchronní zpracování událostí. Objekt, který se stane delegátem, může implementovat potřebné metody z protokolu `NSNetServiceBrowserDelegate`. Nejpodstatnější z těchto metod je:

```
optional public func netServiceBrowser(browser:
    NSNetServiceBrowser, didFindService service:
    NSNetService, moreComing: Bool)
```

Parametr:

- **browser** obsahuje instanci vyhledávače, který našel novou službu,
- **service** obsahuje instanci nové nalezené služby,
- **moreComing** udává, zda-li nalezených služeb bylo více a tedy zda-li se tato metoda bude volat vícekrát pro různé nalezené služby.

Tato metoda je volána vždy, když vyhledávač narazí při vyhledávání po síti na novou službu, která jím zatím nebyla nalezena. Poslední parametr `moreComing` slouží především pro grafické rozhraní aplikace, kde v závislosti na tomto parametru může aplikace změnit své rozhraní anebo počkat, až se všechny nalezené služby zpracují a až poté změnit rozhraní.

Součástí protokolu je samozřejmě i metoda, která se zavolá ve chvíli, kdy se nějaká služba odpojila. Tato metoda však slouží především pouze pro upravení rozhraní aplikace a není nutné ji zde diskutovat.

4.4 Propojení zařízení

Po nalezení služby se lze k této službě připojit. Žádost o propojení je realizováno pomocí metody

```
public func resolveWithTimeout(timeout:
    NSTimeInterval)
```

jenž je zavolána na zvolené instanci nalezené služby. Parametr:

- **timeout** určuje časový interval, po který se čeká, zda-li služba vrátí informace o koncovém zařízení. Pokud se tomuto parametru předá číslo `0`, probíhá vyčkávání až do chvíle, než nastane úspěšné nebo neúspěšné předání dat.

V případě neúspěšného získání dat, zavolá se metoda protokolu `NSNetServiceDelegate`

```
optional public func netService(sender: NSNetService,
    didNotResolve errorDict: [String : NSNumber])
```

V případě úspěšného získání dat (adresy) ze služby se zavolá metoda

```
optional public func netServiceDidResolveAddress(
    sender: NSNetService)
```

kde parametr `sender` obsahuje instanci služby, které byla o adresu požádána. Parametr `sender` obsahuje vlastnost[1] `addresses`, což je pole adres, ze kterých je daná služba dostupná. S využitím těchto adres můžeme vytvořit socket a připojit ho k jedné ze získaných adres. Tento postup demonstruje následující metoda:

[1] myšleno anglické property

```
public func connectToAddress(remoteAddr: NSData!)
    throws
```

Předaným parametrem je adresa, na kterou se má socket připojit. Jelikož je vlastnost `addresses` pole, je potřeba zavolat tuto metodu v cyklu na každé z adres v poli.

V případě úspěšného připojení se k zadané adrese, je na socketu zavolána metoda

```
optional public func socket(sock: GCDAsyncSocket!,
    didConnectToHost host: String!, port: UInt16)
```

po jejímž zavolání můžeme začít provádět síťovou komunikaci mezi propojenými zařízeními. Současně s touto metodou je v serverové části zavolána metoda

```
optional public func socket(sock: GCDAsyncSocket!,
    didAcceptNewSocket newSocket: GCDAsyncSocket!)
```

Pokud se obě metody zavolají, je jisté, že se zařízení povedlo úspěšně propojit.

4.5 Odesílání dat

Data odesílaná z jednoho zařízení na druhé mohou obsahovat různé typy informací. Tyto typy jsou potřeba systematicky seskupit a sjednotit, aby bez větších problémů na obou koncích spojení došlo ke správné interpretaci předaných dat. Z tohoto důvodu byla vytvořena třída `Packet`, která slouží pro přenos dat s jednotným rozhraním pro obě verze aplikace.

K odeslání dat je potřeba vytvořit instanci třídy `Packet` pomocí konstruktoru

```
init(object: Sendable, type: PacketType)
```

Parametr:

- **`object`** je instance objektu, který se má poslat skrz komunikační kanál,
- **`type`** je hodnota z výčtového typu `PacketType`, který obsahuje hodnotu pro každý typ zasílaných dat. Např. pro data zasílaná při přihlášení má tento parametr hodnotu `.Login`[2].

