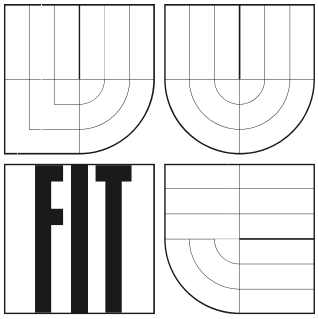
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Agenda majetku VUT v Brně

Ročníkový projekt

2006 Jiří Šindelka

Agenda majetku VUT v Brně

© Jiří Šindelka, 2006.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Prohlášení

Prohlašuji, že jsem tento ročníkový projekt vypracoval samostatně pod vedením Ing. Jaromíra Marušince, Ph.D., MBA.
Další informace mi poskytli Ing. Rudolf Musil, Kaloušková Radka a Václav Bezděk.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Abstrakt

Tento ročníkový projekt se zabývá prostudováním evidence majetku a vytvořením jednoduché inventarizace majetku. Cílem je prostudovat inventarizaci majetku pomocí programu ProBase, prostudovat IS Apollo, evidenci majetku v systému SAP, vytvořit inventarizaci majetku pomocí PDA, modul pro přenos a zpracování dat.

Aplikaci do PDA jsem realizoval ve vývojovém prostředí Microsoft eMbedded Visual Studio C++ 4.0. Modul jsem implementoval ve vývojovém prostředí Borland Dephi 7.

Klíčová slova

ProBase, Apollo, SAP, PDA, Microsoft eMbedded Visual Studio C++, Borland Dephi

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. Musilovy za poskytnutí všech potřebných informaci a rad při řešení projektu.

Abstract

This project deal with study of property evidence and implementation of simply property inventorying. Purpose of this project is study the property inventorying by the program ProBase, peruse the IS Apollo, study the property evidence in SAP, make simply property inventorying through the use of PDA and create module for data transmission and processing.

PDA application is realized in develop environment Microsoft embedded Visual Studio C++ 4.0. Module is realized in develop environment Borland Dephi 7.

Keywords

ProBase, Apollo, SAP, PDA, Microsoft eMbedded Visual Studio C++, Borland Dephi.

Obsah

Seznam obrázků

1 Úvod

Správa majetku na VUT a jeho inventarizace není jednoduchou problematikou. Dobrá evidence majetku a jeho pravidelné kontrolování umožní šetřit prostředky a zpřehledňují operace s majetkem. Dále dobrá evidence 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. Pro identifikaci majetku je nejlepší použít č. kód nebo čip.

Majetek by měl být také evidován na osobu. Evidence na osobu nám umožní jednoduše určit odpovědnou osobu a v případě ztráty nebo poškození není těžké 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, ale 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.

Inventarizace majetku je v současné době prováděna pomocí programu ProBase. Tento systém zavádí automatickou identifikaci majetku pomocí č. kódů, zpřehledňuje a urychluje inventuru. Problémem při zavedení podobného systému je prvotní inventura a označení majetku č. kódem. Tato operace ještě nebyla na všech částech VUT dokončena.

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, vytvořit 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.

Výsledkem tohoto projektu by mělo být prostudovat stávající systém ProBase a navrhnout další kroky potřebné k integraci inventarizace majetku do modulu IS Apolla pro evidenci majetku. Tento modul je řešen jako bakalářská práce studentem Václavem Bezděkem.

2 Seznámení ze systémy

2.1 Úvodem

Vzhledem k tomu, že inventarizace majetku je úzce spjata s evidenci majetku bylo nutné seznámit se ze stávajícími systémy, které evidence majetku využívá.

2.1.1 SAP

Evidence majetku je vedena v systému SAP a proto je nutné tento systém analyzovat. Struktura uložených dat není moc důležitá, protože systém inventarizace bude využívat modul evidence majetku. Vzhledem k tomu, že bylo nutné prostudovat stávající systém inventarizace majetku uvádím zde přehled důležitých tabulek v SAP:

- ANLA – základní data o majetku
- ANLZ – Časová data majetku
- ANLC – Vybrané hodnoty majetku

2.1.2 ProBase

ProBase umožňuje inventarizaci majetku a také je v současné době pro tyto účely využíván. Tento program má dvojí způsob použití – autonomní systém nebo jako nástavba účetního programu. Na VUT je využíván jako nástavba ekonomického systému SAP. Je možné jej provozovat na většině běžných PC s operačním systémem Windows 98, Windows 2000, Windows XP. [5]

