Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Komunikační software pro iPad

Bc. Marek Jalovec

! ! !

Místo této strany vložíte zadání diplomové práce.

(Do jedné vazby originál a do druhé kopii)

! ! !

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Komunikační software pro iPad" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.3.2012 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Vojtěchu Merunkovi, Ph.D.

Komunikační software pro iPad

Communication Software for iPad

Souhrn

Obsahem práce je popis nejpoužívanějších technologií pro vývoj aplikací pro mobilní platformu iOS, správných postupů pro tvorbu uživatelského rozhraní podle dokumentu Human Interface Guidelines společnosti Apple Inc. a předních aplikací, které jsou aktuálně k dispozici na obchodě AppStore a řeší komunikační problémy uživatelů. Práce se zabývá základními rysy jazyka Objective-C, který je kompilátem jazyků C/C++ a SmallTalk-80, které obohacuje o principy plně objektového programování s přístupem k nízko-úrovňovým voláním. Praktická část práce pak popisuje tvorbu aplikace pro komunikaci, která řeší problémy komunikace v aktuálně nabízených aplikací a vytváří tak nový precedens pro tvorbu komunikačního software pro mobilní zařízení.

Summary

The content of this thesis is description of the most commonly used technologies for development of mobile applications for the iOS operating system, correct ways to design and create graphical user interface as suggested in the document Human Interface Guidelines from the Apple Inc. company and front applications, that solves communication problem and are available in the AppStore. Thesis describes basic structure and syntax of the Objective-C language, that uses best from C/C++ and SmallTalk-80 languages and extends them further with fully objective principles with access to lower level system calls. Practical part of this thesis shows this technologies and principles on developing whole new communication application that solves communication problems that arises in real-life situations. This creates new precedent for creating of communication software for mobile devices.

Klíčová slova: Apple, iOS, iPad, iPhone, Objective-C, Cocoa Touch, Foundation framework, Interface Builder, Xcode, komunikace, cizí jazyky

Keywords: Apple, iOS, iPad, iPhone, Objective-C, Cocoa Touch, Foundation framework, Interface Builder, Xcode, communication, foreign languages

Obsah

1. Úvod

Mobilní telefony a mobilní zařízení obecně má dnes naprostá většina uživatelů. Slouží jim místo spousty přístrojů a hraček, které byly naprosto běžné ještě pár let zpátky. Zmizelo tak kapesní rádio, hudební přehrávač, pozvolna mizí budíky, vymřely digi-hry a na ústupu jsou i papírové knihy. Každá technologie musí překonat hranici, kdy ji přijme i kategorie uživatelů, kteří ji odmítají. Pak je globálně přijata a nahradí technologii předchozí.

Firma Apple přišla v lednu 2007 s koncepcí telefonu, který navždy změnil svět komunikačních přístrojů a kterému se tuto hranici podařilo překročit - s iPhonem, a v roce dubnu 2010 se jí to samé povedlo podruhé, tentokrát s iPadem. Tato zařízení využívají inovativního operačního systému iOS, který staví na oblíbené a stabilní platformě MacOS X. V současné době tato zařízení tvoří 76% prodejů firmy Apple a pomohly jí na pozici nejcennější značky světa s tržní nominací 500 miliard dolarů. (Apple Inc., 2012)

Díky velkým displejům, pamětem v řádu gigabytů a intuitivnímu dotykovému uživatelskému rozhraní těchto zařízení se tak přímo nabízí je využít jako pomůcku pro komunikaci a výuku jazyka kdekoli na cestách doma, i v zahraničí díky. Tyto pomůcky (v podobě aplikací) jsou aktuálně velmi neohrabané a nenabízejí prakticky žádné možnosti úpravy významu vět, což lze považovat za velkou slabinu, kterou je potřeba napravit a zaplnit tak díru na trhu.

2. Cíl práce

Cílem práce je vytvořit učební a komunikační pomůcku pro anglicky hovořícího uživatele, která zprostředkuje základní komunikační fráze z předem definovaného seznamu frází v jiném evropském. Pomůcka je realizována jako klientská aplikace pro zařízení s operačním systémem iOS od firmy Apple s využitím dotykového uživatelského rozhraní.

3. Metodika

Tvorba aplikace pro operační systém iOS je možná v nativní formě pouze v prostřednictvím jazyka *Objective-C* na systému MacOS X ve vývojovém prostředí *Xcode*. Existují i alternativní prostředí pro vývoj aplikací pro tuto platformu, ta nicméně nenabízí nativní knihovny v plném rozsahu. Tvorba aplikace tak vyžadovala studium primární literatury k jazyku na webu a fóru pro vývojáře firmy Apple na adrese https://developer.apple.com/ a knih s touto tématikou.

Pro maximalizaci použitelnosti aplikace a zachování *UX*[1] platformy bylo součástí studia jazyka a platformy detailní zkoumání dokumentu *Human Interface Guidelines* firmy Apple, tedy doporučených postupů při tvorbě uživatelských rozhraní pro platformu iOS. V práci je probíráno výhradně uživatelské rozhraní zařízení iPad, protože nabízí uživateli větší komfort, než menší iPhone a iPod Touch. Na těchto zařízeních je aplikace také použitelná, nicméně si zde nedostatek prostoru vynutil jiné uspořádání ovládacích prvků. To je možné prostřednictvím univerzálních *NIB*[2] souborů uživatelského rozhraní aplikace *Interface Builder* řešit bez větších zásahů do zdrojového kódu a aplikaci je tak možno vyvinout relativně jednoduše univerzálně bez dalších nákladů. Tento fakt bohužel neplatí u složitějších aplikací, vyžadujících komplexnější uživatelské rozhraní, a her.

Aplikace není alternativou žádného dosud existujícího řešení, většina konkurenčních aplikací pro překlad a komunikaci pro platformu iOS trpí špatnou rychlostí, nízkou variabilitou a počtem frází, či dalšími nedostatky, či má diametrálně odlišné užití.

Aplikace je kompatibilní s iOS 4.3 a novější z důvodu absence podpory iOS 3 na zařízení iPad - aplikaci by nebylo možno vyvinout univerzální. S iOS 4.3 jsou ale kompatibilní iPhone 3G, iPhone 3GS, iPhone 4, iPhone 4GS, iPod Touch posledních dvou generací a všechny verze iPadu. Uživatelská základna je tak dostatečně velká (přes 95% zařízení se systémem iOS) a nebude problém s kompatibilitou. (HAMACHER, 2011)

Práce byla vypracována na základě analýzy dostupných zdrojů dat, dále je použita metoda dedukce, indukce a syntézy, v neposlední řadě pak vlastní poznatky a zkušenosti z oboru získané několikaletou praxí na pozici programátora, UX návrháře, testera.

[1] User experience - uživatelský prožitek je souhrn veškerých akcí, pocitů, reakcí a působení na uživatele při používání aplikace a jejího rozhraní.

[2] NIB soubory obsahují popis uživatelského rozhraní a jeho napojení na akce v kódu pomocí XML souboru.

První podkapitolou v rešeršní částí práce je stručný popis technologií dotykových mobilních zařízení se zaměřením na zařízení firmy Apple.

Druhá podkapitola řeší problematiku *Human Interface Guidelines*, což je dokument firmy Apple od předních odborníků na průmyslový design a uživatelský zážitek z aplikací. Jsou zde stručně popsány a rozvedeny základní principy použitelné a přístupné aplikace, kterou uživatelé platformy iOS i dalších očekávají a pochopí bez větších problémů a budou ji tak schopni okamžitě používat.

Ve třetí podkapitole jsou rozebírány základní konstrukty jazyka *Objective-C* a jeho syntaxe včetně příkladů tam, kde mohou pomoci pochopení problematiky. Jazyk je velmi podobný jazykům C a C++, uživateli těchto jazyků tak bude většina principů povědomá. Kapitola předpokládá jistou minimální gramotnost v oblasti informatiky a programování.

Čtvrtá podkapitola obsahuje stručnou historii firmy Apple a Steva Jobse, který ji založil a stál v roli generálního ředitele a technologického vizionáře. Despotickým vedením se mu podařilo firmu zachránit a postaral se o spoustu technologických inovací v počítačovém průmyslu a průmyslovém designu s pomocí Jonathana Iveho.

Předposlední kapitola rešeršní části se stručně zaměřuje na jazyk novoslověnština, který byl využit, coby univerzální komunikační prostředek, k demonstraci aplikace. Problematika jazyka není cílem práce. Tento jazyk byl zvolen, protože je blízký českému jazyku, je využíván širokou komunitou a bylo tak možné aplikaci na něm kvalitně otestovat. Pro další studium jazyka lze doporučit oficiální zdroje na adresách http://izviestija.info/ a http://tutorial.neoslavonic.org/. Jazyk je dále podrobně popsán v (MERUNKA, 2009).

Poslední kapitola rešeršní části stručně probírá výhody a nevýhody přímo konkurenčního programu *TalkPad*, který je k dispozici v obchodě AppStore v několika jazykových variantách od čínštiny po perštinu.

Vlastní část práce se skládá ze čtyř částí - Návrh struktury kódu, Hlavní okno, Gramatika a Okno s překladem frází.

V první je pomocí metodiky *UML*[3] demonstrováno základní rozdělení zdrojového kódu programu do logických celků včetně principu dědičnosti, druhá obsahuje popis hlavního okna po inicializaci aplikace, třetí popisuje okno s gramatickým přehledem, který slouží k hlubšímu pochopení skloňování a dalších gramatických jevů v jazyce a poslední kapitola se zaobírá problematikou překladatelské části aplikace. Ta se skládá ze dvou oken a komplexních vykreslovacích metod.

[3] Unified Modeling Language - obecný a unifikovaný jazyk pro modelování objektů, jejich vlastností, metod a vazeb mezi nimi.

4. Přehled řešené problematiky

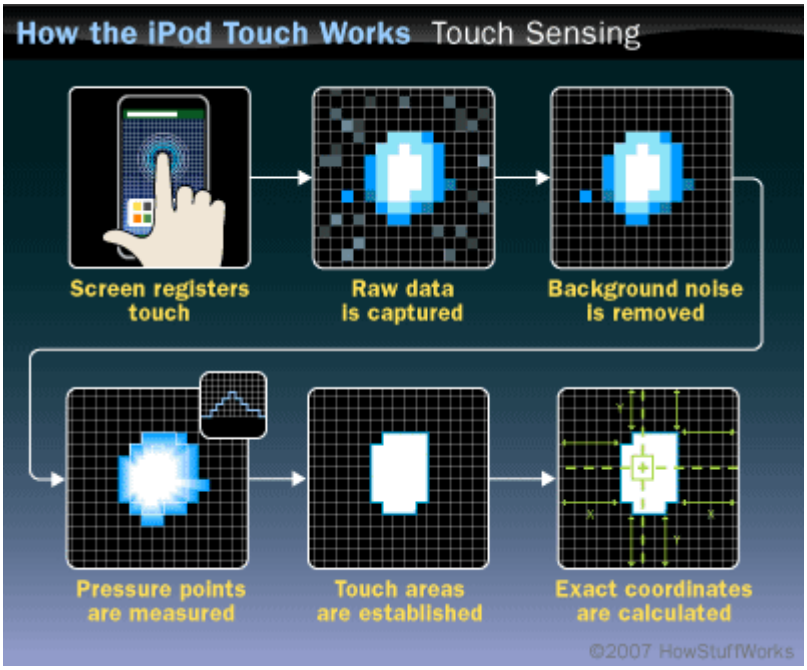
4.1. Mobilní zařízení a dotykový display

První dotykový kapacitní display byl vytvořen již v roce 1965, kdy princip jeho funkčnosti popsal v krátkém článku Samuel Hurst. (Touch display - a novel input/output device for computers, 1965).

Kapacitní display se skládá z vrstvy izolantu (nejčastěji sklo, nebo polykarbonát) potažené vodivou vrstvou. Dotykem prstu dojde ke změně kapacitního odporu, což elektronika převede na souřadnice a předá logice zařízení. Dnes je známo mnoho technologií, které se liší jak cenou a náročností na výrobu, tak rychlostí reakce na vstup uživatele a citlivostí. Druhou nejčastější jsou displeje rezistivní, které však mají výrazně nižší rychlost reakcí. Pracují na principu dvou vodivých vrstev oddělených izolantem (nejčastěji vzduch), kde tlakem dojde ke spojení kontaktu, následnému zjištění pozice a předání řídící elektronice. Tyto displaye byly hojně využívány u zařízení se systémem Windows Embeded CE, kde v kombinaci se špatně navrženým uživatelským rozhraním bez dodržení základních principů použitelnosti způsobily neoblíbenost řešení a odrazovaly uživatele od koupi chytrého telefonu.

Zařízení firmy Apple využívají právě principu kapacitního displeje, i když značně vylepšeného. Rezistivní displej by nebyl u zařízení typu iPad vhodný, neboť dochází k protlaku senzorové vrstvy a takto velká plocha displeje (9,7 palce) by byla velmi zranitelná. Navíc by byl horší uživatelský prožitek, na což firma Apple klade velký důraz.

Jedinečný je u iOS zejména algoritmus na detekci pozice vstupu uživatele, neboť plocha pod prstem je poměrně rozsáhlá. Apple vyvinul sérii procesů, které se snaží maximálně přesně aproximovat záměr uživatele tak, aby byly odpovědi systému jasné, očekávatelné a přitom tolerantní vůči chybám. Je tak možné ovládat systém zcela intuitivně a není třeba řešit dotyk na pixel přesně, což působí velmi příznivě na uživatelský prožitek z těchto zařízení. Tyto postupy zajišťují unikátní ovladatelnost systému, které v současné chvíli ani Google se systémem Android, ani Microsoft s Windows Mobile nedokážou konkurovat. Tyto systémy poměrně často nepochopí gesto uživatele, nebo neprovedou klik ani v případě, že vedle nepřesně stisknutého prvku není žádný jiný, což vede k deprivaci uživatel.



http://static.ddmcdn.com/gif/ipod-touch-touch-sensing.gif

Obrázek 1 - Princip funkce dotykového displeje u zařízení s iOS

V první řadě dojde k zachycení surových dat ze senzoru. Ta jsou však pro další zpracování nepoužitelná - mohou zasahovat na více ovládacích prvků a jsou neostrá. Dojde tak k vyčištění signálu od náhodných ruchů a nepravděpodobných přesahů a pomocí histogramů je nalezena plocha s největší intenzitou dotyku. Ta bývá obvykle stále poměrně velká, je už ale jen zlomkem plochy v surových datech a lze tak přistoupit k detekci ovládacího prvku, který chtěl uživatel stisknout.

Pokud plocha zasahuje více ovládacích prvků stejnoměrně, vyčkává systém na změnu dotyku (např.: "převalením" prstu na nějakou stranu), v případě převahy plochy na nějakém prvku je pak zpráva o dotyku předána tomuto ovládacímu prvku. Inteligentní přístup navíc, oproti konkurenci, zajišťuje částečnou toleranci chyb uživatele, takže v případě osamoceného ovládacího prvku je tolerance výrazně větší a ke stisku dochází při dotyku v rámci malé tolerance.

Zvláštní problematikou jsou také gesta, která displeje zařízení firmy Apple rozeznávají celou řadu včetně několika-dotykových (až 5 dotyků najednou). U těch je detekce pohybu velmi důležitá a postup je obohacen o vyrovnávání pohybu do jednodušších křivek a přímek. Tím je možno pomocí algoritmů na rozpoznávání obrazu odhadnout uživatelův záměr a adekvátně na něj reagovat.

4.2.iOS Human Interface Guidelines

Dokument iOS *Human Interface Guidelines* firmy Apple je obsáhlý dokumentem, pojednávající o zásadách návrhu uživatelského rozhraní pro zařízení s operačním systémem iOS - iPhone, iPod Touch a iPad. Kapitola čerpá z (Apple Inc., 2012) a je doplněna praktickými zkušenostmi z vývoje aplikací pro platformu iOS.

4.2.1. Zaměření na platformu

Cílem programátora je vytvořit aplikaci, se kterou se bude uživateli dobře pracovat. Pro to je nezbytné, aby se aplikace chovala tak, jak je uživatel dané platformy zvyklý - důležitá je homogenita vizuálu prvků uživatelského rozhraní, gest, animací a dalších prvků komunikace s uživatelem.

Základem návrhu uživatelského rozhraní je tak odpověď na tyto otázky:

- Je aplikace kompatibilní se standardy systému iOS?
- Používá správně standardní ovládací prvky?
- Užívá aplikace možnosti systému a zařízení tak, jak bylo zamýšleno při jeho návrhu?

Teprve když je odpověď na všechny otázky ANO, lze pokračovat s vývojem aplikace. Apple sice při schvalování aplikaci nemusí zamítnout, vážné prohřešky se ale projeví na množství uživatelů a jejich hodnocení.

Dotyková zařízení firmy Apple se ovládají čistě gesty na displeji, je tak klíčové dodržet následující:

- Velikost ovládacího prvku musí být minimálně 44 x 44 bodů,
- uživatelské rozhraní musí mít jednotný vizuální styl,
- uživatel se zaměřuje na obsah, ne na ovládání.

iPhone a iPod Touch mají rozlišení 320 x 480 bodů, iPad 768 x 1024. Potřeba je rozlišovat i termín pixel a bod. na iPhone, iPhone 3G, iPhone 3GS, iPod Touch a iPadu 1 a 2 platí, že jeden bod je roven jednomu pixelu. Na iPhone 4/4S a novém iPadu z roku 2012, které mají Retina displej, má jeden obrazový bod velikost 2 x 2 pixely. Aplikace tak jsou kompatibilní, ale grafika je vykreslena jemněji a vypadá lépe.

Dalším důležitým faktorem vývoje aplikací pro mobilní zařízení je fakt, že uživatel může změnit orientaci zařízení a aplikace i systém by na to měly vhodně reagovat. Na místě tak je přesun ovládacích prvků, změna jejich velikosti, či celková přeměna uživatelského rozhraní na efektivnější formát.

Nelze opomenout ani fakt, že oproti webovým a desktopovým aplikacím mají mobilní aplikace pouze jedno okno. Je tak nutno zvažovat, co skutečně uživatel potřebuje vidět a co je vhodnější umístit do jiného pohledu. Komplikované uživatelské rozhraní většinou znamená, že uživatelé aplikaci nechápou a nepoužívají.

Vhodným postupem tak je specifikovat si nezbytné funkce, které aplikace musí mít pro splnění svého účelu, dále pak cílovou skupinu a prostředí, ve kterém se budou při jejím užívání pohybovat - aplikace, pracující s navigací, kterou budou uživatelé provozovat na horském kole, musí mít větší uživatelské prvky, než aplikace na čtení novinek. Seznam funkcí by měl odpovídat jen nezbytnému minimu a aplikace by měla být jednoúčelová, pokud to neubližuje jejímu účelu. Specializované aplikace bývají kvalitnější a umožňují uživateli vybrat si na každou úlohu to nejlepší, co je v nabídce.

Pro univerzální aplikace pro iPhone/iPod i iPad je pak doporučeno následující:

- Pečlivě ladit uživatelské rozhraní pro každé zařízení zvlášť. Co je na iPadu užitečné, může být na iPhone ku škodě. Je lepší uživatelská rozhraní navrhnout pro každé zařízení od nuly, než se snažit jedno převést na druhé. Úzkost iOS ekosystému dovoluje vytvářet rozhraní skutečně pečlivě a na míru.
- Pro každé zařízení vyrobit grafiku na míru - uživatelé iPhone jsou zvyklí na menší míru detailnosti grafiky, než uživatelé iPadu. Prosté škálování menší grafiky na iPad rozhodně není dobrým řešením.
- Zachovat primární určení aplikace. I když iPad může nabízet větší kontrolu nad daty, zpracovávanými v aplikaci, nesmí mít uživatelé pocit, že se jedná o dvě zcela odlišné aplikace.
- Nespokojit se s prostou kompatibilitou a využít specifik každého ze zařízení - iPad nabízí několik ovládacích prvků, které dokáží využít větší plochu displeje lépe, než prosté naškálování aplikace z iPhone.