Parametr **object** je typu `Sendable`, což je protokol definující rozhraní, které musí posílaný objekt splňovat (obrázek 4.1). Toto rozhraní slouží pro převedení objektu na datový typ `NSData`, což je objektově orientovaný obal pro surová data. Třída `Packet` odpovídá také protokolu `Sendable`, protože po síti lze poslat právě jen surová data.

Posílání `Packet`u po síti probíhá pomocí metody

[2] `.Login` je zkrácená verze hodnoty `PacketType.Login`

```
protocol Sendable {
    func archive() -> NSData
    static func unarchive(data: NSData) -> Self
}
```

Obrázek 4.1: Rozhraní protokolu `Sendable`

```
func sendOnSocket(socket: GCDAsyncSocket, withTimeout
    timeout: Double = -1, tag: Int = 0)
```

Parametr:

- **`socket`** je instance socketu, pomocí kterého se mají data poslat,
- **`timeout`** udává dobu v sekundách, po kterou se data budou zkoušet odesílat,
- **`tag`** je pomocné číslo pro správnou interpretaci dat.

Před odesláním dat se vytvoří **`buffer`** jako instance třídy `NSMutableData`, což jsou výsledná surová data připravená pro poslání. `Mutable` v názvu třídy je označení pro objekty, které lze v průběhu vykonávání kódu rozšiřovat a měnit. Na posíláním objektu, který byl předám v konstruktoru třídy `Packet`, se zavolá metoda `archive()`. Díky tomu lze získat velikost zasílaných dat. Tato velikost se vloží jako surová data do objektu **`buffer`**, přičemž délka velikosti je fixně daná. Následně se přidají surová data zasílaného objektu. Nakonec je instance třídy `Packet` odeslána pomocí metody `writeData(_:withTimeout:tag:)` na předaném parametru `socket`.

4.6 Načtení dat

Načtení příchozích dat ze socketu je řešeno zavoláním metody na daném socketu

```
public func readDataToLength(length: UInt,
   withTimeout timeout: NSTimeInterval, tag: Int)
```

Parametr:

- **`length`** udává délku očekávaných přichozích dat,
- **`timeout`** je časový interval v sekundách, po který se bude čekat na příchozí data,
- **`tag`** je pomocné číslo pro správnou interpretaci dat.

Pokud je předán záporný časový interval, **timeout** není použit.

Příchozí data jsou zpracována v metodě

```
optional public func socket(sock: GCDAsyncSocket!,
    didReadData data: NSData!, withTag tag: Int)
```

Parametr

- **sock** je instance socketu, ze kterého data dorazila,
- **data** jsou příchozí data,
- **tag** je pomocné číslo pro správnou interpretaci dat.

Parametr **tag** je stejné číslo, jako `tag` předaný metodě `readDataToLength(_:timeout:tag:)`. Úlohu načtení dat lze rozdělit na dvě podúlohy, díky implementaci třídy `Packet`. První podúlohou je načtení velikosti příchozích dat. Protože první část příchozích dat obsahuje velikost následujících dat s fixní délkou, lze tuto hodnotu jednoduše načíst.

Pro načtení první části je použita metoda `readDataToLength(_:timeout:tag:)`, kde předaný parametr `length` je fixní délka první části dat. Příchozí data jsou zpracována v metoda `socket(_:didReadData:tag:)`, kde příchozí data mají žádanou délku a obsahují velikost dat zasílaného objektu. Po získání hodnoty velikosti dat zasílaného objektu je znovu zavolána metoda `readDataToLength(_:timeout:tag:)`, avšak tentokrát je jako parametr `length` předána velikost dat objektu. Následně se opět zavolá metoda `socket(_:didReadData:tag:)`, ve které dorazí zaslaná data. Protože metoda `socket(_:didReadData:tag:)` se volá pro jakákoliv data a je potřeba rozlišit, zda-li je načítaná velikost posílaných dat nebo již samotná data. Proto je zde využito parametru `tag`, pomocí kterého lze rozlišit načítání jednotlivých částí. Pro načtení velikosti dat je použit `tag` s hodnotou `0` a pro načtení samotného posílaného objektu má `tag` hodnotu `1`.

Správná interpretace příchozích dat je zaručena díky hodnotě výčtové typu `PacketType`, se kterým byla data (instance třídy `Packet`) na jednom konci spojení vytvořena. Získání zasláného objektu je díky protokolu velice snadné `Sendable`. Stačí zavolat na třídní metodu `unarchive(_:)`, které jsou předána surová data reprezentující zaslaný objekt.