Systém ProBase se skládá z několika modulů. Moduly využívané na VUT jsou:

- ProBaseMajetek – hlavní aplikace pro inventarizaci
- ProBaseCom – zajišťuje komunikaci s externími zařízeními

Program ProBaseMajetek umožňuje:

- Správu číselníku
  - Majetku
  - Umístění
- Správu obrazové dokumentace k majetku a lokalitám
- Vystavení dokladů
  - Zařazení majetku
  - Vyřazení majetku
  - Převod majetku
- Vedení zápůjček
- Vedení oprav

Systém ProBase je vybaven terminálem připojeným k pracovní stanici pře rozhraní RS232. Terminál je vybaven snímačem č. kódu a jednoduchou aplikaci pro inventarizaci.

Systém ProBase umožňuje provést dva typy inventury – standardní inventura, prvotní inventura. Rozdíl mezi standardní inventurou a prvotní je v tom, že při prvotní inventuře není k dispozici inventární seznam. Tento druh inventury se většinou provádí poprvé, když ještě nejsou nelepeny č. kódy a systém nezná pozice majetku.

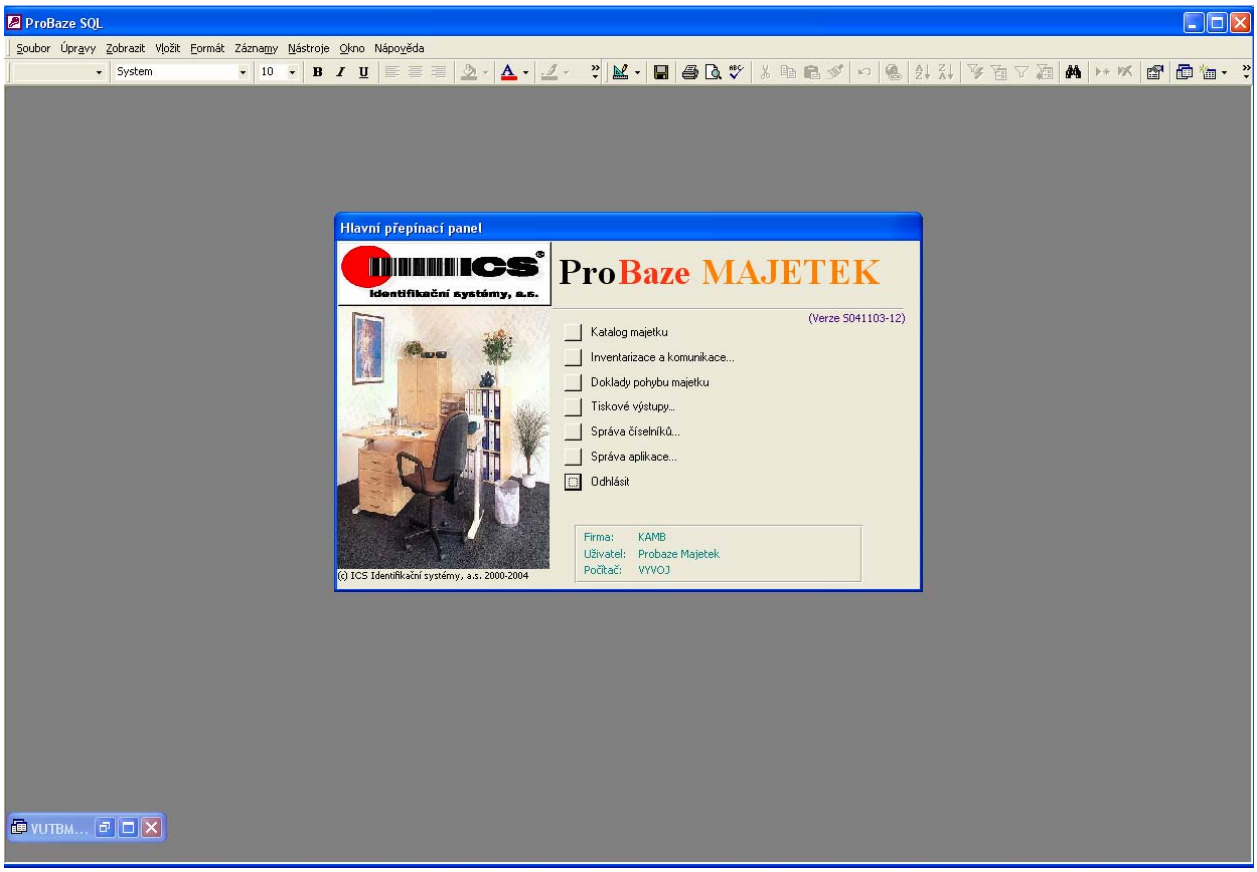
Import dat do systému se provádí pomocí volby „AUTO komunikace za SAP“. Ze systému SAP se data pouze importují. Do systému se žádosti o změny (vyřazení majetku, přesun majetku…) zadávají na základě vytištěných dokumentů.