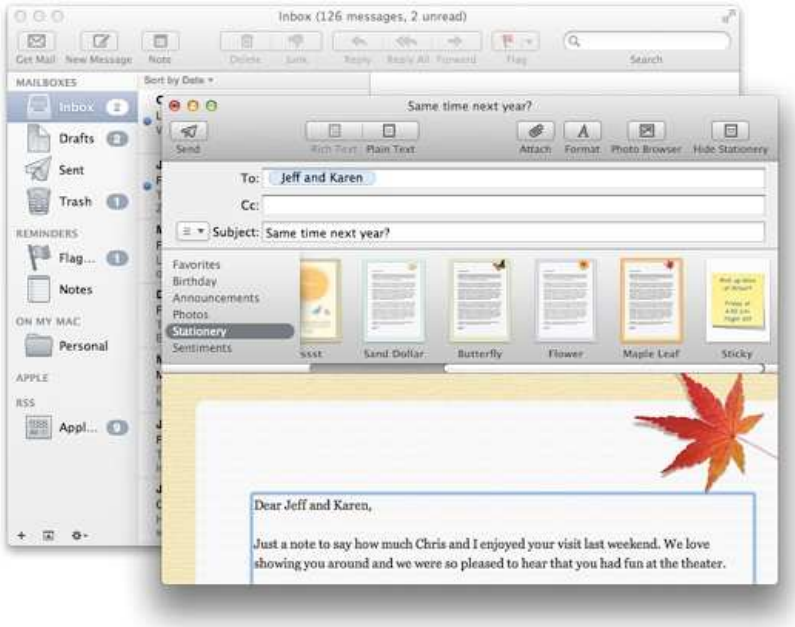
4.2.2. Přechod na iOS z jiné platformy

Při přechodu z desktopových a webových aplikací je nutno brát v potaz omezené rozlišení a jiné způsoby ovládání, jak bylo zmíněno v předchozí kapitole. Specifika jdou ale dále:

- Platí pravidlo 80-20: 80% uživatelů aplikace potřebuje pouze základní funkce aplikace, pouze relativně malá část (typicky pod 20%) využije všechny možnosti desktopové aplikace. Při návrhu mobilní aplikace je tak potřeba zaměřit se na většinu uživatelů - 80%.
- Uživatelé mobilních aplikací se pohybují v prostředích s velkým počtem ruchů, hůře se soustředí. Neměli by být nuceni hledat ovládací prvky a informace, které je zajímají.
- Uživatelé očekávají jiné informace prezentované jiným způsobem a nevadí jim jiné ovládání.

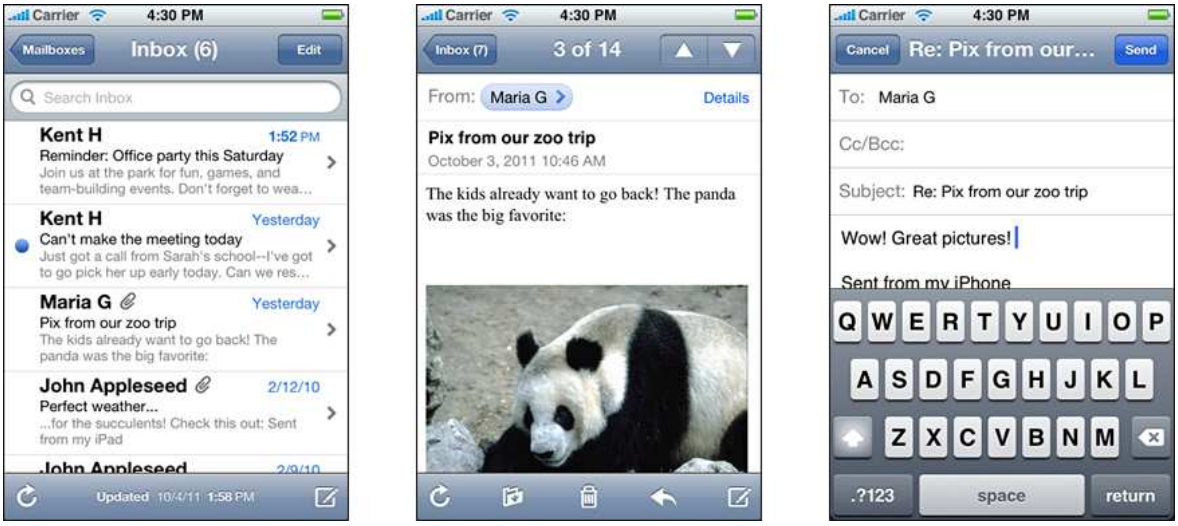
Dobrým příkladem konverze desktopové aplikace je Mail na systému MacOS X:

- Konverze na iPhone si vyžádala omezení na jedno otevřené okno, což vedlo k využití navigačního pruhu v hlavičce aplikace a panelu s nástroji v patičce.
- Průchod emailovými schránkami a detailem zprávy je tak hierarchická struktura tabulek a rámců s jasnou navigací. Každé okno tak řeší jeden konkrétní úkol, čímž zjednodušuje uživatelské rozhraní a zvyšuje přehlednost.
- Akce jsou zobrazeny pomocí piktogramů - na první pohled je zřejmé, které tlačítko provede jakou akci, potencionálně destruktivní akce vyžadují ověření.



https://developer.apple.com/library/ios/DOCUMENTATION/UserExperience/Conceptual/MobileHIG/Art/ds_mailondesktop.jpg

Obrázek 2 - Desktopové provedení aplikace Mail

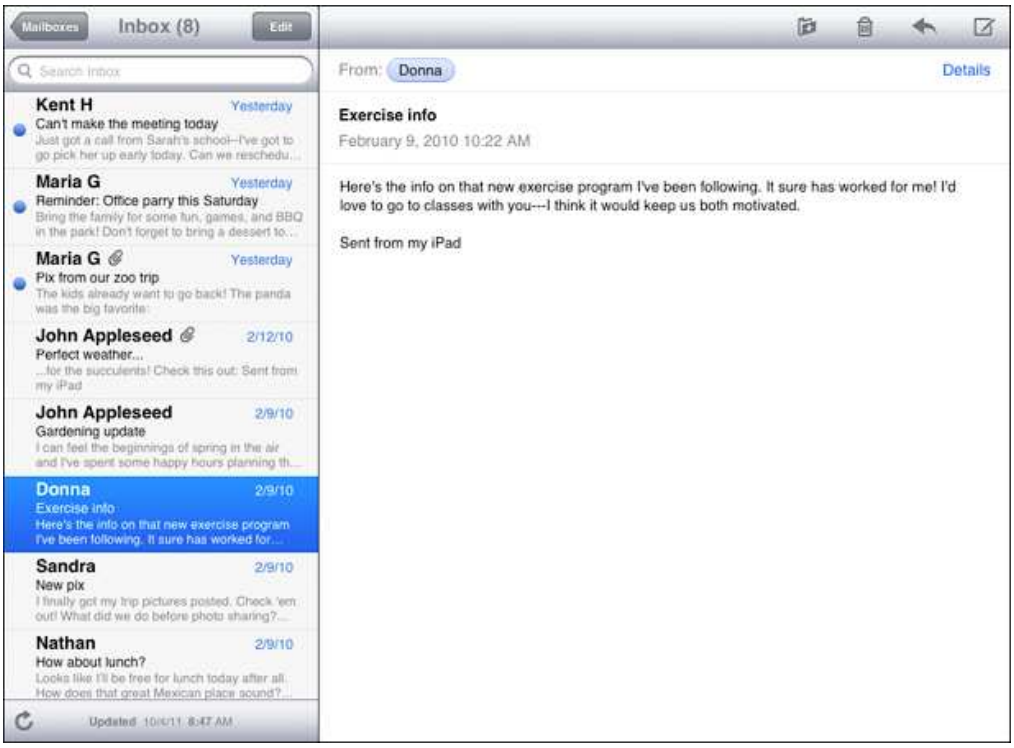


https://developer.apple.com/library/ios/DOCUMENTATION/UserExperience/Conceptual/MobileHIG/Art/ds_mailscreens.png

Obrázek 3 - iOS provedení aplikace Mail na iPhone

Při vydání iPadu bylo nutno vytvořit druhou konverzi - vyšší rozlišení iPadu dovoluje zobrazit více informací najednou a jinak pracovat s obsahem. Klíčové změny jsou tak tyto:

- Aplikace reaguje na otočení zařízení, při obou polohách je dostatek prostoru pro zobrazení obsahu,
- znatelné rozšíření rámce pro editaci zprávy, uživatel tak stále vidí hlavně to, co ho zajímá,
- plošší architektura uživatelského rozhraní, zpřístupňující přímočařejší přístup k datům,
- využití nových ovládacích prvků, jako jsou *Popover*[4] dialogy a *modální panely*[5] (např. pro editaci zprávy).



https://developer.apple.com/library/ios/DOCUMENTATION/UserExperience/Conceptual/MobileHIG/Art/land_mail_orientation.png

Obrázek 4 - iOS provedení aplikace Mail na iPadu

[4] Dialog, který se objeví jako plovoucí rámec nad obsahem se šipkou vycházející z ovládacího prvku, který jeho zobrazení vyvolal. Slouží pro rychlou volbu hodnot z předem daného seznamu.

[5] Modální panel blokuje další běh aplikace a vyžaduje akci uživatele.

4.2.3. Cíle uživatelského rozhraní

- Zaměření na primární úlohu - rozhraní musí obsahovat jen to nejdůležitější pro splnění svého účelu.
- Zaměření na obsah - pokud uživatel tvoří obsah, je rozumné skrýt ovládací prvky, které nepotřebuje a efektivněji tak využít omezený prostor kapesního zařízení.
- Uvažovat odshora dolů - nejdůležitější ovládací prvky a informace by měly být ihned na očích.
- Průchod aplikací by měl být logický a ucelený vzhledem k datům - uživatel by neměl být nucen přeskakovat z jedné částí aplikace do druhé, naopak by měl být veden po jasně dané lince.
- Jasná primární funkce - v každém rámci aplikace musí být na první pohled jasně patrné, co zde má uživatel za úkol a k čemu slouží.
- Popisky a vysvětlivky musí být zaměřeny na běžného uživatele - nemá smysl zahltit uživatele technickými detaily tam, kde nejsou potřeba a nejsou klíčové pro běh aplikace.
- Maximálně odstínit uživatele od práce se soubory - na zařízeních s iOS není k dispozici souborový systém tak, jak je zvyklý uživatel desktopu (každá aplikace má svůj oddělený prostor, do kterého ostatní aplikace nemají přístup). Uživatel by tak neměl být zatěžován volbou umístění souborů, nebo uživatelskými právy. Soubory by měly být uloženy vzdáleně v *iCloudu*[6].
- Sdílení a spolupráce - uživatelé by měli být, tam kde to je žádoucí, zapojeni do s dalšími uživateli. Jde zejména o spolupráci na tvorbě obsahu, nebo soupeření ve hrách pomocí globálních žebříčků.
- Zjednodušení, nebo odstranění nastavení aplikace - aplikace by se už ve výchozím stavu měla chovat tak, jak bude potřebovat naprostá většina uživatelů.
- Citlivý branding - aplikace by neměla být reklamní plochou, ve které loga firmy zabírají místo pro obsah. Uživatelé chtějí pracovat s obsahem, nebo se zabavit, ne obdivovat logotyp.

[6] Řešení společnosti Apple pro synchronizaci uživatelských dat - veškerý vytvořený obsah je synchronizován se vzdáleným úložištěm a odtud je distribuován do všech zařízení uživatele.

- Rychlé a kvalitní vyhledávání - pokud aplikace obsahuje velké množství dat, je žádoucí připravit si indexaci předem a nečekat na vyvolání akce vyhledávání. Aplikace pak může velmi snadno způsobit negativní první dojem a dostat špatné hodnocení. Pokud je vyhledávání časově náročné, je vhodné zobrazit animaci načítání a informovat tak uživatele o této skutečnosti.
- Stručné, kvalitní a jasné popisky - aplikace nesmí obsahovat překlepy, gramatické a jiné chyby v textech, jinak může vyvolat negativní první dojem. Uživatel musí z textů v aplikaci přesně a rychle pochopit, co se od něj vyžaduje.
- Konzistentní dojem a funkčnost uživatelského rozhraní - je nezbytné využívat ikony ve stále stejném významu a na stejných místech. Opak vede k negativnímu dojmu z aplikace.
- Realistický vzhled - pokud aplikace kopíruje vzhled objektu z reálného prostředí, působí méně uměle a pomáhá uživateli pochopit své funkce názorným příkladem. Textura kvalitního materiálu může výrazně zvýšit uživatelský požitek z užívání aplikace i přes omezenější funkce.
- Reakce na změnu polohy - pokud to obsah dovoluje, měla by aplikace reagovat na změnu polohy zařízení přizpůsobením svého vzhledu a rozložení uživatelského rozhraní.
- Ovládací prvky musí být ovladatelné - nejmenší plocha, reagující na stisk, by měla být velká 44 x 44 bodů a měla by být adekvátní svému účelu, důležitosti a prostředí, v němž bude aplikace užívána.
- Animace při přechodech a změnách v aplikaci - každá změna v aplikaci by měla být doprovázena animací, která podpoří právě provedenou akci. Využití animací je velmi žádoucí i pro prodloužení času mezi přechody a získání času pro výpočetně náročné operace.
- Podpora gest - aplikace by měla podporovat gesta, která jsou přirozená práci s konkrétním obsahem tak, jak je zvykem pro systém iOS.
- Automatické ukládání - uživatel by neměl být rušen žádostí o uložení, jedinou akcí, vyžadující explicitní potvrzení je zrušení a smazání.

- Modální okna - modální okna by měla být využívána pouze tam, kde je kritické získat uživatelský vstup pro další pokračování. Pokud může aplikace pokračovat sama, neměla by uživateli blokovat další práci.
- Okamžitý start - aplikace by neměla uživatele po spuštění obtěžovat akcemi, které si nevyžádal a naopak by mu měla okamžitě zpřístupnit obsah, kvůli kterému ji zapnul. Měla by také obsahovat prvky úměrně poslednímu stavu a aktuální orientaci zařízení.
- Neustále konzistentní stav dat - ukládání dat by mělo probíhat neustále tak, aby při požadavku na ukončení aplikace nemusela proběhnout časově náročná aplikace.
- Samovolné ukončení aplikace - aplikace by nikdy neměla ukončit svůj běh sama od sebe, pokud nevolá jinou aplikaci. Působí to negativním dojmem na uživatele.
- Rozhraní iPadu - při vytváření iPad konverze aplikace je žádoucí zkvalitnit rozhraní a nepřidávat funkce jen proto, že je na ně místo.
- Celoobrazovkové animace - animace při změně stavu prvku by měly probíhat pouze na něm a nerušit uživatele při přechodu na další činnost.
- *Popover* - *popover* dialog by měl být použit tam, kde se nastavují parametry jiného prvku.

4.2.4. Technologické cíle

- *iCloud* - od iOS 5 by měla být datově větší uživatelská data ukládána do *iCloudu*. Je však nezbytné zajistit korektní chování v případě absence přístupu k službě.
- Multitasking - aplikace musí být připravena na přerušení hovorem, nebo další aktivitou. Návrat do aplikace musí být do stejného místa, kde ji uživatel opustil.
- Centrum upozornění - je žádoucí umožnit uživateli nastavení vzhledu, četnosti a obsahu upozornění.
- Tisk - nastavení tisku z aplikace musí být přehledné a jednoznačné.

- Zvuky - pokud aplikace produkuje zvuky, mělo by být toto chování nastavitelné. Není vhodné nutit uživatele ztlumit celé zařízení kvůli několika nechtěným zvukům z aplikace.
- Editační menu - editační menu s možností kopírování textu by mělo být povoleno všude tam, kde je logicky využitelné. Uživatelé často chtějí přenést informace z aplikace do aplikace. Stejně tak akce Zpět a Vpřed musí být dostupné tam, kde je uživatelská aktivita potencionálně nechtěná.
- Klávesnice a uživatelský vstup - pro různé typy obsahu je rozumné zobrazit různé klávesnice. Pokud je uživateli předložen vlastní pohled, pracující s textem (např. tokenizér), je nezbytné zpracovat ho tak, aby uživatel na první pohled pochopil princip jeho funkce.
- Geolokace - geolokace musí reagovat na uživatelské akce, musí být zřejmé, proč má uživatel poskytnout informace o své poloze. Potvrzení po startu aplikace by mělo být pouze tam, kde je primárním úkolem aplikace práce s GPS souřadnicemi.

4.2.5. Grafika v aplikaci

- Ikony aplikace - rozlišení aplikačních ikon je 57 x 57 bodů pro původní displej, 117 x 117 bodů pro retina displej. Není třeba vytvářet kulaté aplikace, úprava ikony se provádí automaticky.
- Retina displej - je doporučeno vytvářet grafiku zvlášť pro retina displej tak, aby dodržovala původní layout, ale obsahovala detailnější textury a větší hloubku. Uživatelé těchto zařízení očekávají detailnější grafiku.
- Úvodní obrázek aplikace - pro zlepšení dojmu z prvního načtení aplikace je doporučeno zobrazit do inicializace rozhraní statický obrázek, identický s prvním oknem aplikace. Problémem je absence lokalizace a nutnost detekovat polohu zařízení.

4.3. Jazyk Objective-C

4.3.1. Historie jazyka

Historie jazyka *Objective-C* sahá do 80. let 20. století k Bradovi J. Coxovi, který při jeho tvorbě vycházel z jazyka *SmallTalk-80*. Šlo o vrstvu nad jazykem C, která rozšiřovala jeho možnosti zejména v oblasti tvorby objektů a práce s nimi.

V roce 1988 si tento jazyk licencovala společnost NeXT Software a vyvinula knihovny a prostředí s názvem *NEXTSTEP*. Ten byl o čtyři roky později začleněn do prostředí Free Software Foundation's GNU pod názvem *GNUStep*. Dostal se tak pod licenci GNU, která umožňuje jeho svobodné užívání, modifikování a rozšiřování. V Roce 1994 byla pak ve spolupráci NeXT Software a Sun Microsystem navržena standardizovaná specifikace pod názvem *OPENSTEP*.

Na konci prosince 1996 byla společnost NeXT Software převzata společností Apple Computer (dále jen Apple) a *OPENSTEP* modifikuje na nový balík *Cocoa*. Ten se stal základem pro nový operační systém AppleOS X společně s nástroji pro tvorbu aplikací *Project Builder* a *Interface builder*.

V roce 2007 byl pak jazyk povýšen na verzi 2.0, což mu přineslo srozumitelnější syntaxi, tisíce nových *API*[7] pro komunikaci novým hardware, jakými byly *GPS*[8], akcelerometr,kompas, moderní grafické karty, atd. (KOCHAN, 2010)

S příchodem iPhone v roce 2007 se dostala společnost Apple do sporu s vývojáři, neboť přáním Steva Jobs bylo vytvořit dokonalé a funkční zařízení. Měl pocit, že by neschopní vývojáři toto zařízení ničili nekvalitními aplikacemi. Teprve po čase bylo uvolněn nástroj *Xcode* (dříve *Project Builder*) s podporou iOS (operační systém mobilních zařízení Apple) a vývojářům byl tak poskytnut kvalitní nástroj, který pro vývoj aplikací používal sám Apple. Svůj podíl kontroly si však Apple zachoval nutností instalace aplikací přes *AppStore* (centralizovaný obchod s aplikacemi pro iOS), kde veškeré aplikace vývojářů prochází podrobnou kontrolou a analýzou a jsou často pro sebemenší pochybení zamítnuty bez dalšího odůvodnění. (KAHNEY, 2009)

[7] Application Programming Interface - aplikační rozhraní pro práci s jinou knihovnou/hardwarem.
[8] Global Positioning System - celosvětový systém pro určení polohy uživatele (respektive zařízení).

4.3.2. Syntaxe jazyka

Syntaxe jazyka *Objective-C* je kombinací jazyků C/C++ a *SmallTalk-80*, ze kterého vychází. Obsahuje tak základní funkci `main()`, která je mateřskou třídou každého programu v C a zároveň volání zpráv na objektech uvnitř hranatých závorek, jako jazyk *SmallTalk-80*. Příkladem budiž tradiční příklad `Hello World!`, který se standardně používá pro výuku programování. (KOCHAN, 2010)

```
#import <Foundation/Foundation.h>

int main (int argc, const char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init]

    NSLog(@"Hello World!");

     [pool drain]
    return 0;
}
```

V této ukázce dochází k importu hlavičkového souboru frameworku *Foundation*, který se postará o načtení dalších potřebných souborů pro práci s *Objective-C*. Dále je zde definice metody `main()`, která má návratovou hodnotu typu `int` a přebírá dva parametry - `argc` typu `int` a `argv`, což je ukazatel na pole prvků typu `char`. v Metodě `main()` dochází k alokaci a inicializaci automaticky uvolňovaného zásobníku, zalogování zprávy "`Hello World!`" pomocí volání `NSLog`, uvolnění zásobníku voláním zprávy `drain` na objektu `pool` a ukončení programu vrácením hodnoty 0 - ta značí normální ukončení programu, jakákoli nenulová hodnota by byla vyhodnocena jako chyba. Automaticky uvolňovaný zásobník slouží pro správu paměti v prostředí s počítáním odkazů a bude dále rozebrán v kapitole Správa paměti.

4.3.3. Preprocesor

Preprocesor nabízí nástroje pro psaní čitelných programů, definici programových maker a konstant a přizpůsobení jazyka svým zvyklostem (není doporučeno). Všechna makra se zapisují se znakem # na začátku a jsou provedena při procesu sestavování programu do binární aplikace.

Makro `#define` umožňuje definici konstant

```
#define PI 3.141592654
#define TRUE 1
#define FALSE 0
#define kMenuRowCount 8
#define kSectionTitle @"Awesome section title"
```

a jednoduchých i složitějších výpočtů a příkazů.

```
#define DVE_PI PI * 2.0
#define JE_SUDE num % 2 == 0
#define NA_DRUHOU(x) ((x) * (x))
#define MAX(a, b) (((a) > (b)) ? (a) : (b))
#define tiskint(x) printf (# x " = %i\n", x)
```

Všechna tato makra slouží pro zjednodušení a tím i zpřehlednění kódu, koncentraci konstant, *magic-numbers*[9] a základních výpočtů na jedno místo. Jejich použití je silně doporučeno, po delší době bývá velice obtížné vyznat se v kódu i pro samotného autora a definice srozumitelných a zapamatovatelných názvů napomáhá k udržitelnosti zdrojových kódů aplikace. Při definici vlastních názvu je doporučeno používat názvy s "k" prefixem znázorňující konstantu, nebo psát identifikátor kompletně kapitálkami. Na první pohled pak další programátor pozná, že se jedná o konstantu preprocesoru. Obecně by konfigurace programu měla být na jednom místě - v definovaných konstantách v hlavičce souboru, nebo inicializaci nějaké konfigurační třídy. Pokud jsou různé hodnoty rozesety napříč kódem (např. ve hře bodové hodnocení za jednotlivé akce), bývá takový kód po delší době zdrojem chyb.

[9] "kouzelná čísla" - číselné konstanty, které ovlivňují běh programu a mohou se vyskytovat kdekoli napříč kódem.

Dalším makrem preprocesoru je `#import`. Import byl zařazen už v první ukázce syntaxe v příkladu "`Hello World!`", kde sloužil pro načtení hlavičkových souborů frameworku *Foundation*. Toto makro je při zpracování nahrazeno obsahem uvedeného souboru a ten se tak stává součástí zdrojových kódů programátora. Jedinou zvláštností je ohraničení názvu souboru dvěma uvozovkami, nebo ostrými závorkami (<>). Závorky se užívají u systémových hlavičkových souborů, které jsou součástí frameworku, uvozovky jsou užity u vlastních kódů programátora. Linker tak pozná, kde má uvedené hlavičkové soubory hledat. Tato syntaxe je poděděná z jazyků C a C++.

Posledním makrem, nebo spíš sadou maker, jsou makra pro podmíněnou kompilaci programu. Patří do ní tato: `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif` a `#endif`. Tato makra slouží zpravidla pro pokrytí rozdílů mezi operačními systémy, případně verzemi knihoven. Díky tomu je kód schopen zkompilovat a spustit i jiný programátor na jiném stroji - lze doplnit definice chybějících konstant, načíst jiné hlavičkové soubory, doplnit jednoduché funkce a datové typy například na základě detekce procesoru. U MacOS se toto hojně využívalo s přechodem z procesorů *PowerPC* na procesory Intel, kde byly značné rozdíly v architektuře procesoru a bylo nutno načítat rozdílné verze frameworku a dalších knihoven. `#if` uvozuje začátek podmíněného bloku, `#ifdef` je zkrácenou verzí `#if defined`, tedy kontroly definice (existence) makra preprocesoru v některém z načtených souborů. Tím se zpravidla zajišťuje unikátní načtení souboru a zabraňuje se chybě vícenásobným načtením. `#ifndef` je negací `#ifdef`, zjišťuje tedy naopak neexistenci makra. `#else` a `#elif` jsou výhybkou při neplatnosti výše zmíněných podmínek - `#elif` umožňuje navíc, na rozdíl od `#else`, zadat podmínku i pro tuto další větev. `#endif` pak označuje konec podmíněného bloku. Zvláštním makrem v této skupině je ještě `#undef`, pomocí kterého lze nadefinované makro nebo konstantu odstranit. Funguje však pouze dopředně, neovlivní tak již proběhnuté rozhodovací bloky. (KOCHAN, 2010)

4.3.4. Datové typy

Základní datové typy jazyka *Objective-C* vycházejí z jazyka C a vyšší objektové typy jsou pak stavěny na těchto jednoduchých typech (zejména strukturách). Všechny vyšší objektové typy jsou pak v jazyce *Objective-C* odkazovány pouze ukazateli, programátor je striktně oddělen od nízkoúrovňové reprezentace objektů v paměti a stará se pouze o členské proměnné objektů. V následující ukázce je vidět vytvoření objektů typu `NSString` a `UITableViewCell`, které jsou však pomocí svých názvů pouze odkazovány.

```
- (UITableViewCell *) tableView: (UITableView *) tableView cellForRowAtIndexPath:
(NSIndexPath *) indexPath {
    static NSString *identifier = @"MyCell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
identifier];

    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:
UITableViewCellStyleDefault reuseIdentifier: identifier] autorelease];
    }

    return cell;
}
```

Mezi základní typy patří: `int`, `float`, `double`, `char`. `Int`, `float` a `double` slouží k ukládání číselných hodnot. Pro `int` (a částečně i `double`) lze využít kvantifikátorů `short`, `long`, `long long`, `signed` a `unsigned`. `Signed` je výchozí hodnota a značí, že číslo umožňuje ukládat kladná i záporná čísla. `Unsigned` naopak záporná čísla obsahovat nemůže, naopak ale umožňuje využít bit, sloužící pro uložení znaménka, pro rozšíření (zdvojnásobení) rozsahu proměnné. `Long` říká, že daná proměnná bude obsahovat dlouhé číslo. Na většině systémů však `long int` i `int` mají stejný rozsah: 2^{31}-1, tedy 2 147 483 647. `Long long` už přepíná na dvojnásobnou přesnost, číslo je tak reprezentováno minimálně 64 bity. Tyto příznaky je nutno vést v patrnosti a při výpisu proměnných je uvádět spolu s identifikátorem datového typu v podobě prefixu `l`, nebo `ll`. `Signed`, ani `unsigned` žádný prefix nemají, nemění totiž délku datového typu. (KOCHAN, 2010)

```
NSLog(@"%lli", num);
```

Pro číselné datové typy existuje v *Objective-C* nadstavba objektem `NSNumber`. Jde o kompozitní (složený) objekt, který pro různé datové typy jazyka C volá různé metody a ukládá data do jiných úložišť. Metod pro vytvoření objektu tak je celá řada - `numberWithChar:`, `numberWithDouble:`, `numberWithFloat:`, `numberWithInt:`, `numberWithInteger:`, `numberWithLong:`, `numberWithLongLong:`, `numberWithShort:`, `numberWithUnsignedChar:`, `numberWithUnsignedInt:`, `numberWithUnsignedInteger:`, `numberWithUnsignedLong:`, `numberWithBool:`, `numberWithUnsignedLongLong:`, `numberWithUnsignedShort:` a jejich ekvivalenty s prefixem `initWith...` volané na alokované instanci objektu `NSNumber`. Nevýhodou tohoto objektu je nutnost pamatovat na datový typ, s nímž byl vytvořen, neboť při inicializaci s datovým typem `unsigned int` vrátí objekt na volání `float` hodnoty 0. To je způsobeno způsobem konstrukce objektu, kdy pro každý datový typ obsahuje oddělené úložiště a může vést k velmi obtížně odhalitelným chybám v kódu a v jednodušších programech je tak výhodnější využívat základní datové typy pro práci s čísly, případně typ `NSInteger`.

```
float perc = 28.3443f;
NSNumber *num = [NSNumber numberWithFloat:perc];
```

`NSInteger` je definován jednoduchým pojmenováním datového typu `int`, případně `long` pro 64-bitové systémy. (Apple Inc., 2008)

```
#if __LP64__ || NS_BUILD_32_LIKE_64
  typedef long NSInteger;
  typedef unsigned long NSUInteger;
#else
  typedef int NSInteger;
  typedef unsigned int NSUInteger;
#endif
```

Datový typ `char` slouží ukládání znaků v jazyce C. *Objective-C* naštěstí přináší lepší nástroje pro práci s řetězci v podobě objektu typu `NSString`.

Konstantní řetězce se vytváří speciální syntaxí @"...", která vrací neměnitelný řetězec. Tento typ řetězců je v systému optimalizován do té míry, že dva řetězce se stejným obsahem sdílí stejný blok v paměti i když o to programátor explicitně nežádá. Na tuto funkčnost ovšem nelze spoléhat, není garantována specifikací. Pro ověřování identity řetězců je nutno výhradně využívat metodu `compare:` vracející objekt typu `NSComparisonResult`. (Apple Inc., 2011)

```
NSString *str = @"Hello World!";
```

Pokud programátor potřebuje generovat řetězec programově na základě obsahu dalších objektů a proměnných, je mu k dispozici například metoda `stringWithFormat:` objektu `NSString`, která jako první parametr očekává řetězec popisující formát a v dalších parametrech objekty a proměnné, sloužící pro náhradu zástupných symbolů ve formátu. Metoda vrací objekt, který již dostal zprávu `autorelease` a není potřeba ho uvolňovat. Pokud je nezbytné vlastnictví objektu převzít, existuje alternativní metoda `initWithFormat:` volaná na alokované instanci třídy objektu `NSString`. (Apple Inc., 2011)

```
float perc = 28.3443f;
NSString *str = [NSString stringWithFormat:@"%.02f%%", perc];
```

Datovým typem, který přináší speciálně *Objective-C* je datový typ `id`, což je ukazatel na obecný objekt. Podrobněji bude popsán v kapitole Dynamické typování. (KOCHAN, 2010)

4.3.5. Výrazy

Objective-C, stejně jako jazyk C a většina ostatních, obsahuje základní matematické operátory a pracuje s nimi s obecnými matematickými prioritami. Má tak všechny běžné operace, jako +, -, *, /, % (modulo - celočíselné dělení). Potřeba je dávat pozor pouze na dělení v případě, že všechny operandy jsou celočíselného typu - i výsledek je pak celočíselný (např.: 5 / 2 = 2). Pro získání desetinného výsledku je nutné alespoň jeden operand zadat v (nebo převést na) datovém typu `float`. Aritmetické operátory je možno uvádět i ve zkrácené podobě:

```
int a = 5;
float b = 2.0;
a /= b; // a obsahuje hodnotu 2.5
```

Zvláštními výrazy jsou výrazy pro převody datových typů. Lze je využít pouze u kompatibilních typů, jako je převod `int` na `float` a `double` (a naopak), nebo z objektových například z `UIView` na `UIControl` - `UIControl` je podtřídou `UIView`, obsahuje tak vše z `UIView` a není problém provést přetypování nebo přiřazení. Přetypování se provádí uvedením požadovaného typu v závorkách před proměnnou:

```
int i = 5;
float f = (float) i; // f obsahuje hodnotu 5.0
UIControl *control = (UIControl *) [sender mainView]; // v promenne control je
ukazatel na objekt typu UIControl, prestoze byl vracen objekt typu UIView
```

Ve většině programů málo využívané, nicméně důležité, jsou i bitové operátory. Slouží pro práci s bitovými maskami a čísly na bitové úrovni. Umožňují snadné spočítání doplňku, či změnu operátoru. Do množiny patří tyto: & - bitové AND, | - bitové OR, ^ - bitový XOR, ~ - bitový komplement, >> - bitový posun o 1 bit doprava, << - bitový posun o jeden bit doleva. (KOCHAN, 2010)

```
short int w1 = 0x15; // 0001 0101
short int w2 = 0x0c; // 0000 1100
w3 = w1 & w2; // w3 obsahuje hodnotu 0x04 (0000 0100)
w3 <<= 1; // w3 obsahuje hodnotu 0x08 (0000 1000)
```

4.3.6. Smyčky a rozhodování

Během programování je velmi často nutné provádět nějaký blok kódu opakovaně a o počtu iterací rozhoduje až stav programu při jeho běhu u uživatele. Psát několikrát pod sebe stejný kód tak není efektivní a často ani možné. Pro tento případ jazyk *Objective-C* přejímá z jazyka C smyčky `for`, `while` a `do-while`.

`For` je smyčka s předem daným počtem iterací a podmínkou na začátku. Není u ní garantován ani jeden průchod za předpokladu, že vstupní podmínka není splněna ani v první iteraci. Nejčastěji se užívá na iteraci skrz pole se známou velikostí. Její syntaxe vypadá následovně:

```
int n;
for (n = 1; n <= 10; n++) {
    NSLog(@"%i. loop", n);
}
```

Syntaxe se skládá ze čtyř částí: inicializační příkaz (`n = 1`), podmínka (`n <= 10`), výraz smyčky (`n++`) a tělo cyklu (`NSLog(@"%i. pruchod smyckou", n)`). Inicializační příkaz se provede jen před prvním průchodem smyčkou, slouží k nastavení iterační proměnné. Podmínka se provádí před vstupem do každé jednotlivé iterace a rozhoduje o jejím provedení. Pokud je platná, je povolen vstup do iterace. Výraz smyčky se provádí na konci každého průchodu cyklem, před ověřením podmínky. Slouží k zvýšení/snížení hodnoty čítače a vstupu do další iterace. Tělo cyklu je pak jeden, nebo více příkazů, které se provádí v každé iteraci po dobu platnosti podmínky.

V těle cyklu se mohou objevit dva příkazy, které ovlivňují jeho průběh. Jde o `continue` a `break`. `Continue` přeruší aktuální iteraci a cyklus pokračuje další, `break` ukončuje aktuální iteraci i celý cyklus a to i v případě, že podmínka cyklu stále platí.

Dalším cyklem je cyklus `while`. `While` nemá předem známý počet iterací a, stejně jako cyklus `for`, nezaručuje ani jeden průchod smyčkou. Jeho syntaxe obsahuje pouze podmínku na začátku a tělo cyklu - neobsahuje inicializační sekci, ani výraz smyčky. Úprava podmínkového výrazu/proměnné musí probíhat v těle cyklu.

```
while (isValid) {
    isValid = [table numberOfRowsInSection: section];
}
```

Cyklus `while` je také možno ovlivňovat příkazy `continue` a `break`.

Posledním příkazem cyklů je `do`. Do se oproti předchozím dvěma garantuje minimálně jeden průchod cyklem z důvodu umístění podmínky na konci těla cyklu.

```
int num, actNum;
num = 159;
do {
    actNum = num % 10;
    NSLog(@"%i", actNum);
    num /= 10;
} while(num != 0);
```

S výjimkou podmínky na konci těla, se cyklus `do-while` nijak neliší od cyklu `while`. (KOCHAN, 2010)

Objective-C 2.0 přineslo ještě jednu novinku k cyklům a sice rychlou iteraci (*fast enumeration*) pomocí speciální podoby `for` cyklu:

```
MSArray *myArray = [NSArray arrayWithObjects:@"a", @"b", @"c", nil];
for (id myObject in myArray) {
    NSLog(@"%@", myObject);
}
```

Její primární užití je v iteraci poli a objekty, kde výrazně zvyšuje přehlednost kódu. Pro srovnání kód provádějící stejnou operaci bez rychlé iterace:

```
MSArray *myArray = [NSArray arrayWithObjects:@"a", @"b", @"c", nil];
NSEnumerator *enumerator = [myArray objectEnumerator];
id myObject;
while ((myObject = [enumerator nextObject]) != nil) {
    NSLog(@"%@", myObject);
}
```

4.3.7. Třídy

Implementace tříd se v *Objective-C* skládá ze dvou souborů - `<NazevTridy>.h` s deklarací rozhraní třídy a jejích metod a stejně pojmenovaný soubor s koncovkou `.m` s implementačními detaily - definicí třídy.

Příklad deklarace třídy - obsah souboru `MyClass.h`:

```
#import <Foundation/Foundation.h>

@interface MyClass : NSObject <NSCopying> {
    @private
    UIView *myView;
    @public
    UIImage *myImage;
}

@property (nonatomic, retain) UIImage *myImage;
- (void) myMethod1;
- (UIView *) myMethod2: (UITableView *) table;
- (IBAction) myMethod3: (id) sender;

@end
```

Na začátku hlavičkového souboru je umístěno načtení knihoven frameworku potřebných pro běh třídy pomocí makra preprocesoru `#import`. Není třeba se obávat násobného importu stejné knihovny, preprocesor se postará o pouze jediné vložení díky podmíněnému bloku uvnitř souboru - viz kapitola Preprocesor.

Pod importem hlaviček frameworku je deklarace interface třídy pomocí klíčového slova `@interface`. Z této deklarace lze vyčíst rodičovskou třídu (v tomto případě `NSObject`), volitelně deklaraci kategorie (v tomto případě chybí) a přijímané protokoly (v tomto případě `NSCopying`).

Uvnitř složených závorek jsou deklarovány členské (instanční) proměnné včetně svých typů. V *Objective-C* je možno deklarovat členské proměnné jako `@private` (soukromé pro třídu), `@protected` (soukromé pro třídu a podtřídy) a `@public` (veřejné). Přepínače fungují na bázi stavového automatu a lze je libovolně střídat. Výchozím stavem je `@protected`, tedy všechny proměnné jsou dostupné pouze třídě a jejím podtřídám.

Pokud chce programátor nějakou z členských proměnných zpřístupnit ostatním třídám, musí vytvořit metody, s touto proměnnou pracující, nebo využít syntaxi `@property`. Těmto metodám se říká `setter` a `getter`. Tím je kompileru řečeno, že k této členské proměnné mají mít přístup i další třídy a má pro ně vytvořit `setter` (metoda pro nastavení) a `getter` (metoda pro vrácení hodnoty) metody. Součástí makra `@property` jsou parametry, říkající jak se má v metodách pracovat s předávanými hodnotami. Prvním parametrem je nastavení `atomic`, nebo `nonatomic`. `Atomic` je nezbytný pro více-vláknové aplikace, při přiřazení jsou hodnoty uvolněny automaticky uvolňovaným zásobníkem a ne přímo, jako v případě klíčového slova `nonatomic` - ani na okamžik tak není hodnota prázdná. Navíc jsou v metodě použity mutexové[10] zámky, které se postarají o jediný přístup vlákna v této kritické sekci. `Nonatomic` je naopak rychlejší a paměťově efektivnější, u více-vláknové aplikace však hrozí pádem s chybou segmentace, nebo datovou nekonzistencí v objektu napříč vlákny. V druhém parametru lze zadat `retain`, `copy` a `assign`. `Assing` provádí pouhé přiřazení, což znamená nebezpečí v případě, kdy je předaná hodnota uvolněna mimo třídu - může dojít k chybě segmentace a pádu aplikace. `Retain` předané hodnotě posílá zprávu `retain`, třída se tak stává vlastníkem hodnoty. Stále ale existuje nebezpečí, že bude hodnota po přiřazení změněna, což ovlivní i členskou proměnnou třídy. Poslední možností je `copy` - členské proměnné přiřazuje mělkou kopii předané hodnoty, což je nejbezpečnější přístupem Při psaní programu je nutno pečlivě zvážit, který způsob generování metod je pro dané užití nejvýhodnější, při špatné definici mohou vznikat velmi obtížně odhalitelné chyby. Členským proměnným se dá předat hodnota i pomocí zkrácené tečkové syntaxe:

```
[object setMyData: data];
// lze zadat take jako
object.myData = data;
```

Tato syntaxe dovoluje psát čitelnější kód, neboť vícenásobné zanoření v hranatých závorkách velmi rychle způsobí nečitelnost kódu. Stejná syntaxe platí i pro `getter` metody:

[10] Zkratka z "Mutual exclusion" - vzájemně se vylučující zámek.

```
data = [object data];
// lze zadat jako
data = object.data;
```

Pod volitelnou sekcí `@property` jsou deklarace členských metod. Mínus (pomlčka) před metodou znamená, že jde o instanční metodu, plus by znamenalo metodu třídy, tedy volanou na předkovi. Toho lze výhodně užít pro lazy-loading [11] hodnot společných pro všechny instance třídy. Příkladem takové metody je `alloc`.

V závorce je návratový typ metody. Vracet lze jen jednu hodnotu, v případě potřeby vracet více hodnot je nutno vracet slovník `NSDictionary`, pole `NSArray`, množinu `NSSet`, nebo nějaký vlastní výčtový typ na nich založený. Na rozdíl od jiných jazyků, kde je název funkce jeden řetězec, obsahující alfanumerické znaky a podtržítko, je v *Objective-C* název funkce proložen názvy jednotlivých parametrů oddělených dvojtečkami Výhodou je, že při větším počtu parametrů funkce neztrácí přehlednost, nevýhodou je velmi dlouhý zápis, který může snižovat přehlednost kódu a nutí dělit volání funkcí na více řádků. Na první pohled tak *Objective-C* se svými dlouhými identifikátory metod vypadá vedle ostatních jazyků poměrně podivně a spoustu lidí odradí. Příkladem budiž nejdelší název funkce, který současná verze Objective-C obsahuje: `initWithBitmapDataPlanes:pixelsWide:pixelsHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bitmapFormat:bytesPerRow:bitsPerPixel:`

Zvláštním případem příkladu je metoda `myMethod3`, která má návratový typ `IBAction`. Typ `IBAction` je makrem preprocesoru, které se při kompilaci změní na `void`. Je využíván nástrojem *Interface Builder* k identifikaci metod, na které lze připojit akce z uživatelského rozhraní. Parametrem takové metody je vždy datový typ `id`, tedy ukazatel na obecný objekt. To umožňuje na jednu akci připojit libovolné prvky uživatelského rozhraní, nehledě na jejich typ (např. `UIButton` i `UIProgressBar`). Podobným případem je klíčové slovo `IBOutlet`, které je preprocesorem úplně odstraněno. Slouží také programu *Interface Builder*, tentokrát však pro identifikaci zásuvek pro prvky uživatelského rozhraní. (KOCHAN, 2010)

[11] Inicializace proběhne až v momentě, kdy je hodnota zapotřebí.

Definice třídy - obsah souboru `MyClass.m`:

```
#import "MyClass.h"

@implementation MyClass

    @synthesize myImage;

    - (void) myMethod1 {
        ...
    }

    - (UIView *) myMethod2: (UITableView *) table {
        ...
    }

    - (IBAction) myMethod3: (id) sender {
        ...
    }
@end
```

V implementaci je možné zopakovat jak rodičovskou třídu, tak protokol. Případná kategorie je povinná.

Makro `@synthesize` je doplňkem k `@property`, je preprocesorem nahrazeno `setter` a `getter` metodami dle specifikace v deklaračním souboru. `@synthesize` není nutno volat vícekrát, je možno zadat více členských proměnných oddělených čárkou.

Dále už následují pouze definice metod - vždy je uvedena celá hlavička metody z deklaračního souboru následovaná složenými závorkami s tělem metody.

Pro vyšší přehlednost kódu je možno používat makra #pragma preprocesoru, které následně *Xcode* v seznamu metod použije k vytvoření separátorů a nadpisů sekcí: (LAMARCHE, a další, 2010)

```
#pragma mark -
#pragma mark Some methods...
```

4.3.8. Dědičnost

Dědičností se označuje princip, při kterém se vytvoří třída, která má své instanční proměnné a metody odvozené z jiné třídy - dědí je. Takto definovaná nová třída implicitně obsahuje vše, co obsahovala rodičovská třída a je jí navíc umožněno definici rozšířit, či překrýt původní kód.

Příklad překryté metody `dealloc`:

```
- (void) dealloc {
     [nameLabel release]
     [super dealloc]
}
```

U metody `dealloc` je nezbytné uvolnit (ale ne dealokovat) všechny vlastněné členské proměnné (jinak by došlo k úniku paměti) a následně zavolat metodu `dealloc` rodičovské třídy, která se postará o uvolnění instančních proměnných mateřské třídy.

Pokud je třeba zavolat metodu v rodičovské třídě, volá se přes klíčové slovo `super`. Volá-li se metoda, systém hledá v hierarchii směrem vzhůru. První třída, která metodu implementuje, je zavolána. Stejný princip je využit u příjemců zpráv v uživatelském rozhraní. V případě, kdy metoda není v hierarchii nalezena, je zobrazena následující chybová hláška: (KOCHAN, 2010)

```
warning: 'MyTrida' may not respond to '-myMethod'
```

Pokud je nutné zavolat metodu na objektu samotném, nebo ho vrátit jako delegáta nějaké akce, lze to zařídit pomocí klíčového slova `self`:

```
NSAlert *alert = [[[NSAlert alloc] init] autorelease];
[alert addButtonWithTitle: @"OK"];
[alert addButtonWithTitle: @"Cancel"];
[alert setMessageText: @"Delete the record?"];
[alert setInformativeText: @"Deleted records cannot be restored."];
[alert setAlertStyle: NSWarningAlertStyle];
[alert beginSheetModalForWindow: [searchField window] modalDelegate: self
didEndSelector: @selector(alertDidEnd:returnCode:contextInfo:) contextInfo: nil];
```

4.3.9. Polymorfizmus

Polymorfizmus se v programování užívá ve významu vlastnosti jazyka, která dovoluje:

- přetěžování operátorů metod, tedy možnost volat metodu s různými datovými typy parametrů,
- přetěžování operátorů, kdy se jeden operátor chová různě k proměnným různých datových typů,
- volání stejně pojmenovaných metod na různých objektech.

Polymorfizmus v *Objective-C* je omezen pouze na volání stejně pojmenované metody v rámci kontextu různých tříd. Parametrický polymorfizmus (možnost volat jednu metodu s různými parametry), ani přetěžování operátorů zde neexistuje. Obě chybějící možnosti byly vyhodnoceny jako příliš nebezpečné pro rozumnou udržitelnost kódu a nebyly zahrnuty do jazyka.

Volba metody k zavolání se děje v rámci běhového modulu podle typu objektu, na kterém je zavolána. Při vyvolání selektoru je ověřeno, zda sama třída objektu, nebo některá rodičovská třída obsahuje implementaci volané metody. Je tak vyloučeno, že by nastala chyba vyvoláním nekompatibilní metody. (KOCHAN, 2010)

4.3.10. Dynamické typování

Dynamické typování je v jazyce *Objective-C* řešeno pomocí datového typu `id`. Stručně zmíněn byl již v kapitole Datové typy.

Typ `id` je využit jako návratová hodnota metod, kde se počítá s překrytím (např. `alloc` a `init`) a není tak možné odhadnout, jaký datový typ se bude ve skutečnosti vracet. Druhé užití je jako typ parametru u metod `IBAction` uživatelského rozhraní v podobě proměnné `sender`. Zde není jasné, jaký ovládací prvek metodu zavolá, je tak nutno zadat obecný typ.

Definice datového typu `id` je zanesena v souboru `objc.h`. Typ je vytvořen jako ukazatel na obecnou třídu jazyka *Objective-C*.

```
typedef struct objc_class *Class;
typedef struct objc_object {
    Class isa;
} *id;
```

Pohodlným řešením by bylo všechny objektové proměnné v aplikaci deklarovat jako typ `id`, nebezpečí se však skrývá v rozdílu kontroly kódu při kompilaci a až za běhu. Pokud programátor užívá správné objektové typy proměnných a návratových hodnot, může kompiler zkontrolovat kód na syntaktickou správnost již při sestavování aplikace a ušetřit tak uživatele mnoha pádů na chybách, které lze odhalit až za běhu - příkladem může být volání neexistující metody na objektu, která způsobí chybu a pád aplikace, případně přiřazení nekompatibilního parametru funkci. Typ `id` by tak měl být využíván pouze tam, kde opravdu není znám typ proměnné a kde je dostatek kontroly na kompatibilitu proměnné před dalším zpracováním. (KOCHAN, 2010)

```
objc: MyClass: does not recognize selector -myMethod:
```

4.3.11. Výjimky

Problém s datovým typem id je možno řešit ex-post použitím odchytávání výjimek. Není to ale dobrý programátorský zvyk. Zachytávání výjimek by mělo být použito jen u potencionálně nebezpečných bloků, kde lze očekávat chybu neovlivnitelnou v průběhu návrhu aplikace. Příkladem budiž navázání HTTP spojení a stažení souboru - nelze předem vědět, zda bude server dostupný a zda na něm bude soubor stále umístěn.

Pro odchycení výjimek se užívá makra `@try-@catch-@finally`.

```
@try {
    ...
} @catch (NSException *ex) {
    NSLog(@"Exception @%@%", [ex name], [ex reason]);
} @finally {
    ...
}
```

Mezi složenými závorkami za makrem `@try` je potencionálně nebezpečný kód. Může to být cokoli - série příkazů, nebo jediné volání metody na nějakém objektu. Výhodou jazyka Objective-C je, že volání libovolné zprávy na objektu typu `nil` nevyvolá výjimku. Umožňuje to psát jednodušší metody bez zbytečných ověřování existence hodnoty. Velkou výhodou je to například u metody `dealloc`, kde se volá metoda `release` na všech členských proměnných a je jedno, že již některé prvky mohly být uvolněny.

V případě, že je uvnitř bloku vyvolána výjimka, provedou se příkazy uvnitř bloku za makrem `@catch`.

Pokud je přítomen blok `@finally`, provede se pokaždé nehledě na vyvolání výjimky nebo správný průběh. Zde by mělo být umístěno vyčištění proměnných, apod. (KOCHAN, 2010)

4.3.12. Kategorie

Kategorie jsou velmi příjemnou funkcí v *Objective-C*, která dovoluje lépe dokumentovat a rozšiřovat definice objektů (modularizace) i bez přístupu k původnímu kódu třídy a bez nutnosti využívat podtřídy. Využít je lze například při práci ve velkém týmu, kde část týmu pracuje na implementaci cachování zdrojů, část na inicializaci třídy a vytvoření struktury dat a část na zpracovávání dat. Dalším využitím je rozšíření základní třídy frameworku *Foundation* o metody, které nezbytně potřebujeme ve své aplikaci bez nutnosti vytvářet podtřídu, která by mohla vést k nekompatibilitě v dalších částech kódu. (KOCHAN, 2010)

Velmi kvalitním a často užívaným příkladem využití kategorií a rozšíření třídy jazyka *Objective-C* je knihovna *JSONKit* (https://github.com/johnezang/JSONKit). Ta pomocí kategorie přidává do objektů typu `NSString` metodu `objectFromJSONString` - z textového obsahu ve formátu JSON vrátí objekt typu `NSDictionary`.

```
@interface NSString (JSONKitDeserializing)
- (id)objectFromJSONString;
- (id)objectFromJSONStringWithParseOptions:(JKParseOptionFlags)parseOptionFlags;
- (id)objectFromJSONStringWithParseOptions:(JKParseOptionFlags)parseOptionFlags
error:(NSError **)error;
- (id)mutableObjectFromJSONString;
-
(id)mutableObjectFromJSONStringWithParseOptions:(JKParseOptionFlags)parseOptionFl
ags;
-
(id)mutableObjectFromJSONStringWithParseOptions:(JKParseOptionFlags)parseOptionFl
ags error:(NSError **)error;
@end
```

https://github.com/johnezang/JSONKit/blob/master/JSONKit.h

4.3.13. Protokoly

Protokoly jsou tím, čím jsou v jiných jazycích abstraktní třídy a interfacy. Definují povinné a volitelné metody, které musí objekt, přijímající protokol, splňovat. Příkladem budiž různé enumerátory (čítače), nebo protokol `NSCopying`, který musí přijmout a naplnit každý objekt, který chce podporovat kopírování metodami `copy:` a `copyWithZone:`. Definice třídy, používající třídu `NSObject` jako rodičovskou třídu a přijímající protokol:

```
@interface MyClass: NSObject <NSCopying>
...
@end
```

Je dokonce možné vynutit přijetí protokolu objektem při převzetí parametru:

```
id <NSCopying> myVar;
```

Definice protokolu probíhá pomocí klíčového slova `@protocol`

```
@protocol NSCopying
- (id)copyWithZone: (NSZone *) zone
@end
```

a může využívat přepínačů `@optional` a `@required` - ty určují, zda jsou všechny následující definice metod povinné k naplnění, nebo jsou jen k dispozici a naplnění povinné není. Přepínače fungují na bázi stavového automatu a lze je libovolně střídat.

Speciálním případem protokolů jsou tak zvané neformální, někdy také abstraktní protokoly. Jsou realizovány za pomocí kategorií a jsou určeny k implementaci v podtřídách objektu, na němž jsou definovány. S příchodem parametru `@optional` v protokolech však neformální protokoly není třeba realizovat pomocí kategorií a pro nové kódy to není ani doporučeno. (KOCHAN, 2010)

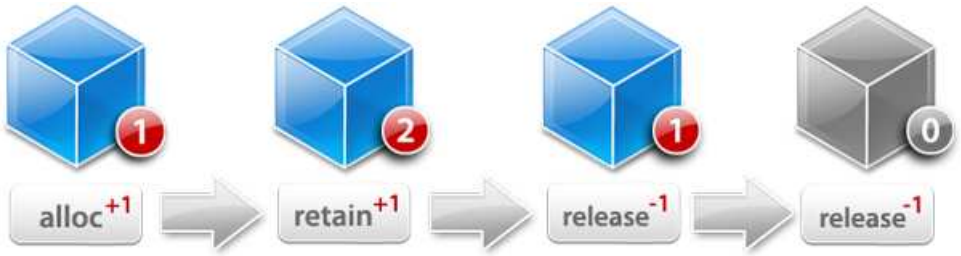
4.3.14. Správa paměti

Objective-C obsahuje podporou dvou režimů správy paměti: *Garbage Collection*[12] a režim počítání odkazů. Režim počítání odkazů je závislý na tzn. automaticky uvolňovaných zásobnících (`NSAutoreleasePool`) a umožňuje programátorovi atomickou kontrolu existence objektů napříč aplikací. To v sobě nese problém v podobě snadného úniku paměti, případně dealokace využívaného objektu a pádu celé aplikace.

Počítání odkazů funguje na jednoduchém principu - kdykoli je objekt vytvořen metodou `alloc`, `copy`, `allocWithZone`, `copyWithZone`, nebo `mutableCopy`, má počet odkazů nastaven na 1 a programátor odpovídá za jeho uvolnění. Při každém zavolání zprávy `retain` (přidržení) na objektu, nebo přiřazení do pole či slovníku, dojde k inkrementaci čítače. Odesláním zprávy `release` (uvolnění), nebo odebráním z pole či slovníku se hodnota čítače sníží o 1. Pokud je hodnota čítače rovna 0, je na objektu zavolána metoda `dealloc` a dojde k zničení objektu. Problém s tímto přístupem nastává, když je v metodě vytvořen objekt a je nutno ho předat metodě, která aktuální metodu zavolala. Pokud by nedošlo k uvolnění, nastal by tzv. *memory leak* (únik paměti), není však možno objektu předat zprávu `release`, protože by mohlo dojít k jeho zničení ještě před předáním, přístupu do neplatné adresy paměti a pádu aplikace s chybou segmentace. Pro tento případ existuje automaticky uvolňovaný zásobník - `NSAutoreleasePool`.

Automaticky uvolňovaný zásobník musí vždy v aplikaci existovat alespoň jeden a musí obsahovat celé tělo funkce `main()`, ve kterém se pracuje s frameworkem *Foundation*. Uvnitř jednotlivých metod tříd je možno definovat vlastní zásobníky, které postupně utváří hierarchickou strukturu - pokud je nějakému objektu zaslána zpráva `autorelease`, je jeho odkaz přidán do naposledy inicializovaného zásobníku (hodnota čítače odkazů snížena není). Při uvolnění tohoto zásobníku pak dojde k zavolání zprávy `release` každému objektu v něm tolikrát, kolikrát obdržel zprávu `autorelease` v daném zásobníku. Definice vlastních zásobníků dává smysl zejména při opakovaných operacích, při kterých se inicializuje velké množství objektů a automatické uvolnění po skončení metody by nemuselo stačit. Pokud systém iOS detekuje nedostatek operační paměti, pošle aplikaci zprávu a pokud není do 5ti vteřin paměť uvolněna, dojde k ukončení aplikace.

[12] Systém se stará sám o počet referencí na objekty a periodicky uvolňuje paměť, která se nevyužívá.



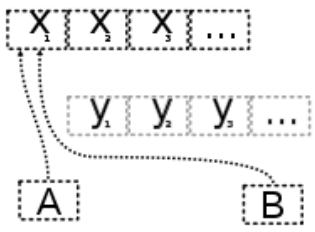
http://cocoadevcentral.com/images/articles/000094/learnobjectivec-referencecounting.png

Obrázek 5 - Počítání referencí na objekty

Garbage Collection funguje pouze na systémech MacOS, na systému iOS k dispozici není. V tom to režimu jsou všechna volání `autorelease`, `release`, `retain` a `dealloc` ignorována a běhový systém sám spravuje seznam objektů, na které již neexistují žádné reference. Tento seznam je periodicky zasílán k uvolnění paměti, uživatel ho nemá pod kontrolou. Zpravidla to znamená, že objekt existuje ještě dlouho poté, co s ním byla ukončena práce. Proto tento způsob není vhodný pro mobilní zařízení se systémem iOS, kde je množství paměti palčivým problémem. *Garbage Collection* musí být pro projekt manuálně zapnut v nastavení projektu změnou hodnoty z `Unsupported` na `Required` u položky *Objective-C Garbage Collection*. (KOCHAN, 2010), (STEVENSON, 2008)

4.3.15. Kopírování

Kopírování objektů slouží k vytvoření klonu objektu. V základní implementaci vytváří mělkou (shallow) kopii objektu, to znamená, že kopie pole obsahuje všechny prvky původního pole v podobě nových odkazů na prvky původního pole. Změnou obsahu prvků v jednom poli tak dojde ke změně hodnot i v poli druhém. Problematika hloubkového (deep) kopírování je popsána v kapitole Archivace.



http://upload.wikimedia.org/wikipedia/en/thumb/e/e1/Shallow_copy_done.svg/200px-Shallow_copy_done.svg.png

Obrázek 6 - Mělká kopie objektu

Každý objekt v *Objective-C*, který chce podporovat kopírování, musí přijmout protokol `NSCopying` (metoda `copyWithZone:`), případně i `NSMutableCopying` (metoda `mutableCopyWithZone:`). `NSCopying` vytváří v základu kopii bez rozlišení měnitelnosti, pokud lze na objektu vytvářet měnitelné i neměnitelné kopie, je nutno přijmout protokoly oba. Při zavolání metody `copy:` je zavolána metoda `copyWithZone:` s parametrem `nil` v parametru zóny, není tak nutno ji implementovat nějak specificky. Podobně se chová i metoda `mutableCopy:`.

```
MyClass *x = [[MyClass alloc] initWithData: data];
MyClass *y = [x copy];

...

- (id) copyWithZone: (NSZone *) zone {
    MyClass *new = [[MyClass allocWithZone: zone] init];
    new.data = [self data];
    return new;
}
```

Při implementaci kopírování je nutno brát v potaz způsob práce s členskými proměnnými - pokud je například generovaná metoda pomocí makra `@synthesize` typu `assign`, dojde při přiřazení hodnoty původního objektu k nežádoucímu provázání hodnot. Vhodnější je použít typ `setter` typu `copy`, který zadanou hodnotu duplikuje. (KOCHAN, 2010)

4.3.16. Archivace

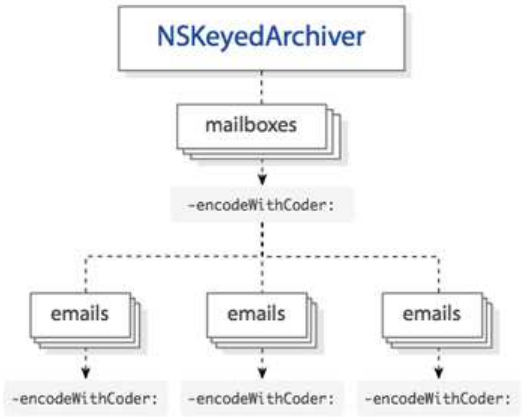
Archivace v *Objective-C* slouží k serializaci hodnot a objektů do uložitelného a později obnovitelného formátu. Tuto schopnost v sobě obsahují v základu `NSArray` a `NSDictionary` pomocí metod `writeToFile:atomically:encoding:error:`, `dictionaryWithContentsOfFile:`, respektive `arrayWithContentsOfFile:` a dalších, často je ale potřeba ukládat i jiné objekty a struktury.

Archivování pomocí `NSKeyedArchiver` je vylepšeným postupem z dřívější verze MacOS, kde byla archivace pomocí `NSArchiver` závislá na pořadí - bez znalosti struktury

celého obsahu archivu byl jeho obsah k ničemu. `NSKeyedArchiver` umožňuje přistupovat k datům prostřednictvím páru klíč-hodnota, což je výrazně flexibilnější.

Archivace začíná zavoláním metody `archiveRootObject:toFile:` na třídě `NSKeyedArchiver`, což označí rodičovský objekt celé kolekce. `NSKeyedArchiver` následně rekurzivně zavolá metodu `encodeWithCoder:` na každém prvku v rodičovském objektu a archivovaný obsah skládá do archivu. Každý objekt tak musí implementovat tuto metodu a musí s ní umět pracovat.

```
- (void) encodeWithCoder: (NSCoder *) coder {
     [coder encodeObject:properties forKey:@"properties"];
}
```



http://cocoadevcentral.com/images/articles/archiving-diagram.gif

Obrázek 7 - Rekurzivní volání encodeWithCoder:

Opětovná inicializace objektu z archivu může proběhnout metodou `initWithCoder:`, která rovnou provede inicializaci objektu. Nejprve se přiřadí proměnné `self` návratová hodnota inicializace rodičovského objektu a následně nastaví členské proměnné na hodnoty z archivu. (STEVENSON, 2005)

```
- (id) initWithCoder: (NSCoder *)coder {
    if (self = [super init]) {
         [self setProperties: [coder decodeObjectForKey: @"properties"]];
    }
    return self;
}
```

Občas je nezbytné archivovaná data přenést po síti, či použít jinak, než je prosté uložení do textového souboru. To je možné pomocí archivace do struktury `NSMutableData`, která obsahuje binární data. Volání archivace pak vypadá následovně:

```
NSMutableData *data = [NSMutableData data];
NSKeyedArchiver *archiver =[[NSKeyedArchiver alloc]
initForWritingWithMutableData: data];
NSString * str = @"Hello World!";

[archiver encodeObject: str forKey: @"myStr"];
[archiver finishEncoding];

[data writeToFile:@"archive" aromically: YES];
[archiver release];
```

Uložení objektu `NSMutableData` na vzdálené URL je velmi jednoduché, volá se metodou `writeToURL:atomically:`, nebo `writeToURL:options:error`.

Archivování se také využívá k tvorbě hlubokých kopií, jak bylo zmíněno v kapitole Kopírování. Využito je právě uložení dat do struktury `NSMutableData` a následné rozbalení na originální struktury a objekty. (KOCHAN, 2010)

```
NSArray *array = [[NSArray alloc] arrayWithObjects: obj1, obj2, obj3, nil];
NSArray *new;
NSData *data = [NSMutableData data];

data = [NSKeyedArchiver archivedDataWithRootObject: array];
new = [NSKeyedArchiver unarchiveObjectWithData: data];

NSLog(@"%@", new);
```

4.4.Historie firmy Apple

Steve Jobs se narodil v roce 1955 dvojici studentů, kteří ho kvůli osobním a sociálním problémům předali k adopci. Novými rodiči se stali Paul a Clara Jobsovi z Moutain View v Kalifornii - budoucího rodiště Silicon Valley.

Školní docházku ukončil jen díky vstřícnému přístupu pedagogů, kteří ho upláceli penězi a bonbony aby vůbec chodil do školy. To ho dle jeho slov zachránilo od budoucího vězení. V dětství se dostal do styku s elektronikou prostřednictvím stavebnice pro mládež, což se mu stalo do budoucnosti osudným. Vysokou školu, jejíž studium bylo jednou z podmínek adopce, opustil v prvním ročníku a navštěvoval neoficiálně pouze předměty, kterého zajímaly. Získal tak dle svých slov znalosti v oborech, které vůbec neměl v předpisu předmětů a které ovlivnily jeho další nahlížení na průmyslový design.

Celé Jobsovo mládí bylo na hranici legality a je s podivem, že dal nakonec vzniknout firmě, která získala celosvětové jméno, má jednu z nejhodnotnějších značek na světě a je jedním s největších inovátorů na poli osobní elektroniky. (KAHNEY, 2009)

Společně se starým přítelem Stevem Wozniakem založili společnost Apple Computer, Inc. Steve Wozniak se bavil stavbou vlastního počítače, neměl však v plánu ho masově prodávat. Za pomoci několika kamarádů začali v garáži Jobsových rodičů montovat počítače podle Wozniakova modelu Apple II a nabízeli je firmám v okolí.

V roce 1980 firma vstoupila na burzu a zaznamenala raketový vzestup. Téměř okamžitě získala hodnotu několika milionů dolarů a do roku 1983 už byla hodnota Jobsových akcií přes 100 milionů dolarů. Díky tomu se v roce 1985 objevil na 411. místě v žebříčku nejbohatších lidí světa v časopisu Fortune. (KAHNEY, 2009)

Vzhledem ke své nezvladatelné povaze byl Jobs v roce 1985 z Apple vyhozen tehdejším generálním ředitelem Johnem Sculleym. V touze po vyřazení Apple z trhu založil NeXT Computer, která měla vytvářet přímo konkurenční produkty. NeXT Computers byl minoritním dodavatelem několika velkých firem a agentury CIA, prodal necelých 50 tisíc strojů. Co bylo ovšem doménou NeXT, byl software, který na strojích běžel. Ten se také stal důvodem pozdějšího návratu Jobse do Apple.

V roce 1994 ovládal Apple 10% celého trhu s osobními počítači, v roce 1995 prodal 4,7 milionu Maců. Problematická byla snaha managementu o maximální rozšíření systému Mac a licencování operačního systému dalším výrobcům. Zákazníky tak nic nemotivovalo ke koupi dražších originálních strojů od Apple. V roce 1996 mel již Apple ztrátu 69 milionů dolarů a ani změna generálního ředitele z Michaela Spindlera na Gila Amelia nebyla šťastným krokem - Gil za rok působení dostal Apple do ztráty 1,6 miliardy dolarů a podařilo se mu špatnými kroky snížít tržní podíl z 10% na 3%.

V této době se Gil Amelio začal zajímat o operační systém BeOS, který měl nahradit zastaralý Macintosh. Náhodou mu mezi nabídkami přišla i nabídka od NeXTu, který mu nabídl svůj vlastní systém. Ten byl, na rozdíl od BeOS, dokončený, měl vývojové prostředí pro tvorbu nových aplikací, rozsáhlou uživatelskou základnu a smlouvy s vývojáři software i hardware. Jobs byl pozván do Apple, aby odprezentoval systém NeXT představenstvu a během krátké doby byly domluveny podmínky - Apple koupil NeXT za 427 milionu dolarů. Jobs se tak 20.12.1996 vrátil do Apple jako poradce Gila Amelia pro integraci systému NeXT. (KAHNEY, 2009)

Návrat do vedení firmy se odehrál v polovině roku 1997 v době, kdy byl Apple na pokraji krachu a peníze na provoz měl na necelého pul roku. Problémem byl extrémně rozvětvený počet variant produktů s nelogickým pojmenováním, kterým se zákazníkům nechtělo probírat. Apple prodával spektrum necelých 40ti produktů od tiskáren po kapesní počítače Newton. Jobs nabídku po diskuzi se všemi týmy proškrtal a přišel s kategorizací počítačů na 4 hlavní skupiny: mobilní osobní, stacionární osobní, mobilní profesionální, stacionární profesionální. Mezi vedením Apple toto řešení zpočátku nevyvolalo nadšení, prodejní výsledky však hovořily jasně.

Velkým uspěchem bylo i urovnání sporu s Microsoftem. Žaloba na Microsoft tvrdila, že Windows 95 ukradly několik klíčových technologií Apple. Výměnou za stažení žaloby Bill Gates slíbil pokračovat ve vývoji kancelářského balíku pro Mac, který byl naprosto klíčovou softwarového vybavení.

Jobsovi se také podařilo snížit skladové zásoby Apple z 400 na méně než 100 milionů. Před Jobsovým příchodem měl Apple roční ztrátu 1,6 miliardy dolarů, už po prvním roce jeho vlády byl zisk 2 miliony. To je pro tak velkou společnost málo, rozdíl oproti předchozímu roku však mluví jasně. Další rok navíc zisk dále raketově rostl. (KAHNEY, 2009)

Nová politika Apple, která směřovala k maximální jednoduchosti a účelnosti se projevila na všech produktech, které vznikly pod Jobsovým vedením. Prvním byl Power Macintosh G3 v roce 1997. Toho se hned v prvním roce prodal 1 milion kusů. Úspěšná série pokračovala iBookem a PowerBookem, dvojicí notebooků pro běžné i firemní uživatele. iBook byl prvním notebookem, který byl nabízen v několika barevných variantách. Velice úspěšný byl i iMac, který byl navržen do tvaru kapky v barvě ovoce a stál za zcela novým trendem v průmyslovém designu. Mezi produkty, které započaly novou éru v elektronice, je třeba samozřejmě uvést iPod, který dnes kraluje trhu mp3 přehrávačů a kterému patří kolem 70% trhu. V roce 2007 měla již plná třetina nových modelů automobilů možnost připojit iPod přímo k rádiu. Masivní úspěch zaznamenal samozřejmě i iPhone a iPad. Operační systém iOS, který pohání iPod, iPhone a iPad je maximálně zjednodušenou variantou desktopového OS X a poskytuje stejnou uživatelskou přívětivost a stejné vývojářské nástroje. Díky tomu je pro vývojáře možné obsáhnout celou produktovou řadu Apple téměř identickým software. V MacOS X verze 10.7 Lion začala zpětná portace funkcí z iOS do desktopového systému, uživatelé mají podobné funkce na kapesních zařízeních i pracovní stanici. Verze 10.8 Mountain Lion, která má vyjít v létě 2012, jde ještě dále a propojuje OS X s dalšími funkcemi iOS 5 - iCloud, nová verze připomínek, notifikační centrum a Game Center. (KAHNEY, 2009)

Na rok 2012 je plánována kompletní obnova produktů iPhone, iPad, iPod, Mac mini, MacBook Pro a MacBook Air. Třetí verze iPadu byla představena na Apple konferenci 7. března 2012, představení dalších produktů proběhne během léta a podzimu.
(Apple Inc., 2012)

4.5.Novoslověnština

4.5.1. Vznik jazyka

Pro konkrétní implementaci aplikace byl vybrán jazyk novoslověnština, který je projektem Evropské Unie pro vzájemnou komunikaci mezi (nejen) slovanskými vědci v prostředí vědeckého zařízení CERN v rámci projektu LA08015 - Spolupráce ČR s CERN. Projekt vznikl v rámci přibližně 300-členné komunity. Jazyk staví na živých slovanských jazycích. Tato skupina obsahuje ruštinu, ukrajinštinu, polštinu, češtinu, slovenštinu, slovinštinu, chorvatštinu, srbštinu a další. Slovanské národy obývají téměř polovinu Evropy a slovanská populace dosahuje celosvětově cca 400 milionů. (CERN, 2012)

Slovanské jazyky jsou poměrně konzistentní skupinou, znalost jednoho jazyka tak většinou stačí k alespoň hrubému porozumění v jakémkoli dalším slovanském jazyce. Na tomto faktu novoslověnština staví, podobně jako staroslověnština Konstantina a Metoděje z 9. století a několik dalších jazykových projektů z 16. století a století pozdějších.

Návrh novoslověnštiny stojí na třech pilířích:

- sjednocení gramatiky a základní slovní zásoby z aktuálně živých a užívaných slovanských jazyků,
- velmi strmá učící křivka, dovolující za minimum času ovládnout velmi značnou část jazyka, což může neslovanským národům poskytnout alespoň pasivní znalost jazyka, případně usnadnit učení nějakého konkrétního jazyka,
- "kompatibilita" s původním jazykem věrozvěstů Konstantina a Metoděje, který je stále často užíván v ortodoxních kostelích a římsko-katolickou obcí v mnoha slovanských zemích.

Je nutno zmínit, že novoslověnština není ojedinělým projektem, za posledních dvě stě let vzniklo mnoho umělých jazyků. Největší přínos v této oblasti mají slovinský kněz a lingvista Matija Majar Ziljski a český překladatel Václav František Bambas. Novoslověnština je také do značné míry inspirována projekty na rekonstrukci moderní srbštiny, češtiny, slovenštiny a hebrejštiny.

Je známo, že více než polovina Slovanů hovoří rusky. Pokud by ruština byla jednoduchým jazykem, nebylo by třeba vytvářet nový umělý jazyk. Bohužel se nachází poměrně daleko od pomyslného středu slovanských jazyků svou specifickou abecedou, výslovností, gramatikou a slovní zásobou. Podobně jsou na tom ostatní slovanské jazyky - žádný z nich není univerzálním jazykem i přes společné kořeny. Důvodem, který dává podnět k vzniku novoslověnštiny, tak je snaha o vznik prostředku k dorozumění napříč slovanskými národy bez nutnosti překladu do několika podobných jazyků.

(MERUNKA, 2012)

V aplikaci nabízené věty a slovní zásoba vychází z generátoru jazyka na adrese http://generator.neoslavonic.org/ a přidruženém webu http://www.neoslavonic.org/.

4.5.2. Abeceda

Na rozdíl od angličtiny jsou v novoslověnštině slova vyslovována tak, jak jsou napsána. Není tak třeba užit se výslovnost každého slova, stačí výslovnost jednotlivých znaků. Důraz je dáván na první slabiku ve slově.

Novoslověnštinu je možno zapisovat latinkou, cyrilicí, nebo ASCII přepisem. Je možné využít i původní hlaholici.

- Mezi znaky `i` a `y` (měkké a tvrdé i/y) není ve výslovnosti žádný rozdíl,
- slabika `ju` změkčuje předcházející souhlásku,
- na rozdíl od jiných slovanských jazyků, novoslověnština nezměkčuje souhlásky, pokud za nim následuje `e`, nebo `i`, kombinace `ne` nebo `ni`, jsou tak vyslovovány tvrdě,
- novoslověnština má slabičné l a r, které se chovají jako samohlásky při tvorbě slabik jako v sanskrtu. Jsou v tom případě psány s prefixovým apostrofem jako `'l` a `'r` (nebo `ъл` a `ър`) v cyrilici. Většina jazyků tuto vlastnost nemá, musí tak tyto hlásky prokládat samohláskami.

(MERUNKA, 2012)

4.5.3. Změkčování

V novoslověnštině existuje stejně, jako v mnoha slovanských jazycích, změkčování. Jeho použití je, například oproti češtině, značně limitováno na velmi malý počet změkčovaných hlásek:

- k → č
- h → š
- g → ž

Příkladem budiž změkčení `človiek` na `človiečе!`, nebo `prah` na `prašny`.

Aby jazyk zněl jako přirozený slovanský jazyk, bylo nutno přidat i některá zakončení slov, která vzniknou aplikací gramatických pravidel.

- cju → ču, cie → če,
- sju → šu, sie → še,
- zju → žu, zie → že.

(MERUNKA, 2012)

4.5.4. Vzory podstatných jmen

Novoslověnština, stejně jako čeština, zná tři rody podstatných jmen - mužský, ženský a střední.

Dále existuje sedm pádů: nominativ, genitiv, dativ, akuzativ, vokativ, lokativ a instrumentál. Pro singulár platí symetričnost dativ - lokativ, pro plurál pak nominativ - akuzativ a vokativ, genitiv - lokativ a dativ - instrumentál.

Pokud v plurálu vyjde na poslední hlásku `-a`, je nahrazeno `-am` pro dativ, `-ah` pro lokativ a `-ami` pro instrumentál.

Stručný přehled skloňovacích tabulek je uveden v příloze A na konci práce.

(MERUNKA, 2009)

4.6.Konkurenční aplikace - TalkPad

Přímo konkurenční aplikací je aplikace *TalkPad*, která využívá podobného principu. Obsahuje jedinou obrazovku, na které jsou základní slovíčka, číslovky a fráze, které jsou aplikací po stisknutí vysloveny pomocí nahrávky rodilého mluvčího. Problémem aplikace je však prakticky nulová slovní zásoba, která nepostačuje ani na základní dorozumění v cizí zemi a nulová flexibilita - aplikace není rozšiřována o další výrazy (není na ně místo) a neumožňuje upravit větu dle aktuální potřeby - např. fráze "*Where is the hospital?*" ("Kde je tady nemocnice?") nejde upravit na hledání nádraží, banky, ani jiného objektu.

Obrázek 8 - TalkPad v řečtině a perštině

5. Praktická část

5.1. Návrh struktury kódu

Každá aplikace v *Objective-C* má svého delegáta aplikace, což je třída, která přijímá zprávy od systému ohledně svého spuštění, ukončení, přechodu na pozadí a do popředí, varování o docházející paměti, atd. Tato třída je tak neměnnou součástí aplikací pro systém iOS i MacOS. Povinným souborem pro platformu iOS je také alespoň jeden *NIB* soubor s oknem aplikace. Jeho výchozí název je MainWindow.xib a není doporučeno ho měnit. Nachází se v něm definice obsahu okna a vazby na základní *outlety* (zásuvky pro prvky rozhraní) delegáta. Většinu z metod delegáta není třeba manuálně implementovat, pokud jde o jednoduchou aplikaci, výchozí systémová implementace je dostatečně pružná. U větších aplikací již naopak bývá vlastní kód nutností.

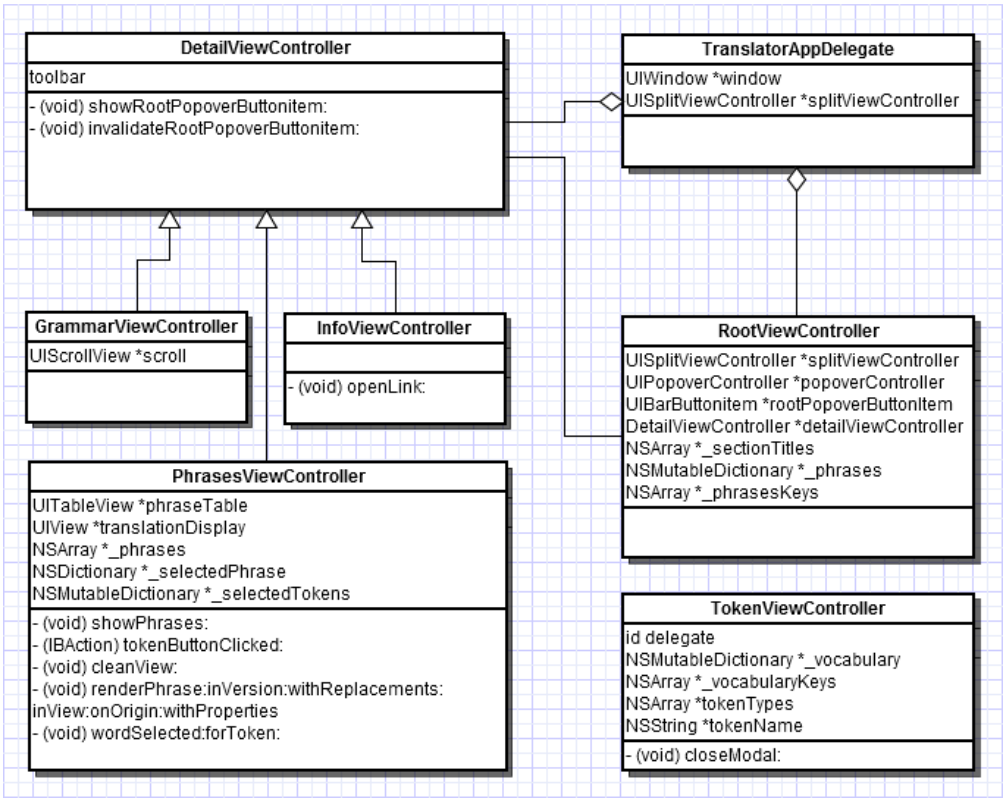
Vzhledem k stavbě aplikace nad `UISplitViewController`, musí v aplikaci existovat nejméně dva další kontrolery uživatelského rozhraní - levý a pravý. Za tímto účelem byly založeny `RootViewController` a `DetailViewController`. `RootViewController` je poděděn z `UITableViewController` (jeho obsah bude pouze tabulka) a bude mít na starost rámec s menu, `DetailViewController` bude rodičovskou třídou pro všechny pohledy zobrazované v pravé straně okna aplikace - stará se obsluhu `UIToolbar` v hlavičce pohledu a o přidávání/odebírání ovládacího prvku (`UIToolbarItem`) pro polohu zařízení na výšku. Nemá žádný NIB soubor s uživatelským rozhraním, ty mají až poděděné třídy. Outlet je pro tyto soubory společný v mateřské třídě.

```
- (void)showRootPopoverButtonItem:(UIBarButtonItem *)barButtonItem {

    [toolbar setItems:[NSArray arrayWithObject:barButtonItem] animated:YES];
}

- (void)invalidateRootPopoverButtonItem:(UIBarButtonItem *)barButtonItem {

    [toolbar setItems:[NSArray array] animated:YES];
}
```

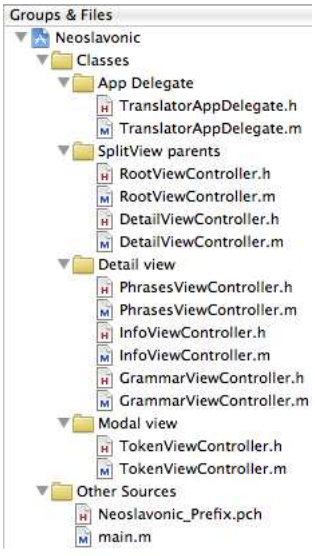
Z pohledu `DetailViewController` jsou poděděny tři kontrolery, starající se o celý zbytek běhu aplikace: `PhrasesViewController`, `InfoViewController`

a `GrammarViewController`. Obsah těchto souborů bude probrán v dalších kapitolách, všechny mají svůj NIB soubor s rozhraním.

Posledním kontrolerem je `TokenViewController`, což je kontroler modálního okna s výběrem tokenů pro doplnění vět.



Obrázek 9 - UML diagram tříd aplikace



Obrázek 10 - Přehled souborů projektu

5.2.Hlavní okno

Hlavní okno je realizováno pomocí `UISplitViewController`. Tento typ pohledu je specialitou iPadu a umožňuje zobrazit najednou dvě okna s obsahem, zpravidla dvě úrovně pohledu na jedna strukturovaná data. Rámec obsahuje dva prvky `UIView` a sám o sobě žádné akce neprovádí. Je však nezbytné nastavit mu delegáta, kterému jsou zasílány zprávy v případě změny orientace okna. Roli delegáta zastává objekt `RootViewController` - ten existuje po celou dobu běhu aplikace i přes to, že může být ve vertikální poloze skrytý. Jde o zprávy:

- `splitViewController:shouldHideViewController:inOrientation:`
- `splitViewController:willHideViewController:withBarButtonItem:forPopoverController:`
- `splitViewController:willShowViewController:invalidatingBarButtonItem:`
- `splitViewController:popoverController:willPresentViewController:`

První zpráva je volána za účelem zjištění, zda má být první pohled skryt v aktuální poloze zařízení. Pokud je vrácena hodnota YES, je na obrazovce zobrazeno zjednodušené rozložení pouze s rámcem detailu.

Další dvě metody dlouží pro informování delegáta, že došlo k změně orientace zařízení a je třeba přizpůsobit rozložení prvků, protože dojde ke skrytí/zobrazení daného rámce. V aplikaci je volání této metody užito k přidání/odebrání tlačítka na `UIToolbar` v hlavičce detailu. Toto tlačítko zobrazí pohled s menu pomocí `UIPopoverControlleru`.

(Apple Inc., 2011)

Podobnou funkčnost by šlo simulovat i pomocí modálního okna, `UINavigationController` se zanořením do druhého rámce, případně manuálně vytvářeným `UIPopoverController`, zvoleno ovšem bylo výše zmíněné řešení - nejvíce odpovídá doporučovaným postupům firmou Apple pro platformu iPad. Je navíc

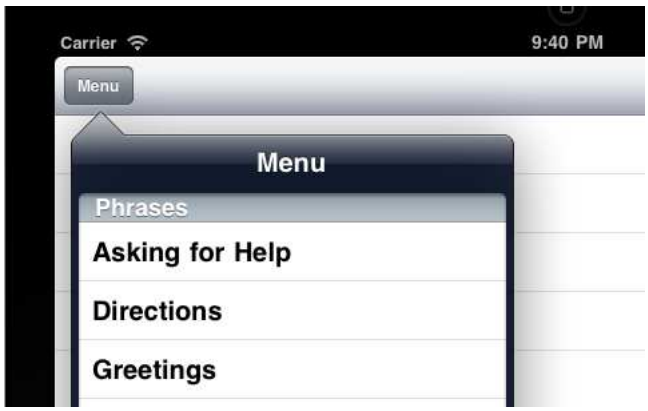
uživatelsky velmi příjemné v obou polohách zařízení a uživatelé jsou na něj zvyklí ze zabudovaných aplikací (např. iMail).

V levé části je umístěn `RootViewController` (dceřiná třída `UITableViewController`) vnořený v `UINavigationController`. `RootViewController` zde slouží jako menu aplikace, je rozdělen na dvě sekce - `Phrases` a `Others`. Dvě sekce byly zvoleny z důvodu rozdělení na překládací část aplikace a ostatní funkce. V `Phrases` uživatel najde jednotlivé kategorie vět, které jsou často potřeba při komunikaci. Tento seznam je dynamicky načítán ze souboru vlastností `phrases.plist`, který je součástí aplikace a lze ho jednoduše aktualizovat pomocí aktualizací na AppStore.

V `Others` jsou položky `Grammar` a `Info` - obě budou popsány později. `UINavigationController` zobrazuje nad tabulkou navigační lištu s popiskem okna. V případě této aplikace se v něm zobrazuje řetězec `Menu`, což je titulek řídícího rámce `RootViewController` nastavený v metodě `viewDidLoad:`.

```
- (void) viewDidLoad {
    ...
    self.title = @"Menu";
    ...
}
```

Převzetí titulku probíhá automaticky, je součástí přirozené funkce `UINavigationController` a využije se i při otočení zařízení do polohy na výšku, kdy se stane popiskem hlavičky `UIPopoverController`.

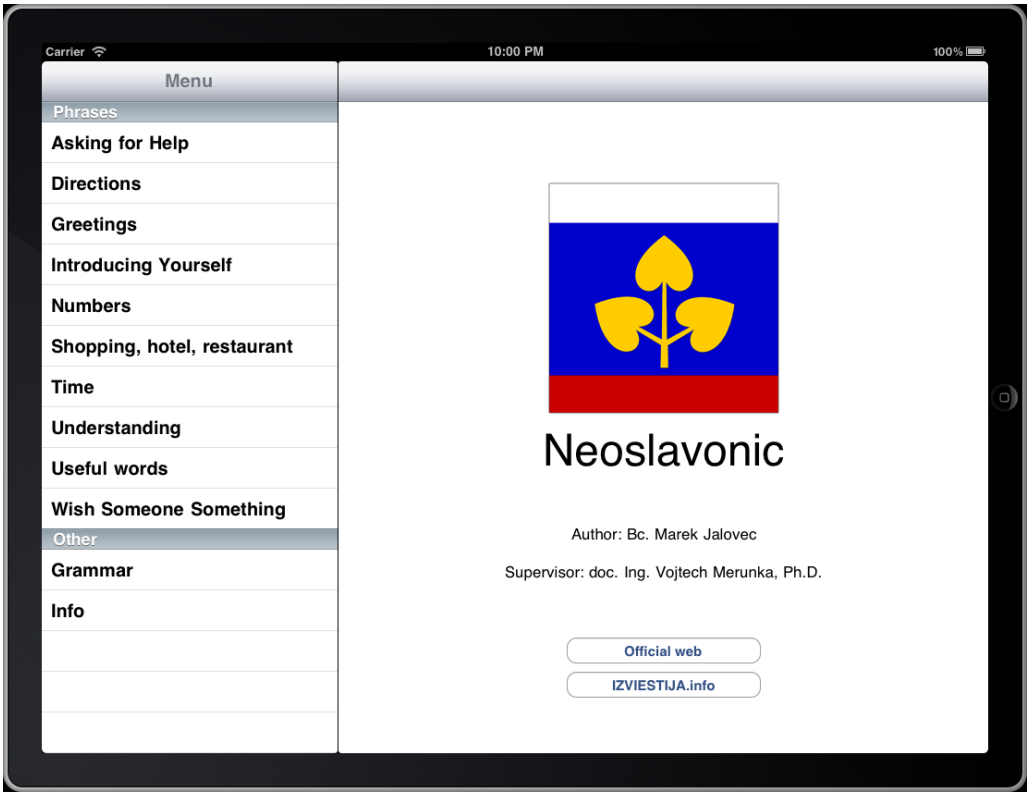


Obrázek 11 - UIPopoverController

Pravá část okna je po otevření zaplněna uvítací obrazovkou s informacemi o programu, která je také dostupná prostřednictvím položky `Info` v menu `Others` v `RootViewControlleru`. Okno s informacemi o programu je realizováno pomocí podtřídy `UIViewController`, do které je vložen `UIImageView` s logem aplikace, třemi prvky `UILabel`, které obsahují informace o autorovi, garantovi práce a název aplikace a odkazy na weby s problematikou novoslověnštiny v podobě tlačítek `UIButton`. Stisknutí tlačítek aplikaci převede na pozadí a spustí internetový prohlížeč Safari s daným odkazem.

```
- (IBAction)openLink:(id)sender {

    // Use URL by sender button.
    NSString *link;
    if ([sender tag] == kButtonOfficial) {
        link = @"https://sites.google.com/site/novoslovienskij/";
    } else { // kButtonIzviestija
        link = @"http://izviestija.info/";
    }

    // Open Safari with selected URL.
    NSURL *url = [NSURL URLWithString:link];
    [[UIApplication sharedApplication] openURL:url];
}
```

Otevření vlastního okna prohlížeče je v tomto případě preferováno před vložením internetového rámce přímo do aplikace - lže očekávat, že uživatel na webu stráví více času zkoumáním materiálů k jazyku a dokonce bude přecházet mezi jednotlivými weby. V aplikaci by neměl k dispozici nativní chování a ovládací prvky prohlížeče a používání by tak bylo méně komfortní. Vzhledem k učebnímu charakteru aplikace, nevadí odvedení pozornosti uživatele mimo hlavní okno. V případě komerční aplikace by bylo vhodnější integrovat internetové okno přímo v aplikace a maximalizovat tak dobu užívání na jedno spuštění.



Obrázek 12 - Vzhled aplikace po inicializaci

V úvodním rámci bylo nutno provést několik kompromisů a nastavení z důvodu flexibility okna při změně pozice a při jiném rozlišení. Všechny prvky zde tak mají nastaveny parametry tak, aby při přechodu na jiné rozlišení, nebo na jinou orientaci zařízení bylo vše viditelné a čitelné:

- Logo novoslověnštiny se zmenší se zachováním poměru stran,
- `UILabel` prvky s texty v případě potřeby přejdou na víceřádkové a zalomí se za titulky,
- všechny prvky se přesunou blíže hornímu okraji okna tak, aby byly viditelné a ovladatelné.

5.3. Gramatika

Okno s gramatickým přehledem je podtřídou `DetailViewController`, v jehož pohledu `contentView` je umístěn `UIScrollView`. Tento typ pohledu slouží k prohlížení obsahu většího, než je velikost jeho okna (vlastnost `frame`) pomocí *drag-and-drop*[13] posunu - uživatel přitiskne prst na plochu displeje a tažením posunuje obsahem.

```
#import <UIKit/UIKit.h>
#import "DetailViewController.h"

@interface GrammarViewController : DetailViewController {
    UIScrollView *scroll;
}

@property (nonatomic, retain) IBOutlet UIScrollView *scroll;

@end
```

Souhrn změn oproti výchozímu nastavení rámce:

- Barva pozadí byla nastavena na bílou barvu, neboť obrázek má bílé pozadí a přesahující obsah jiné barvy by vypadal nepatřičně. Bílá barva je výchozí hodnotou, ale není garantována. Aby byla zaručena kompatibilita s budoucími verzemi, je barva nastavena ručně.
- Vypnuto bylo `canCancelContentTouches`, což je vlastnost dovolující ignorovat v některých případech dotyky uživatele. Jejím vypnutím je zaručena maximální ovladatelnost a ideální reakce na gesta nad pohledem.
- Protože obsah obrázku s gramatikou obsahuje převážně světlé plochy, byla barva posuvníků nastavena na černou (`UIScrollViewIndicatorStyleBlack`).
- Hodnota `clipsToBounds` byla nastavena na YES, aby nedocházelo k vykreslování mimo rámec `UIScrollView`. `Bounds` je parametr všech rámců v *Cocoa*, liší se od parametru `Frame` umístěním výchozího bodu:

[13] Táhni a pusť - stisk na prvku, tažení a uvolnění na jiné pozici

`Frame` má vždy počátek relativní vůči mateřskému objektu, `Bounds` má počátek nulový. Počítá se relativně k obsahu.

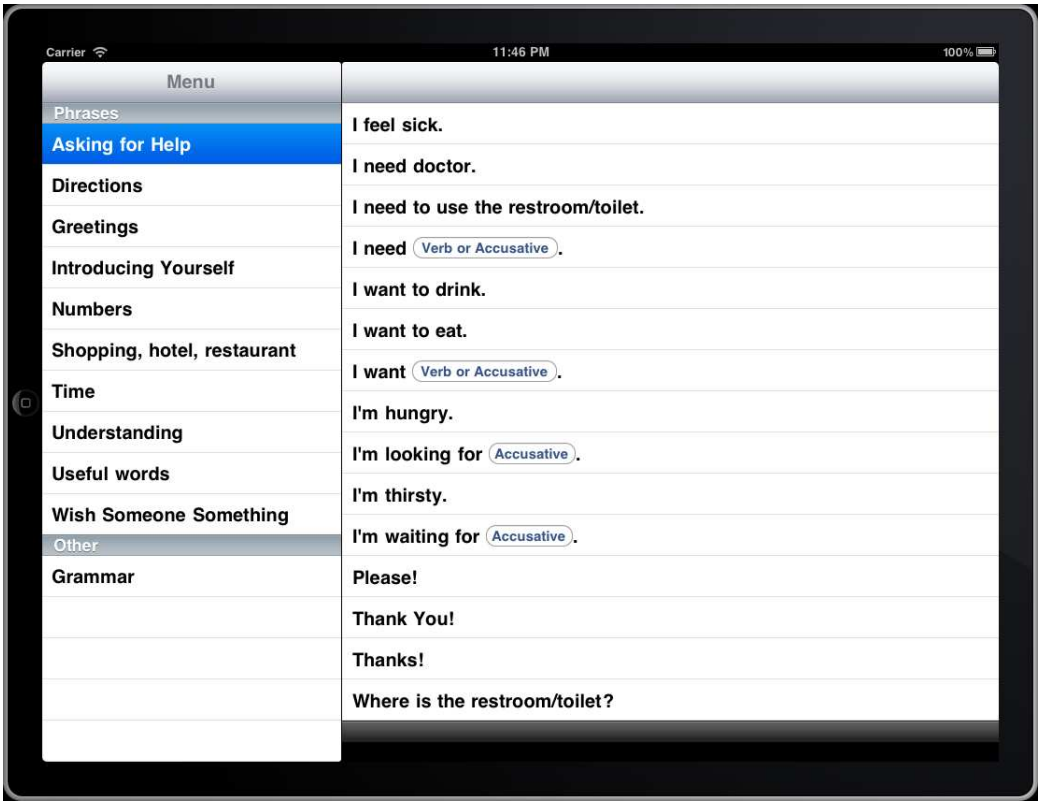
- Pro maximální pohodlí při prohlížení byla vypnuta vlastnost `Paging Enabled`, která při posunu přichytává obsah na ucelené stránky - uživatel tak nemůže volně posouvat obsahem dle libosti, je vázán na plochu displeje. To by při posunu gramatikou bylo kontraproduktivní.
- Dále je zapnuto `Bounce Scroll`. Tato vlastnost dovolí posun obsahu i za jeho mez a následně obsah posune zpět do viditelné části - posun pak působí přirozenějším a méně strojovým dojmem.

Po nastavení rámce je do něj vložen `UIImageView` s obrázkem souhrnu gramatiky a je nastavena velikost obsahu - toto je nezbytné pro správně zobrazení a posun obsahu

```
- (void)viewDidLoad {

    [super viewDidLoad];

    // Init and setup UIScrollView.
    [scroll setBackgroundColor:[UIColor whiteColor]];
    [scroll setCanCancelContentTouches:NO];
    scroll.indicatorStyle = UIScrollViewIndicatorStyleBlack;
    scroll.clipsToBounds = YES;
    scroll.scrollEnabled = YES;
    scroll.pagingEnabled = NO;
    UIImageView *image = [[[UIImageView alloc] initWithImage: [UIImage
imageNamed:@"grammar.png"]] autorelease];
    [scroll addSubview: image];
    [scroll setContentSize: image.frame.size];
}
```

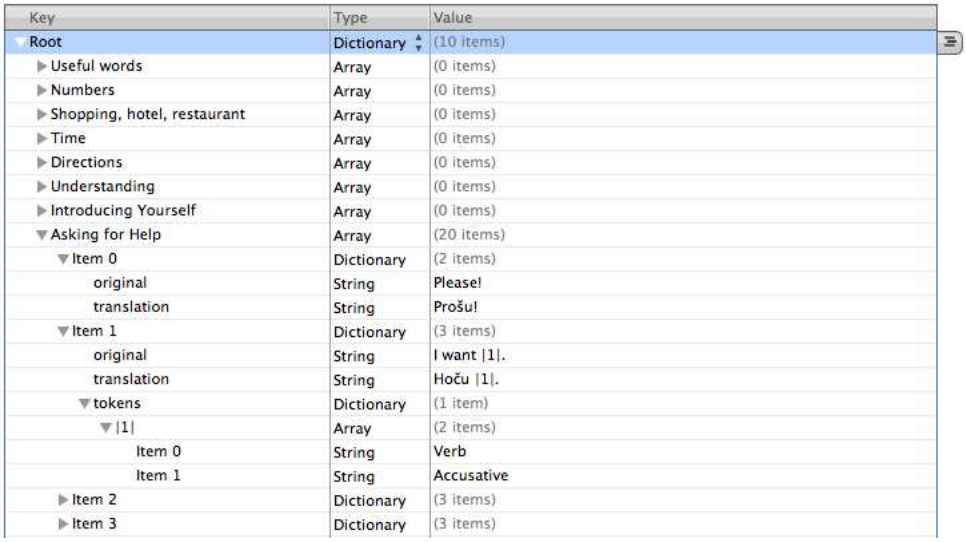
5.4. Okno s překladem frází

Okno s výběrem a překladem frází (`PhrasesViewController`) je složeno z `UIToolbar` v horní části pohledu, ve kterém se zobrazuje při otočení zařízení na výšku tlačítko pro zobrazení `UIPopoverControlleru` s menu, `UITableView` se seznamem frází a dole pak druhou instanci `UIToolbar` s překladem zvolené fráze.



Obrázek 13 - Okno pro překlad frází

Po výběru některé položky z menu `Phrases` je v pravé části obsah nahrazen tabulkou s frázemi, které do dané kategorie patří. Seznam je, jak již bylo zmíněno, dynamicky načítán ze souboru `phrases.plist`. Výběrem fráze v pravé části obrazovky je ve spodním panelu zobrazen její znění v novoslověnštině.



Key	Type	Value
Root	Dictionary	(10 items)
Useful words	Array	(0 items)
Numbers	Array	(0 items)
Shopping, hotel, restaurant	Array	(0 items)
Time	Array	(0 items)
Directions	Array	(0 items)
Understanding	Array	(0 items)
Introducing Yourself	Array	(0 items)
Asking for Help	Array	(20 items)
Item 0	Dictionary	(2 items)
original	String	Please!
translation	String	Prošu!
Item 1	Dictionary	(3 items)
original	String	I want |1|.
translation	String	Hoču |1|.
tokens	Dictionary	(1 item)
|1|	Array	(2 items)
Item 0	String	Verb
Item 1	String	Accusative
Item 2	Dictionary	(3 items)
Item 3	Dictionary	(3 items)

Obrázek 14 - Struktura souboru phrases.plist

Řazení položek je abecední pro snadné vyhledání požadované fráze, probíhá již při načtení seznamu při inicializaci aplikace. Tím je zaručena minimální doba reakce rozhraní na akce uživatele. Inicializace aplikace dále obsahuje nastavení rozměru obsahu `UIPopoverController` pro polohu na výšku, nastavení tabulky s frázemi (vybraný řádek zůstává označený) a inicializaci menu. Řazení položek využívá nové funkčnosti v Objective-C 2.0 pro MacOS X 10.6 Snow Leopard a iOS 3.0 - bloky. Bloky jsou alternativou anonymních (lambda) funkcí z jiných jazyků, umožňují inline[14] definici funkce, čímž výrazně zvyšují přehlednost kódu. Programátor díky vytvoření řadící funkce přímo tam, kde je potřeba, nemusí definovat metodu v hlavičkovém souboru i souboru se zdrojovým kódem.

```
- (void)viewDidLoad {

    [super viewDidLoad];

    // Init view.
    self.clearsSelectionOnViewWillAppear = NO;
    self.contentSizeForViewInPopover = CGSizeMake(320.0, 600.0);

    // Init menu.
    self.title = @"Menu";
    _sectionTitles = [[NSArray alloc] initWithObjects:@"Phrases", @"Other", nil];
```

[14] Přímo v kódu.

```
    // Load phrases and sort child arrays by original text.
    NSString *path = [[NSBundle mainBundle] pathForResource:@"phrases"
ofType:@"plist"];
    NSDictionary *phrases = [NSDictionary dictionaryWithContentsOfFile:path];
    _phrases = [[NSMutableDictionary alloc] init];
    for (NSString *key in phrases) {
        NSArray *object = [[phrases objectForKey: key]
sortedArrayUsingComparator: ^(id x, id y) {
            return [[x objectForKey:@"original"] compare:[y
objectForKey:@"original"]];
        }];
        [_phrases setObject:object forKey:key];
    }
    _phraseKeys = [[[_phrases allKeys]
sortedArrayUsingSelector:@selector(compare:)] retain];
}
```

Některé věty umožňují dynamické doplňování slov tak, aby více odpovídaly záměru hovořícího. V aktuální databázi frází jde o slovesa v infinitivu a podstatná jména v nominativu, akuzativu a dativu.

Key	Type	Value
Root	Dictionary	(3 items)
Accusative	Array	(21 items)
Dative	Array	(21 items)
Verb	Array	(12 items)
Item 0	Dictionary	(2 items)
Item 1	Dictionary	(2 items)
Item 2	Dictionary	(2 items)
original	String	to sleep
translation	String	spati
Item 3	Dictionary	(2 items)
original	String	to sing
translation	String	pievati
Item 4	Dictionary	(2 items)

Obrázek 15 - Struktura souboru vocabulary.plist

Výpis frází je řešen pomocí výchozí tabulky `UITableView`, ale jednotlivé buňky jsou na míru upraveny potřebě vkládat ovládací prvky na nestandardní pozice. Buňka je tak vytvořena jako výchozí `UITableViewCellStyleDefault`, jsou z ní ale odstraněny všechny prvky `UILabel` a `UIButton` a textové segmenty jsou spolu s případnými tlačítky pro volbu tokenů vyrenderovány ručně.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    static NSString *identifier = @"cell";

    // Dequeue or create a cell of the appropriate type.
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
identifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:
UITableViewCellStyleDefault reuseIdentifier: identifier] autorelease];
        cell.accessoryType = UITableViewCellAccessoryNone;
    }

    NSDictionary *phrase = [_phrases objectAtIndex:indexPath.row];

    // Render cell content from phrase text and tokens.
    NSMutableDictionary *properties = [[[NSMutableDictionary alloc] init]
autorelease];
    [properties setValue: [UIFont boldSystemFontOfSize:20] forKey: @"font"];
    [properties setValue: [UIColor blackColor] forKey: @"color"];
    [properties setValue: [UIColor whiteColor] forKey: @"background"];
    [self renderPhrase:phrase inVersion:@"original"
withReplacements:[_selectedTokens objectForKey:indexPath] inView:cell.contentView
onOrigin:CGPointMake(11, 11) withProperties:properties];

    return cell;
}
```

Za tímto účelem byla vytvořena metoda `renderPhrase:(NSDictionary*)inVersion:(NSString*)withReplacements:(NSDictionary*)inView:(UIView*)onOrigin:(CGPoint)withProperties:(NSDictionary*)`, která přebírá slovník s frází, název prezentované verze fráze (originál, nebo překlad - `original`/`translation`), seznam uživatelem vybraných náhražek tokenů, ukazatel na nadřazený rámec, v němž má být fráze vykreslena, výchozí pozici v nadřazeném rámu v obrazových bodech, neboť renderování fráze v `UITableViewCell` a `UIToolbar` vyžaduje jinou pozici a nakonec slovník s nastavením renderování. Ten obsahuje barvu popisků, nastavení fontu a barvu pozadí.

Nejprve v metodě proběhne vyčištění rodičovského rámce pomocí vlastní metody `cleanView:` volané na rámci, následně jsou z předaného konfiguračního slovníku načteny parametry fontu a pozadí textu a jsou předpřipraveny objekty textu fráze a tokenů. Pokud fráze tokeny neobsahuje, není třeba komplikovat rendering tlačítky a pomocí rozhodovacího bloku je do rodičovského rámce vykreslen jediný popisek s textem na zadané souřadnice a se zadanými parametry. Pokud ovšem fráze obsahuje nějaký volitelný blok (v textu je rozpoznatelný podle dvou znaků svislé čáry a čísla mezi nimi). Každý tento blok má v sobě v souboru `phrases.plist` vázáno pole s názvy tvarů slov, které je možno doplňovat. Pokud pole obsahuje více položek, je dovoleno vybírat z více kategorií najednou. V popisku tlačítka jsou pak oba tyto tvary spojeny spojkou "or" (nebo) a v modálním dialogu tabulka obsahuje dvě sekce s tvary.

Vyhledávání tokenů bylo možno udělat pomocí regulárních výrazů (stavové automaty sloužící k porovnávání dat vůči předpisu), nebo jednoduchým vyhledáním podřetězce zabudovanými funkcemi. Zvoleny byly z důvodu vyšší rychlosti zabudované funkce, regulární výrazy jsou výhodnější až pro složitější a variabilní řetězce. Text fráze je tak procházen cyklem a při každé iteraci je vyhledáván jeden klíč z pole tokenů. Pokud je nalezen, je vykreslen `UILabel` s textem, který token bezprostředně předchází a šířka tohoto popisku je připočtena ke startovací pozici, předané funkci. Na nové pozici je vykresleno tlačítko s popiskem povolených tvarů (např. "*Verb or Accusative*" - sloveso, nebo podstatné jméno v akuzativu), je mu nastavena akce po stisku a štítek pro identifikaci tlačítka - fráze může obsahovat i více tokenů, je tak nezbytné vědět, které tlačítko uživatel stiskl.

Výjimkou je případ, kdy již uživatel pro tento token vybral nějakou hodnotu ze slovníku - pak je text zobrazen na tlačítku místo povolených tvarů. Je tak vizuálně součástí věty a umožňuje jednoduché přečtení uživatelem. Po skončení vyhledávání tokenů může nastat situace, že ještě nebyla část textu fráze zpracována. Z toho důvodu probíhá ještě kontrola na zbytkový text a ten je případně vykreslen za posledním tlačítkem.

```
- (void)renderPhrase:(NSDictionary *)phrase inVersion:(NSString *)version
withReplacements:(NSDictionary *)replacements inView:(UIView *)view
onOrigin:(CGPoint)origin withProperties:(NSDictionary *)properties {

    // Cleanup.
    [self cleanView:view];

    // Load properties.
    UIFont *font = [properties objectForKey: @"font"];
    UIColor *color = [properties objectForKey: @"color"];
    UIColor *background = [properties objectForKey: @"background"];

    NSString *phraseText = [phrase objectForKey:version];
    NSDictionary *tokens = [phrase objectForKey: @"tokens"];

    // Are there some tokens to replace?
    NSUInteger matchPos = 0;
    if (tokens) {
        for (NSString *tokenKey in tokens) {
            // Find token key ("|0-9|"), find preceding text and calculate text
size for label.
            NSRange range = [phraseText rangeOfString: tokenKey];
            NSString *match = [phraseText substringWithRange:
NSMakeRange(matchPos, range.location - matchPos)];
            CGSize matchSize = [match sizeWithFont: font];

            // Create label for preceding text.
            UILabel *label = [[[UILabel alloc] initWithFrame:
CGRectMake(origin.x, origin.y, matchSize.width, matchSize.height)] autorelease];
            label.font = font;
            label.textColor = color;
            label.backgroundColor = background;
            label.text = match;
            [view addSubview: label];

            // Move origin by label width.
            origin.x += matchSize.width;

            // Create button for token.
            NSDictionary *replacement = [replacements objectForKey:tokenKey];
            NSString *buttonTitle;
            if (replacement) {
                buttonTitle = [replacement objectForKey:version];
            } else {
```

```
                buttonTitle = [[tokens objectForKey: tokenKey]
componentsJoinedByString: @" or "];
            }
            CGSize buttonSize = [buttonTitle sizeWithFont: [UIFont
systemFontOfSize: UIFont.labelFontSize]];
            buttonSize.width += 10;
            UIButton *button = [UIButton buttonWithType:
UIButtonTypeRoundedRect];
            button.frame = CGRectMake(origin.x, origin.y, buttonSize.width, 24);
            button.tag = [[tokenKey stringByReplacingOccurrencesOfString:@"|"
withString:@""] intValue];
            [button addTarget: self action:
@selector(tokenButtonClicked:withEvent:) forControlEvents:
UIControlEventTouchUpInside];
            [button setTitle: buttonTitle forState: UIControlStateNormal];
            [view addSubview: button];

            // Move origin by button width.
            origin.x += buttonSize.width;

            // Move matchPos by location and length of token key.
            matchPos = range.location + range.length;
        }
    }
    // Is there any remaining text?
    if (matchPos != [phraseText length]) {
        NSString *match = [phraseText substringWithRange: NSMakeRange(matchPos,
[phraseText length] - matchPos)];
        CGSize matchSize = [match sizeWithFont: font];
        UILabel *label = [[[UILabel alloc] initWithFrame: CGRectMake(origin.x,
origin.y, matchSize.width, matchSize.height)] autorelease];
        label.font = font;
        label.textColor = color;
        label.backgroundColor = background;
        label.text = match;
        [view addSubview: label];
    }
}
```

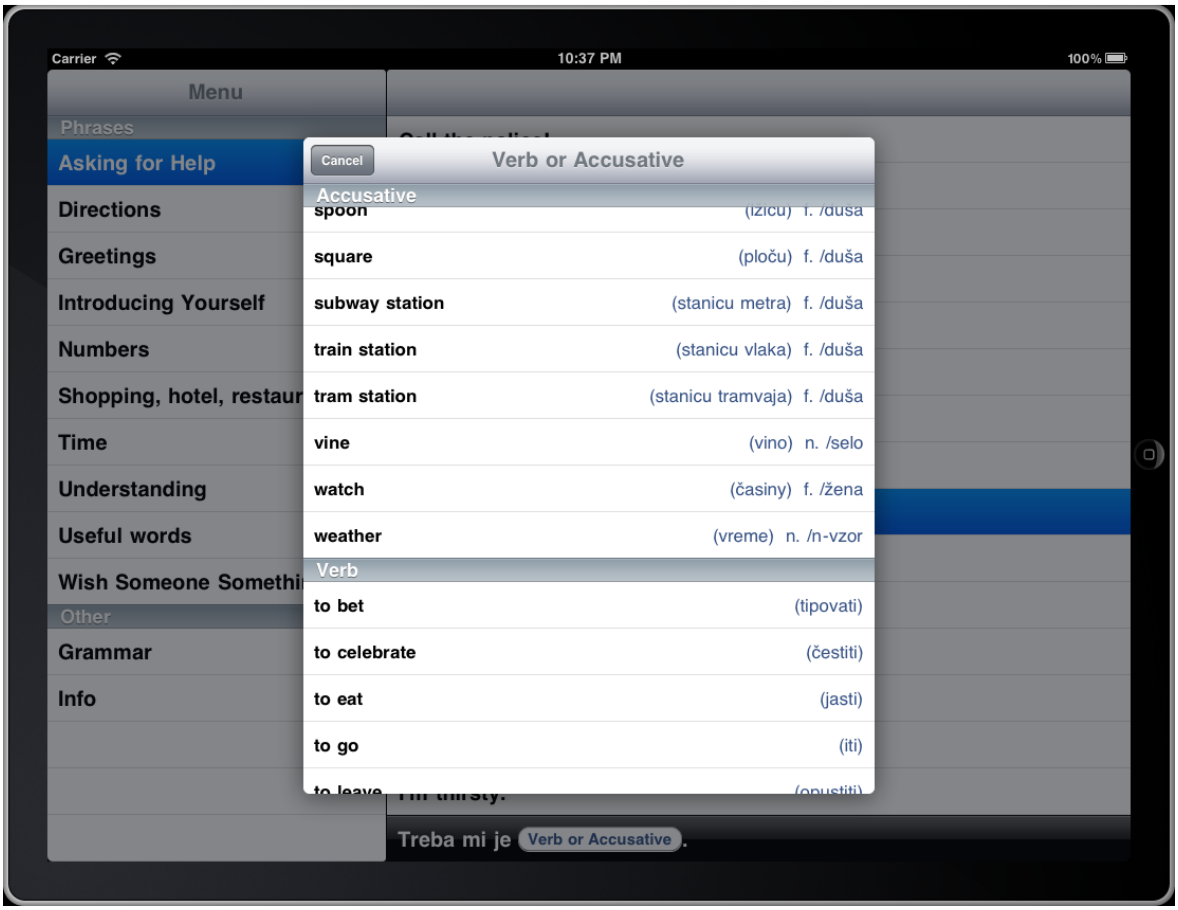
Stisk tlačítka pro doplnění tokenu vyvolá modální dialog se seznamem doplnitelných slov. Modální dialogy existují i pro iPhone, na iPadu však vypadají lépe díky prostoru, který jim je možno věnovat. Dialogy slouží k zablokování aplikace do doby, než uživatel provede nějakou volbu. Volba tokenu je přesně tím případem, kdy je takové chování žádoucí.

Průběh metody obsahuje:

- Inicializaci `TokenViewController`, což je podtřída `UITableViewController`, starající se o výpis slovíček,
- nastavení delegáta metod pro modální okno,
- programový výběr řádku a zobrazení nekompletního překladu, pokud bylo stisknuto tlačítko v tabulce - detekce probíhá prozkoumáním zaslané události a dohledáním řádku v tabulce podle souřadnic v rodičovském pohledu,
- předání identifikátoru tokenu a seznamu typů tokenů modálnímu dialogu,
- přidání navigační lišty s tlačítkem pro zrušení výběru slova,
- zobrazení modálního okna se stylem zobrazení `UIModalPresentationFormSheet` - toto zobrazení vytvoří malé okno přes cca jednu čtvrtinu plochy a zbytek plochy překryje poloprůhlednou vrstvou. Výsledkem je velmi elegantní vzhled.

Na iPhone je nutno řešit výběr tokenu pomocí vložení rámce do navigačního rámce, což způsobí nahrazení aktuálního pohledu pohledem s tokeny. Tento přístup je běžný u procházení hierarchických dat na malé ploše displeje iPhone, je užit například u zabudované aplikace iPod.

```
- (IBAction)tokenButtonClicked:(id)sender withEvent:(id)event{
    UIButton *button = (UIButton *)sender;
    button.highlighted = NO;

    // Create tokenController.
    TokenViewController *tokenController = [[[TokenViewController alloc] init]
autorelease];
    tokenController.delegate = self;

    // If not called from toolbar, set current UITableViewCell selected.
    if ([[button superview] class] != [UIToolbar class]) {
        CGPoint touchPoint = [[[event allTouches] anyObject] locationInView:
phraseTable];
        NSIndexPath *indexPath = [phraseTable indexPathForRowAtPoint:touchPoint];
        [phraseTable selectRowAtIndexPath:indexPath animated:NO
scrollPosition:UITableViewScrollPositionNone];
        [self tableView:phraseTable didSelectRowAtIndexPath:indexPath];
    }

    // Setup tokenController.
    tokenController.tokenName = [NSString stringWithFormat:@"|%i|", button.tag,
nil];
    NSArray *tokenTypes = [[_selectedPhrase objectForKey:@"tokens"]
objectForKey:tokenController.tokenName];
    tokenController.tokenTypes = tokenTypes;

    // Setup UINavigationController (contains Cancel button).
    UINavigationController *nav = [[[UINavigationController alloc]
initWithRootViewController:tokenController] autorelease];
    nav.modalPresentationStyle = UIModalPresentationFormSheet;
    nav.modalTransitionStyle = UIModalTransitionStyleFlipHorizontal;
    nav.navigationBar.topItem.title = [button titleLabel].text;

    // Show modal view.
    [self presentModalViewController:nav animated:YES];
}
```



Obrázek 16 - Okno s tokeny pro doplnění

U podstatných jmen je zobrazen i rod a vzor jako pomůcka při učení jazyka. Všechna slova jsou zároveň zobrazena včetně překladů ve tvaru, který bude v případě jejich vybrání použit pro doplnění vety v novoslověnštině. Výběr slova lze zrušit tlačítkem "*Cancel*" typu `UIToolbarItem` v `UIToolbar` v hlavičce rámce, pak se uživatel vrací animací k původní obrazovce. Stejná animace a zavření modálního dialogu se provádí i bezprostředně po volbě nějakého slova. Animace je zvolena tak, aby uživatel nepřehlédl změnu uživatelského rozhraní a zároveň ho zbytečně nerušila.

Po potvrzení tokenu je zavolána metoda delegáta modálního okna `wordSelected:forToken:`, která oznamuje volbu slova a identifikátor tokenu. Vybraná fráze je identifikována podle vybraného řádku v tabulce - díky volbě modálního panelu neměl uživatel šanci volbu změnit, identifikace je tak jednoznačná. Pokud ještě neexistuje slovník tokenů k dané frázi, je vytvořen, jinak do něj je přidán objekt slova s klíčem identifikátoru tokenu a je zavoláno překreslení daného řádku v tabulce frází

a je nasimulováno nové vybrání tohoto řádku uživatelem. Tím dojde k překreslení i fráze v `UIToolbar` ve spodku rámce a aktualizaci překladu o nově zvolený token.

```
- (void)wordSelected:(NSDictionary *)word forToken:(NSString *)tokenName {

    // Add selected token to collection.
    NSIndexPath *indexPath = [phraseTable indexPathForSelectedRow];
    NSMutableDictionary *tokens = [_selectedTokens objectForKey:indexPath];
    if (!tokens) {
        tokens = [[[NSMutableDictionary alloc] init] autorelease];
    }
    [tokens setObject:word forKey:tokenName];
    [_selectedTokens setObject:tokens forKey:indexPath];

    // Reload table and translation display.
    [phraseTable reloadRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
withRowAnimation:YES];
    [phraseTable selectRowAtIndexPath:indexPath animated:NO
scrollPosition:UITableViewScrollPositionNone];
    [self tableView:phraseTable didSelectRowAtIndexPath:indexPath];
}
```

6. Zhodnocení

Tvorba aplikace přinesla několik technických problémů, nicméně všechny byly díky zdrojům primární a sekundární literatury překonány. V dynamickém generování uživatelského rozhraní se ukázala být problematická zvláště striktní správa paměti v prostředí jazyka *Objective-C*.

Pomůcek pro výuku jazyků a překladačů existuje v prostředí AppStore pro iOS mnoho, nicméně většina trpí problémy s výkonem, ovladatelností, nebo kvalitou databáze s překlady. Předními aplikacemi jsou *TalkPad*, *Free Translator for iPad* a *iTranslator*. *Free Translator* trpí minimální slovní zásobou a ve většině případů těžko užitelnými frázemi. Navíc postrádá možnost doplňovat věty o konkrétní užití. *iTranslator* dovoluje překládat celé věty tak, jak si je uživatel sám napíše, což je rozhodně zajímavé, překlad však probíhá strojově a výsledkem je často věta, které je problém porozumět. *TalkPad*, který existuje pro celou řadu jazyků, nicméně, stejně jako *Free Translator*, trpí nedostatkem slovní zásoby. Aplikace Neoslavonic pro výuku novoslověnštiny a komunikaci v ní se snaží tyto nedostatky odstranit užitím kvalitních podkladů z webu, který komunitě kolem jazyka slouží již delší dobu a obsahuje tak databázi vět, které jsou opravdu užitečné, dále pak návrhem uživatelského rozhraní přesně podle doporučení firmy Apple, která je autorem systému iOS. Tato doporučení jsou stavěna na dlouholetých zkušenostech špičkových odborníků na průmyslový design a použitelnost, jejich dodržováním a rozšiřováním je tak zaručena jistá minimální kvalita aplikace, která splňuje nároky na použitelnou učební pomůcku.

Pro šíření aplikace je zvoleno prostředí AppStore, neboť v době tvorby aplikace a práce neexistuje, a patrně ani v budoucnu nebude existovat, jiný způsob, jak na zařízení se systémem iOS dostat aplikaci legálně bez pozbytí záruky. Existují sice ještě neautorizované obchody s aplikacemi pro jail-breaknuté [15] telefony (např. *Cydia*), prolomením ochrany ale uživatel ztrácí záruku na své zařízení, navíc i sama firma Apple varuje před neautorizovanými modifikacemi systému. Distribuce aplikace touto cestou by tak nebyla pro většinu uživatelů zajímavá. (Apple Inc., 2010)

[15] Telefony s prolomenou ochranou systému a s povoleným přístupem uživatele root.

7. Závěr

Podařilo se vytvořit unikátní aplikaci pro výuku mladého a perspektivního umělého slovanského jazyka, která v současné chvíli nemá obdoby. Využívá principu, který současné aplikace opomíjejí a dovoluje komunikovat jednoduchou formou i naprosto nezkušenému a neznalému uživateli. Vysokou přidanou hodnotou je značná podoba papírovým konverzacím, na které jsou uživatelé zvyklí, které jsou ale nepraktické pro reálné použití.

Aplikace byla představena nezainteresovaným uživatelům z autorova okolí, šlo o několik různých okruhů uživatelů s i bez hlubšího porozumění počítačům a iOS platformě, s i bez zálibě v cizích jazycích a s různými úrovněmi dosaženého vzdělání. Aplikace byla prezentována anonymně, aby byl z hodnocení vyloučen vztah k autorovi. Přijetí aplikace bylo veskrze kladné, pochopení ovládání bylo perfektní. Výtky lze shrnout do přání o širší pokrytí jazyka dalšími větami a tokeny. Tímto směrem tak jde i doporučení pro další rozvoj - lze doporučit doplňování slovníků frází i tokenů pro doplnitelné segmenty vět, které dovolí lepší komunikaci a zvýši užitečnost aplikace pro větší spektrum uživatelů. Doporučenými sekcemi může být například obchodní styk, základní právní pomoc, věty užitné při automobilové nehodě, či jiných nesnázích v cizí zemi - absence peněz ve správně měně, zdravotní potíže, seznamovací fráze na intimnější úrovni a další. Další alternativou pro rozvoj aplikace je přidání více jazyků, to ovšem na rozdíl rozšíření slovníku není prioritou.

V průběhu léta bude aplikace umístěna na AppStore firmy Apple, do té doby budou doplněny datové zdroje na základě prvotní zpětné vazby uživatelů.

8. Seznam použitých zdrojů

Apple Inc. 2012. Apple Special Event - March 7, 2012. *Apple Events.* [Online] 08. 03 2012. [Citace: 27. 03 2012.] http://www.apple.com/apple-events/march-2012/.

—. 2012. iOS Human Interface Guidelines. *iOS Developer Library.* [Online] 07. 03 2012. [Citace: 12. 03 2012.] http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html.

—. 2012. Join the crowd. *Apple Events.* [Online] 10. 03 2012. [Citace: 10. 03 2012.] http://www.apple.com/apple-events/.

—. 2008. NSNumber Class Reference. *Mac OS X Developer Library.* [Online] 02. 08 2008. [Citace: 10. 03 2012.] https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSNumber_Class/Reference/Reference.html.

—. 2011. NSString Class Reference. *Mac OS X Developer Library.* [Online] 10. 11 2011. [Citace: 10. 03 2012.] https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/nsstring_Class/Reference/NSString.html.

—. 2011. UISplitViewControllerDelegate Protocol Reference. *iOS Developer Library.* [Online] 10. 12 2011. [Citace: 12. 03 2012.] http://developer.apple.com/library/ios/#documentation/uikit/reference/UISplitViewControllerDelegate_protocol/Reference/Reference.html.

—. 2010. Unauthorized modification of iOS has been a major source of instability, disruption of services, and other issues. *Support.* [Online] 27. 09 2010. [Citace: 18. 03 2012.] http://support.apple.com/kb/ht3743.

CERN. 2012. Projekt LA08015 - Spolupráce ČR s CERN (2008-2012, MSM/LA). *Informační systém výzkumu, experimentálního vývoje a inovací.* [Online] 2012. [Citace: 27. 03 2012.] http://www.isvav.cz/projectDetail.do;jsessionid=AA745BC61F76B05FD1F74CB522EC512E?rowId=LA08015.

HAMACHER, MARCUS. 2011. Mobile Quick Facts: iPhone users love updates. *apprupt.* [Online] 28. 06 2011. [Citace: 15. 03 2012.]

http://www.apprupt.com/en/blog/2011/06/28/mobile-quick-facts-apple-user-love-updates/.

KAHNEY, LEANDER. 2009. *Jak myslí Steve Jobs.* Brno : Computer Press, a.s., 2009. ISBN 978-80-251-2794-0.

KOCHAN, STEPHEN G. 2010. *Objective-C 2.0.* Brno : Computer Press, a.s., 2010. ISBN 978-80-251-2654-7.

LAMARCHE, JEFF a MARK, DAVE. 2010. *iPhone SDK.* Brno : Computer Press, a.s., 2010. ISBN 978-80-251-2820-6.

MERUNKA, VOJTECH. 2009. *Jazyk novoslovienskij.* České Budějovice : Nová Forma, 2009. ISBN 978-80-87313-51-0.

—. 2012. Neoslavonic language tutorial. *Novosloviensky Jezyk.* [Online] 09. 03 2012. [Citace: 12. 03 2012.] https://sites.google.com/site/neoslavonictutorial/introduction.

SHILOV, ANTON. 2012. Samsung Hints on Smaller Apple iPad with 7.85" Screen. *X-bit labs: PC Hardware News, Reviews and Benchmarks.* [Online] 14. 03 2012. [Citace: 15. 03 2012.] http://www.xbitlabs.com/news/multimedia/display/20120314232839_Samsung_Hints_on_Smaller_Apple_iPad_with_7_85_Screen.html.

STEVENSON, SCOTT. 2008. Learn Objective-C. *Cocoa Dev Central.* [Online] 16. 03 2008. [Citace: 12. 03 2012.] http://cocoadevcentral.com/d/learn_objectivec/.

—. 2005. Saving Cocoa Application Data. *CocoaDevCentral.com.* [Online] 2005. [Citace: 12. 03 2012.] http://cocoadevcentral.com/articles/000084.php.

Touch display - a novel input/output device for computers. **Johnson, E.A. 1965.** 8, místo neznámé : Electronics Letters, 1965, Sv. 1. ISSN 00135194.

9. Seznam obrázků

10. Seznam tabulek

11. Přílohy

Latin	ASCII	Cyrillic	IPA	example	name
a	a	а	[a]	alphabet, but	a
b	b	б	[b]	beer, bird	be
c	c	ц	[ts]	cats	ce
č	ch (cz)	ц	[tʃ]	chimney, bench	če
d	d	д	[d]	date, do, odd	de
dj	dj	дь	[ɟ]	duty	dje
dž	dzh	дж	[dʒ]	juice, gymnastics	dže
e	e	е	[ɛ]	bed, yes	e
f	f	ф	[f]	photo, leaf	ef
g	g	г	[g]	go, get	ge
h (ch)	h (kh)	х	[x]	house, ham	xa
i	i	и	[i]	city, see, meat	i
ie (ě)	ie	ѣ (е)	[jɛ]	yes, cavalliero (Spanish)	jatj
j	j	ј (й)	[j]	yes, you	je
ju	ju	ю	[jʊ]	you, cute	ju
k	k	к	[k]	kill, skin	ka
l	l	л	[l]	like, loop	el
lj	lj	ль	[ʎ]	cavalliero (Spanish)	elj
m	m	м	[m]	man, ham	em
n	n	н	[n]	no, nose	en
nj	nj	нь	[ɲ]	new, cognac, mañana (Spanish)	enj
o	o	о (ѡ)	[ɔ]	law, talk, no (Spanish)	o
p	p	п	[p]	pen, spin	pe
q	q	к	[k]	kill, skin	kve
r	r	р	[r]	sombrero (Spanish)	er
s	s	с	[s]	see, city, pass	es
š	sh	ш	[ʃ]	she, shame	eš
št	sht	щ	[ʃt]	š + t	šta
t	t	т	[t]	to, motor	te
th	th	ѳ	[θ]	thing, teeth	the
tj	tj	ть	[c]	Tuesday, opportunity	tje
u	u	у	[ʊ]	put, good	u
v	v	в	[v]	voice, have	ve
w	w	в	[v]	voice, have	dvojne ve
x	x	кс (ѯ)	[ks]	sex, six	eks
y	y	ы (ѵ)	[i]	city, see, meat	ypsilon
z	z	з	[z]	zoo, zombie, rose	zet
ž	zh	ж	[ʒ]	pleasure, Jean (French)	žet
'	'	ъ	[ə̆]	short sound from burn, earth, bird	jor

https://sites.google.com/site/neoslavonictutorial/lessons/1

Tabulka 1 - Přehled abecedy novoslověnštiny

Mužsky vzor, tv'rdy vzor BRAT, GRAD						
	j.č. (singular)		mn.č. (plural)		dv.č. (dual)	
1.p. (nom.)	brat	grad	brat-i	grad-y	brat-a	grad-a
2.p. (gen.)	brat-a	grad-a	brat-ov	grad-ov	brat-u	grad-u
3.p. (dat.)	brat-u	grad-u	brat-am	grad-am	brat-ama	grad-ama
4.p. (akuz.)	brat-a	grad	brat-y	grad-y	brat-a	grad-a
5.p. (vok.)	brat-e	grad-e	brat-i	grad-y	brat-a	grad-a
6.p. (lok.)	brat-u	grad-u	brat-ah	grad-ah	brat-u	grad-u
7.p. (instr.)	brat-om	grad-om	brat-ami	grad-ami	brat-ama	grad-ama

Tabulka 2 - Skloňování mužského rodu podle vzorů BRAT, GRAD

Mužsky vzor, mekky vzor MUŽ, KRAJ						
	j.č. (singular)		mn.č. (plural)		dv.č. (dual)	
1.p. (nom.)	muž	kraj	muž-i	kraj-i	muž-a	kraj-a
2.p. (gen.)	muž-a	kraj-a	muž-ev	kraj-ev	muž-u	kraj-u
3.p. (dat.)	muž-u	kraj-u	muž-am	kraj-am	muž-ama	kraj-ama
4.p. (akuz.)	muž-a	kraj	muž-e	kraj-e	muž-a	kraj-a
5.p. (vok.)	muž-u	kraj-u	muž-i	kraj-i	muž-a	kraj-a
6.p. (lok.)	muž-u	kraj-u	muž-ah	kraj-ah	muž-u	kraj-u
7.p. (instr.)	muž-em	kraj-em	muž-ami	kraj-ami	muž-ama	kraj-ama

Tabulka 3 - Skloňování mužského rodu podle vzorů MUŽ, KRAJ

Ženský rod, tvʻrdy vzor ŽENA			
	j.č. (singular)	**mn.č. (plural)**	**dv.č. (dual)**
1.p. (nominativ)	žen-a	žen-y	žen-ie
2.p. (genitiv)	žen-y	žen-ʻ	žen-u
3.p. (dativ)	žen-ie	žen-am	žen-ama
4.p. (akuzativ)	žen-u	žen-y	žen-ie
5.p. (vokativ)	žen-o	žen-y	žen-ie
6.p. (lokal)	žen-ie	žen-ah	žen-u
7.p. (instrumental)	žen-oj	žen-ami	žen-ama

Tabulka 4 - Skloňování ženského rodu podle vzoru ŽENA

Ženský rod, mekky vzor DUŠA (BURJA, ZEMJA, ...)			
	j.č. (singular)	**mn.č. (plural)**	**dv.č. (dual)**
1.p. (nom.)	1.p. (nominativ)	duš-a	duš-e
2.p. (gen.)	2.p. (genitiv)	duš-e	duš-ej
3.p. (dat.)	3.p. (dativ)	duš-i	duš-am
4.p. (akuz.)	4.p. (akuzativ)	duš-u	duš-e
5.p. (vok.)	5.p. (vokativ)	duš-e	duš-e
6.p. (lok.)	6.p. (lokal)	duš-i	duš-ah
7.p. (instr.)	7.p. (instrumental)	duš-ej	duš-ami

Tabulka 5 - Skloňování ženského rodu podle vzoru DUŠA

Stredny rod, tvʻrdy vzor SELO			
	j.č. (singular)	**mn.č. (plural)**	**dv.č. (dual)**
1.p. (nominativ)	sel-o	sel-a	sel-ie
2.p. (genitiv)	sel-a	sel-ʻ	sel-u
3.p. (dativ)	sel-u	sel-am	sel-ama
4.p. (akuzativ)	sel-o	sel-a	sel-ie
5.p. (vokativ)	sel-o	sel-a	sel-ie
6.p. (lokal)	sel-u	sel-ah	sel-u
7.p. (instrumental)	sel-om	sel-ami	sel-ama

Tabulka 6 - Skloňování středního rodu podle vzoru SELO

Stredny rod, mekky vzor POLJE (CESARSTVIJE, ...)			
	j.č. (singular)	**mn.č. (plural)**	**dv.č. (dual)**
1.p. (nominativ)	polj-e	polj-a	polj-i
2.p. (genitiv)	polj-a	polj-ej	polj-u
3.p. (dativ)	polj-u	polj-am	polj-ama
4.p. (akuzativ)	polj-e	polj-a	polj-i
5.p. (vokativ)	polj-e	polj-a	polj-i
6.p. (lokal)	polj-u	polj-ah	polj-u
7.p. (instrumental)	polj-em	polj-ami	polj-ama

Tabulka 7 - Skloňování středního rodu podle vzoru POLJE