Kapitola 5

Budoucí práce

Prototypy aplikací, které implementují vybrané techniky a technologie zkoumané v práci a které demonstrují možné propojení dvou zařízení disponující síťovým rozhraním, byly navrženy tak, aby co nejvhodněji řešily daný problém. Obě aplikace spolu komunikují po lokální síti, jedním z možných rozšíření je připravení propojení většího množství zařízení i v rámci síti Internet. K této funkcionalitě by bylo vhodné zvolit webovou real-time databázi. Díky tomu lze zajistit funkční propojení dvou a více zařízení v rámci různých propojených počítačových sítí.

Implementovaný prototyp lze využít mnoha způsoby. Příkladem může být kvízová hra, pomocí které je propojení demonstrováno. Dalším příkladem využití může být přidání dalšího ovladače do již existující hry. Hra díky tomu může využít všech senzorů, kterými disponuje chytrý telefon.

V rámci práce je vytvořen pouze prototyp. Rozšířením prototypu lze také vytvořit aplikaci, pomocí které by se s toutu technologií mohli setkat studenti předmětů zajímajících se o tuto problematiku.

V prototypu aplikace je implementován postup, jak propojit dvě zařízení. Budoucí prací může být vytvoření knihovny, která by poskytovala rozhraní, jak tento postup využít v dalších aplikacích používajících stejnou technologii. Knihovna by sloužila pro ulehčení práce ostatním vývojářům.

Závěr

Cílem mé bakalářské práce bylo prozkoumat možnosti propojení nové Apple TV s chytrým telefonem a napsat prototypy dvou aplikací, které budou demonstrovat jednu vybranou možnost. Během své práce jsem se detailně seznámil se všemi možnostmi, jak propojit dvě zařízení disponující připojením k počítačové síti a rozšířil si své obory v oblasti počítačových sítí. Všechny cíle, které byly zadané na počátku práce, byly splněny.

Jelikož v současné době neexistuje žádné „standardizované“ řešení, závěry z mé práce mohou být použity u aplikací, které se reálně používají ve světě, a nebudou tak porušena žádná pravidla ani ustanovení, která existují pro programování pro obě platformy.

Literatura

[1] Vašek, J.: VNC a Vzdálená plocha - kouzlo vzdáleného přístupu. http://pctuning.tyden.cz/index2.php?option=com_content&task=view&id=12639&Itemid=1&pop=1&page=0, 2009, [Obrázek; cit. 25.4.2016].

[2] Chau, A.: From the Helpdesk ... Security Concerns of Peer-to-Peer Software. https://www.its.hku.hk/news/ccnews125/p2p.htm, 2007, [Obrázek; cit. 25.4.2016].

[3] Marcin Klimek, L. T.: Sockets. http://www.python4science.eu/sockets.html, 2014, [Obrázek; cit. 25.4.2016].

[4] GG1: Delegation using swift. http://www.xappsoftware.com/wordpress/2016/01/25/delegation-using-swift/, 2016, [Obrázek; cit. 25.4.2016].

[5] BlackHatSamurai: Converting Java code to model view controller architecture pattern. http://stackoverflow.com/questions/29325038/converting-java-code-to-model-view-controller-architecture-pattern, 2015, [Obrázek; cit. 25.4.2016].

[6] Betters, E.: New Apple TV: Release date, price, specs, rumours and everything you need to know. http://www.pocket-lint.com/news/127990-new-apple-tv-release-date-price-specs-rumours-and-everything-you-need-to-know, 2015, [Online; cit. 1.4.2016].

[7] Haslam, K.: New Apple TV (fourth-gen, 2015) review: Pros and cons of Apple's fourth-generation Apple TV, updated with tvOS 9.2. http://www.macworld.co.uk/review/apple-tv/new-apple-tv-2015-review-pros-cons-tvos-siri-app-store-dictation-update-3531549/, 2016, [Online; cit. 2.4.2016].

[8] Clover, J.: New Apple TV Only Supports Two Bluetooth Controllers at Once. http://www.macrumors.com/2015/10/02/apple-tv-two-bluetooth-controllers-maximum/, 2015, [Online; cit. 2.4.2016].