Umístění majetku v systému SAP bere systém ProBase jako **„lokalitu očekávanou“**, tam kde obsluha najde majetek při inventuře je považován za **„lokalitu skutečnou”**. Výsledkem provedené inventury jsou tyto stavy:

- OK – lokalita očekávaná = lokalita skutečná
- MANKO – předmět nebyl nalezen nikde
- PŘEBYTEK – předmět neměl evidovánu lokalitu, nebo byl již dříve vyřazen z evidence aniž by došlo k jeho fyzické likvidaci.
- PŘESUN Z – předmět se vyskytl v jiné lokalitě než očekávané
- PŘESUN DO – předmět nebyl nalezen v lokalitě očekávané, ale byl nalezen v jiné lokalitě. Je to doplňující informace ke stavu „PŘESUN Z“

Při zjištění konfliktu mezi lokalitou určenou a zjištěnou je nutné tento nesoulad odstranit. Výsledkem inventury může tedy být:

- Převodka majetku (PŘESUN DO)
- Zařazení majetku (PŘEBYTEK)
- Vyřazení majetku (MANKO)



Obrázek 2.1: *Hlavní okno programu ProBase*

2.1.3 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 [3]. 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

IS Apollo získává informace z centrálního datového serveru. Tento server je založen na databázovém systému Oracle.

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 rychlí a zajistí, že synchronizace dat trvá jen pár minut.

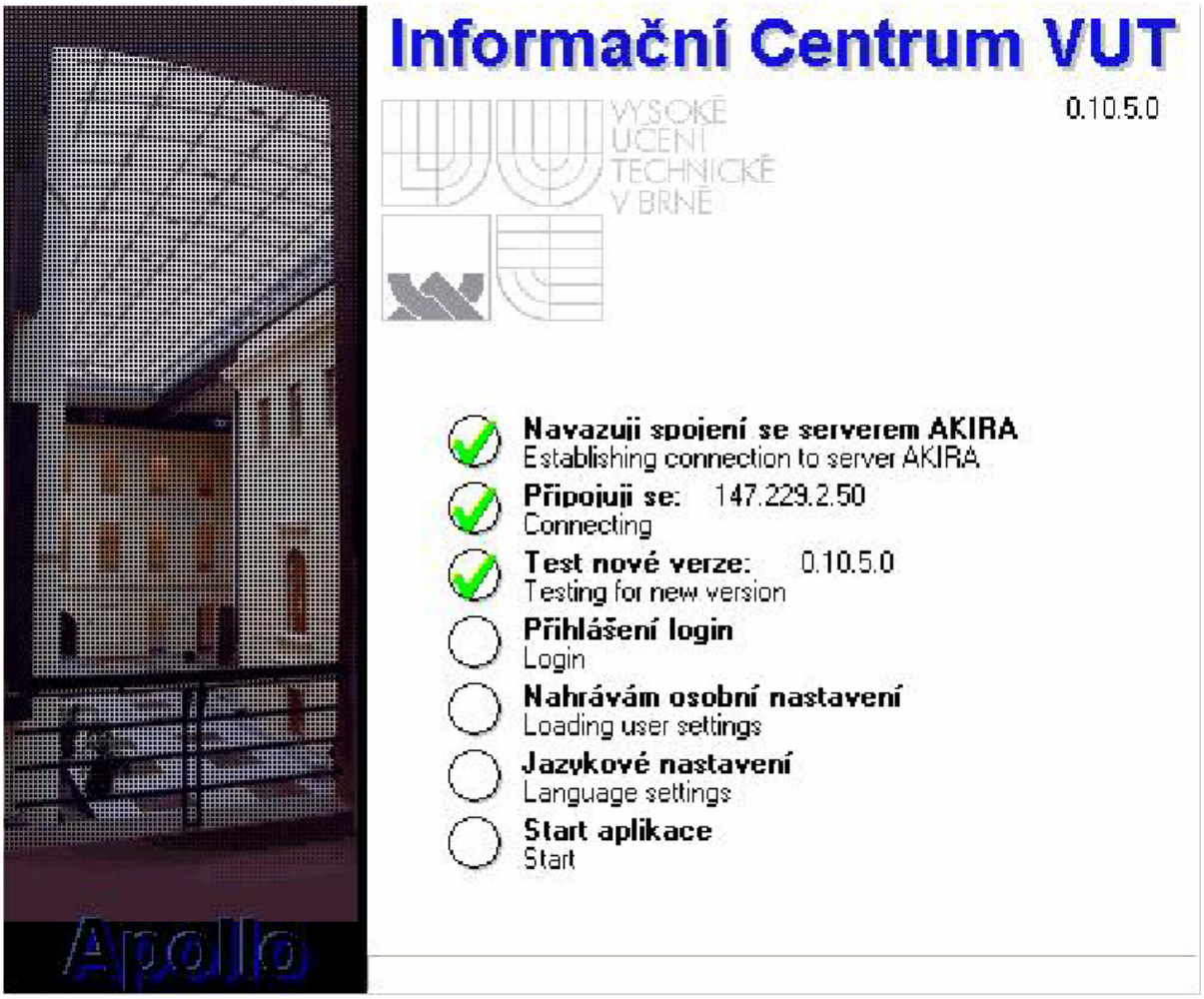
Spuštění

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 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.



Obrázek 2.2: *Spuštění Apolla*

Databázový přístup

Vzhledem k tomu, že Apollo je aplikační klient měl by mít možnost přístupu k datum 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 z 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

2.1.4 Modul evidence majetku

Modul evidence majetku v současné době není spuštěn, protože byl řešen také v tomto akademickém roce. Modul funguje stručně tak, že každá osoba vidí svůj majetek a majetek svých podřízených [1].

Integrace inventarizace do tohoto modulu by spočívala v přidání operací „zahájit inventuru“ a „ukončit inventuru“ nad tímto výběrem. Modul přidává do centrálního datového serveru následující tabulky:

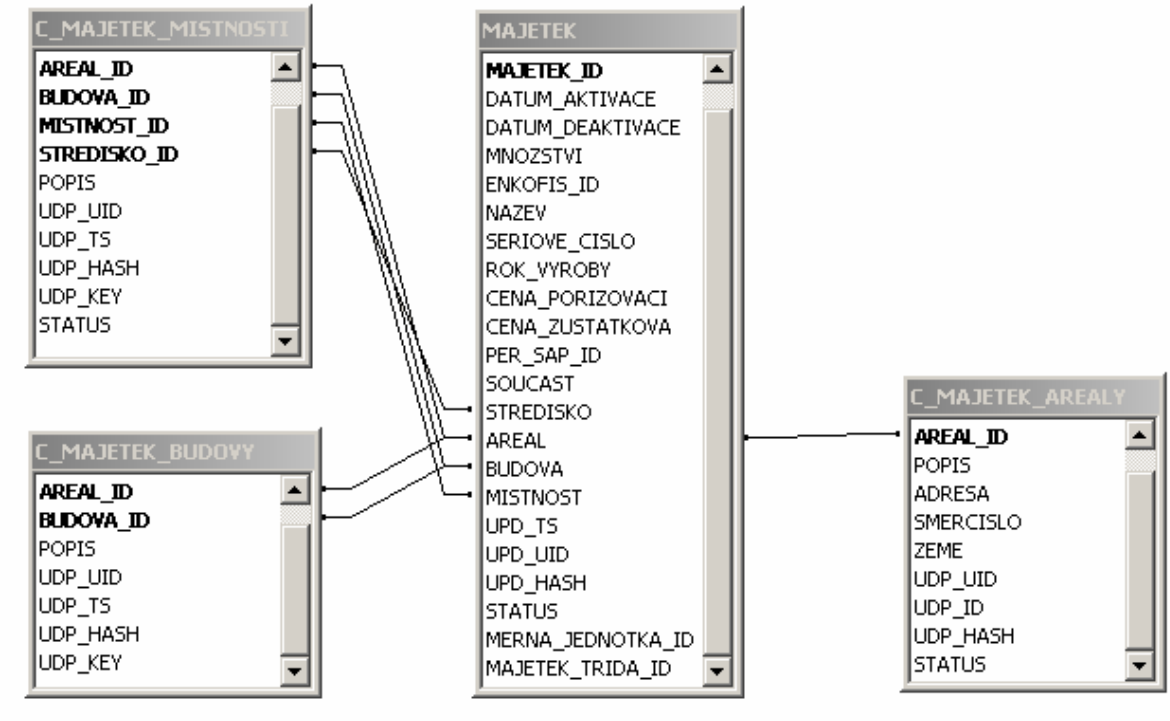
- majetek
- majetek_zmena
- majetek_popis

- 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 načítáni dat o majetku. Hlavní částí tohoto databázového návrhu je tabulka majetek, bude také nejvíce využívána pro inventuru. Tabulka obsahuje všechny potřebné informace o majetku odkazy na jeho popis a umístění.

