**TECHNICKÁ UNIVERZITA V LIBERCI**
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: X2612 – Elektrotechnika a informatika

Studijní obor: 1802T007-Informační technologie

# UTMJ OBU – Jádro a uživatelské rozhraní

# UTMJ OBU – Core a user interface

**Diplomová práce**

Autor: **Jan Dytrych**

Vedoucí práce: Ing. Jindřich Žďánský, Ph.D. tituly

**V Liberci 15.5.2011**

**Zde bude vložen originál zadání DP!**

**Tato stránka není součástí výtisku diplomové práce!**

# Prohlášení

Byl (a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval (a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

## Poděkování

Tímto bych chtěl poděkovat Ing. Jindřichu Žďánskému, Ph.D. za odborné vedení mé diplomové práce a za poskytnuté konzultace a jeho čas. Dále bych chtěl také poděkovat Ing. Tomáši Tvrszkému jednateli skupiny společností Telematix, který mi umožnil prezentovat projekt UTMJ OBU jako diplomovou práci.

## Abstrakt

Diplomová práce se zabývá vývojem aplikace v rámci projektu MPO nazvaného „Výzkum systémových požadavků a architektury pro univerzální telematickou vozidlovou jednotku“. Univerzální vozidlovou jednotkou se rozumí hardware, který je instalován do pozemních dopravních prostředků. Aplikace pro univerzální telematickou jednotku je nazvána OBU a používá nejmodernější technologie, které jsou v době vývoje celého projektu k dispozici. Mezi tyto technologie patří GPS, GSM, DSRC a jiné. Vzhledem k tomu, že na projektu se podílelo více subjektů a také lidí, je diplomová práce zaměřena hlavně na součásti jádra aplikace a uživatelského rozhraní. Architekturu aplikace, která je v práci popsána, byla navržena tak, aby dovolila hlavně flexibilitu aplikace. Architektura aplikace je rozdělena na tři hlavní součásti, které se dále dělí. Mezi tyto součásti patří jádro aplikace, poskytovatelé a moduly. Jádro aplikace v této architektuře zastává funkci centrálního bodu, který dovoluje komunikaci mezi ostatními součástmi aplikace. Jádro aplikace je přesně dané základními rozhraními, tudíž je neměnné. Poskytovatelé v aplikaci poskytují data nebo služby. Poskytovatelé v rámci aplikace mohou být různí a jsou určovány v konfiguraci aplikace. Moduly v aplikaci zpracovávají data od poskytovatelů, nebo z databáze a je také možné načíst libovolné množství podle konfiguračních údajů. Diplomová práce se na konci zabývá problémem uživatelského rozhraní, popisuje vyvinuté řešení, jeho možnosti a strukturu včetně vlastního návrháře běžícího na počítači.

## Klíčová slova

UTMJ, OBU, Telematické aplikace, Windows CE, .NET Compact Framework

## Abstract

The graduation work deals with development of application in resource constrained devices that are placed in land vehicles. This project has been founded by Ministry of Industry and commerce of republic Czech. The name of project was “Development of system requirements and architecture for universal telematic vehicle unit. By universal telematic vehicle unit we mean hardware that is installed in land vehicle (car, truck, etc). A software application that runs on hardware is called in means of this project OBU (on board unit). We use technologies like GPS, GSM, DSRC and others to benefit from those in our project. The project itself has been developed by few companies, subject and people. This graduation work deals only with software application’s core and user interface. Application architecture is also described with function aspects and the application is divided into tree common groups. First group is application’s core that consists of few core services that makes whole pieces of application connected and working as one part. The core is defined by interfaces that contain constant features. Other parts of application are providers and modules. Providers are parts of application that provide some sort of data or service. These data or services are consumed by modules that handle data or services and do their own work like storing data to database or sending data to remote servers. Modules and providers are modular, so they are loaded by application in runtime and they are loaded by core by information provided in application’s configuration. At the end of work the user interfaces and all the related topics are told, like what is the user interfaces, what is the implementation of user interface in OBU application and the designer that has been developed and used to design user interace of entire OBU application.

## Keywords

UTMJ, OBU, Telematic applications, Windows CE, .NET Compact Framework

# Obsah

# Seznam obrázků



# Seznam tabulek

# Seznam zkratek

| | | |
|---|---|---|
| API | Aplikační rozhraní pro programování | Application programming interface |
| CAN | Sběrnice CAN pro průmysl | Controler area network |
| CLR | Společné běhové prostředí .NET | Common language runtime |
| DSRC | Vyhrazená komunikace pro krátkou vzdálenost | Dedicated short range communication |
| EFC | Elektronické placení mýta | Electronic fee calulation |
| GNSS | Globální satelitní síť | Global |
| GPIO | Obecné vstupy/výstupy | General purpose input/output |
| GPRS | Rozšíření GSM o rychlejší datové přenosy | General packet radio service |
| GPS | Globální poziční systém | Global positioning system |
| GSM | Globální systém pro mobilní komunikaci | Global system for mobile communication |
| GUID | Globální unikátní identifikátor | Globaly unique identificator |
| HMI | Rozhraní mezi člověkem za strojem | Human machine interface |
| HW | Hardware | Hardware |
| MPO | Ministerstvo průmyslu a obchodu ČR | Ministry of industry and commerce |
| OBU | Palubní jednotka | On board unit |
| RAM | Paměť s náhodným přístupem | Random access memory |
| RAS | Služba pro vzdálený přístup | Remote access service |
| ROM | Paměť pouze pro čtení | Read only memory |
| UI | Uživatelské rozhraní | User interface |
| USB | Univerzální sériová sběrnice | Universal serial bus |
| UTMJ | Univerzální telematická mobilní jednotka | Universal telematic mobile unit |
| VIN | Identifikace pozemního vozidla | Vehicle identification number |
| VRN | Státní poznávací značka | Vehicle registration number |
| XML | Rozšířitelný značkovací jazyk | eXtensible markup lanaguage |

# Úvod

## Předmluva

V posledních několika letech můžeme vidět vysoký narůst využití telematických aplikací a jejich celkový rozvoj a vývoj. Asi nejvíce je tento trend vidět v oblasti navigací, kdy dochází k jejich mohutnému rozšíření ať už pro osobní použití, tak pro použití komerční. Milióny řidičů již dnes používá navigaci pro spolehlivé a přesné určení své polohy, díky systému GPS provozovaného armádou USA, a také pro snadnější orientaci a navádění na požadovaný cíl své cesty. Navigace se v poslední době stává více propojená s internetem. Toto je možné díky větší penetraci mobilního internetu a také je na trhu více zařízení, které obsahují navigaci jako doplněk. Bavím se zde například o mobilních telefonech, které často již od výrobce obsahují mapy a GPS modul, případně obsahují mapy, které se stahují přímo z internetu od poskytovatelů jakým je například společnost Google a jejich aplikace Google Maps. Využití telematických poznatků nekončí pouze u prostého navigování na místo určení, ale také přidává k navigaci důležitý prvek v podobě dynamických informací o dopravě a počasí. Tyto informace dovolují jistým způsobem řídit dopravu a předcházet případným dopravním kolapsům nebo nehodám. V případě kdy dojde k nehodě, telematické aplikace nabízejí technologii, která dokáže informovat složky záchranného systému o této skutečnosti a také jim předat nějaké další informace, které můžou zefektivnit zásah záchranného systému a také informovat ostatní řidiče o této skutečnosti. Mezi další telematické aplikace může zařadit systém pro sledování vozového parku, jehož hlavním účelem je sledovat, plánovat a efektivně využívat vozový park společnosti.

## Cíl práce

Cílem mé práce bylo navrhnout kompletní architekturu aplikace pro palubní jednotku společnosti Honeywell, realizovat tento návrh v podobě funkčního vzorku aplikace, která správně spolupracuje s HW a také správně komunikuje s uživatelem aplikace. Základním požadavkem na architekturu aplikace byla modularita, spolehlivost, přesnost, dostupnost, spojitost a snadné rozšíření vstupů a výstupů aplikace. Vzhledem k tomu, že palubní jednotka disponuje dotykovým displejem, bylo součástí požadavků na aplikaci vývoj uživatelského rozhraní s podporou dotykového displeje. V této práci se nejdříve seznámíme s operačním systémem, který je součástí HW společnosti Honeywell, dále se seznámíme s technologií. NET Compact Framework. Po nabití znalostí prostředí, ve kterém bude aplikace fungovat, si probereme návrh architektury aplikace, její aspekty, výhody a nevýhody. Poté se podíváme na popis jednotlivých rozhraní aplikace, modularitu aplikace, uživatelské rozhraní a poskytovatele GPS.

## Obor telematika

Co je telematika? Mnoho lidí si pod tímto termínem nepředstaví nic, hlavně z důvodu, že telematika je relativně nový vědní obor, který vznikl spojením poznatků ze dvou starších oborů informatika a telekomunikace. Telematika se značně opírá o nové poznatky a technologie posledních několika desítek let, mezi ty nejdůležitější patří systémy GNSS, mezi kterými jmenujme nejznámější GPS, GLONASS a Galileo. Tyto systémy radikálně změnily

možnosti lidstva určit svojí polohu na Zemi, díky těmto technologiím je možné rychle a snadno lokalizovat polohu postiženého člověka kdekoliv na Zemi a bez zbytečného otálení a hledání mu poskytnout pomoc. V dnešní době je totiž čip schopný přijímat GPS a provést podle něj lokalizaci tak levný a malý, že je možné jej přidat do libovolného zařízení, například mobilní telefony v poslední době vyrobené již tento čip obsahují, a tudíž jsou schopny určit svojí polohu s přesností na metry. Dalším důležitým aspektem je dostupnost internetu, zejména toho mobilního. V dnešní době již není problém v mobilním zařízení (telefon, mobilní terminál, …) být připojen k internetu celý den za relativně rozumné peníze. Tyto trendy určily také hlavní směr, kterým se obor telematika bude směřovat, asi nejvíce známým směrem je navigace, dále systémy pro automatické tísňové volání pod názvem eCall. Mezi další směry, kterými se telematika zabývá, jsou EFC (mýtné) a sledování vozidlového parku neboli fleet. Cílem projektu UTMJ OBU jsou právě tyto směry.

## UTMJ OBU

Zkratka UTMJ znamená „Univerzální telematická mobilní jednotka“ a OBU je zkratka z anglického „On board unit“ neboli palubní jednotka. Tento projekt byl vyhlášen Ministerstvem pro průmysl a obchod neboli MPO jako „Výzkum systémových požadavků a architektury pro univerzální telematickou vozidlovou jednotku“. Na celém projektu se podílely celkem tři subjekty, prvním byla univerzita FD ČVUT v Praze, skupina společností Telematix, která zaštiťovala vývoj software společně s analýzou, a společnost Honeywell, která se soustředila výhradně na hardwarovou část projektu.

Původní znění cílů MPO je následují:

„Výzkum a vývoj v oblasti systémových požadavků a architektury inteligentní univerzální telematické jednotky. Inovativnost řešení telematické jednotky bude spočívat v tom, že jednotka bude realizovat sadu vybrané množiny přesně definovaných funkcí operujících na množině definovaných databází, pravidel a parametrů. Všechny tyto unifikované komponenty budou zvoleny a optimalizovány na základě analýz moderními nástroji systémového inženýrství tak, aby pokryly široký okruh konkrétních telematických aplikací. Z hlediska hardwarového bude pak architektura splňovat požadavky použitím moderních perspektivních rozhraní a komunikačním i jiných standardů a dále produkt s dlouhou morální dobou života použitelná v široké oblasti telematických aplikací nejen v oblasti osobní a nákladní dopravy, ale například v infrastruktuře záchranných systémů, branně bezpečnostních složek, zemědělství i v jiných odvětví národního hospodářství.“

Na základě těchto požadavků byly analyzovány stěžejní systémové požadavky projektu, které byly důležité pro vypracování projektu.

## Systémové požadavky UTMJ OBU

Na základně analýzy byly specifikovány hlavní systémové požadavky projektu UTMJ OBU, které výrazně ovlivnily návrh systémové architektury aplikace.

### Flexibilita

Požadavek flexibilita vznáší požadavek na architekturu aplikace, aby byla v maximální míře modulární. Termínem modulární myslíme variabilitu vstupů a výstupů aplikace, kdy je možné jednoduchým zásahem do konfigurace aplikace změnit vstup, případně jeho zpracování.

### Bezpečnost dat

Vzhledem k tomu, že aplikace může uchovávat citlivé informace o poloze vozidla i několik dní zpět je kladen velký důraz na bezpečnost uchovaných dat. Vzhledem k tomu, že aplikace dále data přenáší na server je také důležité tato data zabezpečit i při přenosu na server, kdy je zranitelnost dat nejvyšší.

### Integrita

Integrita je veličina, která udává pravděpodobnost s jakou je uživatel do stanovené doby informován o nestandardním fungování jednotky.

## Funkční požadavky UTMJ OBU

Při analýze požadavků projektu UTMJ OBU byly mezi systémovými požadavky analyzovány i požadavky funkční, které reflektují hlavně prostředí, ve kterém bude aplikace provozována. Hlavními požadavky jsou vykonávání funkcí eCall, Fleet, Toll a Navigace. Mezi funkční požadavky na aplikaci UTMJ OBU můžeme zařadit:

### Spolehlivost

Spolehlivost aplikace je dána její schopností plnit požadované funkce (eCall, Fleet, Toll, Navigace) v určeném časovém úseku za definovaných podmínek (elektrické a environmentální).

### Přesnost

Přesnost aplikace je definována jako procento shody mezi skutečnou a naměřenou hodnotou funkce, parametru nebo procesu.

### Dostupnost

Dostupnost aplikace je definována jako procento času, po který jsou všechny funkce aplikace plně funkční. Toto procento odpovídá pravděpodobnosti, že jsou současně splněny požadavky na přesnost, integritu a spojitost.

### Spojitost

Spojitost je veličina určující schopnost aplikace plnit požadované funkce bez neočekávaného přerušení během definovaného časového intervalu.

Nyní byly shrnuty obecné požadavky na UTMJ OBU systémového i funkčního charakteru. Dále si řekneme něco o konkrétních požadavcích na aplikaci, které mohou být i hardwarového charakteru, které ovšem značně ovlivňují samotnou aplikaci.

## Požadavky na UTMJ OBU

Požadavky byly při analýze rozděleny do několika kategorií. Pro lepší přehlednost je uvedena tabulka požadavků.

**Tabulka 1 - Požadavky na aplikaci UTMJ OBU**

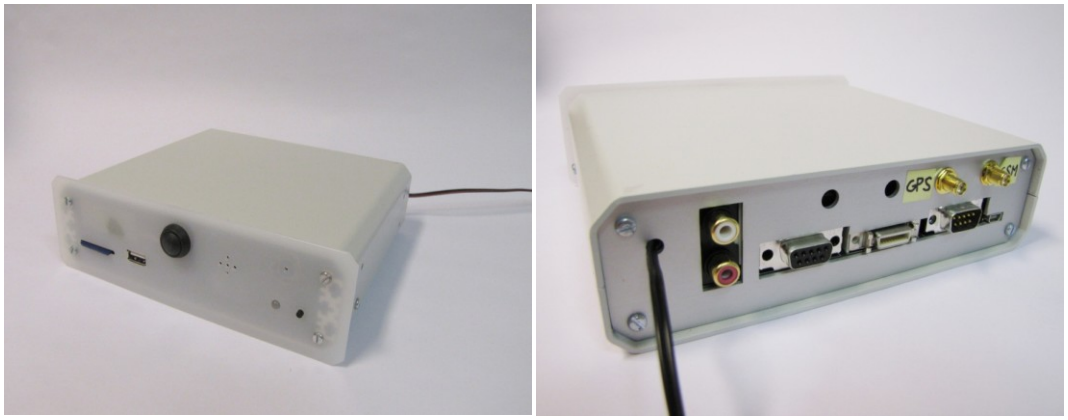
| | |
|---|---|
| **Architektura** | |
| Architektura hardware a software je natolik otevřená, aby bylo možno snadno jednotku uzpůsobit i pro další aplikace. | vyžadováno |
| Součástí programového vybavení jednotky jsou i univerzální funkce, databáze, procesy a pravidla, které umožní snadné naprogramování dodatečných aplikací. | vyžadováno |
| **Autorizace, Autentifikace** | |
| Jednotka obsahuje metody umožňující autentifikaci a autorizaci služeb/aplikací/uživatelů přistupujících k datům uloženým v jednotce | vyžadováno |
| Data uložena v jednotce jsou uspořádána hierarchicky dle míry soukromí, které vyžadují. | vyžadováno |
| **Identifikace** | |
| Jednotka v sobě uchovává UVI, UOBU-ID, VIN, VRN | vyžadováno |
| Údaje UVI, UOBU-ID, VIN, VRN lze v jednotce změnit pouze jednou – při instalaci autorizovanou osobou | vyžadováno |
| Jednotka dává údaje UVI, UOBU-ID, VIN, VRN k dispozici aplikacím | vyžadováno |
| Údaje UVI, UOBU-ID, VIN, VRN jsou zpřístupněny aplikaci až po úspěšné autorizaci a autentifikaci. | |
| **Interface (rozhraní)** | |
| Jednotka obsahuje externě přístupné datové rozhraní umožňující přístup autorizovaných osob k informacím uložených uvnitř OBU a jejich změny. | vyžadováno |
| Jednotka obsahuje externě přístupné datové rozhraní umožňující přístup autorizovaných osob k informacím uložených uvnitř OBU a jejich změny. | vyžadováno |
| Jednotka obsahuje externě přístupné datové rozhraní umožňující nahrávání nebo aktualizování software. | vyžadováno |
| HMI jednotky vhodným způsobem kombinuje chod více aplikací najednou a zároveň řidiče nepřetěžuje svou složitostí. Kritické aplikace mají přednost (výraznější barvy, velikost tlačítek, jednoduchost ovládání apod.) | vyžadováno |
| **Koexistence aplikací** | |
| Hardware i software jednotky umožňuje nerušený chod libovolné kombinace vybraných aplikací | vyžadováno |
| Procesy vyvolané aplikací eCall mají za každých okolností maximální prioritu.<br>Z hlediska priorit je další aplikací v pořadí Toll. Poslední jsou obecně komerční služby (navigace, fleet) | |
| **Komunikace** | |
| Jednotka podporuje tyto bezdrátové komunikační technologie: DSRC, GPRS a GSM | vyžadováno |
| Jednotka je otevřena dalším komunikačním technologiím: Wifi, Bluetooth, WiMax, IrDA, apod. | vyžadováno |
| **Lokalizace** | |
| Jednotka je schopna přijímat a zpracovávat navigační signály systému GPS | vyžadováno |
| **Ostatní** | |
| Jednotka podporuje několik módů s různou spotřebou a funkcionalitou | nutné |

## Hardware UTMJ OBU

[5] Prototyp jednotky UTMJ OBU vyvinula společnost Honeywell, která je celosvětovým výrobcem komponentů pro letecký průmysl, dále se zabývá oblastmi, jako je řízení budov, průmyslových procesů.

Jednotka se skládá z dvou hlavních součást HMI (human machine interface), které obsahuje display s rozlišením 640x480. Display je dotykový, tudíž reaguje na dotyky prstem uživatele. Display dále obsahuje dvě tlačítka pro obecné využití aplikací.

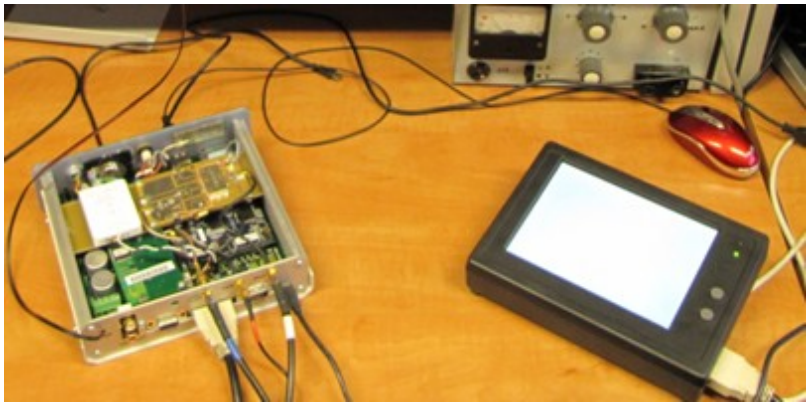
Druhou částí je samotná jednotka UTMJ, která obsahuje čtečku SD karet, USB 2.0 host port, tlačítko Panic, GPS a GSM výstup, CAN port, HMI port, COM port, audio výstup, stavová LED a tlačítko ON/OFF.




Obrázek 1 - Hardware UTMJ

Jednotka dále obsahuje procesor ARMv5 (624 Mhz), 64 MB RAM, GSM/GPRS modem Siemens AC35, GPS přijímač Honeywell a DSRC modul. Jednotka přímo podporuje sběrnice SPI, I2C, CAN a rozhraní GPIO. Jednotka má přeinstalovaný operační systém Windows CE 6.0 R2, který byl společností Honeywell upraven a dovybaven ovladači pro tento hardware.

Ze zkušeností s použitým hardware musím konstatovat, že většina komponentů fungovala při ladění a testovaní bez větších potíží, pouze modem Siemens občas špatně komunikoval.



Obrázek 2 - Jednotka UTMJ a HMI

## Vývoj aplikace pro UTMJ OBU

Na začátku projektu UTMJ OBU byla provedena analýza, která měla za úkol vhodně zvolit operační systém, který poběží na hardwarové jednotce, dále bylo potřeba zvolit vývojové prostředí a jazyk ve kterém bude psána aplikace. Po zvážení všech možných kandidátů, systémových a funkčních požadavků byl analytiky zainteresovaných společností vybrán operační systém Windows CE 6.0. Poté co byl zvolen operační systém začal vývoj samotného hardware a ovladačů pro Windows CE 6.0. Po několika verzích hardware, který byl postupně vylepšován, a rozšiřován do finální podoby začala být aktuální část vývoje samotné aplikace OBU. Vzhledem k tomu, že některé části aplikace OBU již byly dříve napsány pro jiné projekty v jazyce C# a na platformě .NET Compact Framework, byl vybrán právě jazyk C# a platforma .NET Compact Framework. Dalším ne méně důležitým faktorem pro výběr .NET Compact Frameworku a jazyka C# byly i výhody samotného spojení operačního systému Windows CE a .NET Compact Framework. Mezi tyto výhody bylo rychlé nasazení aplikace, snadná správa aplikace, vývojové prostředí Visual Studio.NET a další nástroje pro vývoj na zařízeních s operačním systémem Windows CE.

### Operační systém Windows CE

Operační systém Windows CE je plně 32bitový operační systém, který byl navržen pro běh na zařízeních s malými rozměry a s důrazem na jejich mobilitu a energetickou nenáročnost. Operační systém je vyvíjen a licencován společností Microsoft výrobcům hardware, kteří nemusí vytvářet nový operační systém pro svůj hardware. Systém Windows CE v dodávané verzi od společnosti Microsoft je navržen pro běh na platformách postavených na procesorech ARM, MIPS, SH4 a x86. Protože je operační systém Windows CE hlavně určen pro malá, energeticky nenáročná zařízení různého zaměření je systém možné upravit podle potřeb konkrétního hardware. Z tohoto důvodu je možné si operační systém „poskládat“ z různých komponent, kdy základní jádro systému a základní služby mají 300 kB a přidáním GUI a různých aplikací může systém narůst až do několika MB.

Operační systém Windows CE je oproti klasickému PC systému dodáván a upravován přímo výrobcem pro konkrétní zařízení. Není tudíž možné na libovolné zařízení instalovat libovolnou verzi systému, ale pouze tu verzi dodávanou výrobcem. Což může být nevýhoda, ale i výhoda zároveň. Hlavní nevýhodu vidím v závislosti na dodavateli hardware, kdy většinou není možné nainstalovat na hardware jiný operační systém než jím dodávaný. Na druhou stranu vidím výhodu v možnosti optimalizace systému na konkrétní hardware a tím i jeho efektivní využití. Operační systém má také dostupné zdrojové kódy pro výrobce hardwaru nebo pro vlastníka aplikace Platform builder, což umožňuje výrobcům provést různá vylepšení a úpravy samotného systému podle daného hardware. Platform buidler je určen k vytváření „image“ operačního systému, který se pak nahraje do paměti ROM zařízení odkud je spouštěn hardwarovým zavaděčem neboli „boot loader“. Díky variabilitě operačního systému Windows CE se s ním lze setkat v různých neočekávaných rolích, jako je například robotický vysavač, spíše typickým nasazení OS Windows CE jsou navigační zařízení tzv. „PND“, kde je tento systém dominantní. Další využití systému jsou například různé průmyslové terminály, sledování výroby, docházkové terminály atp.

## Historie operačního systému Windows CE

První verze operačního systému Windows CE byla na trh uvedena již v roce 1996, tehdy byla uvedena verze Windows CE 1.0, která byla po necelém roce nahrazena další verzí Windows CE 2.0, která přinesla hlavně podporu multimedií, internetu (TCP/IP, RAS, …). V roce 2000 byla uvedena verze Windows CE 3.0, která přidala podporu pro plnou emulaci systému, pro zjednodušení vývoje a také jeho zlevnění. Nebylo totiž nutné pro každého vývojáře pořizovat daný hardware, ale bylo možné jej emulovat na PC vývojáře. V roce 2004 přišla revoluce v podobě Windows CE verze 5.0. Hlavní revolucí bylo dodávání zdrojových kódů operačního systému za účelem úpravy systému výrobci hardware, což podstatě zlepšilo adaptovatelnost operačního systému na hardware.

Poslední verzi Windows CE je verze 6.0, která se vyznačuje kompletně přepsaným jádrem, zvýšení počtu zároveň běžících procesů a také rozšíření v podobě alokace až 2 GB RAM na proces. Předchozí verze Windows CE 5.0 podporovala pouze 32 MB. Dalším vylepšením ve Windows CE 6.0 byla podpora nových souborových systémů jako je exFAT, který je určen pro ukládání velkých souborů, nejčastěji multimediálních jako jsou videa, záznam zvuku atp.

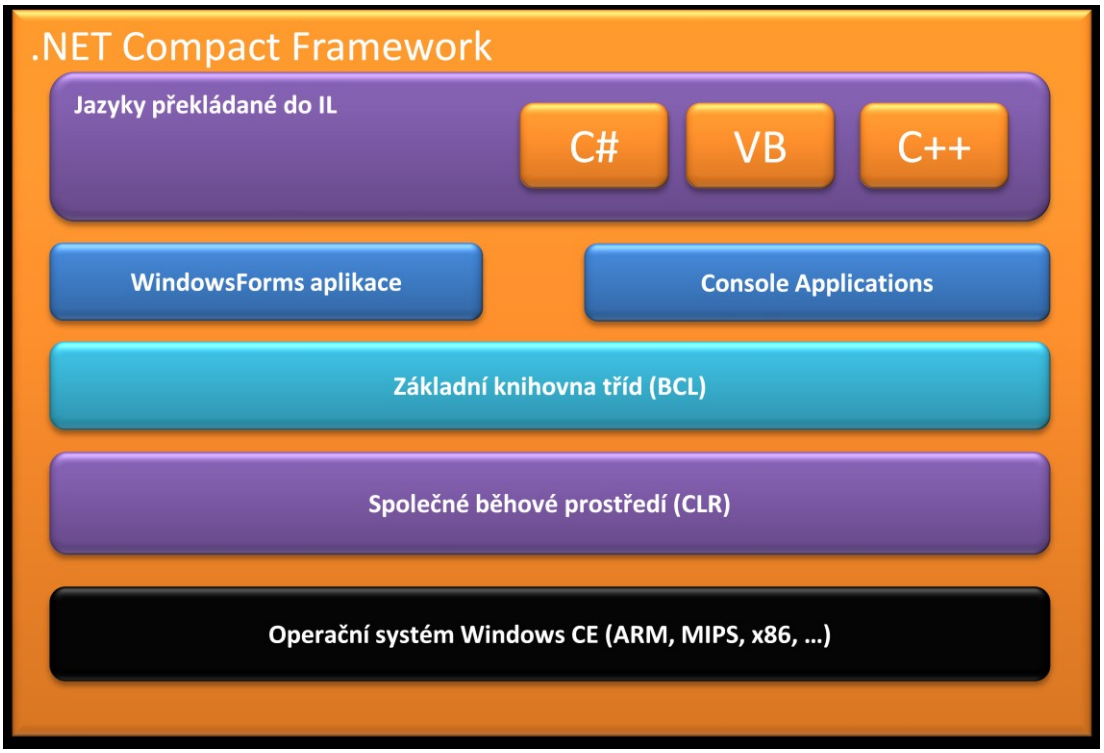
## .NET Compact Framework

[3] Microsoft .NET Compact Framework je platforma pro vytváření a běh aplikací pro mobilní zařízení, na kterých běží operační systém Windows CE a Windows Mobile. Platforma .NET Compact Framework je optimalizovanou verzí platformy .NET Framework pro mobilní zařízení, které často nemají dostatečně velké paměti pro provoz plné verze .NET Framework, také operační systémy, na kterých aplikace napsané pro .NET Compact Framework běží, Windows CE a Windows Mobile se podstatně liší od verzí systémů Windows pro PC, z tohoto důvodu se tvůrci .NET Compact Frameworku rozhodli pro odebrání tříd, které nejsou nezbytně nutné z BCL (základní knihovní třídy) na co nejoptimálnější množinu.

Také bylo nutné vytvořit nové třídy, které jsou svým použitím specifické pro mobilní zařízení typu PDA nebo Smartphone.

Mezi hlavní výhody platformy .NET Compact Framework patří snadná správa paměti, rychlý vývoj aplikací, podpora velkého množství programovacích jazyků (C#, VB.NET, IronPython, F#, atd.) a také částečná kompatibilita s velkým .NET Frameworkem příslušné verze. .NET Compact Framework je určen pro zařízení s malou pamětí, pomalejším procesorem a také menším úložištěm dat než je tomu u stolních počítačů. Z tohoto důvodu je také omezena velikost samotného Frameworku. Omezení je viditelné nejvíce u tříd, jejichž počet je oproti velké verzi značně snížen, právě z důvodu omezené velikost úložišť v řádech stovek kB až několika MB. Na druhou stranu, ale obsahuje .NET Compact Framework třídy specifické pro své využití, jako je například ovládání mobilního telefonu, ovládání portu IRDA atp. Dále .NET Compact Framework obsahuje vylepšenou správu běhu programu zaměřenou na větší úsporu energie.Historie .NET Compact Framework

První verzi platformy .NET Compact Framework byla verze 1.0, která byla vydána v roce 2002. Jak už to většinou bývá první verze .NET Compact Frameworku byla hodně omezena hlavně z hlediska chybějící podpory práce s okny (nebylo dostupné ani window handle) a další velice limitující nedostatky, které byly později doplněny ve verzi 2.0 v roce 2005.




Obrázek 3 - Architektura .NET Compact Framework

Verze 2.0 přinesla znatelné zrychlení aplikací a také rozšíření o podporu generických typů, které podstatně zjednodušují programování, návrh a zvyšují znovu použitelnost výsledného kódu. Aplikace UTMJ OBU byla vyvinuta právě na platformě .NET Compact Framework verze 2.0, v době vývoje již existovala verze .NET Compact Frameworku 3.5, která nebyla v aplikaci UTMJ OBU použita z důvodu firemní politiky a také z důvodu dostupnosti této platformy na ROM zařízeních.

Jak již bylo zmíněno jednou z výhod .NET Compact Framework je, že podporuje mnoho jazyků, mezi asi ty nejvíce používané patří jazyk C#, ve kterém byla i napsána aplikace UTMJ OBU.

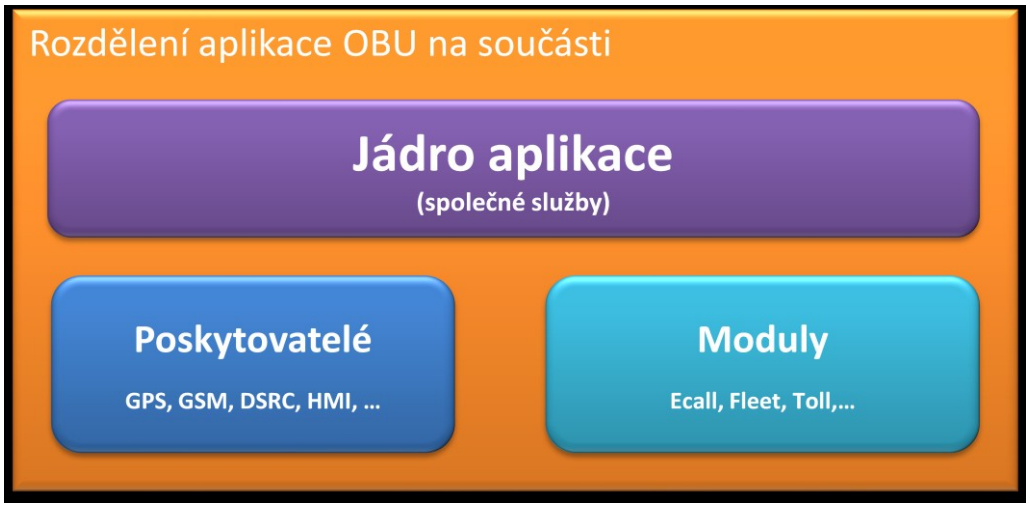
[6] Vývoj aplikací pro .NET Compact Framework 2.0 je podporován v IDE Microsoft Visual Studio 2008 Professional. Zařízení, které podporuje .NET Compact Framework se připojí přes USB k počítači a pomocí software ActiveSync je spojeno s IDE. Ve VS2008 je možné aplikaci napsanou v .NET Compact Framework 2.0 ladit, číst proměnné, měnit běh programu, vyhodnocovat a vykonávat příkazy daného jazyka (C#, případně VB.NET), provádět kopírování zkompilovaného projektu (včetně všech zdrojů) do zařízení automaticky před laděním aplikace. Dále VS2008 dovoluje vývoj všech typů aplikací, jako je WindowsForms aplikace, konzolové aplikace a knihovny pro rozšíření aplikace.

## Jazyk C# 2.0

[7] Jazyk C# byl poprvé představen v rámci vydání první verze vývojového prostředí Visual Studio.NET 2001 s .NET Framework 1.0. Jazyk C# je na první pohled velice podobný jazyku C++, hlavně syntaxe těchto jazyků je velice podobná. Hlavním důvodem pro takovou podobnost je hlavně snadný přechod programátorů, kteří programují v jazyce C++ nebo C, v té době zcela novou platformu .NET. Jazyk C# je plně objektivně orientovaným jazykem, který je vedle jazyka Visual Basic.NET přímo podporován společností Microsoft. Jazyk C# byl navržen pro rychlý vývoj aplikací, které jsou spolehlivé, relativně rychlé, bezpečné a díky podpoře techniky garbage colletion i snadné pro vývoj. Technika garbage colletion dovoluje programátorovi se méně soustředit na správu paměti, kterou aplikace alokuje, protože s jeho programem automaticky běží tzv. garbage collector, který se stará o uvolnění již nepotřebné alokované paměti. Tato technika má výhodu v tom, že když je potřeba je rychle a efektivně uvolněna nepotřebná paměť, ale není nutné jí uvolňovat vždy jak tomu je například u jazyků rodiny C/C++. Což výrazně zlepšuje a zrychluje samotný vývoj a kód je také přehlednější. Na druhou stranu jsou programy napsané v .NET pomalejší hlavně kvůli překladu JIT a také automatické kontrole přístupu k paměti.

## Návrh architektury aplikace OBU

Návrh architektury aplikace OBU musel zohlednit veškeré systémové požadavky, které vyplynuly ze zadání samotného projektu MPO. Mezi nejdůležitější požadavky, které ovlivnili samotný návrh architektury aplikace, patří flexibilita (strana č. 13), která vyžaduje, aby určité části aplikace nebyly pevně dané. Z tohoto důvodu byla softwarová architekta navržena tak, aby bylo možné rozdělit aplikaci na jednotlivé ucelené funkční celky. Architektura aplikace byla rozdělena na tři funkčně ucelené celky:




Obrázek 4 - Rozdělení aplikace na součásti

Hlavní funkcí jádra v architektuře aplikace OBU je poskytovat data, služby a podporu dalším celkům aplikace OBU jako jsou poskytovatelé a moduly.

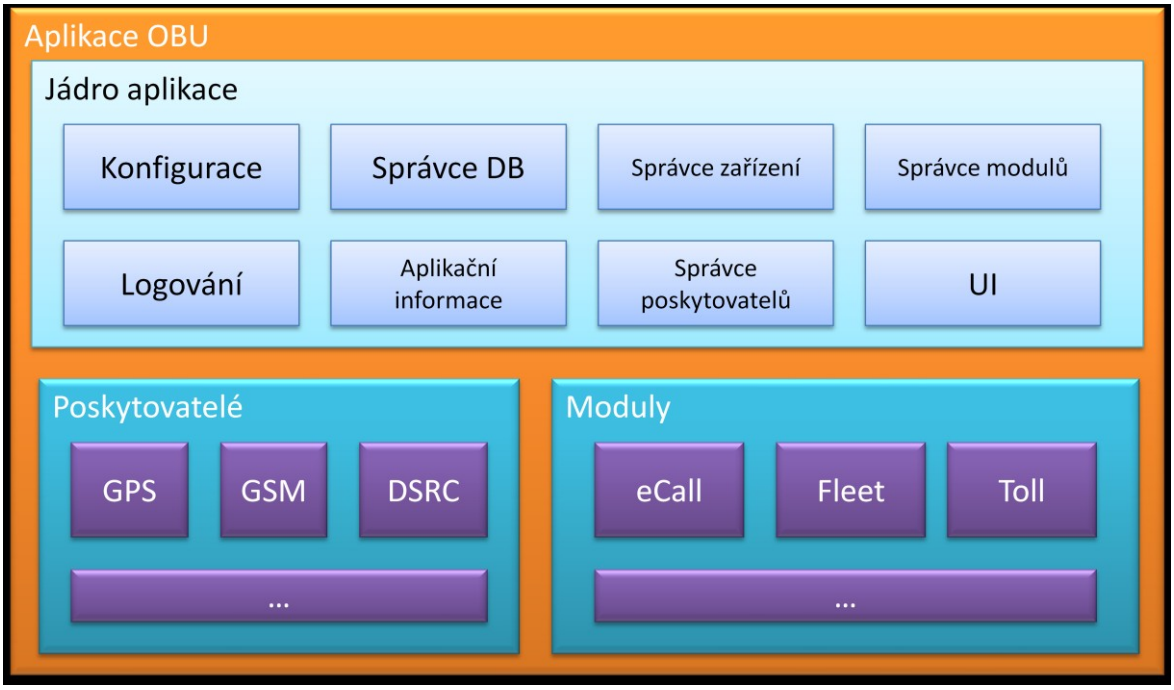
Poskytovatelé plní v architektuře aplikace OBU roli poskytovatelů služeb nebo dat. Kdy poskytovatel je skrze jádro aplikace dostupný modulům a dalším poskytovatelům. Tudíž je

možné pro jednu specifickou službu nebo data využívat pouze jednoho poskytovatele, na kterého je možné navázat další služby případně zpracování dat z modulu.

Moduly plní v architektuře aplikace OBU roli zpracovatelů služeb nebo dat. Moduly často obsahují různé algoritmy pro práci s daty od poskytovatelů a využívají různých služeb poskytovatelů. Přístup ke všem součástím aplikace je pomocí jádra, které obsahuje informace o aktuálně spuštěných funkčně ucelených celků.

## Architektura aplikace OBU

Detailní architektura aplikace OBU je znázorněna na Obrázek 5 - Architektura aplikace OBU, kde můžeme vidět rozložení aplikace do jednotlivých funkčně ucelených celků. Jádro aplikace je označeno světle modrou barvou, naopak rozšiřitelné části aplikace jsou označeny tmavě modrou barvou.




Obrázek 5 - Architektura aplikace OBU

Jádro aplikace obsahuje základní služby, které jsou dostupné pro všechny součásti aplikace pomocí reference na objekt jádra (rozhraní ICore). Tento objekt obsahuje vlastnosti pro přístup k jednotlivým pevně daným službám jádra, které poskytují specifické informace, obsahují určitou funkcionalitu nebo zprostředkovávají interakci s uživatelem. Na Obrázek 5 - Architektura aplikace OBU můžete vidět všechny služby jádra, mezi které patří:

- Konfigurace
- Správce DB
- Správce zařízení
- Správce modulů
- Logování
- Aplikační informace
- Správce poskytovatelů
- Uživatelské rozhraní (UI)

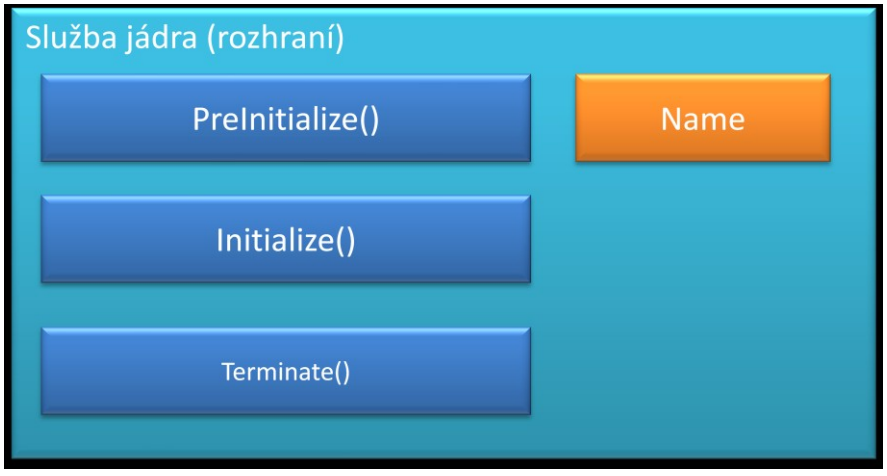
## Jádro aplikace OBU

Jádro aplikace OBU obsahuje základní funkce aplikace, které jsou nezbytné pro spuštění aplikace a běh dalších funkčních celků poskytovatelů a modulů. Tyto základní funkce byly identifikovány při analýze architektury aplikace, podle požadavků na aplikaci a také podle požadavků jednotlivých poskytovatelů a modulů, které byly známé v době návrhu architektury aplikace.

Jádro aplikace je dáno několika rozhraními, které jasně definují všechny metody, vlastnosti a události. Rozhraní ICore definuje, jak vypadá jádro na venek, ostatní rozhraní již definují jednotlivé služby jádra a jejich metody. Mezi tyto rozhraní patří:

- IConfiguration
- IApplicationInformation
- IDatabaseManager
- IDeviceManager
- ILogManager
- IModuleManager
- IProviderManager
- IUI

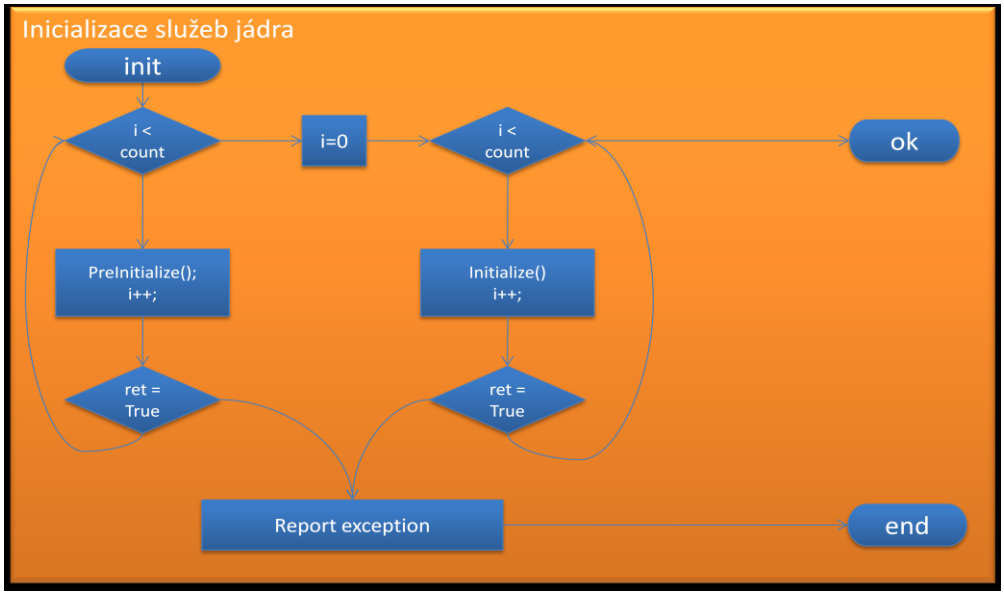
Vzhledem k tomu, že části jádra (dále služby jádra) mají mezi sebou různé závislosti, bylo nutné při inicializaci samotného jádra využít „dvou průchodové“ inicializace služeb jádra. Služby jsou také spouštěny v přesně daném pořadí. Obrázek 6 - rozhraní služby jádra, znázorňuje metody a vlastnosti služby jádra. Mezi základní metody každé služby jádra dostupné pouze v rámci jádra patří PreInitialize, Initialize a Terminate. Dále každá služba jádra pro účely ladění a lepší orientaci obsahuje vlastnost Name, která udává název služby jádra.




Obrázek 6 - rozhraní služby jádra

Dvoj-průchodová inicializace služeb jádra je zobrazena na „Obrázek 7 - spuštění služeb“ kde je znázorněn postup inicializace všech služeb jádra. Nejprve se v daném pořadí zavolá metoda PreInitialize všech služeb, která vrací boolean pro indikaci úspěchu spuštění

služby. Pokud dojde k výjimce PreInitialize nebo metoda vrátí hodnotu False, dojde k zobrazení chybové hlášky a uložení informací do logu jádra (soubor Obu.txt).




Obrázek 7 - spuštění služeb

Pokud všechna volání PreInitialize byla úspěšná, provede se stejným způsobem i zavolání metody Initialize. Pokud volání metody Initialize u všech služeb je úspěšné, služby jádra jsou správě spuštěny a celá aplikace běží.

Hlavním důvodem proč byla použita dvoj-průchodová inicializace je ten, že některé služby jádra vyžadují svojí inicializaci, až když jsou ostatní služby načteny (jsou závislé na jiných službách). Také je dobré rozdělit spuštění služeb do několika fází. První fáze je určena k načtení základních informací (konfigurace) všech služeb, v druhé fázi dochází ke spuštění interních procesů služeb, případně vytvoření interních objektů služeb.

Jak již bylo zmíněno služby se spouští v předem daném pořadí (hlavně podle jednotlivých závislostí). Služby jsou spouštěny v následujícím pořadí:

- Uživatelské rozhraní
- Konfigurace
- Logování
- Správce DB
- Správce HW zařízení
- Aplikační informace
- Správce poskytovatelů
- Správce modulů

Při ukončování aplikace OBU dochází k volání metody Terminate služby jádra v opačném pořadí, nejdříve je ukončen Správce modulů (a všechny spuštěné moduly) a jako poslední je ukončeno Uživatelské rozhraní. V případě selhání nebo vyhození výjimky při volání metody Terminate není přerušeno ukončení dalších služeb.

# Služby jádra aplikace

## Konfigurace

Služba jádra konfigurace plní funkci uchovávání nastavení celé aplikace včetně modulů a poskytovatelů. Konfigurace je načtena ze souboru při spuštění jádra hned po inicializaci uživatelského rozhraní, tudíž je dostupná od samého počátku běhu aplikace hlavně kvůli požadavku na modularitu celé aplikace. Kdy služba jádra konfigurace obsahuje informace pro spuštění jak poskytovatelů, modulů, tak pro nastavení komunikace s HW, nastavení uživatelského rozhraní a v neposlední řadě, také nastavení součástí jádra aplikace.

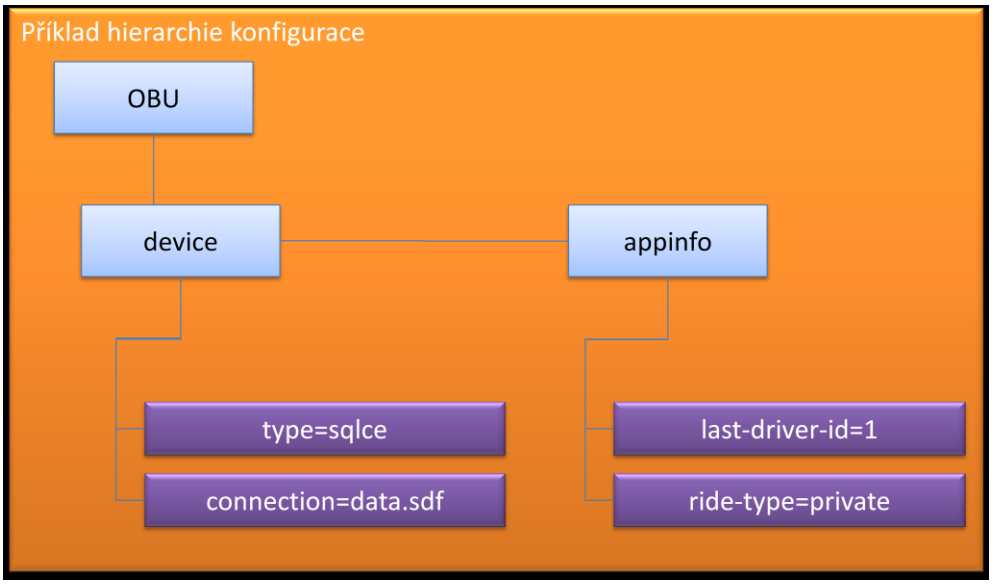
Konfigurace je standardně uložena v souboru s názvem „config.xml“, který je ve formátu XML, kde jsou pomocí značek jazyka XML definovány jednotlivé sekce konfigurace ve více úrovních. V rámci sekce konfigurace existují také hodnoty, které lze identifikovat celým názvem sekce a názvem samotné hodnoty (například root\section\value). Služba jádra konfigurace dovoluje mimo operace načtení konfigurace do paměti RAM zařízení, také zpětné uložení konfigurace zpět do souboru „config.xml“ tohoto je využíváno hlavně v modulech, které vyžadují změnu svého nastavení, jako je například modul „fleet“, který si ukládá v nastavení aktuálně zvoleného řidiče, tak aby po novém spuštění aplikace došlo k automatickému zvolení posledního aktivního řidiče.

```
<?xml version="1.0" encoding="utf-8"?>
<obu>
  <appinfo current-ride="private">
    <current-driver>1</current-driver>
  </appinfo>
  <log directory="logs">
    <filters>
      <filter scope="core" />
    </filters>
  </log>
</obu>
```

Ukázka konfiguračního souboru ve formátu XML

Důvodem pro zvolení formátu XML jako úložiště konfigurace je hlavně snadná čitelnost takového souboru (viz ukázka výše). Dále úpravy v souboru lze provádět i nejjednoduššími editory prostého textu, například v OS Windows CE 5.0 je obsažen editor Wordpad, který byl při vývoji aplikace hojně využíván pro úpravu nastavení přímo v zařízení.

Konfigurace dále podporuje vytváření nových sekcí, mazání sekcí, případně jejich přejmenování. Dále lze vytvářet nové hodnoty, mazat je či přejmenovat. Příklad hierarchie podporovaný v konfiguračním souboru (obecně v konfiguraci) můžete vidět na Obrázek 8 - Příklad hierarchie konfigurace, kde jsou znázorněny vztahy jednotlivých sekcí (světle modrá barva) a hodnot (fialová barva).




Obrázek 8 - Příklad hierarchie konfigurace

## Logování

Služba jádra „logování“ plní funkci záznamu dat o běhu celé aplikace do souboru, který je uložen ve složce „logs“ (v závislosti na konfiguraci). Jednotlivé soubory mohou být pojmenovány jako číselná řada (0→n), nebo lze nastavit (v konfiguraci aplikace) aby se soubor pojmenoval podle časového razítka ve formátu „MMDDYYYY“. Všechny soubory služby jádra „logování“ mají příponu „log“. Služba jádra „logování“ velice usnadňuje ladění aplikace jako celku záznamem informací, které mohou odhalit případné problémy s aplikací bez nutnosti mít připojen k aplikaci ladící nástroj. Služba jádra „logování“ zaznamenává do souboru (tzv. logu) následující údaje:

- Datum a čas
- Typ záznamu
- Zpráva
- Zdroj zprávy (scope name)

Typ záznamu v logu definuje typ dané zprávy, který je uložen v souboru. Typ určuje, zda došlo například k chybě (exception, error) nebo zda je zpráva pouze informativního charakteru (information).

Typy zpráv jsou následující:

Tabulka 2 - Typy zpráv služby jádra "logování"

| Název | **Vysvětlení** |
|---|---|
| **InfoLowPriority** | Informace s nízkou prioritou (pouze pro ladění aplikace) |
| **Information** | Informace se standardní prioritou |
| **InfoHighPriority** | Informace s vysokou prioritou (důležitá informace v rámci aplikace) |
| **Warning** | Upozornění na skutečnost (není dostupné zařízení, které neovlivní úplné spuštění aplikace) |
| **Error** | Vážné selhání součásti aplikace nebo HW. |
| **Exception** | Výjimka, která nebyla očekávána nebo ladící informace o selhání. |
| **IntegrityCheckFast** | Rychlá kontrola integrity aplikace. |
| **IntergrityCheckSlow** | Důkladná kontrola integrity aplikace (pomalá). |

Služba jádra „logování“ přímo závisí na službě „konfigurace“, která poskytuje službě „logování“ informace pro její správný běh. Služba logování je spouštěna hned po inicializaci služby „konfigurace“.

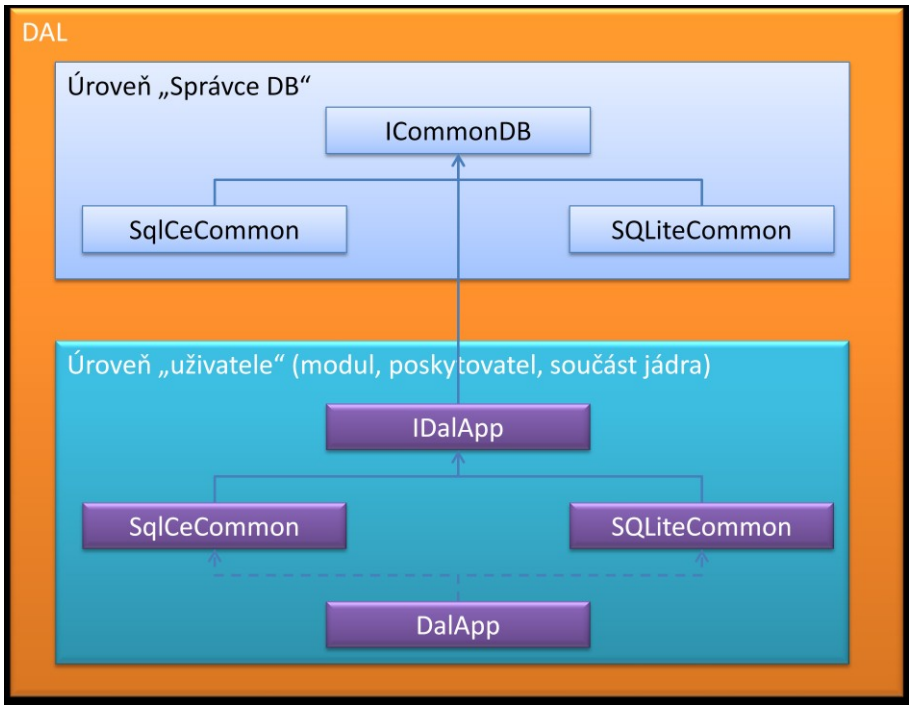
Služba jádra „logování“ obsahuje funkci pro filtrování zpráv, které jsou uloženy do souboru. Tyto filtry jsou uloženy v konfiguraci aplikace a dovolují omezit zprávy uložené do souborů podle typu zprávy nebo zdroje zprávy. Filtrováním lze docílit omezení nedůležitých zpráv při ladění aplikace nebo při spuštění aplikace v plném provozu, kde není nutné zapisovat pouze informativní zprávy, ale naopak je nutné zapisovat selhání součástí a neošetřené výjimky pro identifikaci případného problému.

## Správce DB

Služba jádra „Správce DB“ obsahuje funkce pro správu databázových systému v rámci jádra aplikace OBU. Služba jádra „Správce DB“ obsahuje seznam spojení, jejich typ a dokáže ověřit existenci databáze, integritu a zjistit základní informace o databázi. Služba poskytuje přístup k databázovým systémům pomocí tříd DAL (data application layer), které zjednodušují přístup k databázi a zároveň tvoří vrstvu mezi uživatelem (modul, poskytovatel nebo součást jádra) a samotným databázovým systémem (dále DBS).

## Data application layer

Vrstva data application layer (dále DAL) obsahuje třídy a rozhraní (interface) pro práci s databázovými systémy podporovanými v aplikaci OBU (viz. dále). Obsahuje několik vrstev, které obsahují členy (metody, vlastnosti, události), které usnadňují práci s databázovými systémy různých typů. Hierarchie DAL je znázorněna na Obrázek 9 - Data application layer.




Obrázek 9 - Data application layer

Samotnou vrstvu DAL, můžeme rozdělit na dvě úrovně. První je úroveň „Správce DB“, která obsahuje základní rozhraní ICommonDB, které obsahuje základní definice metod, vlastností a událostí pro rozhraní DAL. Toto rozhraní obsahuje základní metody pro přístup k DBS, kontrola integrity daného DBS, konfigurace nastavení, zabezpečení a správy DBS. Dále tato úroveň obsahuje základní třídy, které implementují toto rozhraní pro zvolené DBS (viz níže).

Úroveň DAL „uživatele“ obsahuje již specifické rozhraní IDalApp pro využití v roli uživatele DBS, kdy je ošetřen základní přístup (vykonání příkazu či dotazu) do DBS. Vzhledem k tomu, že zvolené DBS v aplikaci OBU jsou z hlediska práce s nimi odlišné, existují zde implementace rozhraní pro každý typ DBS. Tyto základní implementace jsou dále využívány v abstraktní třídě DalApp pro zajištění nezávislosti na koncových DBS. Uživatel vrstvy DAL, provede konfiguraci služby jádra „Správce DB“ a vytvoří svojí vlastní třídu, která bude dědit od třídy DalApp, kde vytvoří metody, které budou provádět specifické příkazy či dotazy nezávislé na DBS.

Při návrhu aplikace OBU bylo myšleno na podporu více DBS, vzhledem k tomu, že nebylo nutné podporovat velké množství DBS, byly vybrány DBS Microsoft SQL CE 3.5 a DBS Sqlite.

### Microsoft SQL CE 3.5

[2] Microsoft SQL CE (dále SQL CE) ve verzi 3.5 je databázový systém vyvinutý společností Microsoft. Databázový systém SQL CE 3.5 obsahuje základní funkce převzaté z velké edice Microsoft SQL serveru. SQL CE je systém, který ukládá databázi do jednoho souboru, se kterým provádí různé operace na základě příkazů uživatele. K zadávání příkazů slouží jazyk SQL, který definuje dotazy pro práci s DBS.  SQL CE je podporován na

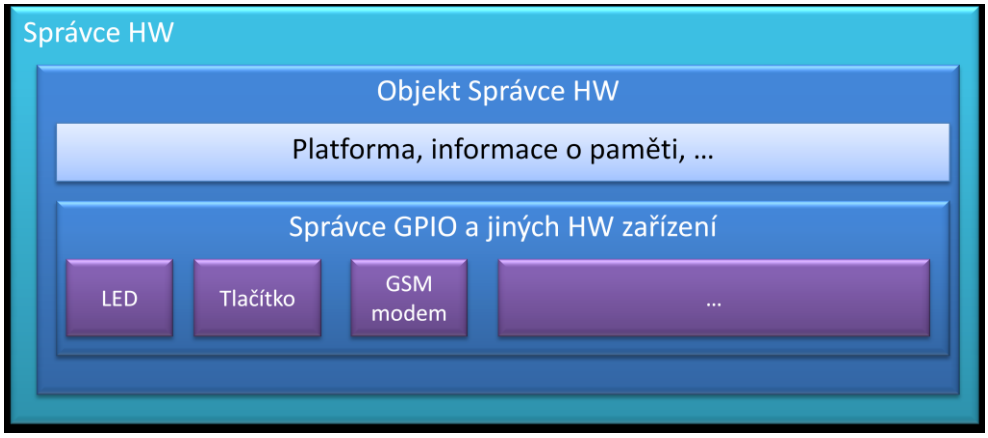
platformách PC (OS Windows), platformě Windows Mobile a Windows CE. Systém je navržen tak, aby bylo možné migrovat a pracovat s daty pouze zkopírováním souboru s databází. Což může mít výhody v podobě snadné přenositelnosti dat (migrace), snadné údržby databáze atd. Hlavní nevýhodu tohoto DBS vidím v absenci uložených procedur, nulové podpory síťového přístupu k DBS (v tomto případě není důležitá funkcionalita) a také problematické podpory přístupu více aplikací či uživatelů najednou. Vzhledem k tomu, že je aplikace programovaná v jazyce C# je SQL CE vhodné také z důvodu dobré provázanosti pomocí technologie ADO.NET a také z důvodu podpory přímo ve vývojovém nástroji Microsoft Visual Studio (od verze 2005).

### SQLite

[3] SQLite je databázový systém vyvíjený komunitou vývojářů z celého světa. Databázový systém podporuje jazyk SQL, jeho výhodou je, že je dostupný jako dynamicky připojovaná knihovna (DLL pro Windows), která obsahuje celý databázový engine, tudíž není potřeba žádného běžícího serveru. Databázový systém SQLite podporuje transakce, nevyžaduje žádnou složitou konfiguraci, pouze název souboru se kterým se bude pomocí SQL dotazů pracovat. Hlavní výhodou databázového systému SQLite je snadná přenositelnost databáze na jiné platformy (od ARM po 32 a 64. bitové systémy). SQLite je celé napsáno v jazyce C a celý zdrojový kód je plně k dispozici, tudíž je možné jej plně modifikovat případně zlepšovat. Další výhodou SQLite jsou automatické testy, které jsou pravidelně prováděny, které zajišťují, že kód je pořád optimální, stabilní a kompatibilní. V projektu UTMJ OBU byl pro použití SQLite využíván wrapper pro ADO.NET, který je rovněž spravován komunitou SQLite.

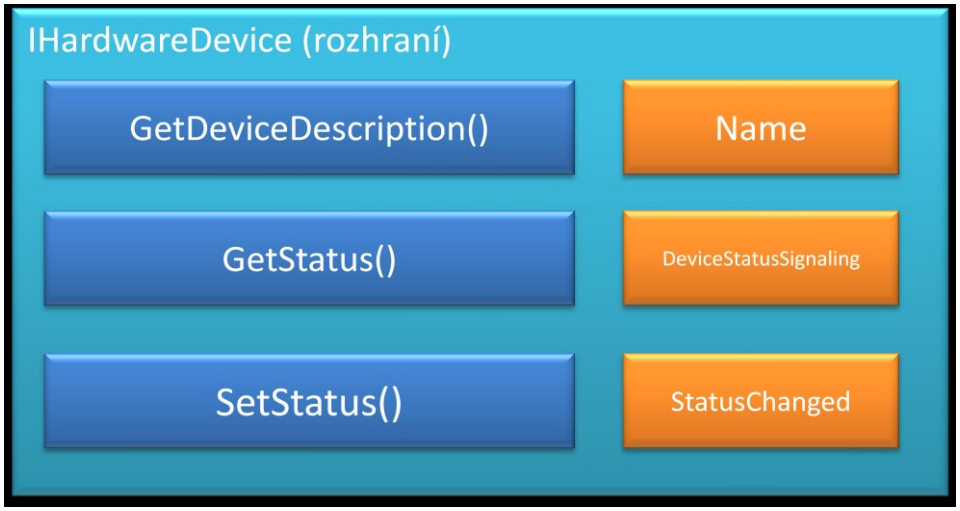
### Správce HW zařízení

Součást jádra „Správce HW“ obsahuje funkce pro správu hardwarového vybavení jednotky UTMJ, uchovává stav jednotky a zajišťuje zaslání události ostatním součástem aplikace o změně stavu jednotky. Dále součást „Správce HW“ spravuje jednotlivé HW moduly, kdy dokáže ovlivnit, zda je daný modul zapnut či vypnut, dále spravuje pomocí GPIO HW periferie, jako jsou například stavové indikátory, tlačítka a jiná HW zařízení a rozhraní (SPI například).




Obrázek 10 - Diagram součástí "Správce HW"

Jak již bylo zmíněno jednotka má svůj obecný stav, který může nabývat tři hodnoty. Prvním stavem je stav „None“ (nedefinováno, žádný), který je nastaven při samotném spuštění součásti „Správce HW“, tento stav indikuje, že nebylo rozhodnuto o stavu jednotky, protože nedošlo k plné inicializaci jádra aplikace OBU. Pokud je jádro inicializováno v pořádku je nastaven stav „Ready“, jednotlivé součásti podle své důležitosti přistupují k objektu součásti „Správce HW“ a upravují stav celé jednotky podle svého stavu. Například pokud je přítomen poskytovatel GPS, a je dostupná pozice jednotky, je nastaven stav „Ready“. Pokud dojde k výpadku signálu je nastaven stav „Problem“, který indikuje, že některá z důležitých součástí jednotky nefunguje či je nedostupná.




Obrázek 11 - Diagram IHardwareDevice

Součást „Správce HW“ mimo jiné spravuje HW zařízení pomocí rozhraní GPIO, kdy dokáže například nastavením předefinovaného pinu zapnout modem, rozsvítit LED nebo zkontrolovat zda nebylo stisknuto tlačítko. Všechny tyto součásti jsou reprezentovány objekty, které implementují rozhraní IHardwareDevice. Rozhraní IHardwareDevice obsahuje metody pro nastavení stavu zařízení (true/false), název zařízení a událost indikující změnu stavu zařízení.

Součást jádra „Správce HW“ také sleduje systémové prostředky zařízení a obsahuje události pro indikaci, zda tyto prostředky docházejí. Mezi sledované prostředky patří například paměť RAM a stav úložiště (SD karta nebo interní flash paměť), pokud stav sledovaných pamětí klesne pod hodnotu určenou v konfiguraci aplikace, dojde k vyvolání události, která informuje o kritické hodnotě paměti.

## Aplikační informace

Součást jádra „Aplikační informace“ obsahuje základní informace o jednotce, vozidlu, řidič, typu jízdy a jedinečnému identifikátoru jednotky. Základní informace jsou uloženy v konfiguraci aplikace a v databázi součásti jádra „Aplikační informace“. Součást jádra „Aplikační informace“ je přímo závislá na součásti jádra „Správce DB“ z důvodu práce s DBS.

Všechny informace dostupné v součásti jádra „Aplikační informace“ jsou dostupné ostatním součástím aplikace.

V konfiguraci aplikace OBU jsou uloženy následující údaje:

- Unikátní identifikátor jednotky – využívá se pro jednoznačnou identifikaci jednotky, používá datový typ GUID, který je reprezentován textově v podobě 32 znaků ve formátu {########-####-####-###-############}, kde znak # reprezentuje hexadecimální číslo. Datový typ GUID lze také reprezentovat jako 128bitový celočíselný typ.
- ID řidiče v databázi – Identifikátor posledního přihlášeného řidiče v podobě čísla
- Typ jízdy – Poslední zvolený typ jízdy

V databázi součásti „Aplikační informace“ jsou uloženy následující údaje:

- **Informace o řidiči**

Tabulka 3 - Uložené informace o řidiči

| Název | Popis |
|---|---|
| **ID_Driver** | Identifikační číslo řidiče v DB (index, PK) |
| **FirstName** | Křestní jméno řidiče |
| **LastName** | Příjmení řidiče |
| **DriverNumber** | Evidenční číslo řidiče v rámci firmy, může být osobní číslo. Datový typ byl použit nvarchar (textový) |

- **Informace o vozidle**

Tabulka 4 - Uložené informace o vozidle

| Název | Popis |
|---|---|
| **ID_Vehicle** | Identifikační číslo vozidla v DB (index, PK) |
| **LicenceNumber** | Registrační značka vozidla (SPZ) |
| **VIN** | Výrobní číslo karoserie vozidla |
| **CarName** | Název vozidla definovaný v rámci firmy (např. Citroën C4) |
| **CarWidth** | Šířka vozidla (určeno pro potřeby navigace) |
| **CarHeight** | Výška vozidla (učeno pro potřeby navigace) |
| **CarLenght** | Délka vozidla (učeno pro potřeby navigace) |
| **CarWeight** | Váha vozidla (učeno pro potřeby navigace) |
| **CargoWeight** | Váha nákladu (učeno pro potřeby navigace) |

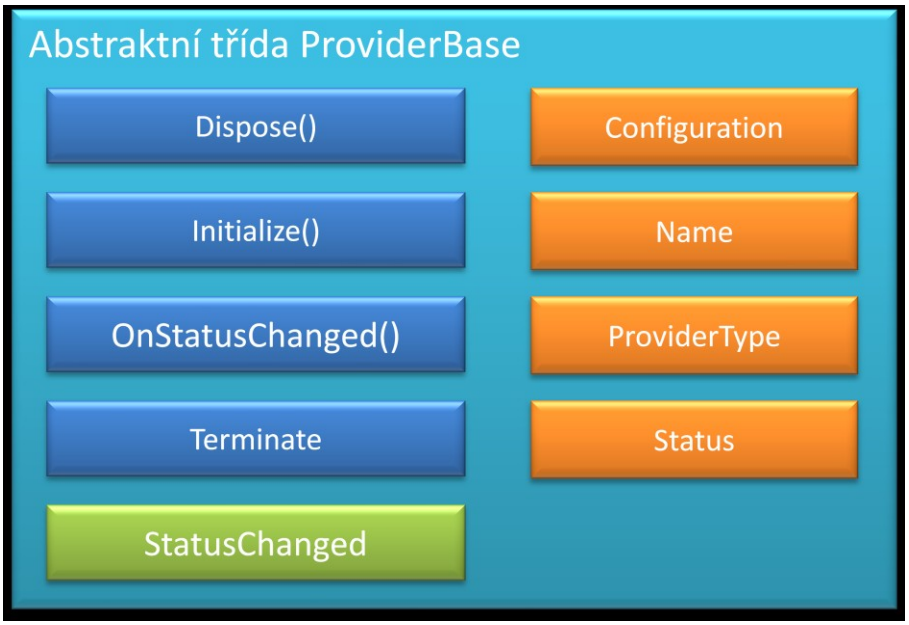
Všechny zmíněné informace jsou ostatním součástem aplikace OBU reprezentovány objekty, které obsahují aktuální informace z databáze. Dále objekt

součásti „Aplikační informace“ obsahuje metody pro provedení změn jako je například změna řidiče, změna typu jízdy, nebo zjištění všech řidičů v rámci jednotky, pro snadný výběr ze seznamu za pomocí UI.

## Správce poskytovatelů

Součást jádra „Správce poskytovatelů“ obsahuje funkce pro správu poskytovatelů. Definice poskytovatelů, kteří se spouštějí po spuštění aplikace OBU, jsou uloženy v konfiguraci a načtení definice probíhá ve fázi PreInitialize inicializace jádra. Samotná inicializace poskytovatelů probíhá podle pořadí načtených definicí ve druhé fázi inicializace Initialize, inicializace všech poskytovatelů je synchronní operace, tudíž dochází k postupné inicializaci poskytovatelů. Vzhledem k tomu, že někteří poskytovatelé mohou být závislé na druhých poskytovatelích je pořadí spouštění poskytovatelů důležité. Poskytovatelé jsou inicializováni, kdy jsou již spuštěny skoro všechny služby jádra (uživatelské rozhraní, konfigurace, logování, správce DB, správce zařízení a aplikační informace), tudíž je možné, aby poskytovatelé využívali skoro všechny služby jádra.

Součást jádra „Správce poskytovatelů“ dovoluje spuštění poskytovatele i při běhu aplikace je možné spustit poskytovatele i dynamicky podle potřeby. Není tudíž nutné deklarovat všechny poskytovatele v konfiguraci. Objekty poskytovatelů musí dědit od abstraktní třídy ProviderBase, která definuje základní metody, vlastnosti a události. Které jsou používány samotnými poskytovateli pro indikaci stavu, typu poskytovatele, názvu poskytovatele, dále abstraktní třída ProviderBase obsahuje vlastnosti typu „protected“, které jsou dostupné pouze třídě, která zdědí od třídy ProviderBase. Mezi tyto vlastnosti patří vlastnost „Configuration“ a „Core“.




Obrázek 12 - Abstraktní třída ProviderBase

Obrázek 12 - Abstraktní třída ProviderBase ukazuje základní členy třídy ProviderBase, modře zabarvené jsou metody, žlutě zabarvené jsou vlastnosti a jsou zelené události. Metoda Initialize musí být přetížena dědící třídou a je určena ke spuštění

poskytovatele, který při správném spuštění vrátí hodnotu True. Metoda Terminate() je volána součástí jádra „Správce poskytovatelů“ při ukončení poskytovatele. Vlastnost Name určuje název poskytovatele  Vlastnost Status určuje stav poskytovatele, který může nabývat následujících hodnot:

Tabulka 5 - Stavy poskytovatelů

| Název | **Popis stavu** |
|---|---|
| **Unknown** | Neznámý stav / nedefinovaný stav |
| **Disconnected** | Odpojen |
| **Disconnecting** | Probíhá odpojení poskytovatele |
| **Connected** | Připojen |
| **Connecting** | Připojování poskytovatele |
| **ConnectionError** | Došlo k chybě při připojení poskytovatele. |

Abstraktní třída ProviderBase obsahuje také vlastnost ProviderType, která určuje typ poskytovatele, který může nabývat následujících hodnot:

Tabulka 6 - Typy poskytovatelů

| **Název** | **Popis stavu** |
|---|---|
| **Other** | Poskytovatel má jiný typ než definovaný. |
| **GPS** | Poskytovatel je typu GPS, tudíž poskytuje aktuální pozici jednotky. |
| **GSM** | Poskytovatel je typu GSM a pracuje s GSM modem. |
| **TMC** | Poskytovatel pracuje s TMC modulem, pro zjištění zpráv (není využito) |
| **CAN** | Poskytovatel pracuje se sběrnicí CAN. |
| **Communication** | Poskytovatel zprostředkovává komunikaci se serverem. |
| **EFC** | Poskytovatel je modulem pro mýto. |

Metoda OnStatusChanged volá událost StatusChanged, která indikuje změnu stavu poskytovatele ostatním součástím aplikace OBU. Indikace změny stavu je v aplikaci OBU velice důležitá, protože je možné sledovat stav poskytovatelů a pří chybě je možné znovu spustit.

Součást jádra„Správce poskytovatelů“ obsahuje metody pro získávání objektů poskytovatelů, k získání poskytovatele je určena metoda GetProvider. Metoda GetProvider existuje ve dvou verzích, které se liší pouze přijímaným parametrem. První verze metody GetProvider přijímá jako parametr datový typ String, který určuje název požadovaného poskytovatele (například „GPS“). Druhá verze metody GetProvider přijímá jako parametr datový typ Type, ve kterém musí být specifikován datový typ, který daný poskytovatel implementuje. Pokud je parametr metody GetProvider(Type type) typ jiný než rozhraní (interface) metoda vyhodí výjimku.

Součást jádra „Správce poskytovatelů“ také obsahuje metody pro zjištění všech dostupných poskytovatelů. Toto metoda je využívána v testovacím modulu

## Poskytovatelé

Poskytovatelé v aplikaci UTMJ OBU poskytují služby či data dalším součástem aplikace, které je poté používají (služby) nebo zpracovávají (data). Většina vyvinutých poskytovatelů obsahují samostatné vlákno, které zpracovává data ze zařízení nebo poskytuje služby.

V rámci projektu UTMJ OBU jsem se podílel na implementaci některých poskytovatelů. Mým úkolem bylo hlavně zajistit správné napojení na jádro.

V rámci projektu UTMJ OBU byly vyvinuty následující poskytovatelé.

Tabulka 7 - Vyvinutí poskytovatelé

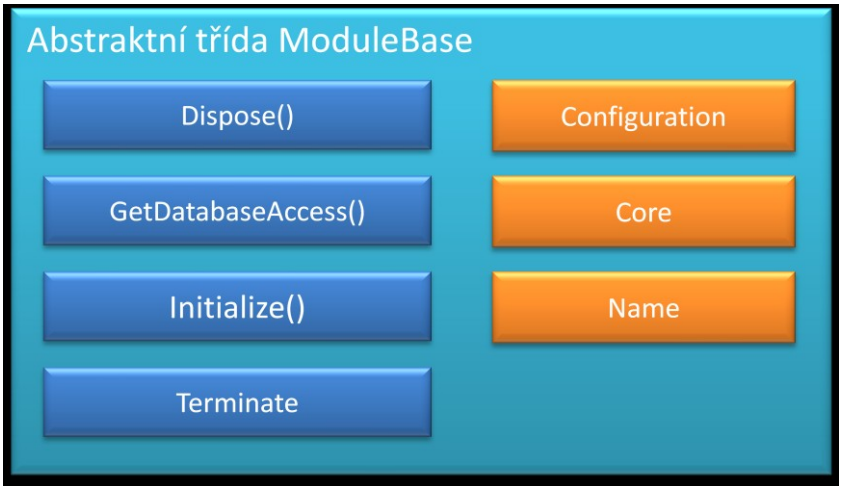
| Název | Popis |
|---|---|
| **GPSProvider** | Poskytovatel pozice za pomocí GNSS zařízení (GPS, Gallileo, …). Poskytovatel čte lokalizační informace pomocí portu COM, kde jsou od GNSS zařízení přijímány NMEA věty obsahující informace o pozici, nadmořské výšce, rychlosti, atp. |
| **GSMProvider** | Poskytovatel GSM modem, který je integrován na základní desce UTMJ OBU. Poskytovatel podporuje ovládání GSM modemu pomocí AT příkazů. |
| **ConnectionProvider** | Poskytovatel síťového připojení, který pracuje s technologií RAS systémů Windows CE a Windows Mobile. |
| **GNSS EFC** | Poskytovatel využívá poskytovatele GPSProvider pro detekci mýtných bran určených souřadnicemi a azimutem. |
| **DSRC EFC** | Poskytovatel využívá DSRC modulu připojený přes rozhraní SPI jednotky UTMJ pro komunikaci s mýtnými branami. |
| **EcallProvider** | Poskytovatel služby ECall, který je spjat s modulem ECall (také jej do aplikace zavádí). |

## Správce modulů

Součást jádra „Správce modulů“ má funkci v aplikaci OBU spravovat běh všech modulů aplikace. Všechny moduly jsou načítány dynamicky při startu aplikace. Moduly, které mají být spuštěny v aplikaci OBU, jsou uloženy v konfiguraci aplikace, tudíž je součást „Správce modulů“ přímo závislá hlavně na součásti jádra „Konfigurace“, která poskytuje informace o spustitelných modulech aplikace. Spustitelný modul je definován názvem souboru assembly (DLL knihovna .NET) a názvem typu objektu modulu. Při spuštění modulu (druhá fáze inicializace služby jádra Initialize) je nejdříve načtena assembly modulu do programu a poté v ní vyhledán typ modulu uvedený v konfiguraci. Pokud je datový typ úspěšně načten je zavolána metoda Start(), která je definována v abstraktní třídě ModuleBase, kterou musí zdědit všechny třídy reprezentující modul. Pokud metoda Start(), která je volána při spuštění modulu, nevrátí False ani nevyhodí výjimku, je modul spuštěn a běží. Opakem metody Start() je metoda Stop(), která spustí ukončení daného modulu.

Součást jádra „Správce modulů“ obsahuje metody pro práci s moduly, hlavně obsahuje metodu GetModule, která přijímá parametr typu String a hledá načtené moduly v rámci aplikace OBU podle jména. Další užitečnou metodou součásti jádra „Správce modulů“ je IsModuleLoaded, která přijímá parametr typu String, a vrací typ bool, pokud metoda vrátí

hodnotu True je modul načten v aplikaci OBU. Dále součást jádra „Správce modulů“ obsahuje metodu LoadModule, která dovoluje spustit modul z jiné součásti aplikace OBU, tohoto se může využívat při zavádění modulů dynamicky (kliknutí uživatelem, nějaká událost, atp).




Obrázek 13 - Abstraktní třída ModuleBase

Abstraktní třída ModuleBase je předkem všech modulů aplikace OBU. Třída obsahuje základní vlastnosti pro usnadnění vývoje modulu. Obsahuje vlastnosti Configuration, která obsahuje konfiguraci modulu, dále obsahuje vlastnost Core, která zpřístupňuje jádro aplikace modulu pro přístup k ostatním součástem aplikace. Každý modul je identifikován vlastností Name, která určuje název modulu. Abstraktní třída ModuleBase, také deklaruje základní metody, které musí modul přetížit. Tyto metody jsou Initialize(), která slouží k inicializaci (spuštění) modulu, Terminate(), která slouží k ukončení modulu. Obě zmíněné metody vracejí datový typ bool, který určuje, zda bylo spuštění nebo ukončení modulu úspěšné. Další metodou definovanou v třídě ModuleBase je metoda GetDatabaseAccess(), která zpřístupňuje, DAL (viz strana 26) na úrovni modulu ostatním součástem aplikace, například UI.

Základní funkcí každého modulu je pracovat s daty od poskytovatelů, provádět na nich různé výpočty, odesílat je pomocí poskytovatelů sítě na vzdálený server a ukládat je do databáze modulu. Moduly jsou důležitou součástí aplikace, protože právě moduly jí dávají funkcionalitu a vykonávají pro uživatele důležité operace z hlediska zpracování dat i služeb. Správce modulů pracuje s moduly, tak aby v případě pádu jednoho modulu nebyly ovlivněny ostatní moduly, tudíž pokud dojde při komunikaci s modulem k nějakému problému je vše zapsáno do logu aplikace OBU a uživatel je notifikován pomocí změny indikátoru jednotky, tudíž je zabezpečeno, že v případě problému nedojde k úplnému selhání jednotky.

V rámci projektu OBU jsem se podílel na implementaci modulů z hlediska optimalizace vláken a komunikací s jádrem. Mezi moduly vyvinuté v rámci aplikace OBU patří ECall, Fleet, Toll a Testovací modul.

## Moduly

Základní ideou modulu ECall je rychlé zavolání záchrany pro účastníky dopravní nehody. Modul ECall při detekci nárazu vozidla spustí automaticky poplach, který aktivuje ECall volání, které volá na linku 112 a přenáší informace o poloze vozidla, identifikátor vozidla a také adresu serveru, kde se linka 112 o vozidle a události dozví více. K tomu modul ECall využívá poskytovatele GSMProvider, CommunicationProvider a součást jádra „Správce HW“. Modul ECall má v rámci aplikace OBU nastavenu vysokou prioritu z důvodu rychlé odezvy software.

Modul Fleet má jako hlavní funkci zasílat pozici jednotky UTMJ na server, kdy je možné dále zobrazit pozici dané jednotky na mapě. Modul Fleet na server odesílá následující informace:

- WGS84 souřadnice jednotky
- Nadmořská výška
- Rychlost
- ID řidiče
- Datum a čas z GPS v UTC

Modul Toll využívá poskytovatelů EFC pro placení mýta na úsecích silnic a dálnic, které jsou zpoplatněny. Dále provádí modul Toll záznam do databáze při průjezdu mýtnou bránou. Do databáze modul Toll ukládá aktuální souřadnice brány, směr průjezdu, cena, datum a čas. Modul Toll podporuje různé režimy práce s mýtnými branami, podporuje jak fyzické brány (technologie DSRC), tak virtuální brány (technologie GNSS), dále podporuje tzv. „Hybridní mýto“, kdy jsou použity jak fyzické, tak virtuální brány.

# Uživatelské rozhraní aplikace OBU

## Úvod

Vzhledem k tomu, že jednotka UTMJ obsahuje HMI, které má jako součást dotykový barevný LCD display s rozlišením 640x480 a barevnou hloubkou 16bitů, je součástí jádra i uživatelské rozhraní (UI), které zajišťuje interakci s uživatelem jednotky. Uživatelské rozhraní je určeno pro přímou komunikaci mezi jednotkou a uživatelem (řidičem) a komunikace probíhá pomocí elementů UI zobrazených na obrazovce HMI. Uživatel na zobrazení dat může reagovat dotykem, pomocí dotykové vrstvy HMI, která rozpozná přesné souřadnice doteku uživatele. Tím je vytvořena obousměrná komunikace mezi uživatelem a jednotkou.

Uživatelské rozhraní aplikace OBU dovoluje grafické zobrazení dat a jednotlivých elementů UI pomocí objektů navržených pro tento účel.

Základním prvkem celého UI je zobrazení neboli „View“. Tento prvek definuje jedno zobrazení UI s rozmístěním elementů, navázání na kód aplikace (viz dále code behind) pro interakci

## Definice objektů v XML

Všechny elementy UI jsou definovány v souboru XML dané struktury, která podporuje rozmístění elementů na obrazovce, hierarchie elementů, lokalizaci obsahu elementů podle aktuálního jazyka, úpravu vzhledu podle aktuálního rozlišení displeje.

Jazyk XML byl vybrán v aplikaci z důvodu snadného zpracování ve světě aplikací postavených na .NET Frameworku, dále také podpory vnoření elementů, která je důležitá pro vyjádření vztahů jednotlivých elementů (podřízený, nadřízený, kontejner, atp). Další výhodou použití XML je dobrá čitelnost dat uložených v XML i pouhým textovým editorem, tudíž je možné snadno upravovat vzhled aplikace i bez nutnosti použití vývojového prostředí. Hlavní nevýhodou jazyka XML oproti binárním typům souborů je relativně pomalé zpracování dat, kdy je nutné zpracovávat textové části souboru.

Soubor definující zobrazení „View“ má příponu TXUI (telematix UI) a má přesně danou strukturu, kterou zpracovává vyvinutý parser. Jak můžeme vidět na ukázce níže XML elementem na nejvyšší úrovni je element View, který má několik vlastností:

Tabulka 8 - Atributy XML elementu View

| Název | **Popis** |
|---|---|
| **name** | Název zobrazení pro identifikaci. |
| **codebehind** | Úplný název třídy včetně jmenného prostoru, která implementuje třídu View. Tato třída je po načtení zobrazení vytvořena a spuštěna. |
| **backgroundimage** | Obrázek pozadí zobrazení, který je automaticky roztažen na celou obrazovku. |

Elementy UI jsou dále definovány pod XML elementem „Controls“. Element s názvem „Controls“ je také důležitý dále, protože dovoluje přidání elementů UI jako pořízených jiným elementům UI, tzv. kontejnerům. XML element „Controls“ obsahuje další elementy, které jsou již elementy UI, jejich název značí typ elementu UI. XML elementy definující UI element mohou mít pouze atributu „name“, která značí název UI elementu. Tento název je poté využíván při hledání a práci s daným elementem UI a musí být v rámci zobrazení unikátní, aby bylo možné jednoznačně identifikovat daný element. Další podmínkou názvu UI elementu je slovo nezačínající číslicí a obsahující znaky A-Z,a-z, a 0-9. Ostatní znaky zde nejsou dovoleny.

Každý XML element definující element UI obsahuje elementy, které nastavují vlastnosti UI objektů. Tyto elementy jsou pojmenovány podle názvu vlastnosti objektu a obsahují pouze atribut s názvem „value“, kde je uložena hodnota pro danou vlastnost. Vzhledem k tomu, že vlastnosti objektů UI mohou mít různé datové typy, dokáže podle typu vlastnosti parser XML provést převod z textové reprezentace na objektovou. Parser XML zobrazení podporuje následující datové typy:

- String – Textová hodnota různé délky.
- Font – Definice stylu písma, jeho velikosti, tučné, atp.
- Color – Definice barvy v prostoru RGB.
- UIImage – Obrázek ve formátu PNG, JPEG a GIF.
- Image – Obrázek ve formátu BMP, GIF s určením průhledné barvy RGB.
- Int32 – Celé číslo typu integer.
- Byte – Jeden bajt (0-255).
- Double – Reálné číslo s dvojitou přesností.
- Boolean – Logická hodnota (true/false).
- ImageList – Seznam obrázků ve formátu BMP a GIF s určením průhledné barvy.
- Enum – Výčet hodnot (podpora názvů ve výčtech).

```
<?xml version="1.0" encoding="utf-8"?>
<View name="testview" codebehind="Namespace.SomeClass" backgroundimage="image.png">
  <Controls>
    <ImageButton name="ibGPS">
      <Dimensions value="10,8,53,46" />
      <ForeColor value="@white" />
      <NormalImage value="@gps_up" />
      <DownImage value="@gps_down" />
    </ImageButton>
    <ImageBox name="pbPlaceHolder">
      <Dimensions value="0,0,100,140" />
    </ImageBox>
    <Label name="lblText">
      <Dimensions value="0,10,width-110,35" />
      <TextAlign value="center" />
      <ForeColor value="@colorHead" />
      <Font value="@fontHead" />
      <Text value="~MainText" />
      <AutoSize value="false" />
    </Label>
  </Controls>
</View>
```

Ukázkový XML souboru zobrazení neboli „View“.

Vzhledem k tomu, že v UI obecně je nutné definovat elementy, které se budou přizpůsobovat velikosti zobrazení, byl v rámci parseru XML vyvinut vyhodnocovač výrazů s podporou proměnných, které obsahují hodnoty jako šířka obrazovky, výška obrazovky. Tudíž je možné jednoduchým výrazem umístit element UI na celou obrazovku nastavením vlastnosti Dimensions takto:

```
<?xml version="1.0" encoding="utf-8"?>
<View name="testview" codebehind="Namespace.SomeClass" backgroundimage="image.png">
  <Controls>
    <ImageButton name="ibFullStreen">
      <Dimensions value="0,0,width,height" />
      <ForeColor value="@white" />
      <NormalImage value="@gps_up" />
      <DownImage value="@gps_down" />
    </ImageButton>
  </Controls>
</View>
```

Parser UI také podporuje speciální případ souboru s příponou TXUI, který se nazývá App.txui. Tento soubor neobsahuje žádné elementy UI, ale pouze obsahuje zdroje dat (Resources) a lokalizační texty včetně výchozího jazyka aplikace. Ukázkový soubor App.txui je uveden níže. Obsahuje element „App“, který je kořenový a může obsahovat elementy „ResourceSets“ a „Localizations“. Element „ResourceSets“ dále obsahuje elementy „Resources“, které musí mít atribut „resolution“. Tento atribut může mít hodnotu „all“, což znamená, že všechny zdroje v této části jsou dostupné nehledě na rozlišení. Další hodnotou atributu „resolution“ může být „default“, která definuje zdroje pro standardní rozlišení. Dále může atribut nabývat číselných hodnot, které vyjadřují šířku rozlišení obrazovky v pixelech. Element „Localizations“ musí obsahovat atribut „default“, který definuje výchozí lokalizaci. Element „Localizations“ obsahuje elementy s názvem „Localization“, které obsahují vlastnost „language", která definuje jazyk lokalizace. Tento atribut musí obsahovat ISO název jazyka o délce dvou znaků. Například „en“, „cz“,“de“, atp. Element Localization“ dále obsahuje textové hodnoty, které jsou identifikovány názvem a obsahují hodnotu.

Zde uvádím ukázku souboru App.txui:

```
<?xml version="1.0" encoding="utf-8"?>
<App>
  <ResourceSets>
    <Resources resolution="all">
      <Text name="big_font" value="Tahoma,14" />
      <Text name="gps_status" value="images/640/service_status.gif 16 16 255,0,0" />
      <Text name="colorVersion" value="255,255,200" />
      <Text name="right_panel_rect" value="width - 110,0,110,height" />
    </Resources>
    <Resources resolution="default">
      <Text name="ikeyboardkeyup" value="images\640\combobox.png" />
    </Resources>
  </ResourceSets>
  <Localizations default="en">
    <Localization language="cs">
      <Text name="main_lblHome" value="Vítejte v OBU" />
```

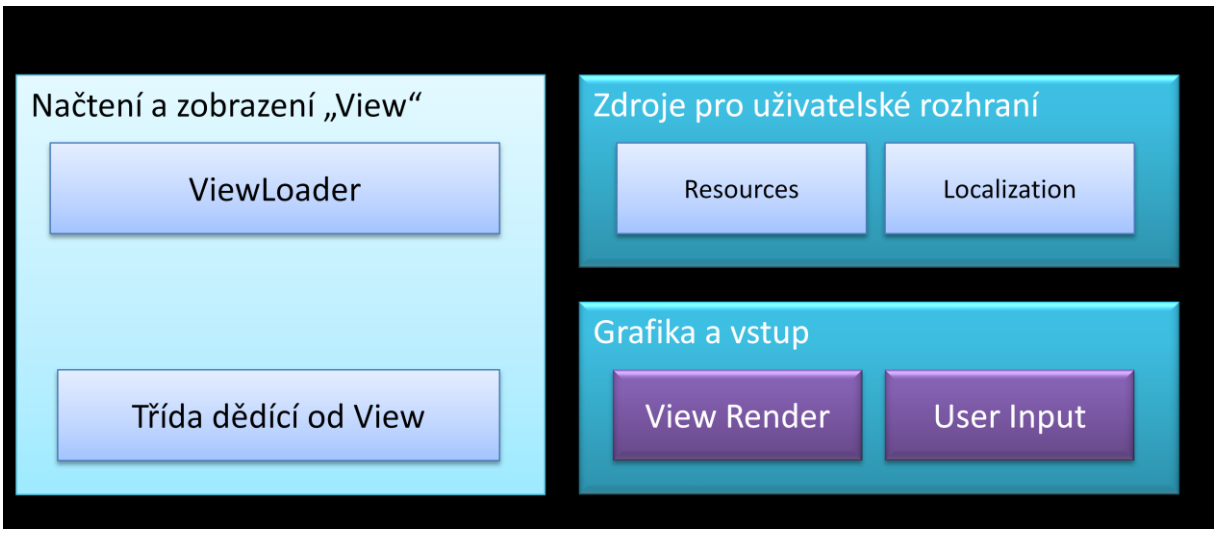
```
      <Text name="gps_status_lblHead" value="Stav GPS" />
    </Localization>
    <Localization language="en">
      <Text name="main_lblHome" value="Welcome to OBU" />
      <Text name="gps_status_lblHead" value="GPS status page" />
    </Localization>
  </Localizations>
</App>
```

Každé zobrazení může definovat i tzv. kód na pozadí (code behind), který může reagovat na různé události vyvolané uživatelem, případně zobrazovat aktualizované informace (například GPS souřadnice, atp). Kód je generován automaticky při spuštění příkazu build v IDE Visual Studio. Výsledkem generování je kód, který usnadňuje přístup k prvkům UI definovaným v XML, tyto prvky jsou pak deklarovány jako členské proměnné a programátor k nim může přistupovat pouze napsáním názvu objektu ve zdrojovém kódu. Tato vlastnost velice usnadňuje psaní kódu na pozadí, protože není nutné hledat objekty a přiřazovat si je do proměnných manuálně. Zdrojový kód je pak zkompilován společně s aplikací OBU a je spouštěn podle aktuálně zobrazeného „View“, kde je definován v atributu „codebehind“ elementu „View“.

Další výhodou oddělení kódu od definice UI je, že kromě objektových vazeb jsou oboje části UI oddělené, tudíž je možné, implementovat UI pomocí jiné technologie vykreslování (například DirectDraw) bez nutnosti změnit kód běžící na pozadí.


Obrázek 14 - Ukázka uživatelského rozhraní OBU

Uživatelské rozhraní je rozděleno do několika součástí, které se starají o načtení, inicializaci, interakci a vykreslení uživatelského rozhraní. Tyto součásti můžeme rozdělit do tří částí, které jsou viditelné na Obrázek 15 - Součásti uživatelského rozhraní.




Obrázek 15 - Součásti uživatelského rozhraní

Objekty uživatelského rozhraní pro načtení a zobrazení „View“ jsou „ViewLoader“, který načítá elementy UI z XML souboru do paměti zařízení, dále mezi tyto objekty patří třída dědící od třídy „View“, která obsahuje hlavně kód na pozadí, správu objektů a propojení mezi zobrazením a zbytkem aplikace.

Objekty uživatelského rozhraní pro správu zdrojů jsou „Resources“, které zajišťují zdroje (obrázky, texty, fonty, barvy, atp.), a objekt „Localization“, který obsahuje jazykové texty pro překlad aplikace do více jazyků.

Poslední skupinou objektů uživatelského rozhraní je grafika a vstup, kdy objekt „View Render“, který zajišťuje prezentaci elementů UI na obrazovce, a objekt „User input“, který zpracovává vstupy uživatele.

## Vytvořené objekty UI

Uživatelské rozhraní aplikace OBU obsahuje několik specializovaných objektů, které jsou vytvářeny v rámci „View“ a reprezentují různé typy elementů UI. Tyto elementy UI mají předem dané rozhraní a dědí od třídy „Control“, která je základní třídou definující základní vlastnosti, metody a události. V rámci projektu UTMJ OBU byly vyvinuti následující objekty UI:

| Název elementu UI | **Popis elementu** |
|---|---|
| **Button** | Element UI tlačítko, které je vykresleno pomocí čar, bez grafiky. Obsahuje text a událost Click. |
| **ImageBox** | Element UI, který zobrazuje na obrazovce na určité pozici obrázek. |
| **ImageButton** | Element UI tlačítko, které je reprezentováno dvěma obrázky, které jsou střídány podle stavu tlačítka. Stav tlačítka může být stisknuto nebo puštěno. Obsahuje událost Click. |
| **ImageComboBox** | Grafický rozbalovací seznam, který je kreslen pomocí částí obrázků, který je nastaven. |
| **Label** | Zobrazuje krátký text uživateli, lze nastavit písmo a barvu. |
| **Listbox** | Seznam položek, který podporuje posun zobrazení tažením prsty po displeji. |
| **ProgressBar** | Zobrazuje stav akce a kolik zbývá k dokončení. |
| **SattelitesView** | Zobrazuje satelity na obloze v kruhovém diagramu, u každého satelitu ukazuje sílu signálu a jeho identifikační číslo. |
| **StatusIndicator** | Zobrazuje část obrázku, podle nastaveného indexu. Může zobrazovat například dostupnost GSM spojení, GPS signálu. Interně používá elementu ImageList. |
| **ImageList** | Nevizuální element UI, který vrací část obrázku podle indexu. Tento index definuje dlaždici ve větším obrázku, která má určité rozměry. |
| **LayoutContainer** | Element UI, který dovoluje elementy vložené do něj umístit podle svého nastavení a podle jeho rozměrů. |

## Návrhář UI

Návrhář uživatelského rozhraní byl v projektu UTMJ OBU používán jako návrhová aplikace pro jednotlivé zobrazení v rámci aplikace. Návrhář podporuje uživatelsky přívětivou formou upravovat soubory typu App.txui, tak ostatní View.txui, které obsahují definici View. Návrhář zobrazuje uživatelské rozhraní stejně jako je tomu v aplikaci OBU, k tomu bylo uživatelské rozhraní zkompilováno pro běh na PC, tímto bylo dosaženo výsledku kdy to co je zobrazeno v návrháři je i zobrazeno na jednotce UTMJ. Díky této aplikaci mohlo být uživatelské rozhraní jednoduše navrženo a upravováno.


Obrázek 16 - Návrhář uživatelského rozhraní aplikace OBU

Aplikace zobrazuje náhled uživatelského rozhraní, dovoluje jej upravovat, tak že změny se okamžitě projeví v zobrazení. Dále lze jednotlivé elementy UI umísťovat na plochu zobrazení a libovolně s nimi pohybovat. Návrhář dokáže simulovat různá rozlišení, tak aby bylo možné vyzkoušet nastavení jednotlivých elementů v jiných rozlišeních. Také podporuje změnu zobrazovaného jazyka, tudíž je možné zkontrolovat správnost a funkčnost lokalizace.

V případě otevření souboru s názvem App.txui návrhář zobrazí uživatelské rozhraní, které dovoluje upravovat jednotlivé zdroje dat (obrázky, texty, atp) a lokalizace.

Aplikace také dovoluje zobrazení celého souboru typu txui v poznámkovém bloku pro přímou editaci obsahu, nebo pro snadné kopírování elementů.

## Závěr

Diplomová práce obsahuje popis univerzální telematické mobilní jednotky OBU po softwarové stránce. Univerzální telematická mobilní jednotka OBU je zařízení do vozidla, které je určeno pro provoz telematických služeb. Mezi tyto služby můžeme zařadit záchrané volání v případě nehody eCall, dohledový systém vozidla fleet (vozový park) a služba pro výběr elektronického mýta (Toll).

Diplomová práce se konkrétně zaměřuje hlavně na samotné jádro aplikace a uživatelské rozhraní, které využívají ostatní součásti, jako jsou moduly a poskytovatelé. Jádro aplikace je velice důležité pro funkčnost ostatních částí aplikací, protože poskytuje důležité informace, služby a v neposlední řadě celou aplikaci spojuje, tak že funguje jako jeden celek. V práci je popsána realizace samotného jádra a jeho služeb. Tyto služby jsou rozděleny do několika částí podle svého zaměření, tak aby všechny metody dané služby jádra měly souvislost a tvořily tak službu poskytující ucelené funkce.

Všechny služby jádra byly identifikovány jako znovu použitelné a tudíž samotné jádro aplikace může sloužit k vývoji odlišných aplikací bez nutnosti programovat nové služby jádra znova a znova. Dále je aplikace navržena, tak že si lze pouze postavit aplikaci z již vytvořených částí a pouze podle potřeb upravit konfiguraci a aplikace může běžet.
V diplomové práci není popsána pouze samotná realizace, je zde i popsán úspěšný návrh architektury mobilní aplikace běžící na zařízení v automobilu, který úspěšně splnil požadavky na aplikaci a také testy, které byly prováděny jak při vývoji, tak po něm pro ověření funkcionality.

Mezi základní služby jádra patří: konfigurace, logování (tvorba záznamů o běhu aplikace), aplikační informace, správce hardware zařízení, správce databáze, správce poskytovatelů, správce modulů a uživatelské rozhraní. Všechny služby jádra jsou plně konfigurovatelné, tudíž lze změnit jejich chování snadnou úpravou konfiguračního souboru.

Práce seznamuje i s dalšími důležitými aspekty celého projektu, jako je například samotný hardware UTMJ, který je popsán hned zpočátku práce. Hardware UTMJ byl vyvinut společností Honeywell, se kterou jsem spolupracoval na vývoji jádra. Dále jsou v rámci diplomové práce krátce popsány všechny vyvinuté moduly, na kterých jsem spolupracoval hlavně z pohledu správné provázanosti s jádrem celé aplikace a také vylepšování rozhraní mezi jádrem a ostatními součástmi.

Přínos projektu UTMJ OBU jako celku vidím ve zlepšení situace na silnicích, hlavně v podobě zrychlení příjezdu záchranných složek na místo nehody v podstatně kratším čase, což v případě nehody, kdy u těžších případů hrají roli i minuty vidím jako podstatné zlepšení záchrany života. Přínos samotné práce vidím v popsání navržené architektury aplikace, která má svá specifika plynoucí z požadavků projektu. Například modularitu, spolehlivost, flexibilitu, přesnost, spojitost a integritu. Poslední část diplomové práce je věnována návrhu uživatelského rozhraní, které odděluje vzhled uživatelského rozhraní reprezentovaným elementy jazyka XML, od výkonného kódu aplikace, v neposlední řadě se diplomová práce

zabývá vytvořením návrháře UI, který dokáže pracovat se UI souborem a upravovat jej uživatelsky intuitivním způsobem.

V rámci projektu UTMJ OBU jsem si mohl vyzkoušet jak roli programátora, analytika, tak i vedoucího programátora, který zodpovídá za celé dílo a jeho funkčnost. Což mnohdy znamenalo být v práci do pozdních večerních hodin. V rámci role vedoucího programátora malého týmu pro mne znamenalo se dobře orientovat v kódu ostatních programátorů, což bylo důležité v situacích, kdy se program nechoval korektně nebo docházelo k chybám a bylo nutné chování nebo chyby opravit. Pro mne osobně byl tento projekt velkou zkušeností, která mne v mnohých směrech posunula dále.

## Citovaná literatura

[1] Ministerstvo průmyslu a obchodu, Projekt 2A-1TP1/138 [online]. 07/2006. Dostupný z WWW: <http://www.isvav.cz/projectDetail.do?rowId=2A-1TP1%2F138>

[2] Microsoft corporation, Microsoft SQL Server 2005 Compact Edition Overview [online]. 2005. Dostupný z WWW: < http://download.microsoft.com/download/A/4/7/A47B7B0E-976D-4F49-B15D-F02ADE638EBE/SSCEOverview.doc>

[3] SQLite, About SQLite [online]. 03/2011. Dostupný z WWW <http://www.sqlite.org/about.html>

[4] GUID, Wikipedia.org [online]. 03/2011. Dostupný z WWW

<http://en.wikipedia.org/wiki/Globally_unique_identifier>

[5] O nás – Honeywell Česká Republika [online]. 03/2011. Dostupný z WWW <http://www.honeywell.com/sites/cz/O-nas.htm>

[6] LACKO, Luboslav. Programujeme mobilní aplikace ve Visual Studiu .NET. Computer press, 2004, 470 s. ISBN: 80-251-0176-2

[7] Microsoft corporation, Visual C# [online]. 03/2011. Dostupný z WWW: < http://msdn.microsoft.com/library/kx37x362.aspx>