[9] Trdlička, J.: Procesy a vlákna. https://edux.fit.cvut.cz/courses/BI-OSY/_media/lectures/02/bi-osy-p02-threads.pdf, 2016, [Online; cit. 7.4.2016].

[10] Proces (program). https://cs.wikipedia.org/wiki/Proces_(program)/, 2016, [Online; cit. 5.4.2016].

[11] Knakal, L.: IP paket. http://home.zcu.cz/~lknakal/, [Online; cit. 7.4.2016].

[12] Smotlacha, V.: Transportní vrstva. https://edux.fit.cvut.cz/courses/BI-PSI/_media/lectures/p6-transport.pdf, 2015, [Online; cit. 9.4.2016].

[13] Beal, V.: What is The Difference Between IPv6 and IPv4? http://www.webopedia.com/DidYouKnow/Internet/ipv6_ipv4_difference.html, 2014, [Online; cit. 9.4.2016].

[14] IP Adresa. https://cs.wikipedia.org/wiki/IP_adresa, 2016, [Online; cit. 9.4.2016].

[15] IANA: Service Name and Transport Protocol Port Number Registry. http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, 2016, [Online; cit. 13.4.2016].

[16] ORACLE: What Is a Socket? https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html, 2015, [Online; cit. 14.4.2016].

[17] FreeBSD: Explaining BSD. https://www.freebsd.org/doc/en/articles/explaining-bsd/article.html, 2016, [Online; cit. 16.4.2016].

[18] IPv6.cz: BSD sockets - Úvod. https://www.ipv6.cz/BSD_sockets_-_Úvod, 2008, [Online; cit. 16.4.2016].

[19] Stanislav, G. A.: Chapter 7. Sockets. https://www.freebsd.org/doc/en_US.ISO8859-1/books/developers-handbook/sockets.html, 2016, [Online; cit. 16.4.2016].

[20] tutorialspoint: Unix Socket - Server Examples. http://www.tutorialspoint.com/unix_sockets/socket_server_example.htm, 2016, [Online; cit. 19.4.2016].

[21] tutorialspoint: Unix Socket - Client Examples. http://www.tutorialspoint.com/unix_sockets/socket_client_example.htm, 2016, [Online; cit. 19.4.2016].

[22] Hoffman, J.: *iOS and OS X Networking Programming Cookbook*. Packt Publishing Ltd., 2014, ISBN 978-1-84969-808-5.

[23] FJFI: Zeroconf. https://nms.fjfi.cvut.cz/wiki/Zeroconf, 2006, [Online; cit. 21.4.2016].

[24] Inc., A.: About Bonjour. https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices/Introduction.html, 2013, [Online; cit. 21.4.2016].

[25] Inc., A.: Domain Naming Conventions. https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices/Articles/domainnames.html#//apple_ref/doc/uid/TP40002460-SW1, 2013, [Online; cit. 21.4.2016].

[26] Lynn, E.: What are Long-Polling, Websockets, Server-Sent Events (SSE) and Comet? http://stackoverflow.com/questions/11077857/what-are-long-polling-websockets-server-sent-events-sse-and-comet, 2015, [Online; cit. 17.4.2016].

[27] Hanson, J.: What is HTTP Long Polling? https://www.pubnub.com/blog/2014-12-01-http-long-polling/, 2014, [Online; cit. 17.4.2016].

[28] Bernard, B.: Úvod do architektury MVC. https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/, 2019, [Online; cit. 27.4.2016].

[29] Furrow, A.: Introduction to MVVM. https://www.objc.io/issues/13-architecture/mvvm/, 2014, [Online; cit. 25.4.2016].

[30] Žoltá, L.: Deklarativní programování. http://lucie.zolta.cz/index.php/softwarove-inzenyrstvi/38-deklarativni-programovani, 2016, [Online; cit. 25.4.2016].

Příloha **A**

Seznam použitých zkratek

API Application programming interface

BSD Berkley Software Distribution

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

HTTP Hypertext Transfer Protocol

IP Internet Protocol

JSON JavaScript Object Notation

mDNS Multicast Domain Name System

REST Representational State Transfer

SIG Bluetooth Special Interest Group

TCP Transmissin Control Protocol

TCP/IP Transmission Control Protocol / Internet Protocol

UDP User Datagram Protocol

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `source` ..... adresář se zdrojovými kódy
  - `implementation` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF