VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

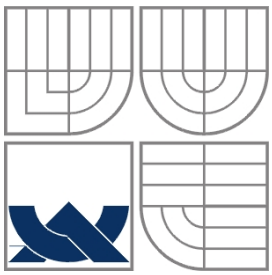
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# RFID INVENTARIZAČNÍ SYSTÉM

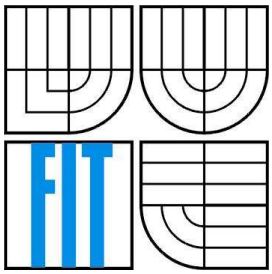
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE JIŘÍ ŠINDELKA
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# RFID INVENTARIZAČNÍ SYSTÉM

RFID INVENTORY SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE JIŘÍ ŠINDELKA
AUTHOR

VEDOUCÍ PRÁCE ING. JAROSLAV MARUŠINEC, PH. D., MBA
SUPERVISOR

BRNO 2007

# Abstrakt

Tato diplomová práce se zabývá studiem systémů souvisejících s inventarizací a evidencí majetku. Cílem je prostudovat systém SAP, IS Apollo, databázovou platformu Oracle 10g, technologie RFID a provést průzkum trhu na zařízení pro zadaný čip EPCglobal UHF Gen2. Na základě průzkumu trhu navrhnout a implementovat inventarizaci pomocí RFID. Při implementaci se zaměřit na hromadné sejmutí RFID tagů. Dále bude implementována aplikace simulující modul evidence majetku a bude navržen přenos dat do této aplikace, který se bude dát využit i při přenosu dat do IS Apolla. Implementaci inventarizace majetku jsem se rozhodl řešit ve vývojovém prostředí Microsoft Visual Studio 2005 Standart Edition C#. Aplikaci simulující modul evidence majetku, přenos dat do této aplikace jsem se rozhodl řešit ve vývojovém prostředí Borland Delphi 7.



# Klíčová slova

SAP, IS Apollo, Oracle 10g, RFID, EPCglobal UHF Gen2, Microsoft Visual Studio 2005 Standard Edition C#, Borland Delphi 7.



# Abstract

This project deals with a survey of inventory system and accounting property. The purpose of the project is to study the SAP system, IS Apollo, the database platform Oracle 10g, the RFID technology and to make a marketing research of a device for the chip EPCglobal UHF Gen2. On base marketing research propose and implement RFID inventory. At implementation target will be collective reading RFID tags. Further I will implement application simulating evidence systems and I will design data transmission to those application that will be also used at transmission data to the IS Apolla. I decided to implement the inventorying in the development environment Microsoft Visual Studio 2005 Standard Edition C#. Application simulated accounting property and data transmission I decide to implement in the development environment Borland Delphi 7.



# Keywords

SAP, IS Apollo, Oracle 10g, RFID, EPCglobal UHF Gen2, Microsoft Visual Studio 2005 Standart Edition C#, Borland Delphi 7.


# Citace

Jiří Šindelka: RFID inventarizační systém, diplomová práce, Brno, FIT VUT v Brně, 2007

# RFID inventarizační systém

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaromíra Marušince, Ph.D., MBA. Další informace mi poskytli Ing. Rudolf Musil, Ing. Markéta Přidalová. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Šindelka
22.5.2007

## Poděkování

Chtěl bych poděkovat svému vedoucímu projektu Ing. Marušincovi, Ph.D., MBA za poskytnutí všech potřebných informací a kontaktů při zahájení projektu. Dále bych chtěl poděkovat Ing. Musilovi za poskytnutí všech potřebných informaci a rad při řešení projektu.

# Obsah

# Seznam obrázků



# Seznam tabulek

# 1 Úvod

RFID inventarizace majetku není jednoduchou problematikou. Projekt by měl navázat na projekt inventarizace pomocí č. kódu nazvaný Agenda majetku na VUT, který jsem řešil minulý rok.

Příchod RFID čipů je dalším logickým krokem v oboru automatické identifikace. Vytváří nám lepší komfort při sběru dat a nenahraditelného pomocníka v dalších odvětvích. Většina obchodních sítí by jsi dnes těžko dokázala představit hlídání zboží bez pomoci RFID čipů. Také ve velkých skladech hraje RFID velkou roli při sledování zboží. V nedaleké budoucnosti se RFID začne čím dál tím více prosazovat. Zavedením samoobslužných pokladen bylo dříve jen přání odborníků a dnes si již zákazníci jiných zemí mohou tohoto komfortu dopřát. Stejně tak zavedení elektronického mýtného je v první fázi opřeno o technologii RFID. I když je RFID novou technologií hraje v mnoha odvětvích velkou roli a proces evidence a inventarizace majetku by neměl být výjimkou. Dalším příkladem použití RFID technologie je vytvoření řízeného skladu v hale bez skladových pozic. RFID tagy jsou umístěny v pravidelné síti v podlaze a při uložení palety na místo je automaticky zaznamenám čárový kód palety a pozice určená RFID tagem v podlaze.

Z příchodem RFID neodpadají problémy spojené s evidencí majetku. Dobrá evidence majetku a jeho pravidelné kontrolování šetří prostředky a zpřehledňují operace s majetkem, může upozornit na opatření, které je nutné provést (např. hodnota majetku v místnosti je příliš velká na aktuální zabezpečení…). Důležitou součástí evidence majetku je umístění tohoto majetku. Je-li evidence majetku kvalitní a kontrola pravidelná, není těžké jakýkoliv majetek bez problému nalézt. Evidence majetku na osobu nám také zajistí odpovědnost za majetek a v případě ztráty nebo poškození jednoduše sjednat nápravu.

## 1.1 Současný stav

V současné době je evidence majetku vedena v systému SAP. SAP je komplexní ekonomický systém a evidence majetku je jen jedním z mnoha modulů. SAP umožňuje všechny potřebné operace s majetkem. SAP je moderní, známý, kvalitní produkt a také je velmi drahý. Pro každou instalaci je nutné zakoupit licenci. Rozšíření systému SAP pro všechny zaměstnance VUT je nereálné a hlavně zbytečné. SAP byl vybrán jako nový ekonomický systém pro VUT v Brně. Předchůdcem byl ekonomický systém Ekonfis, který po rozšíření VUT a zvyšováním nároků přestal vyhovovat.

V minulém roce byl realizován modul IS Apolla evidence majetku. Tento modul umožní každé osobě na VUT evidenci svého majetku a majetku svých podřízených. Tento modul je součástí IS Apolla, takže je přístupný všem osobám na VUT bez nutnosti zakoupení licence.

Inventarizace majetku je v současné době prováděna pomocí programu ProBase. Tento systém zavádí automatickou identifikaci majetku pomocí č. kódů, zpřehledňuje a urychluje inventuru. Cílem

projektu Agenda majetku na VUT byla integrace inventarizace majetku do modulu evidence majetku. Vzhledem k souběžnému vývoji modulu evidence majetku a inventarizace tento cíl splněn nebyl.

## 1.2 Cílový stav

Protože je v současné době možné provádět inventuru majetku jen pomocí programu ProBase, rozhodlo se vedení Centra výpočetních a informačních služeb, využít modul pro evidenci majetku a integrovat inventarizaci do tohoto modulu. Tento přístup umožní každé osobě na VUT provést inventuru svého majetku a majetku svých podřízených. Dalším rozšířením inventarizace a také stěžejním bodem tohoto projektu je zavedení technologie RFID.

Výsledkem tohoto projektu by mělo být prostudovat stávající systém Apollo a navrhnout další kroky potřebné k integraci inventarizace majetku do modulu pro evidenci majetku. Tento modul byl řešen jako bakalářská práce studentem Václavem Bezděkem. Dále bych se měl seznámit se systémem Oracle 10g, který figuruje jako databázový server centrálního databázového skladu. Pro pochopení evidence majetku je také potřeba seznámit se se systémem SAP. Hlavní části projektu bude prostudování technologii RFID, dostupných zařízení pro zadaný čip Gen2 standart EPCglobal na frekvenci 868 MHz, realizace inventarizace majetku pomocí RFID a návrh modulu pro komunikaci s vybraným zařízením. Součástí modulu pro komunikaci by měl být také přenos dat do IS Apolla.

Výsledkem inventury by měla být tisková sestava nebo dokument v elektronické formě, který bude dále zpracován manuálně v systému SAP. Zpět do systému SAP se nebudou žádná data předávat.

# 2 Analýza technologií a systémů

Před samotnou implementací bylo nutné seznámit se se stávajícími systémy a potřebnými technologiemi. Vzhledem k tomu, že inventarizace je úzce spojena s evidencí majetku, zaměřil jsem studium požadovaných systémů tímto směrem.

## 2.1 SAP

Evidence majetku je vedena v systému SAP, a proto je nutné tento systém analyzovat. Nejdůležitější je prostudování jednotlivých tabulek, které se týkají majetku. Analýza systému SAP je pouze informativní, protože tuto část by měl řešit modul evidence majetku systému Apollo.

- ANLA – Základní data o majetku. Tabulka obsahuje pro každý majetek jeden záznam.
- ANLZ – Časová data majetku. Tabulka obsahuje pro každý majetek 1 – N záznamů.
- ANLC – Hodnoty majetku. Tabulka obsahuje pro každý majetek 1 – N záznamů.

Následující tabulky zobrazují jednotlivá pole analyzovaných tabulek a jejich stručný popis. Analýze jsem podrobil jen vybrané sloupce tabulek, které by mohly být užitečné při evidenci a inventarizaci majetku.

| Pole | Popis |
|---|---|
| MANDT | klient |
| BUKRS | účetní okruh |
| ANLN1 | hlavní číslo majetku |
| ANLN2 | vedlejší číslo majetku |

**Tabulka 2.1. Společná pole tabulek**

| Pole | Popis |
|---|---|
| ANLKL | třída majetku |
| AKTIV | datum aktivace majetku (zařazení do systému) |
| DEAKT | datum deaktivace majetku |
| MENGE | množství |
| MEINS | měrná jednotka |
| INVNR | inventární číslo ze systému EKONFIS |
| TXT50 | název majetku |
| TXA50 | název majetku (možné pokračování) |
| SERNR | sériové číslo |
| URJHR | rok výroby |

**Tabulka 2.2 Vybraná pole tabulky ANLA**

| Pole | Popis |
|---|---|
| ADATU | počátek doby platnosti intervalu |
| BDATU | konec doby platnosti intervalu |
| GSBER | součást, které majetek patří |
| KOSTL | nákladové středisko, kterému majetek patří |
| WERKS | areál umístění majetku |
| STORT | budova umístění majetku |
| RAUMN | místnost umístění majetku |
| PERNR | číslo pracovního poměru zodpovědného pracovníka za majetek |

**Tabulka 2.3 Vybraná pole tabulky ANLZ**

| Pole | Popis |
|---|---|
| GJAHR | fiskální rok |
| AFABE | oblast ocenění majetku |
| KANSW | kumulované pořizovací a výrobní náklady KAUFW |
| KMAFA | přenos kumulovaných rezerv |
| KNAFA | kumulované normální odpisy |
| NAFAG | roční zaúčtovaný normální odpis |
| NAFAL | roční podílový normální odpis |
| NAFAV | podílový kumulovaný normální odpis |
| KANSW | kumulované pořizovací a výrobní náklady KAUFW |
| KMAFA | přenos kumulovaných rezerv |
| KNAFA | kumulované normální odpisy |

**Tabulka 2.4 Vybraná pole tabulky ANLC**

## 2.2 IS Apollo

IS Apollo je součástí třívrstvé architektury a plní funkci klienta. Celý systém je založen na modularitě. Uživatel si stáhne pouze moduly, které chce využívat. Kvůli své modularitě není systém náročný pro uživatele a podporuje týmový vývoj. Každý z vývojářů si může pracovat na svém modulu. Klient zajišťuje:

- Automatickou aktualizaci modulu
- Připojení k aplikačnímu serveru
- Komunikaci s aplikačním serverem
- Stahování a spouštění potřebných modulů
- Uložení vlastního nastavení

- Automatické odhlášení a uzavírání spojení po delší době nečinnosti

V třívrstvé architektuře byla mezi klienta a databázi vložena vrstva nazývána aplikační server. Tento aplikační server má název Akira a jeho hlavním úkolem je snížení nároků na klienta. Veškeré datové a funkční součásti jsou přesunuty na tento server. Klient má za úkol pouze zobrazení dat a interakci s uživatelem.

Mezi hlavní výhody třívrstvé architektury patří možnost zvýšení výkonu systému přidáním dalších, výkonnějších serverů. Centralizace dat a aplikací (modulů) umožňuje lepší zabezpečení dat a upgrade aplikací.

Apollo komunikuje pouze s aplikačním serverem pomocí zabezpečeného protokolu SSL (Secure Socket Layer). Server Akira také zprostředkovává připojení k centrálnímu databázovému serveru, který tvoří poslední vrstvu této třívrstvé architektury. Tento server je založen na databázovém systému Oracle 10g.

Synchronizace mezi systémem SAP a centrálním datovém serveru probíhá pravidelně každých 24 hodin a to ve 4:03 v noci. Je spuštěn skript napsaný v jazyce *Perl*, který načte data z databáze SAP a převede je do databáze centrálního datového serveru. Skript pracuje tak, že se přihlásí do databáze SAP a provede výběr potřebných dat. Provede kontrolní součet *udp_hash* řádků tabulek. Poté tyto kontrolní součty porovná s hodnotu *udp_hash* tabulek centrálního datového skladu. Aktualizace se provádí pouze u řádků s různou hodnotou kontrolního součtu. Tento přístup je velice rychlý a zajistí, že synchronizace dat trvá jen několik minut.[2]

**Spuštění**

Po spuštění Apolla jsou zaslány zprávy všem aplikačním serverům a provede se připojení na nejméně vytížený server. Neodpoví-li žádný server, Apollo o této události uživatele informuje a sám se ukončí.

Po připojení na nejméně vytížený server se provede kontrola verze požadovaného modulu a případné stáhnutí novějšího modulu.

Jsou-li předchozí kroky úspěšné, přichází na řadu autorizace uživatele. Uživatel je vyzván k zadání loginu a hesla. Po verifikaci zadaných údajů je Apollu umožněno připojení.

Následuje stažení osobního nastavení uživatele a spuštění základního modulu *Desktop*, který umožňuje zobrazení oprávnění a základní nastavení.[2]


**Obrázek 2.1 Spouštění Apolla**

### Databázový přístup

Vzhledem k tomu, že Apollo je aplikační klient, měl by mít možnost přístupu k datům na aplikačním serveru. Data z databáze zprostředkovává aplikační server *Akira*.

Nelze zadávat přímo SQL dotazy, ale lze spouštět dotazy, které jsou na tomto serveru uloženy. SQL dotazy jsou uloženy v oddělené tabulce. Jsou volány svým názvem a parametry dotazu. Každý dotaz je svázán se svým modulem a je omezen přístupovými právy, který musí uživatel mít, aby jej mohl vykonávat.

Vzhledem k povaze projektu jsem se zajímal pouze o práci s výběrovým dotazem. Unita *uAkira.pas* implementuje rozhraní spouštění dotazů. Funkce *FastQueryOpen* spouští SQL dotaz typu SELECT. Prefix „Fast“ je v tomto případě na místě, protože lze funkci specifikovat, zda se má výsledek této funkce provést postupně nebo naráz. Parametry *FastQueryOpen*:

- Název dotazu
- Seznam parametru
- Proměnná pro vrácení výsledků
- Příznak již zmiňované možnosti postupného vracení výsledku.

Výsledkem je proměnná odvozená z třídy *TDataset*. Tato třída je základní třídou pro všechny databázové komponenty. Práce s ní je jednoduchá a intuitivní. Důležité metody:

- *First* – nastaví kurzor na první záznam
- *Last* – nastaví kurzor na poslední záznam
- *Next* – přesune se na další záznam
- *Prev* – přesune se na předchozí záznam
- *FieldByName* – Vrácení pole aktuálního záznamu jako objekt třídy *TField*. Tento objekt je variantní a může být takto i uložen, a nebo pomocí metod třídy *Variant* přetypován na požadovaný typ.

Důležité vlastnosti:

- *Eof* – indikace konce TDatasetu
- *Bof* – indikace začátku TDatasetu

**Princip komunikace s modulem**

Je-li dán požadavek na spuštění modulu na aplikační server, provede se kontrola na novější verzi modulu. V případě nalezení novější verze se provede aktualizace a spuštění příslušného modulu. Poté je vyvolána událost *AfterCreate* a běh programu přebírá modul. Při požadavku na uzavření modulu je vyvolána událost *OnCanClose* a při uzavření *OnDestroy*.

Komunikace mezi modulem a Apollem probíhá prostřednictvím zasílání zpráv. Zprávy vyvolají události, které lze obsloužit. Zaslání zprávy z modulu lze realizovat funkcí *FceCMD*, která má jako první parametr typ zprávy a druhý parametry zprávy. Obsluha zpráv zaslaných klientem je řešena událostmi, které definuje třída *TAppFrame*.[2]

| **Událost** | **Popis(klávesová zkratka)** |
|---|---|
| *OnDBRefresh* | obnovení dat (F5) |
| *OnDBFirst* | požadavek přesunutí na první záznam v tabulce (F7) |
| *OnDBNext* | požadavek přesunutí na další záznam v tabulce (F9) |
| *OnDBPrev* | požadavek přesunutí na předchozí záznam v tabulce (F8) |
| *OnDBLast* | požadavek přesunutí na poslední záznam v tabulce (F10) |
| *OnDBExport* | požadavek uživatele na export dat |
| *OnDBImport* | požadavek uživatele na import dat |
| *OnDBFind* | požadavek uživatele na hledání dat (CTRL + F) |
| *OnDBNew* | požadavek na vytvoření nového záznamu |
| *OnDBDelete* | požadavek na smazání vybraných záznamů |
| *OnDBPrintBefore* | inicializace před tiskem |
| *OnDBPrint* | tisk dokumentu |

| | |
|---|---|
| *OnDBPrintAfter* | ukončení tisku |
| *OnDBSave* | požadavek uživatele na uložení dat (CTRL + S) |
| *OnMessage* | událost pro posílaní zpráv |
| *OnMessageData* | událost pro posílaní zpráv s ukazatelem na poskytovaná data |

**Tabulka 2.5 Popis událostí TAppFrame**

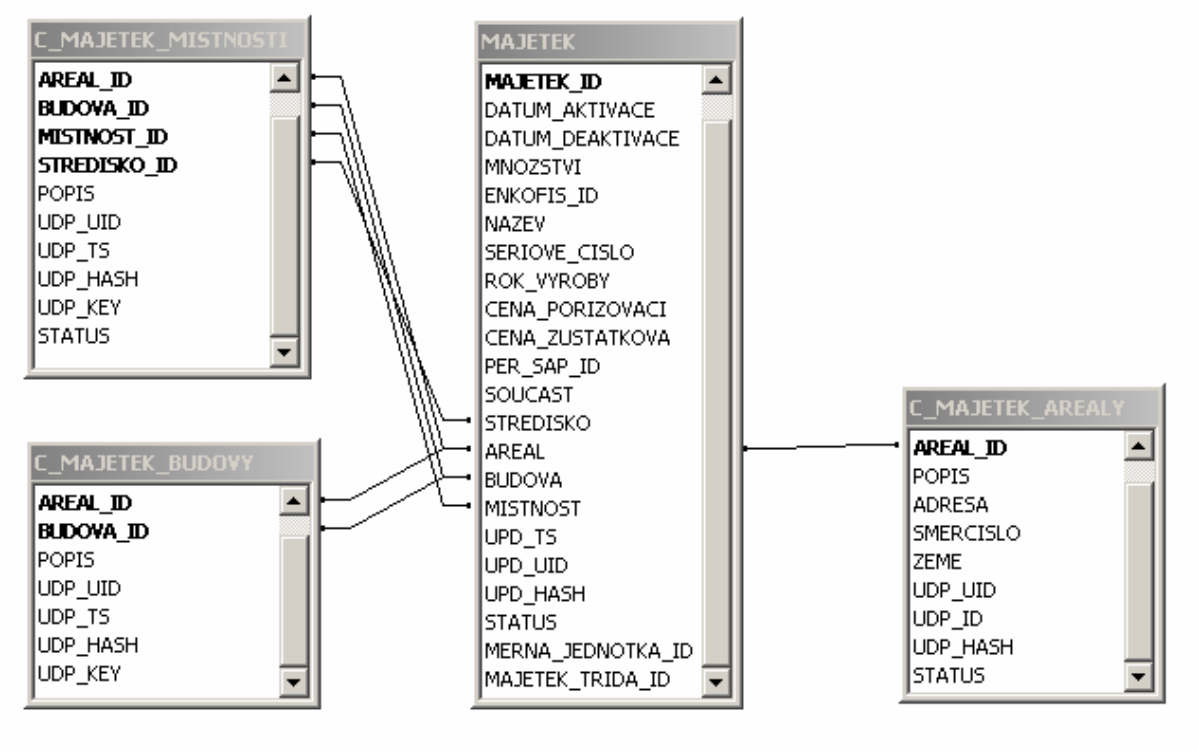
### 2.2.1 Modul evidence majetku

I když součásti zadání není studium modulu evidence majetku, považoval jsem za nutné prostudovat databázovou strukturu, který tento modul využívá.

Podstata modulu evidence je taková, že každý vidí svůj majetek a majetek svých podřízených. Integrace inventarizace do tohoto modulu by spočívala v přidání operací „zahájit inventuru“ (zkopírování dat o vybraném majetku do zařízení) a „ukončit inventuru“ (zkopírování data ze zařízení). Modul přidává do centrálního datového serveru následující tabulky:

- majetek
- majetek_zmena
- majetek_popis
- c_majetek_arealy
- c_majetek_budovy
- c_majetek_mistnosti
- c_majetek_mnozstvi
- c_majetek_popis_typ
- c_majetek_tridy

Inventarizace by měla využívat jen část těchto tabulek pro načítání dat o majetku. Hlavní částí tohoto databázového návrhu je tabulka majetek, která bude také nejvíce využívána pro inventuru. Tabulka obsahuje všechny potřebné informace o majetku a odkazy na jeho popis a umístění.




**Obrázek 2.2 Struktura vybraných tabulek**

# 2.3 Oracle 10g

Oracle 10g je komerční databázový systém od firmy Oracle pro střední a větší firmy. Je to robustní platforma s garancí pro velké objemy dat. Je označován za první relační databázi podporující grid computing, který umožňuje využití výpočetního výkonu neomezeného počtu geograficky oddělených počítačů. Mezi další vlastnosti patří objektová podpora, přístup XML, bezpečnost zajišťovaná virtuálními privátními databázemi, analytické funkce pro datové sklady a mnohé další. Další funkcí, která by se dala využít v příští fázi tohoto projektu, je možnost práce s prostorovými daty (grafické znázornění místnosti a majetku). Databázová platforma Oracle 10g je dostupná pro většinu dnes známých operačních systémů včetně Linux a Windows.

Centrální datový sklad VUT je postaven na databázovém systému firmy Oracle 10g. Existuje několik instancí databáze: CDBX, CISB a CISC. CDBX je oficiální datový sklad VUT v Brně, k němuž se připojuje nejenom celoškolský informační systém Apollo, ale i Portál VUT v Brně a další. CISB má identické nastavení jako CDBX a slouží pro výzkum a vývoj. CISC je kopie CDBX, její data jsou nepravidelně obnovována a slouží hlavně jako testovací databáze pro uživatele.

Přístup k databázi je chráněn nejen uživatelským jménem a heslem, ale i IP adresou. Toto opatření zvyšuje zabezpečení serveru a snižuje pravděpodobnost neoprávněného přístupu. [3][4]

## 2.4 RFID

RFID je zkratka anglických slov Radio Frequency Identification. Do češtiny by šla tato zkratka přeložit jako radiofrekvenční systém identifikace. Je to moderní technologie identifikace objektů pomocí radiofrekvenčních vln. Technologie RFID je nasazována do systému s požadavkem na rychlé a přesné zpracování dat. Další obrovskou výhodou je bezkontaktní čtení informací.

Informace jsou v elektronické podobě ukládány na čipy, kterým se říká tagy. Tagy lze opakovaně číst i do nich zapisovat pomocí radiových vln. Čtení a zápis tagů je prováděn hromadně na rozdíl od čárových kódů. Současná zařízení dokáží zpracovat až stovky tagů za minutu.

RFID je považován za přímého nástupce čárových kódů i když se nepředpokládá, že by RFID zcela nahradilo čárové kódy. Na trhu jsou odvětví, kde bude RFID dominovat, ale to samé se dá říct o čárových kódech. V současné době se většinou používá kombinace obou technologií.

S myšlenkou na identifikaci objektu pomocí radiových vln přišla firma WalMart, která před několika desítky let stála u zrodu čárových kódů. Hlavní myšlenkou byla identifikace objektů na větší vzdálenost, bez nutnosti přímé viditelnosti. Dalším neméně důležitým požadavkem bylo zpracování více objektů současně. V současné době se technologie RFID velice rozvíjí o dochází k nasazení v mnoha odvětvích trhu. Hlavně v logistice, výrobě, sledování objektů, sledování majetku, sledování zavazadel a také evidence osob a majetku.[5]

### 2.4.1 Rozdělení tagů

Stejně jako u čárových kódu je informace zaznamenána na nosič dat tzv. RFID tag, který je připevněn na sledovaný objekt. Tag obsahuje malý čip s anténou a pamětí. Na čipu je uložen EPC (electronic produkt code) tagu. EPC je jednoznačná identifikace tagu. V současné době se používá EPC o délce 96 bitů, které se výrobcům přiděluje centrálně v řadách.

| Část EPC | Význam |
|---|---|
| 8 bitů | hlavička EPC |
| 28 bitů | informace o firmě |
| 24 bitů | třída výrobků |
| 36 bitů | unikátní číslo výrobku |

**Tabulka 2.6 Struktura EPC**

Současně používaných 96 bitů by mělo poskytnout dostatečný číselný prostor - 268 milionům výrobců, 16 miliónů druhů výrobků a 68 miliard sériových čísel. Vzhledem k tomu, že zatím není znám způsob upotřebení tolika EPC, je možné používat EPC o délce 64 bitů.

EPC přiděluje standardizační organizace EPCglobal. Mezi členy EPCglobal dnes patří stovky renomovaných světových firem. V České republice je zastoupena prostřednictvím sdružení GS1 Czech republic.

**Rozdělení dle napájení**

Aktivní čipy vysílají sami své informace do okolí, používají metodu TTF (tag talks first). To, že čipy vysílají své informace umožňuje miniaturní baterie, kterou jsou tyto čipy vybaveny. Nevýhodou aktivních čipů je snížená odolnost, způsobená přítomností baterie, pravidelná výměna této baterie a vysoká pořizovací cena. Výhodou je několikanásobná vzdálenost čtení (až 100m). Velikost paměti může být až 100Kb. Největší uplatnění je v oboru sledování lidí, zvířat, vozového parku a všude tam, kde se počítá s opětovným použitím čipu.

Pasivní čipy jsou cenově výrazně levnější, mají různou vzdálenost čtení od několika centimetrů do 10m, dlouhou životnost čipu a používají metodu (RTF reader talk first). Pomocí vln vyzářených z čtecího zařízení je nabita anténa a informace uložené v tagu jsou vyslány zpět do zařízení. V současné době jsou nejvíce rozšířeny pasivní čipy a to zejména kvůli své nízké ceně, nenáročnosti na obsluhu a odolnosti. Velikost čipu může být od 64 bitů do 256 bitů.

Dalším typem jsou semiaktivní čipy. Jsou to v podstatě aktivní čipy, které obsahují baterii pouze pro posílení dosahu. Tyto čipy mohou mít dosah až 18 m.

**Rozdělení dle možnosti zápisu**

Možnosti zápisu jsou úzce spojeny s rychlostí čipu, velikostí EPC a jsou rozděleny do různých tříd.

| Třída tagu | Popis |
|---|---|
| Class 0 | pouze pro čtení, programováno ve výrobě, 64 nebo 96 bit, čtení 1000 tagů/sec |
| Class 1 | zápis jednou/zápis mnohokrát, programováno při použití, 64 nebo 96 bit, čtení 200 tagů/sec |
| Class 0+ | čtení/zápis, programováno kdykoliv, 256 bit, čtení 1000 tagů/sec |
| Gen 2 | čtení/zápis, programováno kdykoliv, 256 bit, čtená 1600 tagů/sec |

**Tabulka 2.7 Rozdělení dle tříd**

**Rozdělení dle frekvence**

Systémy RFID se provozují na různých vlnových délkách. Volba nejvhodnější frekvence je jedna z nejdůležitějších fází návrhu takovéhoto řešení. Z této volby totiž vyplývá celá řada dalších omezení, jako například dosah čtecího zařízení, zákonná omezení, rychlost čtení a zapisování, použitelnost v různém prostředí a další.

| Frekvence | Název | Popis |
|---|---|---|
| 125 – 134 KHz | LF -Nízká frekvence | • dosah pod 0.2 m<br>• malá rychlost čtení<br>• kontaktní snímání<br>• vysoké výrobní náklady<br>• možnost snímání na kovu a přes kapalinu |

| | | |
|---|---|---|
| 13.56 MHz | HF - Vysoká frekvence | • použití: čipování zvířat, docházkové karty<br>• dosah do 1 m<br>• dostatečná rychlost čtení<br>• vysoké výrobní náklady<br>• obtížné čtení přes kapalinu |
| 860 – 930 MHz | UHF - Velmi vysoká frekvence | • použití: docházkové systémy, smart karty<br>• dosah do 3 m<br>• velká rychlost čtení<br>• nelze číst přes kapalinu<br>• obtížné čtení z kovu<br>• levná výroba tagu<br>• požití: třídění zásilek na letištích, sledování pohybu vratných obalů, sledování palet ve výrobě, sledování kontejnerů v překladištích |
| 2.45, 5.8 GHz | MW - Mikrovlnná frekvence | • dosah do 10 m<br>• možnost čtení při extrémně vysokých rychlostech<br>• velká cena RFID tagu |

**Tabulka 2.8 Rozdělení dle frekvencí**

Frekvenční pásmo UHF je dále rozděleno pro různé regiony dle následující tabulky.

| Označení | Frekvence | Region |
|---|---|---|
| Region1 | 865 - 869 MHz | Evropa, Afrika |
| Region2 | 902 - 928 MHz | USA, Kanada, Mexiko |
| Region3 | 950 - 956 MHz | Japonsko, Asie |

**Tabulka 2.9 Rozdělení pásma UHF**

### Rozdělení dle použití

RFID tagy se dnes vyrábějí v několika velikostech a z různých materiálů. Rozhodujícím faktorem při výběru tagu by měl být způsob použití v systému. Tagy mohou být nalepeny na objekt, a nebo mohou být umístěny přímo v objektu. Zapouzdřené tagy mají větší odolnost a používají se i v případě umístění tagů na kovový materiál. Pouzdro zde plní funkci oddělovače antény od kovového podkladu.

Zvláštním typem tagu je Smart Label, která obsahuje semiaktivní čip a baterie je přímo tištěna do etikety. Jedná se o kombinaci čárového kódu a RFID tagu.

# 3 Průzkum trhu

Před analýzou problému bylo nutné provést průzkum trhu a seznámit se s dostupným zřízením pro zadaný čip UHF EPCglobal Gen2. Průzkum trhu jsem zahájil prostřednictvím internetu. V první fázi bylo nutné shromáždit informace o dostupných zařízeních.

## 3.1 Seznámení s RFID zařízeními

- Čtečky
  - Stacionární čtečky
  - Ruční kabelové čtečky
  - Mobilní terminály
- RFID tiskárny
- RFID aplikátory

**Stacionární čtečky**

Většinou se umísťují na čtecí brány nebo do výrobních hal pro sledování toku produktů.Vzhledem k nutnosti dosažení velké čtecí vzdálenosti, jsou vyráběny pro frekvenci UHF. Čtečka může obsahovat větší počet externích antén, připojených přes externí konektory.

Použití:

- příjem a výdej ze skladu
- transport a logistiku

**Mobilní terminály**

Jsou vhodné pro aplikace, které vyžadují mobilitu. Většinou se jedná o PDA s integrovanou čtečkou RFID popřípadě č. kódů. Výjimkou nejsou ani kombinované čtečky. V případě, že čtečka RFID není integrována, je možné ji připojit přes nějaké rozhraní např. jako PCMCI kartu, infračervený port a další. Mobilní terminály podporují bezdrátovou komunikaci např. pomocí WiFi technologie, což umožňuje online řešení.

Použití:

- inventarizace
- skladové hospodářství
- speciálních aplikace

**RFID aplikátory**

Umožňují automatickou identifikaci produktů bez nutnosti manuální činnosti obsluhy. Většinou jsou dodávány jako součást dopravníkových a třídících systémů. Jsou určeny především pro tisk a kódování Smart Etiket. Výjimkou není ani integrovaná čtečka RFID tagů, která provádí kontrolu zakódování. Variantou může také být pouze tisk etikety na základě již dříve aplikovaného RFID tagu.
Použití:

- Výroba
- Zpracovatelský průmysl

**RFID tiskárny**

Jsou dostupné v několika provedeních od stolních až po průmyslové tiskárny. Většinou využívají termotransfér tisk s možností kódování RFID tagů. Tiskárny je možné připojit přes různá rozhraní. Rozhraní lze měnit modulem, který je v tiskárně zasunut např.: RS232, Paralelní port, Síťový port, Wifi modul a další. Stejně jako u RFID aplikátorů mohou být tiskárny vybaveny snímačem pro kontrolu zakódování.
Použití:

- všude, kde se používají Smart Etikety

## 3.2 Oslovení firem

Další fází průzkumu bylo oslovení firem zabývajících se oborem automatické identifikace. Vzhledem k povaze projektu jsem požadoval cenové nabídky pro mobilní terminál s integrovanou nebo externí čtečkou RFID tagu. Firmy jsem kontaktoval telefonicky nebo e-mailem. Oslovené firmy:

- Eprin spol. s r. o.
- Kodys spol. s .r. o.
- Barco spol. s. r. o.
- Identifikační systémy spol. s r.o.
- JM partners.CZ spol. s r.o.

## 3.3 Výběr zařízení

Frekvenční pásmo UHF se ukázalo jako velký problém. Pro toto frekvenční pásmo se vyrábějí hlavně stacionární čtečky. Mobilní terminály mi byly nabízeny od dvou různých dodavatelů. Jeden od firmy Symbol a druhý od firmy Intermec. Zařízení od firmy Symbol nepodporovalo frekvenci stanovenou pro Evropu (868 MHz), takže výběr byl jednoduchý. Vítězem se stal mobilní terminál Intermec 7*51 Hendled Computer* s externí čtečkou RFID tagů připojenou přes infračervený port *IP4 Intellitag Portable Reader*. Cenová nabídka je uvedena v příloze.


**Obrázek 3.1 Vybrané zařízení s externí čtečkou RFID tagů.**

## 3.4 Závěr průzkumu trhu

Výsledek průzkumu trhu byl pro mě velice překvapivý. Vysoká cena vybraného zařízení a omezený výběr, určily další směr vývoje projektu. Vybrané zařízení se nebude pořizovat. Hromadné čtení RFID tagů bude v další fázi nahrazeno snímačem č. kódů s pamětí a následným dávkovým přesunem do PDA. V případě, že vybrané zařízení půjde zapůjčit z Fakulty informačních technologií, bude aplikace na toto zařízení přenesena, otestována a demonstrováno hromadné čtení RFID tagů.

# 4 Analýza problému

Projekt bude vzhledem k výsledku průzkumu trhu rozdělen do několika fází. Ještě před samotnou realizací bude potřeba vybrat a otestovat dávkový snímač, který bude simulovat hromadné čtení RFID tagu. Další fází projektu bude návrh a implementace aplikace na PDA s vybraným dávkovým snímačem. Bude potřeba navrhnout přenos dat mezi PDA a aplikací na PC. Aplikace bude implementována pro PDA DELL Axim X5.

## 4.1 Stanovení požadavků na aplikace

Před samotnou implementací bude nutné definovat požadavky na obě aplikace. Tento krok ulehčí samotnou implementaci a zamezí chybám vycházejících z nedostatečné analýzy problému.

**Aplikace na PC – požadavky**

- Simulace modulu evidence majetku.
- Zobrazení dat položek.
- Nahrání zvolených dat do CE zařízení.
- Stáhnutí výsledku inventury z CE zařízení.
- Při nahrání a stáhnutí dat by měl být umožněn výběr adresáře.

**Aplikace na PDA – požadavky**

- Nahrání dat do aplikace z vybraného adresáře
- Přerušení inventury – uložení dat do původního adresáře
- Ukončení inventury – Tvorba výsledného souboru bez možnosti nahrání zpět do aplikace.
- Kontrola položek pomocí čárových kódů.
- Podpora hromadného čtení RFID tagů.
- Zobrazení nahraných místností a jejich stavů.
- Zobrazení položek místnosti při vstupu do místnosti.
- Manuální vstup do místnosti.
- Manuální kontrola položky zadáním jejího čísla a nebo vybráním ze seznamu.
- Odlišení normální a manuální kontroly speciálním stavem položky.

# 5 Simulace RFID čtečky

V první fázi projektu bude prozkoumána možnost simulovat hromadné čtení RFID tagů dávkovým scannerem s pamětí. Toto zařízení bude používáno spolu s PDA DELL Axim X5, a proto bylo při výběru přihlédnuto k možnostem připojení. Nejjednodušší a odzkoušenou variantou by bylo připojení přes rozhraní RS232. Bohužel PDA toto rozhraní nepodporuje. Proto bude použita čtečka komunikující přes bluetooth rozhraní. Toto rozhranní sice není cílovým PDA podporováno, ale existuje modul do Compact Flash slotu, který bude pořízen.

## 5.1 Vybraná čtečka

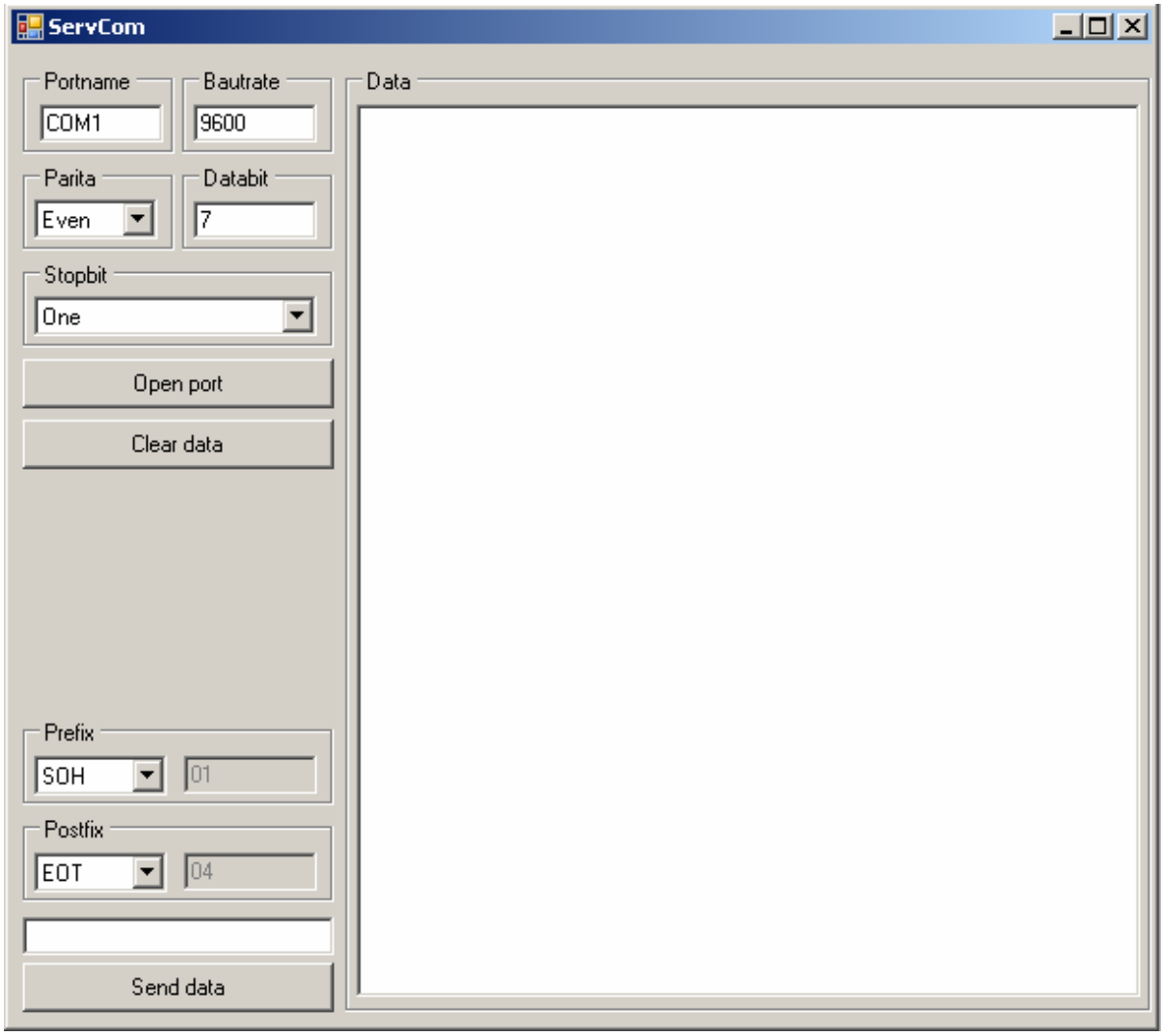
Byla vybrána čtečka komunikující jak přes rozhraní RS232 pomocí kabelu, tak přes rozhraní bluetooth. Jedná se o čtečku Cordless Scanner. Výběr byl zaměřen pouze na čtečky splňující směrnici RoHS, která udává výrobci elektrozařízení povinnost zajistit, aby elektrozařízení uvedené na trh po 30.6.2006 neosahovalo nebezpečné látky, jako např.: olovo, rtuť, kadmium, šestimocný chrom a další. I když výrobci už nemohou vyrábět zařízení, které nesplňuje směrnici RoHS, stále tato zařízení prodávají do vyprázdnění svých zásob.

Dále bylo nutné zakoupit modul bluetooth do Compact Flash slotu pro PDA Dell Axim X5, aby PDA mohlo se čtečkou komunikovat.


**Obrázek 5.1 Vybraná dávková čtečka**

## 5.2 Testování čtečky

Před samotnou tvorbou aplikace bylo nutné otestovat funkčnost čtečky a zjistit, zda bude vyhovovat. Čtečka může pracovat v několika módech, které se nastavují buď čtením předtištěných servisních čárových kódů nebo zasíláním servisních paketů přes rozhraní RS232. Pro zasílání a vyčítání dat ze čtečky je možné použít standardní součást operačního systému Microsoft Windows - Hyperterminál. Čtečka byla testována na vlastní aplikaci ServCom, která byla implementována. Aplikace umožňuje zasílání a vyčítání dat ze sériového portu podobně jako Hyperterminál.




**Obrázek 5.2 Aplikace pro práci ze sériovým portem**

### 5.2.1 Módy a nastavení čtečky

Čtečka může data přenášet do zařízení ve dvou různých formátech. Prvním je XML dokument s předem definovanou strukturou. Druhým způsobem je přímé zasílání dat na sériový port. Pro simulaci hromadného čtení bude vhodnější druhá varianta. Čtečka může pracovat ve dvou různých módech:

- Real-time – Scanner je připojen kabelem nebo bluetooth k hostujícímu zařízení. Scanner zasílá data do zařízení hned po přečtení.
- Batch – Tento mód je nazýván také jako dávkový. Jedná se o to, že scanner není připojen přímo k zařízení. Data jsou po přečtení ukládána do paměti čtečky. Po připojení kabelu a zaslaní příkazu jsou data přenesena do zařízení.

Dalším zajímavým nastavením je zapnutí potvrzovacího protokolu ACK/NAK. Po zaslání dat do zařízení čeká čtečka potvrzení přijetí ACK, v případě zaslání zprávy NAK se vysílání dat opakuje. Toto nastavení je velmi důležité při zapnutí automatického stahování, kdy jsou data přenesena do zařízení hned po připojeni k rozhraní, ne až po zaslaní paketu pro stáhnutí dat.

Vzhledem k tomu, že nastavení čtečky servisními čárovými kódy je omezené právě na existenci těchto čárových kódů, rozhodl jsem se pro nastavení přes servisní pakety. Nastavení probíhá ve 4 různých skupinách. Každá skupina nastavení odpovídá jednomu bytu. Jednotlivé příkazy jsou uzavřeny mezi znak SOH a CR, které uvozují a ukončují příkaz.

Po připojení čtečky přes rozhranní bluetooth je v hostujícím zařízení vytvořen virtuální sériový port. S tímto portem je možné pracovat stejně jako s jakýmkoliv jiným sériovým portem. Této vlastnosti mělo být při simulaci hromadného čtení využito.

### 5.2.2 Čtení kódu a přenos do zařízení

Po prostudování nastavení čtečky mohlo začít testováni. Testování bylo zahájeno se sériovým kabelem a místo PDA Dell X5 bylo použito PC a již zmiňovaný ServCom. Čtečka byla nastavena do Batch modů, byl povolen protokol ACK/NAK. Čtení a zasílání dat probíhalo podle předpokladů:

- Při odpojeném kabelu bylo přečteno několik kódů.
- Byl připojen kabel a zaslána žádost o data do zařízení.
- Data byla v případě potvrzení ACK jednotlivě zasílána.
- Při zaslání NAK bylo poslední vysílání opakováno.

Poté bylo zahájeno testování funkčnosti rozhraní bluetooth. Prvním problémem bylo spárování PC a čtečky. Spárování čtečky se provádí pomocí klíče passkey, který je zaznamenán v manuálu. Spárování čtečky je trochu odlišné od spárování dvou PC, kde je klíč pro spárování volen uživatelem.

Bylo provedeno totožné nastavení již fungující varianty se sériovým kabelem a mohlo začít testováním. Bohužel čtečka se nechovala podle předpokladů. Data byla zasílána hned po přečtení scannerem, i když nebylo navázané spojení. Docházelo ke ztrátě přečtených dat a hlavně se nedařilo nasimulovat hromadné čtení.

Módy čtečky Batch a Real-time nejdou nastavit, čtečka si tyto módy detekuje a nastavuje sama podle připojení. Při volbě rozhranní bloutooth se nepodařilo nastavit takové podmínky, aby čtečka pracovala v Batch modu.

Navíc byla data posílána z postfixem 00002, což by mohlo poukazovat na špatné nastavení virtuálního portu. Port byl ale nastaven stejně jako u varianty ze sériovým kabelem. Při zahájení vysílání dat čtečka přestala reagovat na servisní pakety. Opětovné navázání spojení nastalo, až po vymazání paměti čtečky servisním čárovým kódem.

### 5.2.3 Závěr z testování čtečky

Po prostudování a otestování čtečky lze říct, že tato možnost nepůjde použít pro simulování hromadného čtení RFID tagů v první fázi projektu. Čtečka se chová dle předpokladů, pouze při propojení kabelem, ale PDA Dell Axim X5 rozhraním RS232 nedisponuje.

V první fázi projektu bude implementováno softwarové simulování hromadného čtení RFID tagů. Hromadné čtení bude simulovat vlákno, které poběží samostatně a bude v různých časových intervalech zasílat předem definovaná data. I když tato varianta neumožní správnou funkčnost, bude možné otestovat princip hromadného čtení.

# 6 Implementace - PC

Aplikace na PC by měla sloužit k simulaci modulu evidence majetku. Pro implementaci aplikace bylo zvoleno vývojové prostředí Borland Delphi 7, protože je to implementační jazyk IS Apolla. Hlavním úkolem aplikace je simulace modulu evidence a realizace komunikace se zařízením.

## 6.1 Databáze aplikace

Aplikace využívá lokální databázi, která má podobnou strukturu tabulek jako modul evidence majetku. Tato skutečnost umožní lehkou a rychlou integraci funkčnosti této aplikace do modulu evidence majetku.

K databázi se přistupuje přes ODBC rozhraní. Databáze je typu Microsoft Access. Pro správnou funkčnost je nutné nastavit ODBC zdroj k této databázi s názvem *db_vut*.

## 6.2 Formuláře aplikace

Aplikace je založena na formulářích, které jsou odvozeny od základní třídy *TForm* knihovny VCL (Visual Component Library). Pro účely aplikace byly definovány následující formuláře:

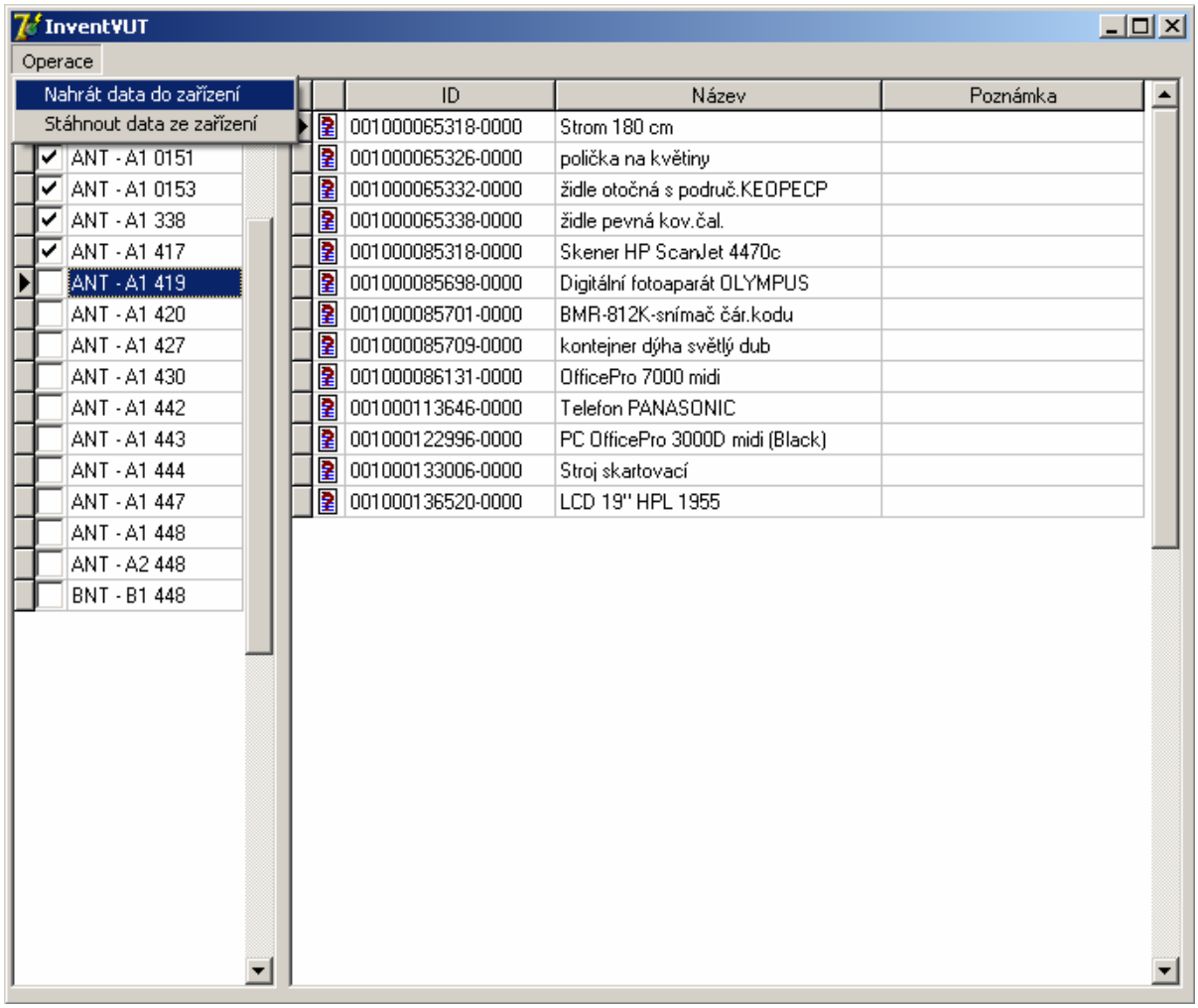
- *TfMain* - Hlavní formulář aplikace.
- *TfSelectDir* – Formulář pro výběr adresáře na CE zařízení.

**TfMain**

Třída *TfMain* je definovaná v unitě *UTfMain.pas*. Formulář je rozdělen na dvě části. V první části je zobrazen seznam místností a možnost jejich označení zaškrtnutím. Nad výběrem místností lze poté provádět operace. Aplikace umožňuje tyto operace:

- Nahrát data – Provede se vytvoření souboru pro přenos dat a jejich nahrání do PDA do vybraného adresáře.
- Stáhnout data – Provede se stažení exportního souboru z PDA z vybraného adresáře a jeho zpracování.

V další části okna aplikace je zobrazen obsah označené místnosti. Detail zobrazuje stav položky místnosti, název položky, její označení a poznámku položky. Stav položky je zobrazen ikonou.




**Obrázek 6.1 Hlavní formulář TfMain**

Formulář *TfMain* provádí operace nad výběrem místností. Místnosti jsou zobrazeny v komponentě *DBGrid*. Tato komponenta neumožňuje zobrazení pole pro zaškrtnutí (*CheckBox*) v prvním sloupci tabulky. Existuje komponenta, která je potomkem komponenty *DBGrid* a tuto funkci umožňuje, ale byla provedena vlastní implementace, z důvodu snížení požadavků na nainstalované komponenty.

Nejprve bylo nutné zjistit, kdy vlastně dochází k překreslení jednotlivých sloupců tabulky. Poté bude zapotřebí místo hodnoty z databáze zobrazit v sloupci *CheckBox*.

K překreslení tabulky dochází na událost *OnDrawColumnCell*. Tato událost má několik parametrů, které umožní identifikovat sloupec, řádek a data pro vykreslení. Kreslící plochu získáme zavoláním metody *Canvas* příslušné tabulky. Dalším problémem bylo vykreslení *CheckBoxu*. Vykreslení lze provést voláním windows API funkci *DrawFrameContro*.

Takto dochází k vykreslení komponenty *CheckBox* v prvním sloupci tabulky, místo hodnot uložených v databázi. Teď je ještě potřeba po kliknutí na příslušný *CheckBox* změnit hodnotu jak, v databázi, tak vizuálně ve formuláři. Tato operace je prováděna na událost *OnScroll* komponenty *TDQuery*, která je s tabulkou místností spojená. Při kliknutí dochází v této komponentě ke změně

vybraného řádku. Nejprve se provede změnu hodnoty pole v komponentě *TDQuery* a poté změna hodnoty v databázi. Automaticky je vyvolána událost *OnDrawColumnCell*, protože se změnil výběr v tabulce a je nutno tabulku překreslit.

Stejným způsobem se provádí vykreslování ikon stavu položek. Místo vykreslení komponenty *CheckBox* se provede vykreslení ikony. Ikony jsou uloženy v komponentě *TImageList* a jejich indexy odpovídají hodnotám stavu položky:

| Stav | Ikona | Popis stavu |
|---|---|---|
| 0 |  | Nezkontrolovaná položka |
| 1 |  | Zkontrolovaná položka |
| 2 |  | Přesunutá položka |
| 3 |  | Nová položka |
| 4 |  | Zkontrolovaná manuálně |

**Tabulka 6.1 Stavy položek**

### TfSelectDir

Třída *TfSelectDir* je definována v unitě *UTfSelectDir*. Formulář slouží k výběru adresáře na cílovém zařízení. Před nahráním nebo stáhnutím dat z CE zařízení je zobrazen tento formulář, kterým uživatel zvolí adresář.

Formulář je zobrazen modálně a je testována vlastnost *ModalResult* po jeho ukončení. V případě *mrOK* byl adresář vybrán. Je-li vrácena hodnota *mrCancel* uživatel formulář zavřel. V třídě jsou definovány další vlastnosti:

- *Root* – Kořenový adresář pro prohledávání na CE zařízení.
- *Dir* – Vybraný adresář v kořenovém adresáři.
- *CanModifi* – Proměnná definující, zda je možné měnit adresářovou strukturu na CE zařízení.

Při konstrukci formuláře je z konfiguračního souboru načtena proměnná *Root*. Po destrukci formuláře je tato proměnná opět uložena do konfiguračního souboru.

Je-li proměnna *CanModifi* nastavena na *True*, je povoleno do kořenového adresáře přidávat nebo odebírat adresáře.


**Obrázek 6.2 Formulář TfSelectDir**

# 6.3 Napojení na lokální databázi

Aplikace *InventVUT* používá komponenty *DeerSoft* pro přístup k databázi přes rozhranní ODBC. V aplikaci jsou použity následující komponenty pro přístup k databázi:

| Komponenta | Popis komponenty |
|---|---|
| *TDMaster* | Komponenta DeerSoft, která představuje zvolený ODBC zdroj v aplikaci. |
| *TDQuery* | Komponenta DeerSoft, která má vazbu na *TDMaster* a umožňuje provádění SQL příkazů a zpracování jejich výsledků. |
| *TDataSet* | Standardní komponenta, která organizuje množinu záznamů z databáze do logické tabulky. |
| *DBGrid* | Standardní komponenta, která zobrazuje *TDataSet*. |

**Tabulka 6.2 Komponenty pro přístup k databázi**

# 6.4 Komunikace s PDA

Pro komunikaci s PDA využívá aplikace komponenty TSM CE. Jedná se o komponenty, které realizují prakticky veškeré možné operace se zařízením, komunikujícím pomocí ActiveSync. Některé zajímavé komponenty TMS CE zobrazuje následující tabulka:

| Název komponenty | Popis komponenty |
|---|---|
| *TCEDeviceInfo* | Komponenta zjišťující informace o zařízení. |
| *TCEFindFile* | Komponenta, která umožňuje vyhledávání souborů na zařízení. |
| *TCEFileOperation* | Provádění operací ze soubory. |

| | |
|---|---|
| *TCEDatabase* | Komponenta zajišťuje přístup k databázi na zařízení. |
| *TCEDatabaseOperation* | Komponenta zajišťující kopírování databáze mezi zařízením a PC. |
| *TCERunApp* | Spuštění aplikace na zařízení. |
| *TCESortcut* | Vytvoření zástupce na zařízení. |

**Tabulka 6.3 Vybrané komponenty CE TSM**

V aplikaci InventVUT jsou využívány pouze komponenty *TCEFileOperation* a *TCEFileFind*. Ostatní komponenty nebylo nutné použít. Možná komponenta *TCERunApp* by našla v aplikaci uplatnění.

Dále byla prostudována možnost práce s komponentu *TDatabase*. Bohužel se nejedná o komponentu pro práci s obecnou databází, ale pouze základní PocketPC databází, která v žádném případě nevyhovuje požadavkům aplikace.

### TCEFileOperation

Tato komponenta umožňuje provádění jedné i několika operací se soubory. K příslušné operaci je zobrazen průběh (progressbar). Vybrané vlastnosti komponenty:

- *FileNames* – Seznam souborů pro vybranou operaci.
- *SourceFilename* – Jméno zdrojového souboru.
- *TargetFilename* – Jméno cílového souboru.
- *TargetDirecotry* – Cílový adresář.
- *TimeOut* – Čas v milisekundách pro neúspěšné ukončení operace.
- *Operations* – Výběr operace.

Vybrané metody komponenty:

- *Execute* – Provedení vybrané operace.
- *CreateCEDirectory* – Vytvoření adresáře na cílovém zařízení.

Hodnoty vlastnosti *Operations*:

- *coCopyDesktopToCE* - Kopírování SourceFilename z PC do zařízení jako TargetFilename.
- *coCopyCEToDesktop* - Kopírování TargetFilename z PC do zařízení jako SourceFilename.
- *coDeleteCE* – Smazání SourceFilename na zařízení.
- *coMoveCE* – Přesun z SourceFilename do TargetFilename na zařízení.
- *coCopyCE* – Kopírování z SourceFilename do TargetFilename na zařízení.
- *coMultiDesktopToCE* - Kopírování všech souborů v FileNames z PC do TargetDirectory na zařízení.
- *coMultiCEToDesktop* - Kopírování všech souborů v FileNames ze zařízení do TargetDirectory na PC.

**TCEFileFind**

Tato komponenta slouží pro vyhledání souboru na cílovém zařízení. Komponenta je využívána pro zobrazení struktury adresářů na zařízení. Vybrané vlastnosti komponenty:

- *Files* – Seznam nalezených souborů.
- *Filter* – Řetězec definující filtr vyhledávání.
- *IncludeDirecotry* – Hodnota definující, zda budou nalezené soubory obsahovat cestu.
- *Attributtes* – Vlastnosti hledání.

Vybrané metody komponenty:

- *Execute* – Provedení hledání.

Hodnoty vlastnosti *Attributtes*:

- *ffReadOnly* – Do výsledku hledání jsou zahrnuty read-only soubory.
- *ffHiden* – Do výsledku hledání jsou zahrnuty skryté soubory.
- *ffDirectory* – Do výsledku hledání jsou zahrnuty adresáře.
- *ffArchive* – Do výsledku hledání jsou zahrnuty soubory.

I když nebyly implementovány vlastní komponenty pro práci s CE zařízením, byl proveden, z důvodu pochopení funkčnosti, rozbor zdrojových kódů těchto komponent. Komponenty využívají *RAPI* rozhraní programu *ActiveSync*, které je definované v knihovně *RAPI.DLL*. V konstruktoru komponent dojde k načtení této knihovny a inicializace objektu *RAPI*. Jednotlivé metody jsou poté realizovány postupným voláním příslušných funkcí a procedur definovaných v knihovně *RAPI.DLL*.

## 6.5 Komunikační soubory

Přenos dat mezi PC a PDA probíhá formou datových souborů. Jedná se o binární soubory, která mají předem definovaný formát. Pole typu řetězec jsou uloženy tak, že je nejdříve uložena délka řetězce na 1 byte a poté samotný řetězec. Maximální délka řetězce je tedy 255 znaků a pro jeho přenos je potřeba znaků 256. Tento přístup umožní uložení prázdného řetězce. Číselná pole jsou kódována do 1 znaku a přenášena jako 1 byte.

Každý komunikační soubor má jako první byte uvedenou verzi, která je aplikací na PC tvořena a na PDA kontrolována. Verzování komunikačních souborů zabrání načtení souboru s odlišnou verzí. Verze se mění se změnou struktury komunikačních souborů. Pro přenos jsou definovány tyto komunikační soubory:

- AREALY.CIS – Číselník areálů v aktuální dávce.
- BUDOVY.CIS - Číselník budov v aktuální dávce.
- MISTNOSTI.CIS - Číselník místnosti v aktuální dávce.
- MAJETEK.DAT – Datový soubor majetku.
- EXPORT.DAT – Soubor změn po ukončení inventury.

**AREALY.CIS**

Jedná se o číselník areálů používaný v aktuální dávce. Soubor je tvořen aplikací na PC a do PDA je kopírován. Soubor má následující strukturu:

| Popis pole | Typ pole |
|---|---|
| Identifikace areálu | Řetězec |
| Popis areálu | Řetězec |

**Tabulka 6.4 Struktura souboru AREALY.CIS**

**BUDOVY.CIS**

Jedná se o číselník budov používaný v aktuální dávce. Soubor je tvořen aplikací na PC a do PDA je kopírován.

| Popis pole | Typ pole |
|---|---|
| Identifikace areálu | řetězec |
| Identifikace budovy | řetězec |
| Popis budovy | řetězec |

**Tabulka 6.5 Struktura souboru BUDOVY.CIS**

**MISTNOSTI.CIS**

Jedná se o číselník místností používaný v aktuální dávce. Soubor je tvořen aplikací na PC a do PDA je kopírován. Pole stav nemá význam při prvotní tvorbě souboru na PC. Při přerušení inventury na PDA jsou data uložena zpět do komunikačních souborů a poté pole stav význam má.

| Popis pole | Typ pole |
|---|---|
| Identifikace areálu | řetězec |
| Identifikace budovy | řetězec |
| Identifikace místnosti | řetězec |
| Popis místnosti | řetězec |
| Stav | číslo |

**Tabulka 6.6 Struktura souboru MISTNOSTI.CIS**

**MAJETEK.DAT**

Jedná se o datový soubor majetku pro aktuální dávku přenosu. Soubor je tvořen aplikací na PC a do PDA je kopírován. Pole starý areál, stará budova, stará místnost, poznámka a stav nemají význam při prvotní tvorbě souboru na PC. Při přerušení inventury na PDA, jsou data uložena zpět do komunikačních souborů a poté pole význam mají.

| Popis pole | Typ pole |
|---|---|
| Identifikace položky | řetězec |
| Identifikace areálu | řetězec |
| Identifikace budovy | řetězec |
| Identifikace starého areálu | řetězec |
| Identifikace staré budovy | řetězec |
| Identifikace staré místnosti | řetězec |
| Popis položky | řetězec |
| Součást | řetězec |
| Středisko | řetězec |
| Jméno | řetězec |
| Čárový kód | řetězec |
| Čipový kód | řetězec |
| Poznámka | řetězec |
| Stav | číslo |

**Tabulka 6.7 Struktura souboru MAJETEK.DAT**

# 7 Implementace – PDA

Aplikace na PDA by měla sloužit k provádění inventury pomocí čárových kódů a hromadného čtení RFID tagů. Vzhledem k výsledku průzkumu trhu musí být implementace rozdělena do dvou fází. Mezi jednotlivými fázemi se bude aplikace přenášet na různá zařízení. Bylo zvoleno vývojové prostředí, které minimalizuje nároky pro přenos aplikací. Pro implementaci bylo vybráno vývojové prostředí Microsoft Visual Studio 2005 a jazyk C#.

Jak již bylo zmiňováno dříve, implementace bude rozdělena do několika fází. V první fázi bude implementována aplikace na PDA Dell Axim X5. RFID hromadné čtení bude simulováno softwarově. V další fázi projektu bude zažádáno o zapůjčení PDA Intermec 751 se čtečkou RFID tagů EPCGlobal UHF Gen2. Fakulta informačních technologií VUT v Brně toto zařízení vlastní. V případě zapůjčení, bude aplikace přenesena na toto zařízení a otestováno hromadně čtení.

Obě zařízení jsou typu PocketPC 2003 Second Edition, a proto je možné psát aplikaci pro platformu .NET Compact Framework 2.0. První fáze implementace probíhala na PDA Dell Axim X5. Scanner a RFID čtečka byly nahrazeny SW emulátorem.

## 7.1 Data

### 7.1.1 Struktura dat

Platforma .NET Compact Framework 2.0. implementuje třídy pro práci se SQL CE Serverem, který bude použit pro uložení dat nahraných v aplikaci. Dále je potřeba mít v PDA uložená data přerušené inventury. Tato data jsou uložena zpět do komunikačních souborů. SQL CE Server je databáze uložená v jednom souboru s koncovkou sdf. Pro práci s databází existují tyto třídy:

| Třída | Popis |
|---|---|
| *DataTable* | Tabulka *DataSetu* – výsledek příkazu SELECT |
| *SQLCeDataReader* | Objekt pro sekvenční průchod *DataTabel* |
| *DataAdapter* | Objekt pro práci s odpojenými databázemi |
| *DataGrid* | Objekt pro zobrazení *DataTabel* |

**Tabulka 7.1 Vybrané třídy pro práci s databází v .NET Compact Framework**

Pro zjednodušení práce s databází byla vytvořena vlastní třída *DBComponnet*.

**DBComponent**

Třída je definována v souboru *CEDatabase.cs*. Jedná se o třídu, která zaobaluje funkčnost SQL CE Serveru a ulehčuje práci s touto databází. Třída definuje tyto vlastnosti:

- *DBName* – Jméno souboru databáze.
- *DBPassword* – Heslo databáze.
- *DBEncrypt* – Kódování databáze.
- *DBDelete* –Smazat před vytvořením databáze.
- *DBConnectionState* – Vrácení stavu připojení.

Třída definuje tyto metody:

- *DBCreate* – Vytvoření databáze. V případě *DBDelete* je před vytvořením původní databáze smazána.
- *DBOpen* – Otevření databáze.
- *DBClose* – Zavření databáze.
- *DBReader* – Metoda, která vrátí objekt *SQLCeDataReader*. Metoda je definovaná pro průchod výsledku SQL příkazu typu *SELECT*.
- *DBQuery* – Metoda, která vrátí *DataTable* a vytvoří objekt *DataAdapteru*. Metoda je definovaná pro práci s komponentou *DataGrid*.
- *DBExecute* – Provedení *INSERT, DELETE* a *UPDATE* SQL příkazu.
- *DBExecuteScalar* – Metoda která vrátí objekt z databáze. Jedná se o skalární *DBReader*. Výsledek obsahuje pouze první řádek a první sloupec *DataTable*.

**Struktura databáze**

Data uložená v databázi mají podobnou strukturu jako komunikační soubory. Z komunikačních souborů je databáze plněna a při přerušení inventury je databáze do komunikačních souborů uložena.

V případě neexistence souboru databáze, je struktura tabulek vytvořena v konstruktoru hlavního formuláře.

Nejjednodušší variantou uložení databáze do souborů, je serializace objektů. Pro serializaci objektu se používá třída *BinaryFormatter*. Tato třída dokáže každý objekt definující rozhraní ISerializable serializovat a následně deserializovat. Třída *BinaryFormatter* není .NET Compact Frameworkem 2.0 podporována, a proto princip serializace objektu nepůjde využít.

Následující tabulky zobrazují strukturu tabulek v databázi. Ve sloupci popis určuje (PK) primární klíč tabulky.

- c_areal – číselník areálu

| Pole | Popis |
|---|---|
| ID | Označení areálu (PK) |
| POPIS | Popis areálu |

**Tabulka 7.2 Struktura tabulky c_areal**

- c_budova – číselník budov

| Pole | Popis |
|---|---|
| AREAL_ID | Označení areálu (PK) |
| BUDOVA_ID | Označení budov (PK) |
| POPIS | Popis budovy |

**Tabulka 7.3 Struktura tabulky c_budova**

- c_mistnost – číselník místností

| Pole | Popis |
|---|---|
| AREAL_ID | Označení areálu (PK) |
| BUDOVA_ID | Označení budov (PK) |
| MISTNOST_ID | Označení místností (PK) |
| POPIS | Popis místnosti |
| STAV | Stav místnosti |

**Tabulka 7.4 Struktura tabulky c_mistnost**

- majetek – tabulka majetku

| Pole | Popis |
|---|---|
| ID | Označení majetku (PK) |
| LOKACE | Umístění majetku |
| LOKACE_OLD | Původní umístění majetku |
| POPIS | Popis majetku |
| STŘEDISKO_ID | Středisko |
| SOUČÁST_ID | Součást |
| OSOBA | Osoba |
| KOD | Čárový kód |
| RFID | RFID tag |
| NOTE | Poznámka |
| STAV | Stav položky |

**Tabulka 7.5 Struktura tabulky majetek**

- confic – Konfigurační tabulka. Ukládá stav aplikace.

| Pole | Popis |
|---|---|
| PATH | Cesta k datovému adresáři. V případě, že je pole NULL, v aplikaci nejsou nahrána žádna data. |

**Tabulka 7.6 Struktura tabulky confic**

### 7.1.2 Načtení a uložení dat

Data do aplikace je možné načíst z komunikačních souborů a v případě přerušení inventury je uložit zpět do souborů. Tento přístup zajistí jednotnost při prvotním načtení a možnost načtení dat po přerušení inventury. Možnost přerušení inventury byla realizována, aby bylo možné nahrát několik datových dávek do zařízení a mezi těmito dávkami libovolně přepínat.

Načtení a uložení dat jak ze souboru, tak z databáze, je realizováno přes pomocné datové třídy, které mají společné rozhraní. Datové třídy i jejich rozhraní jsou definovány v souboru *Typy.cs*. Typy odpovídají struktuře jednotlivých tabulek. Datové třídy mají příponu *Rec*. Jejich společné rozhraní je pojmenováno *IRec*:

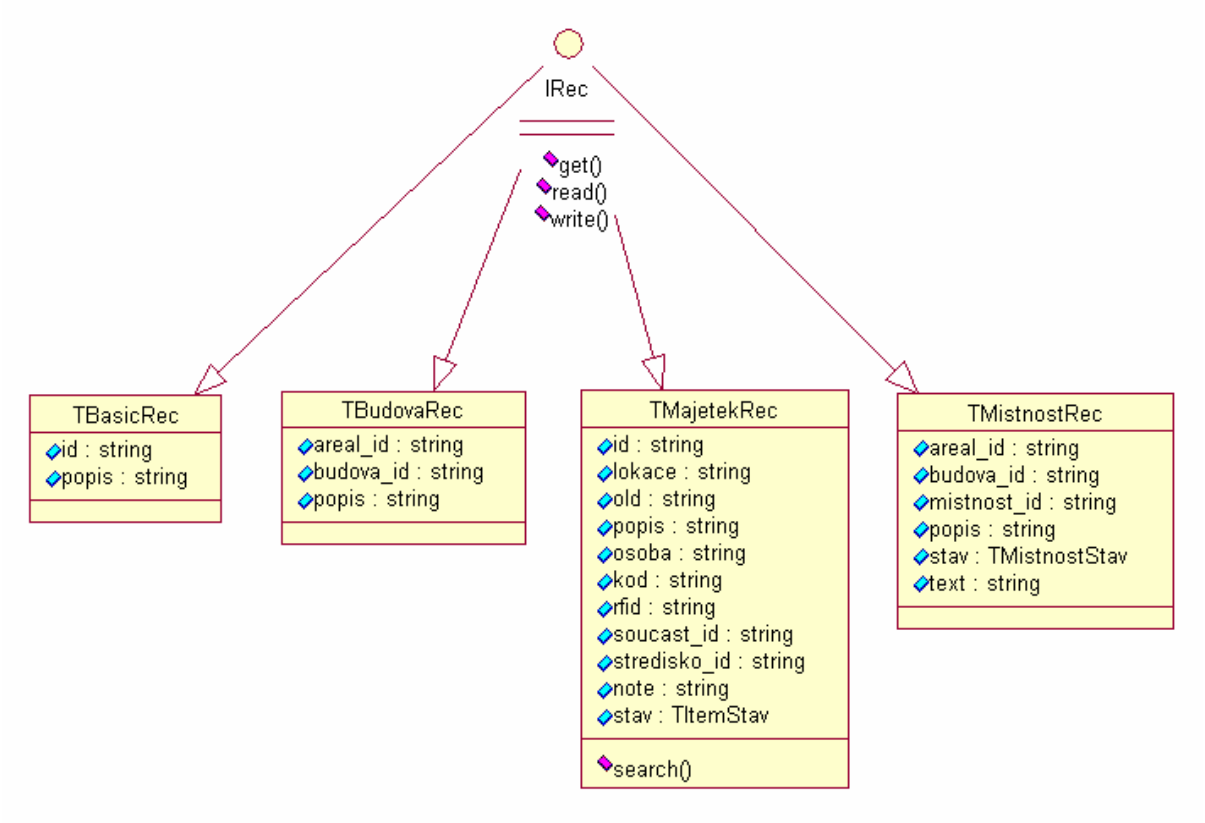
- *get* – Jde o metodu, která má jeden parametr typu *SQLCeDataReader* a vrací hodnotu typu *boolean*. Metoda realizuje naplnění datové třídy ze záznamu z databáze.
- *read* - Jde o metodu, která má jeden vstupní parametr typu *StreamReader*. Třída *StreamReader* je třída pro čtení z datového proudu (streamu). V tomto případě je datovým proudem komunikační soubor. Metoda realizuje naplnění datové třídy z komunikačního souboru a vrací počet přečtených znaků. Vrácení počtu přečtených znaků dává informaci o průběhu čtení. Tato metoda se používá při načítání dat z komunikačních souborů.
- *write* – Jde o metodu, která má jeden vstupní parametr typu *BinaryWrite*. Třída *BinaryWriter* je určena pro zápis do binárního souboru. V tomto případě se jedná o komunikační soubory. Tato metoda je používána při přerušení inventury a nutnosti uložení databáze zpět do komunikačních souboru.

| Třída | Popis |
|---|---|
| *TBasicRec* | Základní datová třída realizující klasický číselník. V aplikaci je používána pro záznamy číselníku areálů. |
| *TBudovaRec* | Pomocná datová třída realizující záznamy číselníku budov. |
| *TMistnostRec* | Pomocná datová třída realizující záznamy číselníku místností. |
| *TMajetekRec* | Pomocná datová třída realizující záznamy tabulky majetek |

**Tabulka 7.7 Přehled pomocných datových tříd**

Datová třída *TMajetekRec* navíc definuje metodu *search*. Tato metoda slouží pro nalezení záznamu v databázi a naplnění datové třídy. Parametrem metody je ID - označení majetku. V případě nenalezení záznamu v databázi je datová třída inicializována a metoda vrátí *false*.

Datová třída *TMistnostRec* navíc definuje pole text. Toto pole obsahuje umístění položky ve tvaru: areal – budova mistnost. Třída definuje dva konstruktory. První má jako parametr pole text a tento parametr transformuje do polí areal, budova, mistnost. Další konstruktor má 3 parametry areal, budovu, mistnost a provádí transformaci do pole text. Uložení umístění ve dvou různých formátech umožní jednoduše párovat a verifikovat pole text kdekoliv v aplikaci.



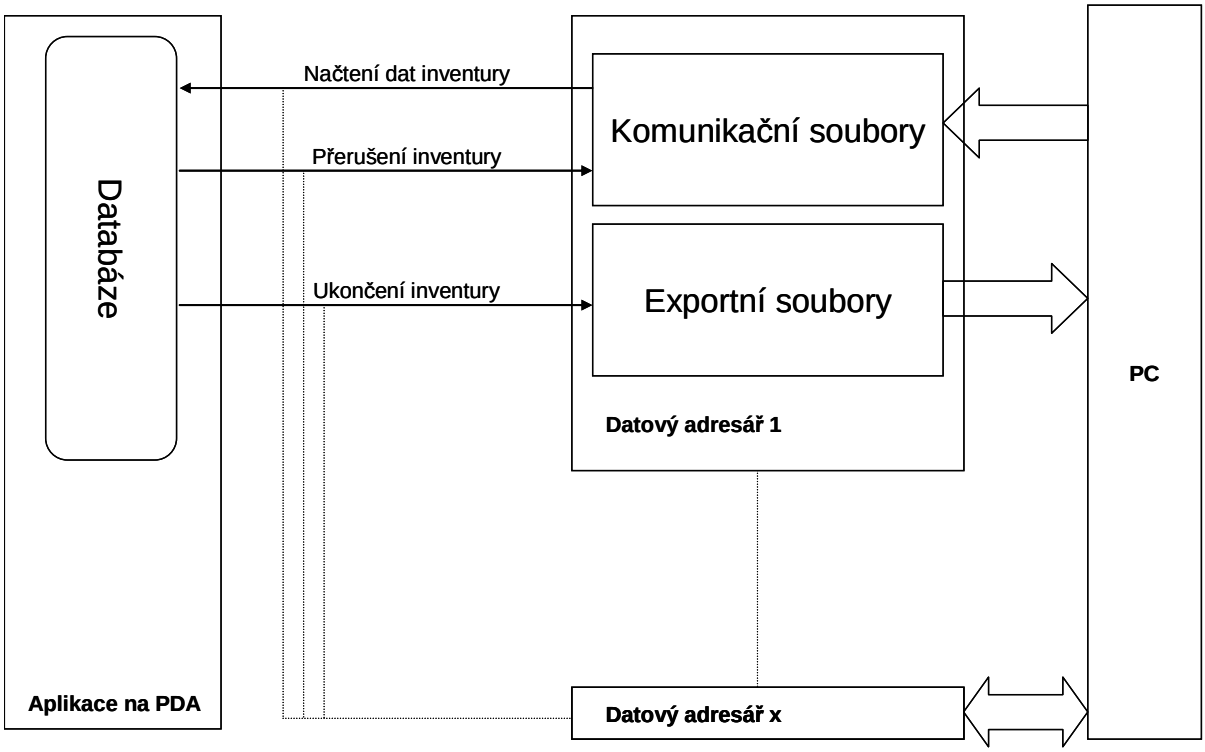

**Obrázek 7.1 Diagram pomocných datových tříd**

### 7.1.3 Exportování dat

Po ukončení inventury je potřeba přenést data vhodně do aplikace na PC. Exportování dat je operace, která neumožní data zpět nahrát. Jedná se o nevratnou operaci a před jejím provedením je zobrazen dotaz pro pokračování. Výsledek inventury je uložen do exportního souboru, do adresáře, z kterého byla data načtena. Jedná se v podstatě o doplněný inventární seznam.

| Pole | Popis |
|---|---|
| ID | Označení majetku. |
| POZNÁMKA | Poznámka generována při inventuře. |
| STAV | Stav položky. |
| AREAL | Nový areál položky. Toto pole je generováno pouze v případě přesunuté nebo nové položky. |
| BUDOVA | Nová budova položky. Toto pole je generováno pouze v případě přesunuté nebo nové položky. |
| MÍSTNOST | Nová místnost položky. Toto pole je zapsáno pouze v případě přesunuté nebo nové položky. |

**Tabulka 7.8 Struktura souboru EXPORT.DAT**




**Obrázek 7.2 Datové toky**

# 7.2 Konfigurační soubor aplikace

Microsoft Visual Studio umožňuje tvorbu standardního konfiguračního souboru. Tento soubor lze upravit ve vlastnostech aplikace. Jedná se o XML dokument, který má pouze jednu úroveň zanoření. K tomuto souboru lze přistoupit přes objekty definované v jmenném prostoru *System.Configuration*.

Nevýhodou tohoto přístupu je právě jedna úroveň zanoření. I když tato aplikace nepotřebuje více úrovní zanoření, je zde implementována vlastní třída nastavení *Settings*, která je definována v souboru *Settings.cs*. Pracuje také s XML dokumentem a umožňuje z dokumentu číst a také ho modifikovat. Výhodou je vlastní formát XML dokumentu, bez omezení počtu úrovní zanoření. Jména parametrů jsou uzly XML dokumentu a jejich hodnota představuje hodnoty parametrů.

Třída definuje dvě metody *Get* a *Set*, které nastavují a čtou parametry z XML dokumentu. V případě neexistence souboru nebo zadané struktury je vyvolána výjimka. Konstruktor obsahuje cestu ke konfiguračnímu souboru aplikace. Bez cesty ke konfiguračnímu souboru nemá tento objekt žádný smysl, a proto je potřeba zakázat standardní konstruktor. Nejjednodušší variantou je označit konstruktor modifikátorem přístupových práv *private*. Tento konstruktor není mimo tuto třídu přístupný, a proto nemůže být také použit pro tvorbu objektu.

## 7.3 Co je to Delegate?

V následujících kapitolách je používán termín delegate. Co to vlastně delegate je? Jazyk C# definuje nový typ a to právě delegate. Jedná se v podstatě o chytrý ukazatel na funkci. Jeho chytrost spočívá v tom, že jsou zabezpečené, variabilní a hlavně typově bezpečné. Další výhodou je, že normální ukazatel na funkci může ukazovat pouze na statickou funkci. Delegate může ukazovat, jak na statickou funkci, tak na metodu vytvořeného objektu, která není statická. Platforma .NET Framework využívá delegáty pro odběr událostí a callback funkce.

Nejobecnějším delegátem pro odběr události je *EventHandler*, který má dva parametry. Prvním je objekt, který událost vyvolal a druhým je třída *EventArgs*. *EventArgs* je základní třídou pro definování argumentů události. Další velkou výhodou modelu delegátů a událostí, je možnost registrace více delegátů pro odběr události. Metody pro obsluhu události jsou volány v pořadí jejich registrace.

## 7.4 Scanner

Každý výrobce zařízení určeného pro čtení čárových kódů, si definuje vlastní knihovny pro nastavení a ovládání příslušné čtečky. Někdy může jít o dll knihovnu (dynamic link library), někdy výrobce definuje přímo komponentu, ale nejobecnějším přístupem je virtuální sériový port. Nastavení probíhá zasíláním konfiguračních paketů a přečtená data jsou opět na tento port zasílaná.

V aplikaci je vytvořeno univerzální rozhraní čtečky *IScanner*. Toto rozhranní je zde definováno kvůli jednotnému použití v celé aplikaci. V případě změny čtečky se nemusí přepisovat všechny výskyty v kódu aplikace, ale pouze se změní objekt, který definuje univerzální rozhranní. Jedná se vlastně o rozhranní, které musí třída scanneru definovat.

**IScanner**

Jak již bylo zmíněno dříve, jedná se o rozhranní, které musí třída zaobalující funkci čtečky definovat. Globální reference *scanner* je právě typu tohoto rozhranní a může ukazovat na každý objekt, který má toto rozhranní definované.

Rozhranní je definované v souboru *IScanner.cs*. V souboru je definovaná další pomocná třída *ScannerEventArgs*. Třída dědí ze standardní třídy *EventArgs* a jedná se o druhý parametr delegáta *EventHandler*. Třída *ScannerEventArgs* je druhým parametrem delegáta *ScannerEventHandler*, který je v tomto souboru také definován. Rozhraní *IScanner* obsahuje tyto prvky:

- *StartRead* – Jedná se o metodu, která by měla implementovat povolení čtečky a zahájení čtení.
- *StopRead* – Jedná se o metodu, která by měla implementovat zakázání čtečky a ukončení čtení.
- *Terminate* – Jedná se o metodu, která by měla implementovat operace nutné před destrukcí objektu. Většinou tato metoda zůstane neimplementována, ale u některých řešeních je čtení

zahájeno voláním nějaké metody, která čeká na objekt dat. V tomto případě musí být před destrukcí objektu toto čekání přerušeno. Pro tyto potřeby je zde tato metoda definována.

- *DataReady* – Jedná se o událost, která by měla být vyvolána v případě přečtení dat čtečkou. Pro tuto událost je ve formulářích aplikace zaregistrován delegát, který data dále zpracuje.

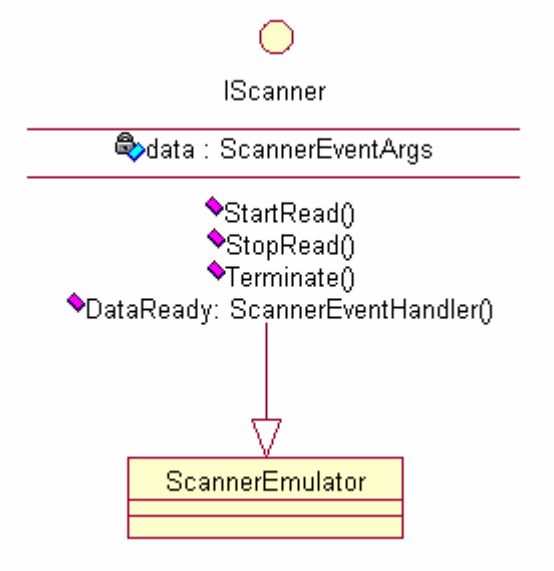
I když událost umožňuje registraci více delegátů, pro účely scanneru je nezbytné, aby v jednom okamžiku byl zaregistrován pouze jeden delegát a to právě delegát aktivního formuláře. Tato skutečnost je v aplikaci řešena takto:

- Každý formulář, který chce odebírat událost od objektu scanneru si v konstruktoru vytvoří delegáta typu *ScannerEventHandler*.
- Při aktivaci formuláře se zaregistruje tento delegát pro odběr události *DataReady*
- Při deaktivaci formuláře se odregistruje tento delegát.
- Událost aktivace formuláře je volána v případě, že formulář dostane vstupní fokus. Vzhledem k tomu, že fokus může mít v jednom okamžiku pouze jeden formulář, je zaregistrován také maximálně jeden delegát pro odběr události *DataReady*.

**ScannerEmulator**

V první fázi projektu je aplikace tvořena na PDA Dell Axim X5. Toto zařízení nedisponuje scannerem, ani žádným zařízením umožňujícím čtení čárových kódů. V aplikaci je definována třída *ScannerEmulator*, která v podstatě nic nedělá, pouze implementuje rozhranní *IScanner*. Třída je definována v souboru *CEScannerEmulator.cs*. Rozhranní je implementováno prázdnými metodami, takže třída žádné operace neprovádí. Tato třída je také důležitá při ladění aplikace na standardním emulátoru CE zařízení, který je součástí Microsoft Visual Studia.

Při přenesení aplikace na zařízení, které bude disponovat čtečkou čárových kódů, stačí pouze definovat zaobalující třídu, která bude implementovat rozhranní *IScanner* a při konstrukci objektu *scanner* tuto třídu použit. Žádné další zásahy v aplikaci nejsou potřeba, což je také hlavním důvodem tvorby univerzálního rozhranní *IScanner*.




**Obrázek 7.3 Diagram tříd scanneru**

## 7.5 Čtečka RFID

Se čtečkou RFID je podobný problém jako u čtečky čárových kódů. Každý výrobce zařízení určeného pro čtení RFID tagů si většinou definuje vlastní knihovny pro nastavení a ovládání čtečky. Stejně jako u čtečky čárových kódů je nejobecnějším řešením zasílání přečtených dat a nastavování čtečky pomocí virtuálního sériového portu.

V aplikaci je vytvořeno univerzální rozhraní čtečky *IReader*. Toto rozhranní je zde definováno, kvůli jednotnému použití v celé aplikaci. V případě změny čtečky se nemusí přepisovat všechny výskyty v kódu aplikace, ale pouze se změní objekt, který definuje univerzální rozhranní. Jedná se vlastně o rozhranní, které musí třída zaobalující funkce čtečky definovat.

**IReader**

Jak již bylo zmíněno dříve, jedná se o rozhranní, které musí třída zaobalující funkce čtečky definovat. Globální reference *reader* je právě typu tohoto rozhranní a může ukazovat na každý objekt, který má toto rozhranní definované.

Rozhranní je definované v souboru *IReader.cs*. V souboru je definovaná další pomocná třída *ReaderEventArgs*. Třída dědí ze standardní třídy *EventArgs* a jedná se o druhý parametr delegáta *EventHandler*. Třída *ReaderEventArgs* je druhým parametrem delegáta *ReaderEventHandler*, který je v tomto souboru také definován. Rozhraní *IReader* obsahuje tyto prvky:

- *StartRead* – Jedná se o metodu, která by měla implementovat povolení čtečky a zahájení čtení.
- *StopRead* – Jedná se o metodu, která by měla implementovat zakázání čtečky a ukončení čtení.
- *Terminate* – Jedná se o metodu, která by měla implementovat operace nutné před destrukcí objektu.

- *DataReady* – Jedná se o událost, která by měla být vyvolána v případě přečtení dat čtečkou. Pro tuto událost je ve formulářích aplikace zaregistrován delegát, který data dále zpracuje.

I když událost umožňuje registraci více delegátů, pro účely čtečky je nezbytné, aby v jednom okamžiku byl zaregistrován pouze jeden delegát. V konstruktoru formulářů, které chtějí využívat objekt *reader* je vytvořen delegát typu *ReaderEventHandler*. Registrování delegátů pro odběr události *DataReady* je řešeno stejně jako u objektu čtečky. Delegát je zaregistrován v případě, že formulář dostane focus. Při ztrátě fokusu je delegát odregistrován.

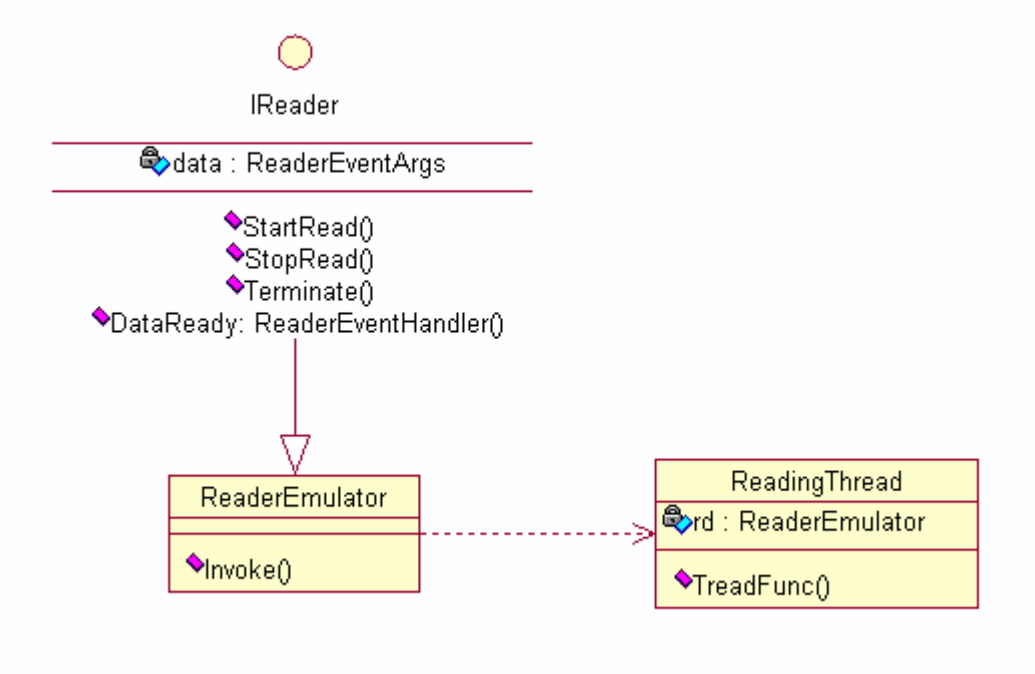
**ReaderEmulator**

V první fázi projektu je aplikace tvořena na PDA Dell Axim X5. Toto zařízení nedisponuje čtečkou RFID tagů. V aplikaci je definována třída *ReaderEmulator*, která simuluje čtení RFID tagů softwarově. Třída je definována v souboru *CEReaderEmulator.cs* a implementuje rozhraní *IReader*, takže může být použita jako objekt *reader*.

V souboru je definována třída *ReadingThread*, která provádí samotnou simulaci čtení RFID tagů. Při konstrukci objektu *ReadingThread* je do této třídy předána reference na objekt třídy *ReaderEmulator*. Třída *ReadingThread* definuje metodu *ThreadFunc*, která je při zahájení čtení spuštěna ve vlastním vlákně. Tato metoda simuluje čtení postupným vyvoláváním události *DataReady*, přes referenci předanou v konstruktoru objektu *ReadingThread*.

Vzhledem k tomu, že je simulace čtení spuštěna v odděleném vlákně, než je vlákno formuláře, bylo potřeba tyto dvě vlákna synchronizovat. Synchronizace je provedena pomocí metody *Invoke* formuláře, která má jako parametr delegáta typu *EventHandler*. Tato metoda vyvolá tohoto delegáta, který je už synchronizován s hlavním vláknem formuláře.

Při přenesení aplikace na zařízení, které bude disponovat čtečkou RFID tagů, stačí pouze definovat zaobalující třídu, která bude implementovat rozhranní *IReader* a při konstrukci objektu *reader* tuto třídu použít. Žádné další zásahy v aplikaci nejsou potřeba, což je také hlavním důvodem tvorby univerzálního rozhranní *IReader*.




**Obrázek 7.4 Diagram tříd čtečky**

## 7.6 Formuláře aplikace

Aplikace je založena na formulářích, které jsou odvozeny od základní třídy *Form*. Pro účely aplikace jsou definovány následující formuláře:

- *fLokace* – Hlavní formulář aplikace
- *fInvent* – Formulář provádění inventury pomocí čárových kódů.
- *fRFID* – Formulář provádění inventury pomocí RFID tagů.
- *fDetailUmisteni* – Formulář detailu umístění
- *fDetail* – Formulář detailu položky.
- *fPrompt* – Formulář zobrazení výzvy uživateli.
- *fSelectDir* – Formulář pro výběr adresáře na CE zařízení.

Vzhledem k tomu, že vytváření objektů formulářů na mobilním zařízení je časově náročné, jsou formuláře vytvářeny při inicializaci aplikace v hlavním formuláři *fLokace* a reference na tyto objekty jsou uloženy jako globální objekty ve statické třídě *glObj*, která je definována v souboru *Global.cs*. Toto řešení umožní přístup ke globálním objektům kdykoliv je to potřeba. Týká se to hlavně formulářů *fPrompt*, *fDetail*, *fDetailUmisteni*, které jsou volány z více míst aplikace.

Další možností je předání referencí na sdílené objekty v konstruktoru formulářů. Tento přístup je možná programátorsky čistější, ale při větším počtu parametrů v konstruktoru, komplikuje čitelnost zdrojového kódu.

Třída *glObj* neobsahuje pouze reference na formuláře aplikace, ale také reference na další globální objekty jako je třeba *IScanner*, *IReader*, *Settings* a *DBComponnet*.

Každý formulář, kromě hlavního, si definuje metodu *Init*, která nahrazuje konstruktor třídy. Tato metoda musí být zavolána před zobrazením formuláře. Metoda zaručuje inicializaci formuláře a naplnění správnými daty. Nutnost volání inicializace je řešena stavovou proměnou formuláře *isInit*, která je po zavolání metody *Init* nastavena na *true* a po zavření formuláře vynulována na hodnotu *false*. Pří pokusu o zobrazení formulář, je tato stavová proměnná testována a v případě nulové hodnoty je vyvolána výjimka a formulář je zavřen. Tento princip velmi ulehčí proces ladění aplikace.

**fLokace**

Třída *fLokace* je definována v souboru *fLokace.cs*. Jedná se o hlavní formulář aplikace, který je po startu aplikace zobrazen. V konstruktoru jsou tvořeny a inicializovány všechny globální objekty. Je provedeno připojení k databázi a v případě neexistence databáze, je databáze i struktura tabulek znovu vytvořena.

Pro zobrazení místností je použita komponenta *ListView*. Tato komponenta umožňuje oproti komponentě *ListBox* zobrazení více sloupců a hlavně jednoduché zobrazení ikony. Jednotlivé řádky jsou tvořeny objekty třídy *ListViewItem*, který má jako jednu z vlastností *ImageIndex*. Ikony jsou uloženy v komponentě *ImageList*. Index ikony odpovídá stavu místnosti *TStavMistnosti*. Jedná se o výčtový typ, který je definován v souboru *Global.cs*.

| Hodnota | Výčtový typ | Ikona | Popis |
|---|---|---|---|
| 0 | *isNone* | | Do místnosti ještě nebylo vstoupeno. |
| 1 | *isCheck* |  | Do místnosti již bylo vstoupeno, ale kontrola ještě není kompletní. |
| 2 | *isComplete* |  | Do místnosti již bylo vstoupeno a místnost neobsahuje žádné nezkontrolované položky. Kontrola je kompletní. |

**Tabulka 7.9 Přehled stavu místností**

Další a možná i lepší variantou, by bylo zobrazení tabulky místností v komponentě *DataGrid*, která je určena pro zobrazení dat z databáze. Bohužel se nepodařilo realizovat zobrazení ikony v prvním sloupci tabulky, tak byla použita vlastnost komponenty *ListView*.

Formulář využívá komponentu *MainMenu*. Jeho položkami jsou operace, které jsou pro tento formulář definovány. Hlavní menu velmi šetří místo na malé obrazovce mobilního zařízení, což je také hlavní důvod proč je zde použito. Použití jiných ovládacích komponent by možná zpřehlednilo rozvržení formuláře, ale stalo by se tak na úkor velikosti hlavní komponenty tohoto formuláře *ListView*.

Průběh načítání, ukládání dat a ukončení inventury zobrazuje komponenta *ProgressBar*, která je za normálních okolností skryta. Pro zobrazení průběhu jsou definovány následující metody formuláře *fLokace*:

- *PBarBegin* – Jedná se o metodu, která má za úkol zobrazení a inicializaci komponenty *ProgressBar*. Inicializací se rozumí nastavení maximální hodnoty, která bývá většinou velikost souborů, který se má zpracovat nebo počet řádků *DataSetu*. Metoda má dva vstupní parametry: minimální a maximální hodnotu.
- *PBarStep* – Jedná se o metodu, která má za úkol posun hodnoty komponenty *ProgressBar*. Metoda má jeden vstupní parametr odpovídající hodnotě průběhu.
- *PBarEnd* – Jedná se o metodu, která má za úkol ukončení zobrazení průběhu. V metodě se provede nejprve nastavení na maximální hodnotu, poté na minimální a komponenta *ProgressBar* se schová. Tato metoda provádí ukončení zobrazení průběhu, a proto by se mělo zaručit, aby byla provedena vždy. Nejjednodušším způsobem je volání této metody v bloku *finally*, který se provede i v případě vyvolání výjimky.

Při zobrazení průběhu bylo nutné zajistit překreslení celého formuláře. Vzhledem k tomu, že program je vykonáván uvnitř metody, smyčka zpráv formuláře se nezpracovává. To je také hlavním důvodem, proč se vykresluje pouze objekt komponenty *ProgressBar*, ale zbytek formuláře ne. Tato vlastnost byla potlačena přidáním zpracování smyčky zpráv do metody *PBarStep* a *PBEnd*. Vynucení zpracování smyčky zpráv zajistí metoda *DoEvent* objektu *Application*. Celý formulář je také možné překreslit zneplatněním jeho obsahu metodou *Invalidate*.

**fDetailUmisteni**

Třída *fDetailUmisteni* je definována v souboru *fDetailUmisteni.cs*. Jedná se o formulář aplikace sloužící pro zobrazení detailu umístění. Objekt je vytvořen v konstruktoru hlavního formuláře *fLokace*.

Metoda *Init* má dva parametry. Prvním je objekt třídy *TUmisteni* a druhým je pole obsahující počet nezkontrolovaných, zkontrolovaných, přesunutých a nových položek v místnosti. Třída *TUmisteni* je definována v souboru *Typy.cs* a realizuje detail místnosti:

- lokace – Lokace místnosti ve formátu: areal – budova mistnot.
- areal – Slovní popis areálu.
- budova – Slovní popis budovy.
- mistnost – Slovní popis místnosti.

Formulář může být ukončen dvěma stavy. *DialogResult.Cancel* v případě zavření formuláře, *DialogResult.Ok* v případě vstupu do místnosti.

**fDetail**

Třída *fDetail* je definována v souboru *fDetail.cs*. Jedná se o formulář aplikace sloužící pro zobrazení detailu položky a využívá se také hlavně pro potvrzení operace s položkou. Objekt je vytvořen v konstruktoru hlavního formuláře *fLokace*.

Třída implementuje metodu *Init* dvakrát. Jedná se tedy o polymorfní metodu. Na základě počtu parametrů je zvolena příslušná implementace. V prvním případě má tři parametry: parametr, nadpis formuláře, objekt třídy *TMajetekRec*. V druhém případě má čtyři parametry: parametr, nadpis, objekt třídy *TMajetekRec*, booleovskou proměnou pro dávkové zpracování.

Jedná se o parametrický formulář. Parametr určuje, zda půjde o detail položky nebo nějakou operaci s položkou. Veškerá aplikační logika je z tohoto formuláře přenesena do nadřazeného formuláře, který formulář vyvolal, kvůli jeho univerzálnosti. Formulář realizuje tyto parametry:

- 0 – Zobrazení detailu nepřesunuté položky.
- 1 – Potvrzení operace nepřesunuté položky.
- 2 – Zobrazení detail přesunuté položky.
- 3 – Potvrzení operace přesunuté položky.
- 4 – Zobrazení detailu nepřesunuté položky s možností manuální kontroly položky.

Formulář obsahuje kromě standardních komponent, také komponenty, které jsou zobrazovány na základě parametru:

- *Button* btOk (potvrdit) – Je zobrazeno pouze v případě potvrzení operace
- *Button* btNote (poznámka) – Je zobrazeno pouze v případě potvrzení operace nebo manuální kontroly.
- *Button* btManual (manuální kontrola) – Je zobrazeno pouze v případě manuální kontroly.
- *Button* btNext (další) –Je zobrazeno pouze v případě dávkového zpracování.
- *RadioButton* rbMove (přesun položky) – Je zobrazeno pouze v případě operace s přesunutou položkou.
- *RadioButton* rbOk (kontrola na původní pozici) – Je zobrazeno pouze v případě operace s přesunutou položkou.

Implementace parametrického formuláře umožnila snížit počet formulářů v aplikaci. Formuláře s podobnou funkcí byly implementovány jako jeden formulář s parametrem.

Čtvrtým parametrem metody *Init* je příznak dávkového zpracování. Dávkové zpracování rozšíří návratové hodnoty formuláře o další stav. Při testování výsledku formuláře je poté možno, buď dávkovou operaci ukončit nebo se přesunout na další položku v dávce. Tato funkce mohla být realizována parametrem formuláře, ale znamenalo by to rozšířit parametry dvojnásobně. Definice druhé metody *Init* umožňuje použití dávkového zpracování se všemi parametry aplikace za cenu minimálního rozšíření.

Formulář může být ukončen třemi stavy. *DialogResult.Cancel* v případě zavření formuláře, *DialogResult.Ok* v případě potvrzení operace a *DialogResul.Ignore* v případě přesunu na další položku při dávkovém zpracování.

**fSelectDir**

Třída *fSelectDir* je definována v souboru *fSelectDir.cs*. Jedná se o formulář aplikace realizující výběr adresáře se zdrojovými daty před nahráním dat do aplikace. Objekt je vytvořen v konstruktoru hlavního formuláře *fLokace*.

Tento formulář nedefinuje metodu *Init* ani stavovou proměnnou *isInit*, protože do formuláře nejsou předávána žádná data ani objekty. Formulář jako jediný využívá globální objekt třídy *Settings*. Z konfiguračního objektu *settings* načítá kořenový adresář datových adresářů. Seznam datových adresářů je získán pomocí statické třídy *Directory*. Tato třída definuje statickou metodu *GetDirecotries*, která má jeden parametr – kořenový adresář. Metoda vrací pole řetězců, které obsahuje adresáře v kořenovém adresáři. Tyto adresáře jsou poté zobrazeny.

V .NET Frameworku existuje další třída pracující z adresářovou strukturou. Jmenuje se *DirecrotyInfo*. Tato třída není statická, takže je potřeba vytvořit objekt této třídy. Třída také poskytuje metodu *GetDirectories*, ale nevrací pouze pole řetězů, ale pole objektu *DirectoryInfo*.

Formulář může být ukončen dvěma stavy. *DialogResult.Cancel* v případě zavření formuláře a *DialogResul.Ok* v případě výběru adresáře.

**fInvent**

Třída *fInvent* je definována v souboru *fInvent.cs*. Jedná se o formulář aplikace realizující inventuru pomocí čárových kódů a zobrazení obsahu místnosti. Objekt je vytvořen v konstruktoru hlavního formuláře *fLokace*. Metoda *Init* má jeden parametr *TMistnostRec*.

V konstruktoru formuláře jsou načteny všechny potřebné ikony a je vytvořen delegát pro odběr události *DataReady* objektu scanneru.

Třída definuje vlastní výsledek po ukončení *Result*. Tato vlastnost je typu *TMistnostRec* a je vyčítána ve formuláři *fLokace*. Podle ní je měněn stav místností.

Hlavní komponentou tohoto formuláře je *ListView* pro zobrazení položek místnosti. Ikony stavu položek jsou uloženy v komponentě *ImageList*. Index ikony odpovídá stavu položky *TItemStav*. Jedná se o výčtový typ, který je definován v souboru *Global.cs*.

| Hodnota | Výčtový typ | Ikona | Popis |
|---|---|---|---|
| 0 | *isNone* |  | Položka ještě nebyla zkontrolována. |
| 1 | *isOK* |  | Položka byla zkontrolována standardní cestou na předpokládaném místě. |
| 2 | *isMove* |  | Položka byla kontrolována a musela být do místnosti přesunuta z jiné místnosti. |
| 3 | *isNew* |  | Položka byla do této místnosti přidána. Nebyla nalezena v této ani v žádné jiné místnosti. |

| | | | |
|---|---|---|---|
| 4 | *isOKManual* |  | Položka byla zkontrolována manuální cestou na předpokládaném místě. |
| 5 | *isCritical* |  | Tento stav může nastat pouze v případě RFID kontroly (hromadné čtení). Tento stav má položka, která nemohla být zkontrolována bez zásahu obsluhy. Tato položka musí být dále zpracována. Jedná se o položky, které by při jednotlivé kontrole měly stav *isMove* nebo *isNew*. |

**Tabulka 7.10 Přehled stavu položek**

### fRFID

Třída *fRFID* je definována v souboru *fRFID.cs*. Jedná se o formulář aplikace realizující inventuru pomocí RFID tagů. Objekt je vytvořen v konstruktoru hlavního formuláře *fLokace*. Metoda *Init* má jeden parametr *TMistnostRec*.

Hlavními komponentami tohoto formuláře jsou dvě komponenty *ListView* pro zobrazení položek místnosti a kritických položek. Kritická položka je položka označená stavem *isCritical*.

V konstruktoru formuláře jsou načteny všechny potřebné ikony a je vytvořen delegát pro odběr události *DataReady* objektu čtečky. Při odchycení události *DataReady* je položka analyzována a zpracována. Zpracování probíhá bez zásahu obsluhy. Položky, které potřebují zásah obsluhy jsou označeny stavem *isCritical* a mohou být zpracovány až po ukončení hromadného čtení.

Tento formulář také využívá formulář *fDetail* v módu dávkového zpracování. Formulář je vyvolán po ukončení a jsou jim zpracovány všechny položky se stavem *isCritical*.

### fPrompt

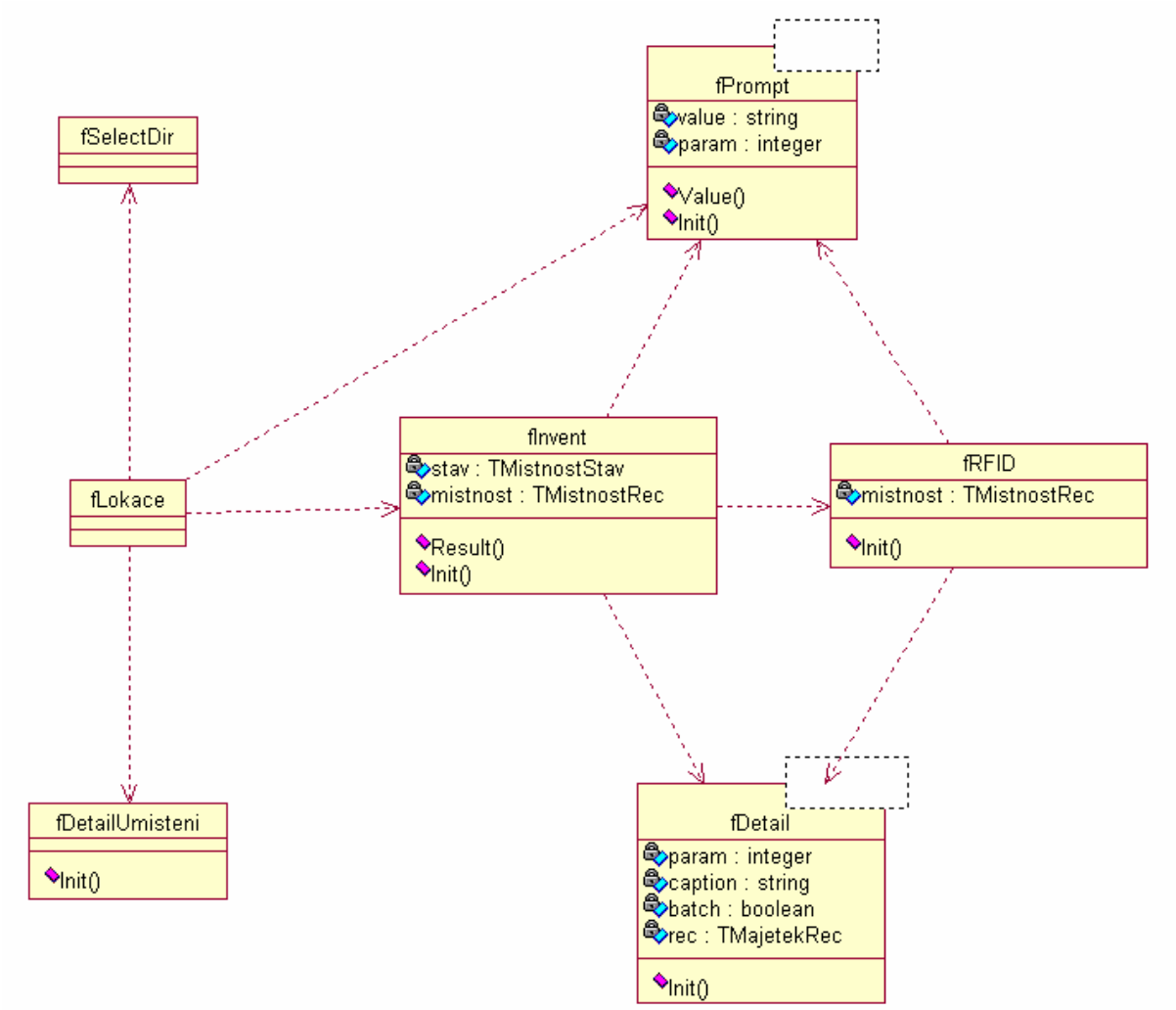
Třída *fPrompt* je definována v souboru *fPrompt.cs*. Jedná se o formulář aplikace sloužící pro zadání dat od uživatele. Objekt je vytvořen v konstruktoru hlavního formuláře *fLokace*.

Metoda *Init* má dva parametry: parametr, výchozí hodnota. Parametr určuje jaká hodnota se bude zadávat. Výchozí hodnota určuje počáteční hodnotu komponenty *TextBox*, která slouží pro zadání hodnoty.

Jedná se o parametrický formulář. Parametr určuje, jaká hodnota se bude zadávat. Podle parametru je zobrazen nadpis formuláře. Nadpis formuláře by se dal také předat jako parametr v metodě *Init* a tím by tento formulář nemusel být parametrický. Formulář je parametrický z důvodu validity zadaných dat. Kontrola dat se může provést hned při zadání, ne až po ukončení formuláře. Každý parametr určuje zadání určité hodnoty a při pokusu o ukončení může být tato hodnota kontrolována. Parametry formuláře:

- 0 – Zadání místnosti.
- 1 – Zadání položky.
- 2 – Zadání poznámky.

Další možností kontroly dat je předání těchto požadavků v parametru metody *Init*. Tím by formulář nebyl parametrický a stal by se ještě více univerzálním.




**Obrázek 7.5 Diagram tříd formulářů a jejich závislostí**

# 7.7 Další třídy aplikace

### Info

Třída *Info* je definována v souboru *Global.cs*. Jedná se o statickou třídu, která poskytuje informace o různých adresářových cestách a jménech souborů. Třída obsahuje statickou proměnnou *pathApp*, která ukládá adresář spuštěné aplikace. Třída pracuje s dvěmi různými cestami, které jsou definované jako vlastnosti třídy:

- *BatchData* – Kořenový adresář datových souborů. Je načítán z konfiguračního objektu *settings*.
- *PathData* – Kořenový adresář dat aplikace. Adresář obsahuje databázi aplikace a další pomocné soubory.

V souboru *Global.cs* je také definován výčtový typ *TTerminalFiles*. Třída *Info* obsahuje privátní pole řetězců *tfName* - jména souborů aplikace. Metoda *FName* třídy *Info* tyto dvě struktury spojuje a dodává jim funkčnost. Parametrem *FName* je právě hodnota výčtového typu *TTerminalFiles* a metoda vrací cestu a jméno k odpovídajícímu souboru.

Při tvorbě cesty a jména souborů jsou využívány public vlastností *BatchData* a *PathData* a ne privátní položky *batchData* a *pathData*, protože vlastnost umožňuje provádění více operací ne jenom vrácení hodnoty proměnné. Vlastnost *PathData* provádí test na existenci tohoto adresáře *pathData*. V případě že zatím neexistuje, tak je vytvořen.

**MyException**

Třída *MyException* je definována v souboru *Global.cs*. Je odvozena od základní třídy *Exception*. Tato třída má za úkol tvorbu vlastních výjimek. Tvorba vlastních výjimek umožní odlišit tyto výjimky od standardních.

**Msg**

Třída *Msg* je definována v souboru *Global.cs*. Tato třída je statická a definuje jednu statickou metodu *Error*. Tato metoda vyvolá standardní chybové hlášení s požadovaným textem a nadpisem. Použití metody zpřehledňuje kód a zavádí jednotný vzhled chybových hlášení.

## 7.8 PDA Intermec 751

Jak již bylo zmíněno dříve, zařízení, které se stalo vítězem průzkumu trhu, nemohlo být kvůli malé konkurenci a vysoké ceně zakoupeno. Další možností bylo zapůjčení tohoto zařízení z Fakulty informačních technologii VUT v Brně. Tato zápůjčka byla realizována na závěr realizace projektu.

Aplikace byla psaná tak, aby přenos na jiné zařízení mohl být proveden s minimálním zásahem do zdrojových kódů. Pro funkčnost čtečky čárových kódů a čtečky RFID tagů, je potřeba pouze implementovat třídy zaobalující jejich funkce. Pro třídy je připraveno rozhranní *IScanner* a *IReader*, které v současné době implementují třídy *ScannerEmulator* a *ReaderEmulator*.

Aplikace byla na zapůjčené zařízení přenesena bez větších potíží. Problém nastal při implementaci tříd scanneru a čtečky. Scanner podle předpokladů komunikuje pomocí virtuálního portu. Čtečka ovšem komunikuje přes nestandardní rozhraní. Komunikace probíhá přes virtuální server, ke kterému je potřeba se připojit pomocí TCP kanálu. V nastavení čtečky je zobrazen i port, na kterém je služba čtečky poskytována. Implementace třídy zaobalující funkci čtečky nemohla být provedena, protože spolu s čtečkou nebyl zapůjčen žádný programátorský manuál. Není znám komunikační protokol čtečky.

**ScannerCOM**

V aplikaci je definována třída *ScannerCOM*, která zaobaluje funkčnost scanneru komunikujícího pomocí sériového portu. Třída je definována v souboru *CEScannerCOM.cs*. V konstruktoru třídy je inicializován a vytvořen objekt sériového portu. Data z toho portu jsou asynchronně vyčítány pomocí události *DataRecieve*. V metodě *StartRead* je zaregistrován delegát pro odběr této události. V metodě *StopRead* je tento delegát odregistrován. Data jsou dále zpracovávána a v případě nalezení znaku # je vyvolána událost *DataReady*. Ve formuláři, který má zaregistrován delegáta pro odběr této události je potřeba provést synchronizace pomocí metody *Invoke* třídy *Form*, protože se jedná o asynchronní operaci. Pro správnou funkčnost třídy *ScannerCOM* je nutné nastavit příponu dat, které posílá scanner. V našem případě je příponou znak #. Tento znak odděluje jednotlivá data, která jsou scannerem posílána.

# 8 Závěr

## 8.1 Zhodnocení dosažených výsledků

V rámci projektu jsem se seznámil s inventarizací a evidencí majetku na VUT. Prostudoval jsem systém SAP, IS Apollo, databázovou platformu Oracle 10g a technologii RFID. Také jsem prostudoval modul evidence majetku, který s inventarizací úzce souvisí.

Po prostudování všech systémů jsem provedl průzkum trhu na dostupná zařízení. Bohužel jsem měl velice omezený výběr, volba a porovnání zařízení od různých výrobců nebyla možná.

Výsledek průzkumu trhu určil směr vývoje projektu. Realizace musela být rozdělena do několika fází a muselo být přijato mnoho náhradních řešení. Další změnu projekt prodělal po otestování prvního náhradního řešení. Dávková čtečka čárových kódů, která měla být použita pro simulování hromadného čtení RFID tagů, nefungovala podle předpokladů a nemohla být použita. Muselo být přijato další náhradní řešení a čtení RFID tagů bylo simulováno softwarově.

Po realizaci výpůjčky zařízení z Fakulty informačních technologií VUT v Brně jsem provedl přenos aplikace. Při přenosu vznikly problémy, spojené s vyčítáním dat z RFID čtečky. Součástí výpůjčky nebyl programátorský manuál, bez kterého jsem nedokázal zjistit komunikační protokol čtečky.

Inventarizace umožní každému uživateli systému Apollo provést inventuru svého majetku a svých podřízených. Zavedením technologie RFID do inventarizace minimalizuje chybu a zvyšuje rychlost provádění inventury.

## 8.2 Další vývoj projektu

Další vývoj projektu bych dokončení třídy zaobalující funkčnost RFID čtečky. Zařízení by se mělo zapůjčit na delší dobu nebo zakoupit. Spolu se zařízením by se měl zapůjčit i programátorský manuál, který ulehčí tvorbu zaobalující třídy RFID čtečky. Po vytvoření této třídy bude systém plně funkční.

Další možnou variantou by bylo rozšíření PDA Dell Axim X5, které vlastní mnoho zaměstnanců VUT. Rozšíření by mělo spočívat v integraci čtečky čárového kódu a čtečky RFID tagů. Výhodou by bylo omezení nákladů spojených s nákupem zařízení, které bylo použitu v závěru projektu.

Úplná integrace do modulu evidence majetku IS Apolla, by také mohla přispět k dokončení systému inventarizace pomocí RFID tagů. Přenos dat ze zařízení do PC je řešen tak, aby se dal při integraci do modulu plně využít.

## 8.3 Vlastní přínos projektu

Projekt byl pro mě přínosem v mnoha ohledech. Při získávání informací jsem musel komunikovat se zaměstnanci CVIS, kteří byli velice ochotní a pomohli mi udělat si představu o výsledném systému. Dalším velkým přínosem byla komunikace s oslovenými firmami a jejich obchodními zástupci.

První fázi průzkumu trhu jsem provedl na internetu a velice jsem ocenil přehledné a jednoduché stránky. V případě tvorby internetových stránek, bych se těmito zásadami určitě řídil.

Dalším velkým přínosem bylo řešení problémů vzniklých během realizace projektu. Projekt prošel několika fázemi vývoje a bylo nutné na tyto fáze vhodně reagovat. Řešení zdánlivě neřešitelných problému, pro mě bude v budoucnu neocenitelnou zkušeností.

Seznámil jsem se s vývojovým prostředím Microsoft Visual Studio 2005 a jazykem C#. Tvorba aplikací byla velice příjemná, rychlá a hlavně na různých platformách velmi podobná. Při řešení dalších projektů budu určitě toto vývojové prostředí preferovat.

Další část projektu jsem řešil ve vývojovém prostředí Borland Delphi 7. Mohl jsem si v praxi vyzkoušet jaké to je, pracovat současně ve dvou různých vývojových prostředích a psát programy ve dvou různých jazycích. I když jsem se seznámil s jazykem C# až teď, řešení některých problémů mě napadalo dříve právě v tomto jazyce a musel jsem hledat alternativy v jazyce Pascal.

Práce s manuály a příručkami byla pro mě také velkým přínosem. Seznámení s formou psaní manuálu mi bude užitečné při práci na budoucích projektech.

# Literatura


[1] Bezděk, Václav: Osobní evidence majetku VUT v Brně [Bakalářská práce]. FIT VUT Brno

[2] Kolektiv autorů: Dokumentační projekt Apollo. CVIS VUT Brno, 2002.

[3] Kolektiv autorů: Telnet, Databázový trh.
URL: http://technet.idnes.cz/tec_technika.asp?r=tec_prakticky&c=A041004_5284398_tec_prakticky (20.12.2006)

[4] Oracle Czech s.r.o. Škrétova 12, 120 00 Praha 2 – Vinohrady: Prezentace produktu.
URL: http://www.oracle.com/global/cz/database/index.html (10.12.2006)

[5] Kolektiv autorů: RFID portál, Co je RFID?. URL: http://www.rfidportal.cz (1.11.2006).

[6] Nagel, Christian: C# 2005: Programujeme profesionálně. Brno, Computer Press 2006.

[7] Sharp, John: Microsoft visual c# 2005: krok za krokem. Brno, Computer Press 2006.

[8] Virius, Miroslav: C# hotová řešení. Brno, Computer Press 2006.

[9] Troelsen, Andrew: C# a .NET 2.0 profesionálně. Brno, Zoner Press 2006.

[10] Sells, Chris: C# a Winforms. Brno, Zoner Press 2005.

[11] Kačmar, Dalibor: Programujeme .NET aplikace ve visual studiu .NET. Praha, Computer Press 2001.

[12] Cantú, Marco: Myslíme v jazyku Delphi 7: knihovna zkušeného programátora. Praha, Grada Publishing 2003.

[13] Lacko, Luboslav: ASP.NET a ADO.NET 2.0: Hotová řešení. Brno, Computer Press 2006.

# Seznam příloh

# Příloha 1. Cenová nabídka

**Nabídka GATC s.r.o. pro:**

**Eprin spol. s r. o.**
**Ing. Markéta Přidalová**
Hybešova 38
Brno 602 00

Naše značka: RFID-EPRIN-06-11-DK
Datum: 24.3.2006

**RFID UHF - GEN2**

| *Item No.* | *Description* | *Ceníková cena 1 ks [Euro]* | *Cena pro Eprin spol. s r. o. 1 ks [Euro]* | *Qty* | *Cena celkem pro: Eprin spol. s r. o. [Euro]* |
|---|---|---|---|---|---|
| **751 Handheld Computer** | | | | | |
| 751B6100E800N803 | Scanner, WIFI 802.11b, 64MB RAM, no BT, | 1920,00 € | 1248,00 € | 1 | **1 248,00 €** |
| 318-013-004 | Spare Battery Pack, 2000 mAh, (7x1 Colour) | 100,00 € | 65,00 € | 1 | **65,00 €** |
| 852-060-002 | charger, 7x1 Colour, single (for one battery pack) | 135,00 € | 87,00 € | 1 | **87,00 €** |
| 851-061-00(7) | Universal AC Adapter, 12V/2.5A with power plug | 68,00 € | 45,00 € | 1 | **45,00 €** |
| 1-974027-025 | Power Cord 220-250 VAC, Europe | 15,00 € | 9,00 € | 1 | **9,00 €** |
| **IP4 Intellitag Portable Reader** | | | | | |
| IP4B002003 | IP4 Intellitag Portable Reader | 1500,00 € | 980,00 € | 1 | **980,00 €** |
| 318-014-00(2) | Spare Battery Pack, IP3 / IP4 scan handle | 95,00 € | 62,00 € | 1 | **62,00 €** |
| 852-062-003 | Battery Charger, 2 Pack for IP3U | 110,00 € | 104,00 € | 1 | **104,00 €** |
| **Shipping Cost** | | | | | |
| | Shipping | 6,00 € | 6,00 € | 1 | **6,00 €** |
| | | | | | **2 606,00 €** |

**Poznámka:**
**Cena platí pro objednání 1 ks zařízení pro DEMO jednotku zákazníka. Zákazník má možnost uplatnit tuto speciální slevu pouze jedenkrát.**
*Ceny jsou uvedeny v EURO bez DPH.*
*Platnost nabídky je 1 měsíc.*
*Dodací lhůta 3-4 týdny*

**Obrázek 1. Cenová nabídka od firmy Eprin spol. s r. o.**

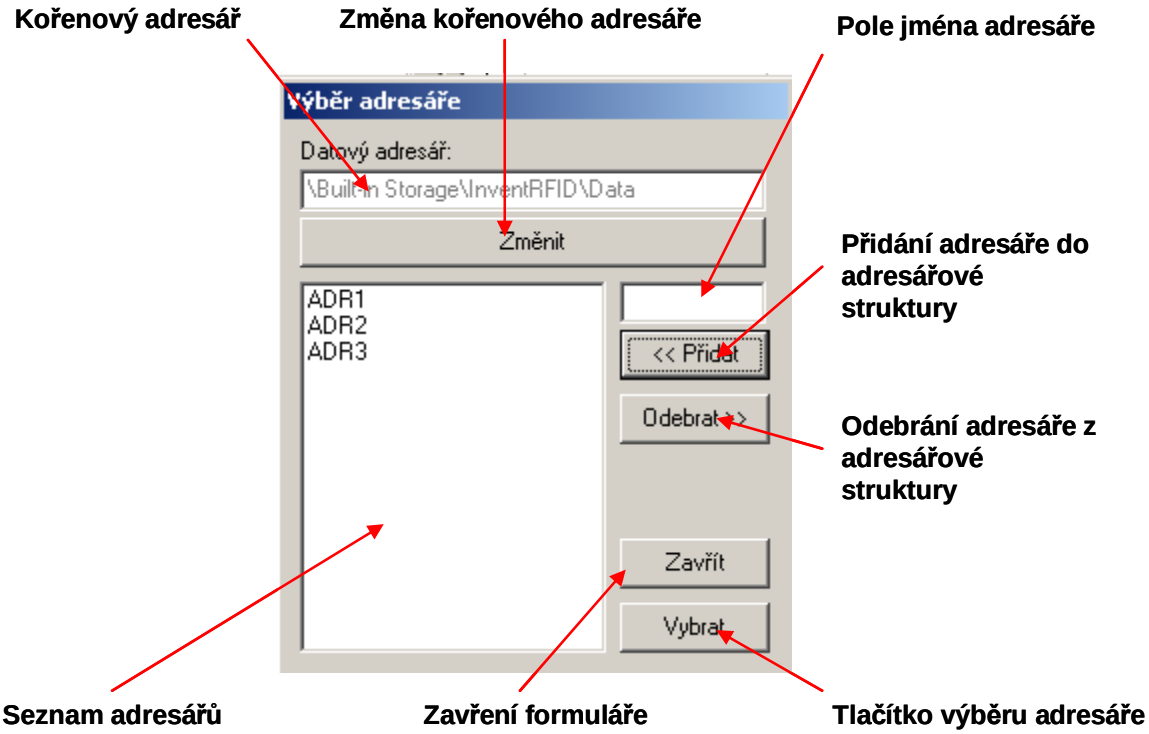
# Příloha 2. Uživatelský manuál - PC

Po spuštění aplikace je obsluze zobrazen hlavní formulář. Formulář je rozdělen na dvě části. V prvním části je zobrazen seznam místností a pomocí zaškrtnutí je možné místnosti vybírat. Nad vybranými operacemi je možné provádět operace, které jsou dostupné v hlavním menu aplikace. Druhá část aplikace zobrazuje detail místnosti, která je v pravém okně vybrána. Nad výběrem místností lze provádět tyto operace:

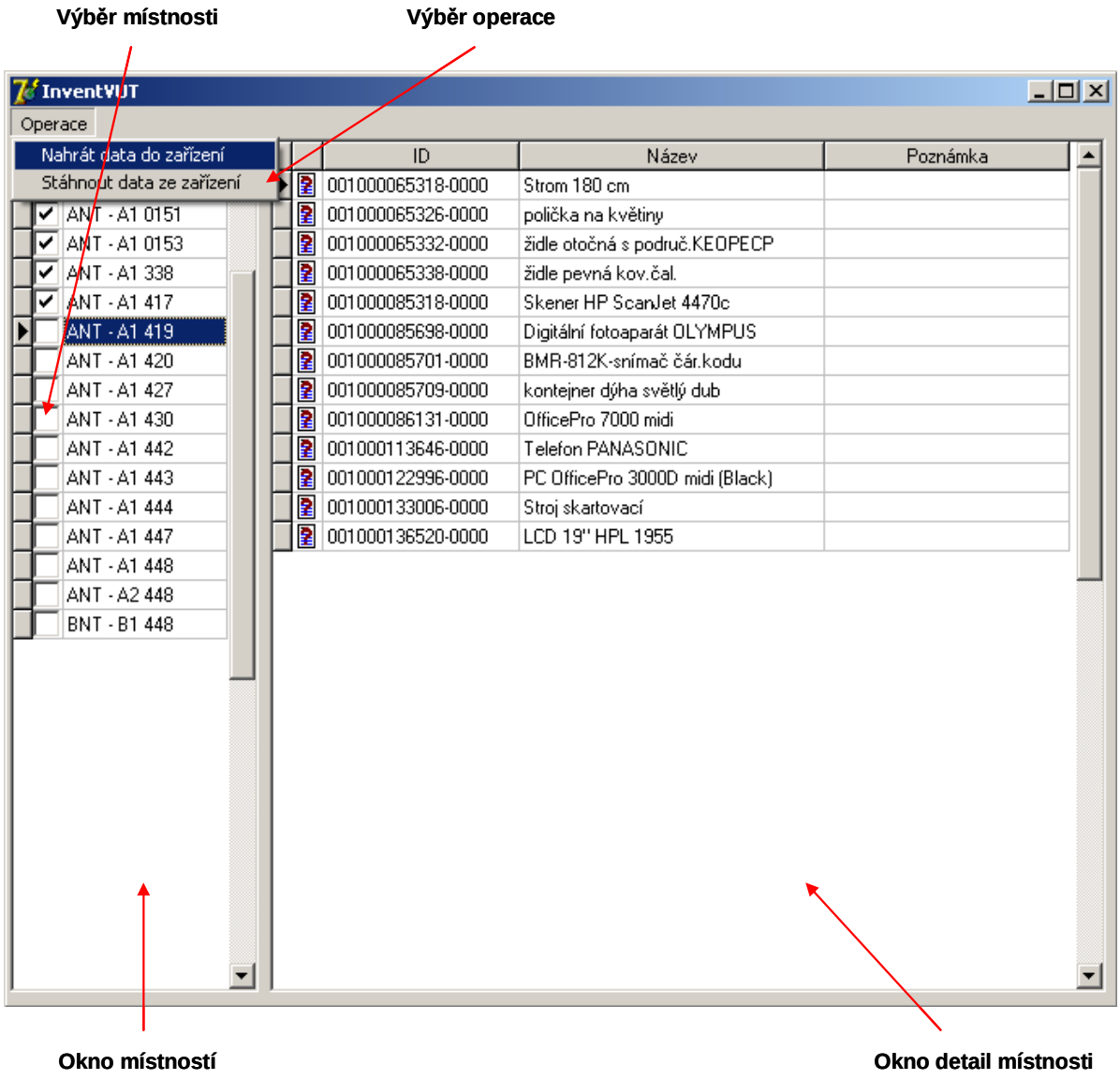
- Nahrát data do zařízení.
- Stáhnout data ze zařízení.

Po zvolení operace je uživateli zobrazen formulář výběru datového adresáře na CE zařízení. V případě operace nahrání dat lze měnit strukturu kořenového adresáře tlačítky *Přidat* a *Odebrat*. V případě operace stáhnutí dat je umožněna pouze operace výběru adresáře, bez možnosti modifikace struktury kořenového adresáře. Kořenový adresář lze měnit tlačítkem *Změnit*. Po opětovném uložení je zobrazena adresářová struktura. Výběr adresáře je nutné ukončit stiskem tlačítka *Vybrat*. Po výběru adresáře se provede zvolená operace.

Není-li zařízení správně připojeno pomocí programu ActiveSync, je obsluha informována o neúspěchu zvolené operace.




**Obrázek 1. Popis okna výběru datového adresáře**

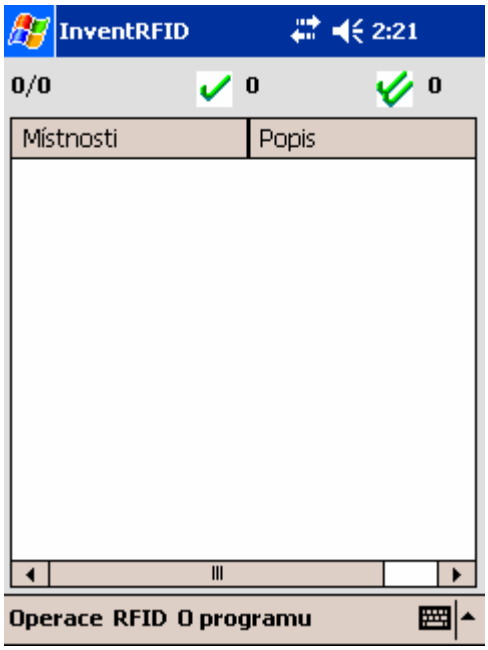



**Obrázek 2. Popis hlavního okna aplikace**

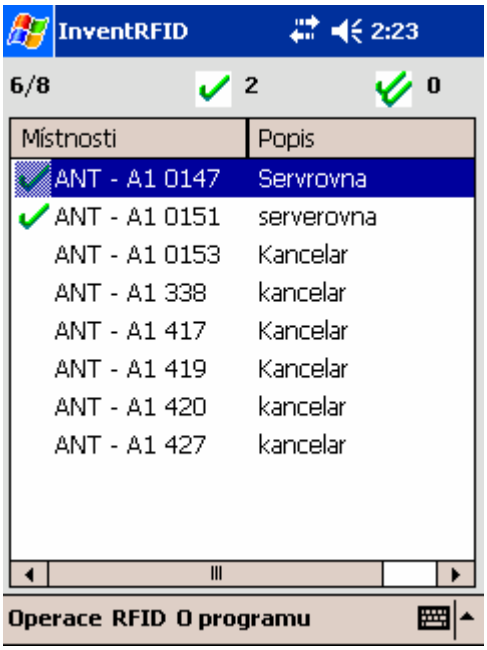
# Příloha 3. Uživatelský manuál – PDA

**Hlavní okno aplikace – seznam místností**

Po spuštění aplikace je obsluze zobrazeno hlavní okno.



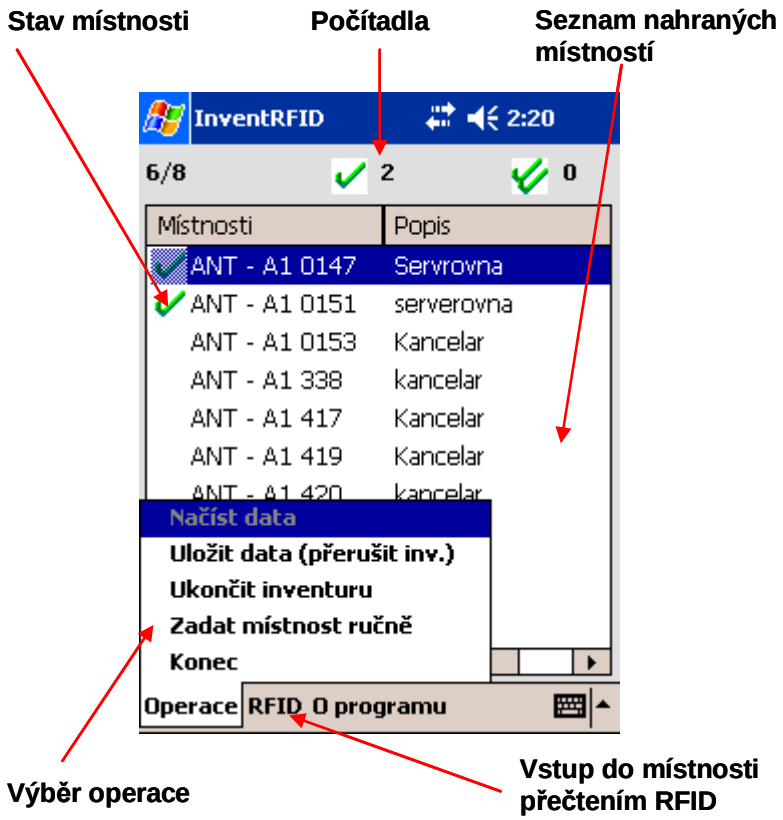

**Obrázek 1. Prázdná aplikace**




**Obrázek 2. Aplikace s nahranými daty**

Hlavní okno slouží pro zobrazení nahraných místností a jejich stav. Formulář zobrazuje stav jednotlivých položek formou ikon. Tyto ikony jsou také použity pro zobrazení počtu jednotlivých místností. Místnost může mít tyto stavy:

- Bez ikony – Do místnosti ještě nebylo vstoupeno.
- ✓ - Do místnosti bylo už vstoupeno, ale stále ještě není kontrola kompletní. Místnost obsahuje nějaké nezkontrolované položky.
- ✓✓ - Kontrola v místnosti je již kompletní. Místnost již neobsahuje žádné nezkontrolované položky.




**Obrázek 3. Popis okna místností**

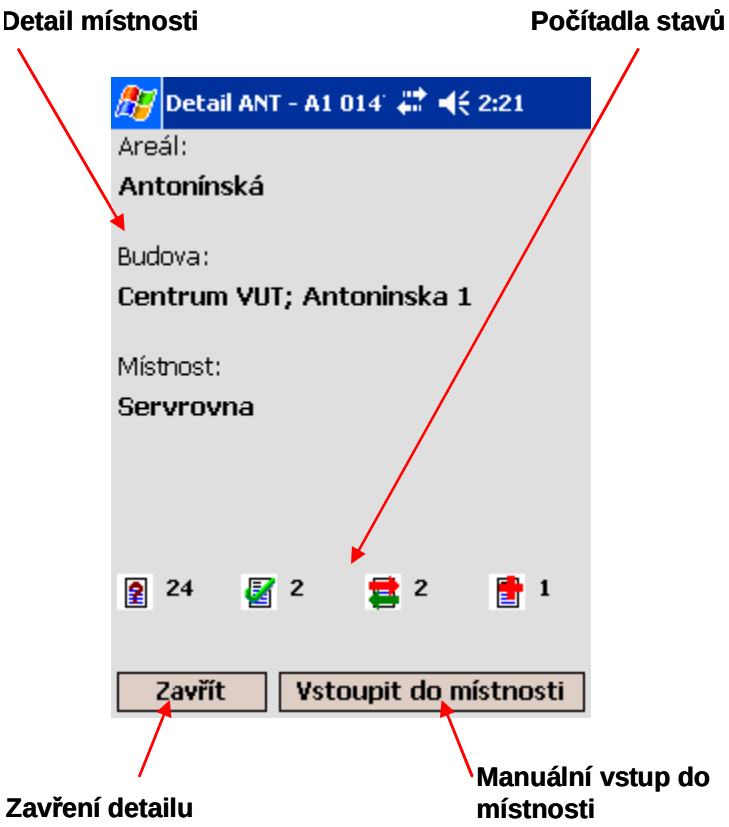
Pomocí hlavního okna je možné manipulovat s daty aplikace. Operace jsou zobrazeny v hlavním menu aplikace. Pro manipulaci s daty jsou definované tyto operace:

- Nahrát data – Nejsou-li v aplikaci nahrána data, je možné data do aplikace nahrát. Data se nahrávají z datového adresáře, který musí být zvolen.
- Přerušit inventuru – Data jsou uložena zpět do datového adresáře na CE zařízení. Do aplikace je možné nahrát jinou datovou dávku.
- Ukončit inventuru – Operace provede nezvratné ukončení inventury. Operaci je nutné potvrdit. Data jsou transformována do exportního souboru do původního datového adresáře. Po ukončení inventury jsou data připravena na přenos do PC.

Před nahráním dat je uživatel vyzván k zadání datového adresáře. Pro zadání slouží další okno aplikace. Vývěr je nutné potvrdit tlačítkem *Vybrat*. Po vybrání datového adresáře jsou do aplikace data nahrána. Další operace hlavního okna:

- Zadat místnost ručně – Obsluze je umožněno zadat kód místnosti ručně.
- RFID – Obsluze je umožněno přečíst kód místnosti RFID čtečkou.
- Ukončení aplikace – Operace provede ukončení aplikace. Po opětovném spuštění je aplikace ve stejném stavu, ve kterém byla před ukončením.

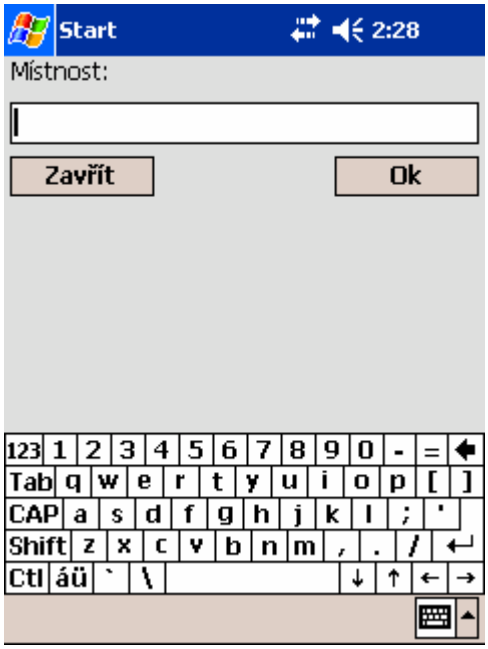
Po dvojitém kliknutí na místnost je obsluze zobrazen detail místnosti. Detail zobrazuje informace o místnosti a jejích položkách.




**Obrázek 4. Popis okna detailu místnosti.**

Počty jednotlivých položek jsou zobrazeny ikonou. Do místnosti je možné vstoupit několika způsoby:

- Ručním zadáním.
- Přečtením čárového kódu.
- Přečtením RFID tagů.
- Kliknutím na tlačítko *Vstoupit do místnosti* v detailu místnosti.

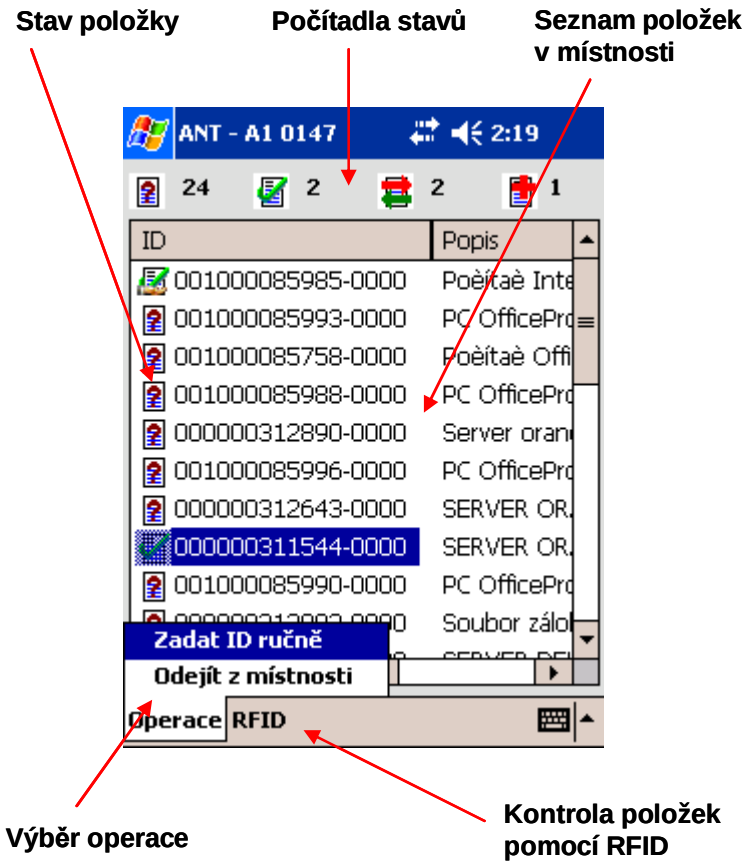



**Obrázek 5. Manuální zadání místnosti**

### Okno kontroly položek – seznam položek místnosti

Po vstupu do místnosti je vyvoláno okno kontroly. Okno zobrazuje položky místnosti a jejich stav formou ikony:

- - Nezkontrolovaná položka.
- - Zkontrolovaná položka. Položka byla kontrolována standardní cestou na předpokládaném místě.
- - Manuální kontrola. Položka byla zkontrolována manuálně obsluhou. Kód položky nebyl čitelný a obsluha musela provést kontrolu výběrem ze seznamu.
- - Přesunutá položka. Položka byla do aktuální místnosti přesunuta z jiné místnosti.
- - Položka byla do místnosti přidána. V žádné místnosti nebyla položka nalezena.

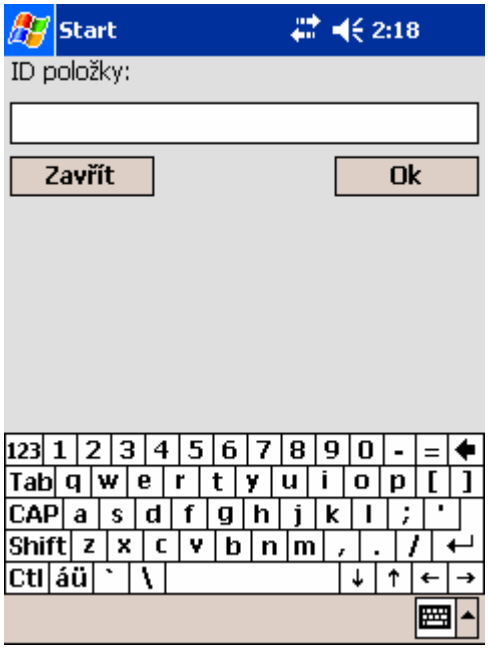



**Obrázek 6. Popis okna kontroly položek**

Položky lze kontrolovat následujícími způsoby:

- Přečtením čárového kódu položky.
- Výběrem ze seznamu položek – manuální kontrola.
- Zadaní čísla položky z klávesnice.

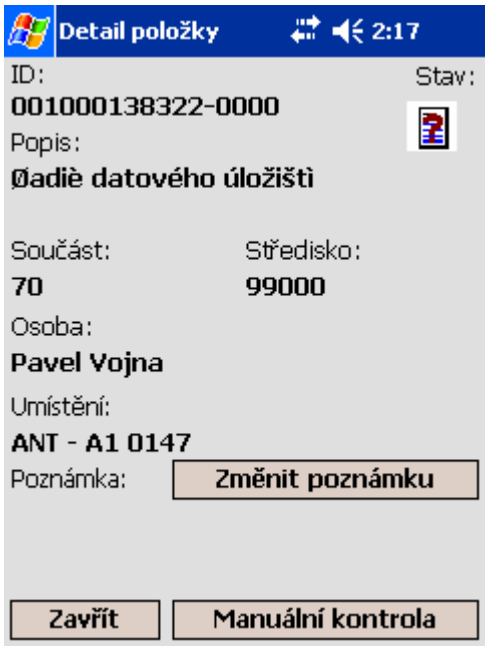
Po zadání položky může dojít k zobrazení detailu položky v případě, že už položka byla kontrolována. Pro nezkontrolované položky dochází k jejich kontrole. Obsluze je zobrazeno okno potvrzení operace. Je-li položka vybrána ze seznamu, je možné provést manuální kontrolu stiskem tlačítka *manuální kontrola*. Formulář kontroly pomocí RFID tagů je vyvolán výběrem *RFID* v hlavním menu kontroly položek.




**Obrázek 7. Manuální zadání ID položky**

### Okna detailu položek

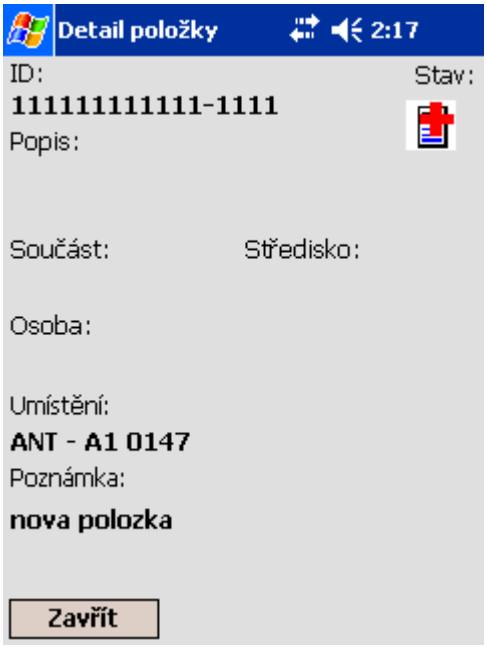
V případě zadání již zkontrolované položky je zobrazen detail této položky. Je-li vybrána položka ze seznamu je také zobrazen její detail. Při výběru nezkontrolované položky je obsluze umožněna manuální kontrola stiskem tlačítka *Manuální kontrola.*




**Obrázek 8. Detail nezkontrolované položky**


**Obrázek 9. Detail zkontrolované položky**


**Obrázek 10.Detail přesunuté položky**

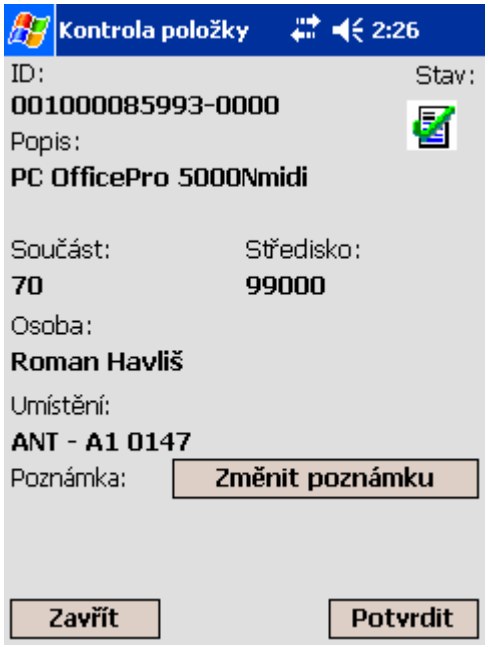



**Obrázek 11. Detail nové položky**

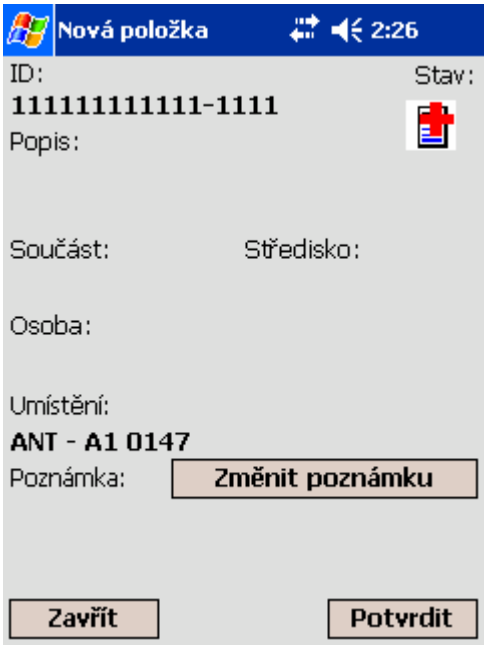

**Obrázek 12. Detail manuálně zkontrolované položky**

### Okna potvrzení operací

V případě zadání nezkontrolované položky je zobrazeno okno potvrzení operace. V případě potvrzení operace je změně stav položky. Operaci lze zrušit stiskem tlačítka *Zavřít*.

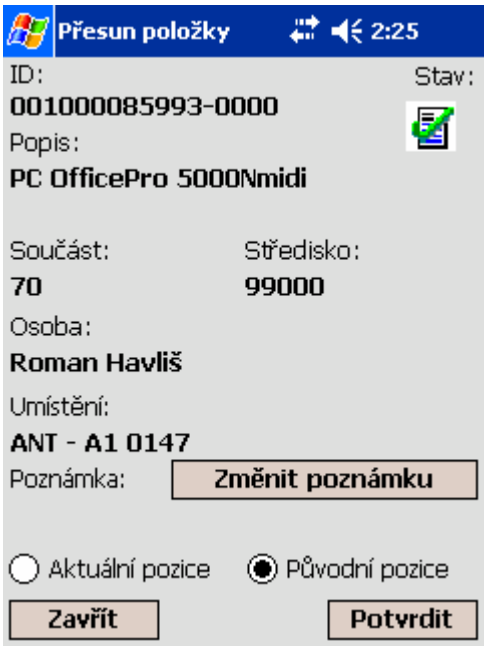



**Obrázek 13. Potvrzení kontroly položky**



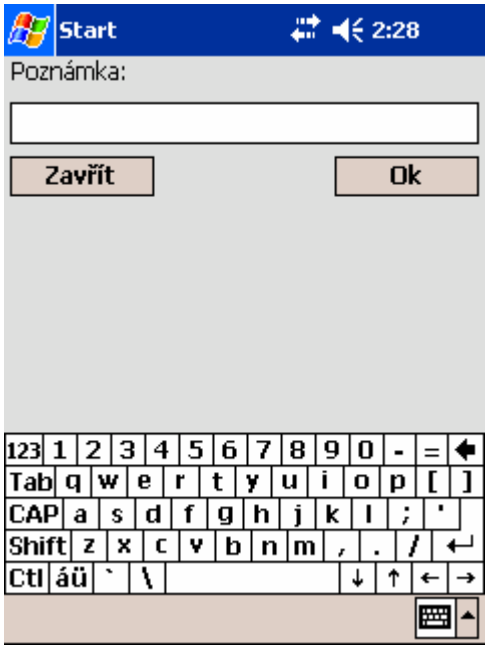

**Obrázek 14. Potvrzení přidání položky**


**Obrázek 15. Potvrzení přesunu položky**




**Obrázek 16. Potvrzení kontroly v jiné místnosti**

Při Potvrzení operace nebo manuální kontrole nezkontrolované položky lze zadat poznámku ke kontrole. Tato poznámka bude zobrazena na výsledném inventárním seznamu. Poznámku ke kontrole lze zadat stiskem tlačítka *Změnit poznámku*. Obsluze se zobrazí okno pro zadání poznámky.



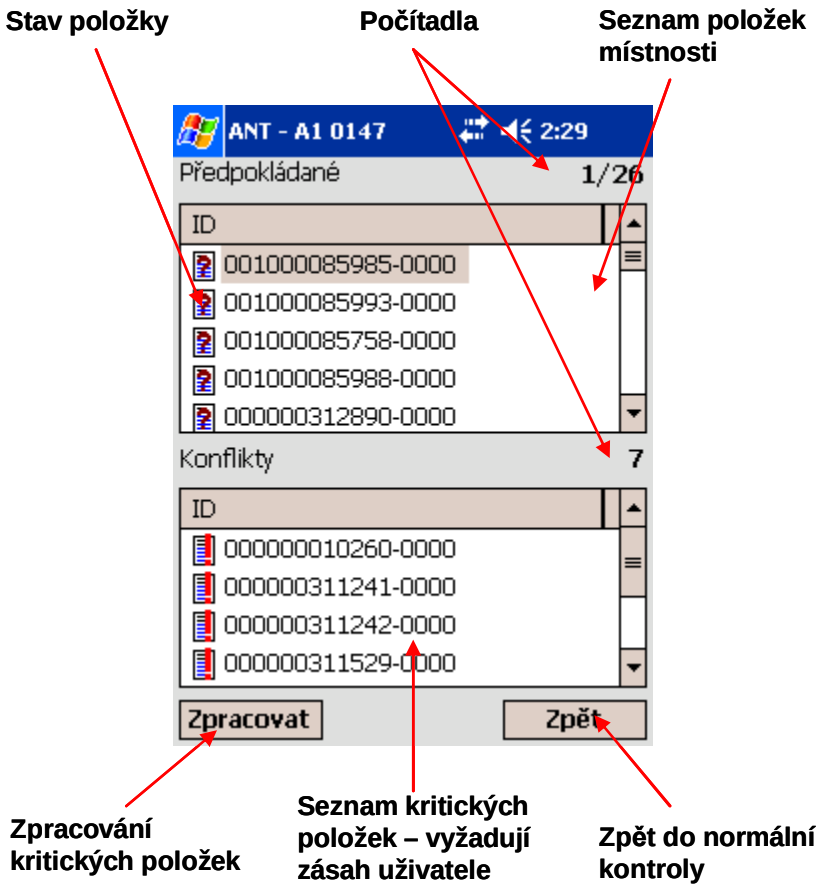

**Obrázek 17. Zadání poznámky**

### RFID

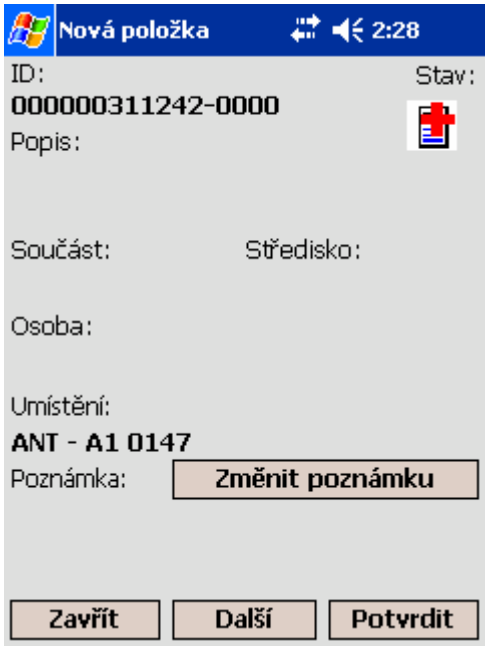
Aplikace umožňuje kontrolu položek hromadným čtením RFID tagů. Okno RFID kontroly lze vyvolat výběrem *RFID* z hlavní nabídky formuláře kontroly položek. Po zobrazení okna RFID kontroly je možné zahájit hromadné čtení. Obsluze jsou zobrazeny dva seznamy položek.

- Předpokládané položky – Jedná se o položky místnosti.
- Kritické položky – Jedná se o položky, které pro dokončení vyžadují manuální zásah obsluhy. Jedná se o položky z jiných místností nebo nové položky.

Tlačítkem *Zpracovat* je možné zahájit zpracováni kritických položek. Zpracování kritických položek probíhá stejným způsobem jako u klasické kontroly. Při potvrzení operace je navíc zobrazeno tlačítko *Další*, kterým je možné přeskočit položku a přesunout se na další.

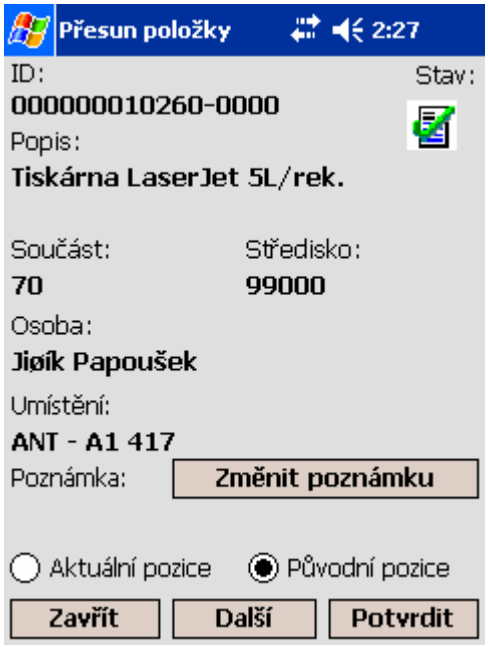



**Obrázek 17. Popis okna RFID kontroly**




**Obrázek 18. RFID - Potvrzení nové položky**


**Obrázek 19. RFID - Potvrzení přesunuté položky**




**Obrázek 20. RFID – Potvrzení kontroly na v jiné místnosti**

### Struktura konfiguračního souboru Settings.XML

<?xml version="1.0"?>

<confic>

<BatchDir>Kořenový datový adresář </BatchDir>

</confic>