Databázový návrh obsahuje také číselníky označené prefixem „c_“. Číselníky jsou užitečnou pomůckou, každé velké databáze. Uchovávají si v sobě informaci o identifikaci položky a další popisující vlastnosti. Tím je zmenšena velikost hlavní databázové tabulky.

Pro potřeby inventury budou využity stávající tabulky modulu evidence majetku, jen bude potřeba do tabulky majetek přidat pole CAROVYKOD a CIPOVYKOD položky.



Obrázek 2.3: *Struktura vybraných tabulek*

3 Návrh řešení

Na základě dosažených informací by bylo vhodné řešit inventarizaci majetku jako součást modulu evidence majetku. Inventura by se spouštěla nad majetkem osoby využívající modul evidence majetku. Po prostudování stávajících terminálů, které využívá program ProBase jsem usoudil, že by bylo vhodné zaměřit se na novější přístup a to PDA s operačním systémem WindowsCE. Tuto myšlenku jsem prokonzultoval s Ing. Musilem a dohodli jsme se, že se zaměřím na tvorbu aplikace pro PDA a modul komunikace s tímto zařízením.

3.1 Modul pro komunikaci s PDA

Modul pro komunikaci by měl být implementován ve vývojovém prostředí Borland Delphi 7, aby byl kompatibilní s modulem pro evidenci majetku a mohl být později do tohoto modulu integrován. Hlavním úkolem tohoto modulu by mělo být komunikace s PDA a zpracování dat získaných z databáze.

Data získané dotazem do databáze je nutné uložit do určitého formátu. Prostudoval jsem následující možnosti:

- XML
- Databáze
- Vlastní struktura souborů

XML

Použití formátu XML by bylo velice elegantní, ale vzhledem k předpokládanému množství dat by mohl mít tento přístup za následek pomalou aplikaci a nedostatek paměti v cílovém zařízení. Prostudoval jsem práci s XML dokumentem a zjistil jsem, že vyhledávání a filtrování tohoto dokumentu předchází nahrání celé struktury do paměti. To by v případě PDA s omezenými prostředky mohlo být nežádoucí.

Databáze

Program MS ActiveSync umožňuje konverzi databáze do formátu CDB což je databáze PoketAccess. Výhodou je určitá abstrakce nad daty, ale databáze může být pouze jednoúrovňová a rozhráni pro práci s touto databázi v PDA je slabé a nedostačující.

Vlastní soubory

Tato možnost není až tak elegantní jako předchozí dva přístupy, ale pro moje účely je nutná. Možnost spočívá v konverzi dat z databáze do vlastní struktury souborů. Soubory jsem tvořil tak, aby bylo vyhledávání co nejrychlejší. Seřazením těchto dat již v PC umožní pozdější možnost binárního vyhledávání v souborech. Data mohou být uložena v souborech, nemusí se nahrávat do paměti, a tím je umožněno zpracování velkého množství dat. Jediná nevýhoda tohoto přístupu je pracnější implementace.

3.2 Aplikace pro PDA

Tato aplikace by měla být uživatelsky příjemná a jednoduchá. Měla by přebírat funkčnost stávajících terminálů programu ProBase a vytvořit tak prostředek pro jednoduchou a rychlou inventarizaci majetku. Pro vývoj této aplikace jsem použil vývojové prostředí eMbedded Visual Studio C++ 4.0 [2][4].

Komunikace se zařízením standardně využívá program MS ActiveSync a nenašel jsem důvod proč tuto skutečnost měnit.

3.3 Volba PDA

Hlavním problémem byla vysoká cena PDA s integrovanou čtečkou č. kódu. PDA by mělo být také libovolně rozšířitelné o čtečku čipových karet. Po domluvě s Ing. Musilem jsem se rozhodl si PDA s integrovanou čtečkou zapůjčit.

PDA jsem si zapůjčil od firmy **Eprin spol. s r. o.,** takže nebyly kladeny žádné finanční nároky na tento projekt. Zařízení má v sobě integrovaný skener č. kódu a slot PCMCI pro vložení čtečky RFID čipu. Jedná se o průmyslové PDA od firmy UNITECH typ PA960. Je vybaveno operačním systémem WindowsCE 4.2 takže pro vývoj aplikace budu muset použít vývojové prostředí eMbedde Visual Studio C++ 4.0.

Obrázek 3.1: *Detail PA960*

4 Implementace

4.1 Modul pro komunikaci s PDA

V rámci tohoto projektu jsem modul řešil jako samostatnou aplikace s vlastní databází, která má strukturu stejnou jako tabulky modulu evidence majetku. Později lze tento modul lehce integrovat do modulu evidence majetku nahrazením databáze výsledkem dotazu. Modul implementuje dvě operace nad majetkem a to:

- Zahájení inventury – provede se výběr potřebných dat z databáze, konverze na lepší formát a uložení do souborů. Poté se datové soubory nahrají do PDA a spustí se aplikace inventury.
- Ukončení inventury – provede se test na ukončení inventury a následné stažení potřebných dat z PDA. Data se zkonvertují a uloží zpět do databáze. V tomto místě by měl být využit modul evidence majetku pro zpracování získaných dat. Zpracování by mělo proběhnout formou tisku těchto dokumentů:
  - Přesun majetku
  - Vyřazení majetku
  - Zařazení majetku

4.1.1 Funkce modulu

Po zahájení inventury modul provádí dotaz do databáze a následnou konverzi dat do datových souborů. Data jsou poté nahrána do PDA a je spuštěna aplikace inventury. Po ukončení inventury jsou data z PDA stažena a zkonvertována zpět do databáze.

4.1.2 Struktura dat

Struktura dat jsem zvolil takovou, aby vyhledávání v datových souborech bylo co nejrychlejší. Každý datový soubor má ještě soubor indexů. Datové soubory mají příponu *.DAT. Indexové soubory mají příponu *.IDX. Indexové soubory jsou řazeny podle zvoleného klíče a obsahují pouze indexy do datových souborů. Vyhledávání položky se poté provádí binárně nad souborem indexu. Rozdělením na datové a indexové soubory jsem získal možnost přenést do PDA tabulku, která je řazena podle různých kriterií. Pro účely inventarizace majetku jsou modulem tvořeny tyto soubory:

- SOUCASTI.DAT, SOUCASTI.IDX – datový a indexový soubor číselníku součástí
- STREDISKA.DAT, STREDISKA.IDX – datový a indexový soubor číselníku středisek
- AREALY.DAT, AREALY.IDX – datový a indexový soubor číselníku areálů
- BUDOVY.DAT, BUDOVY.IDX – datový a indexový soubor číselníku budov
- MISTNOSTI.DAT, MISTNOSTI.IDX – datový a indexový soubor číselníku místností

- MAJETEK.DAT – datový soubor majetku
- MAJETEK_KOD.IDX – indexový soubor do souboru majetku seřazený podle č. kódu
- MAJETEK_MIS.IDX – indexový soubor do souboru majetku seřazený podle místností
- ZMENY.DAT – soubor se změnami položek po inventuře

Struktura jednotlivých souborů je definována v unitě *Exchange.pas.*

Struktura souboru číselníků:

- id – identifikace položky
- popis – popis položky

Struktura souborů majetku:

- stav
- areál
- budova
- místnost
- carovy_kod
- nazev
- majetek_id
- stredisko
- soucast
- jmeno

Indexové soubory mají strukturu identickou. Obsahuji uložené indexy do datových souborů. Index je typu *longint*. Indexové soubory musí mít jednotnou délku záznamu, kvůli již zmiňovanému binárnímu vyhledávání.

Struktura souboru změn:

- položka_id – pozice změněné položky v datovém souboru majetku
- nova_pozice – nová pozice položky (areál, budova, místnost)

Soubor změn nemusí mít indexový soubor, protože je to výstup z PDA a je procházen sekvenčně. Tento soubor obsahuje změny umístění provedené během inventury.

4.2 Aplikace pro PDA

Verze operačního systému (4.2) je dost vysoká, ale program jsem psal z ohledem na to, že může být s minimálními změnami přepsán na operační systém nižší verze. Používal jsem funkce, které jsou podporovány ve verzích 1.0 a 2.0.

Vzhledem k tomu, že vývojové prostředí nepodporuje možnost kotvení komponent v návrhu aplikace a dynamickou změnu formuláře, je návrh aplikace proveden staticky a v případě přesunu na jiný typ zařízení musí být obrazovky přeformátovány.

Aplikaci je implementována pomocí MFC knihovny. Tento přístup mi zajistil možnost programování aplikace založené na událostech. Jednotlivé obrazovky tvoří dialogy děděné ze základní třídy *CDialog*. Dialogy mezi sebou komunikují prostřednictvím předávaných referencí na objekty v konstruktoru dialogu.

4.2.1 Abstrakce nad datovými soubory

Zvolením možnosti předávaní informací vlastní strukturu souboru jsem ztratil výhody databázového přístupu. Toto jsem se rozhodl řešit napsáním vlastních tříd, které se budou tvářit jako tabulky databáze. Třídy pro práci z datovým soubory jsem implementoval do souboru *TDatabase.h TDatabase.cpp*. Potřebné objekty, které mi budou simulovat tabulky, jsou realizovány ve třídě *TData* a změny jsou realizovány třídou *TZmeny*.

TDatabase

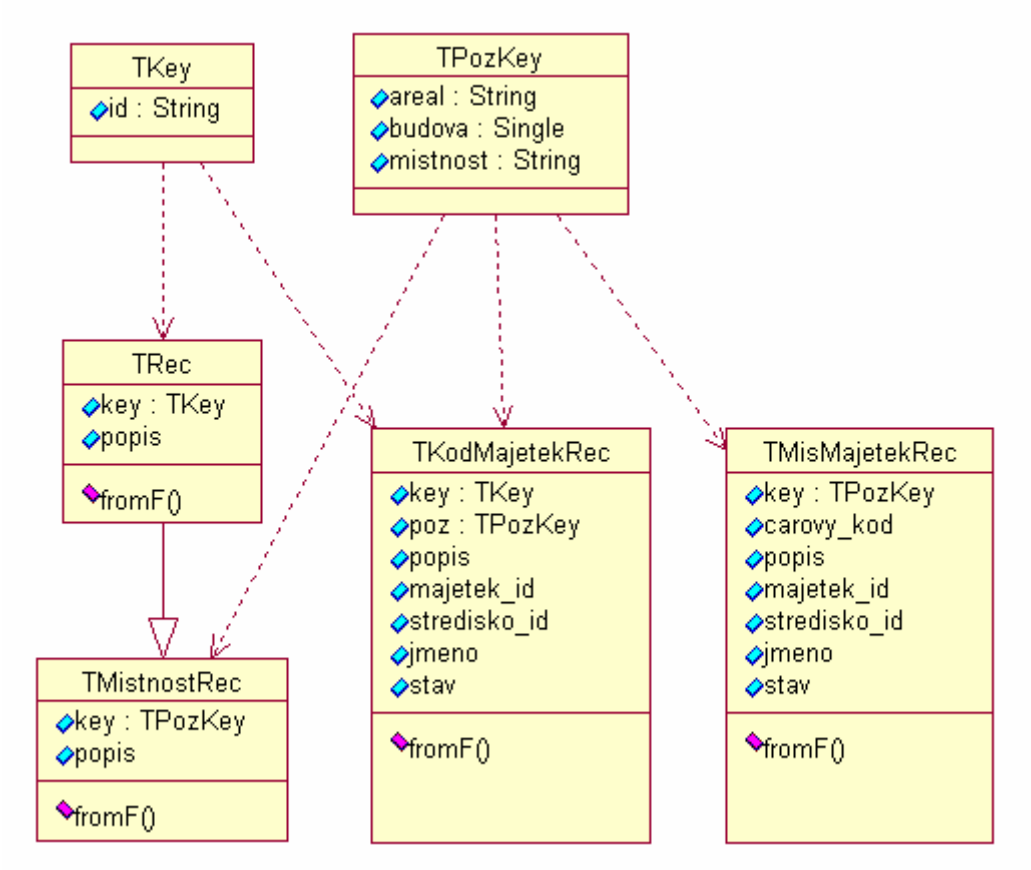
Soubor *TDatabase.h* obsahuje definici struktur souboru, metody pro operaci ze soubory a definici třídy *TTable,* která simuluje databázovou tabulku. Dále obsahuje definici třídy *TMajetekTable*.
Metody pro práci ze soubory:

- *readStrF* – Funkce pro čtení řetězce z otevřeného souboru. Funkce vrací přečtený řetězec a má dva parametry: soubor a délku řetězce.
- *readLenStrF* – Funkce čte ze souboru délku řetězce. Má jeden parametr a to soubor.
- *readLongF* – Funkce čte ze souboru typ long. Má jeden parametr a to soubor.
- *writeStrF* – Funkce provádí zápis řetězce do souboru. Má dva parametry: soubor a řetězec.
- *openFr* – Funkce otevře soubor pro čtení a v případě chyby vyvolá výjimku *TWin32Ex*. Tato výjimka je definovaná v souboru *Excepts.h.*
- *openFw* – Funkce otevře soubor pro zápis a v případě chyby vyvolá výjimku *TWin32Ex*. Tato výjimka je definovaná v souboru *Excepts.h*.
- *rewriteF* - Funkce vytvoří soubor a v případě chyby vyvolá výjimku *TWin32Ex*. Tato výjimka je definovaná v souboru *Excepts.h*.

Definované struktury:

- *TRec* – je základní struktura souboru. Tuto strukturu mají soubory číselníků.
- *TMistnostRec* – je to struktura odvozena ze základní třídy *TRec* a definuje strukturu souboru místnosti. Oproti základní třídě redefinuje klíč *TKey* na *TPozKez* což je složený klíč pozice položky.
- *TKodMajetekRec* – je to struktura definující tabulku majetku řazeného podle kódu. Klíčem je zde č. kód položky *TKey*.
- *TMisMajetekRec* - je to struktura definujíce tabulku majetku řazeného podle místností. Klíčem je zde pozice položky *TPozKey.*

Každá struktura má definovanou metodu *fromF*, která provede načtení dat ze souboru.



***Obrázek 4.1**: Přehled datových struktur a jejich závislostí*

V tomto souboru je také definována šablona tříd *TTable.* Tato šablona umožňuje definovat pouze jednu třídu pro všechny druhy tabulek. Parametry třídy jsou jména indexového a datového souboru. Parametry šablony jsou:

- Typ tabulky – jedna z výše definovaných struktur.
- Typ klíče – tento parametr může nabývat hodnot *TKey* a *TPozKey* nebo jakékoliv jiné struktury, která má definovaný operátor ==(rovno) a <(menší).

Metody šablony tříd *TTable*:

- *get* – Tato metoda provede vrácení i-tého záznamu z tabulky. Vrací pořadí tohoto záznamu v tabulce při úspěchu a nebo -1, když i-tý záznam není nalezen. Hodnota nelezeného záznamu je uložena do druhého parametru funkce.
- *getNext* – Tato metoda provede vrácení dalšího záznamu z tabulky. Vrací pořadí tohoto záznamu v tabulce při úspěchu a nebo -1, když další záznam není nalezen. Hodnota nelezeného záznamu je uložena do prvního parametru funkce. Třída TTable si udržuje informaci o aktuální pozici, a proto je nutné před voláním metody getNext inicializovat aktuální pozici metodami *get, search* nebo *searchFirst.*

- *search* – Tato metoda provede binární vyhledání záznamu v tabulce. Vrací pořadí tohoto záznamu v tabulce při úspěchu a nebo -1, když záznam není nalezen. Hodnota nelezeného záznamu je uložena do druhého parametru funkce. Prvním parametrem je klíč pro vyhledání.
- *searchFirst*– Tato metoda provede binární vyhledání záznamu v tabulce a následuje sekvenční postup tabulkou vzhůru a kontrola zda klíč stále vyhovuje. Vrací pořadí tohoto záznamu v tabulce při úspěchu a nebo -1, když záznam není nalezen. Hodnota nelezeného záznamu je uložena do druhého parametru funkce. Prvním parametrem je klíč pro vyhledání.

Soubor také definuje třídu *TMajetekTable*. Což je třída tabulky majetek. V konstruktory třídy se provádí vytvoření tabulek majetku řazených podle místností a podle kódu. Třída přináší dvě nové metody pro operaci nad tabulkou majetku:

- *updateStavSortMis* – Tato metoda provádí změnu stavu majetku, který je určen pozicí v tabulce majetku řazeného podle místností. Prvním parametrem je pozice do tabulky majetku řazeného podle místností a druhým parametrem je nový stav.
- *updateStavSortKod* – Tato metoda provádí změnu stavu majetku, který je určen pozicí v tabulce majetku řazeného podle kódu. Prvním parametrem je pozice do tabulky majetku řazeného podle kódu a druhým parametrem je nový stav.

TData

Tato třída je definovaná v souborech *TData.h* a *TData.cpp*. Zajišťuje tvorbu objektu jednotlivých tabulek v konstruktoru a správnou destrukci těchto tabulek v destruktoru. Tento přístup umožňuje v aplikaci vytvořit instanci této třídy a máme k dispozici všechny potřebné tabulky a operace nad nimi.

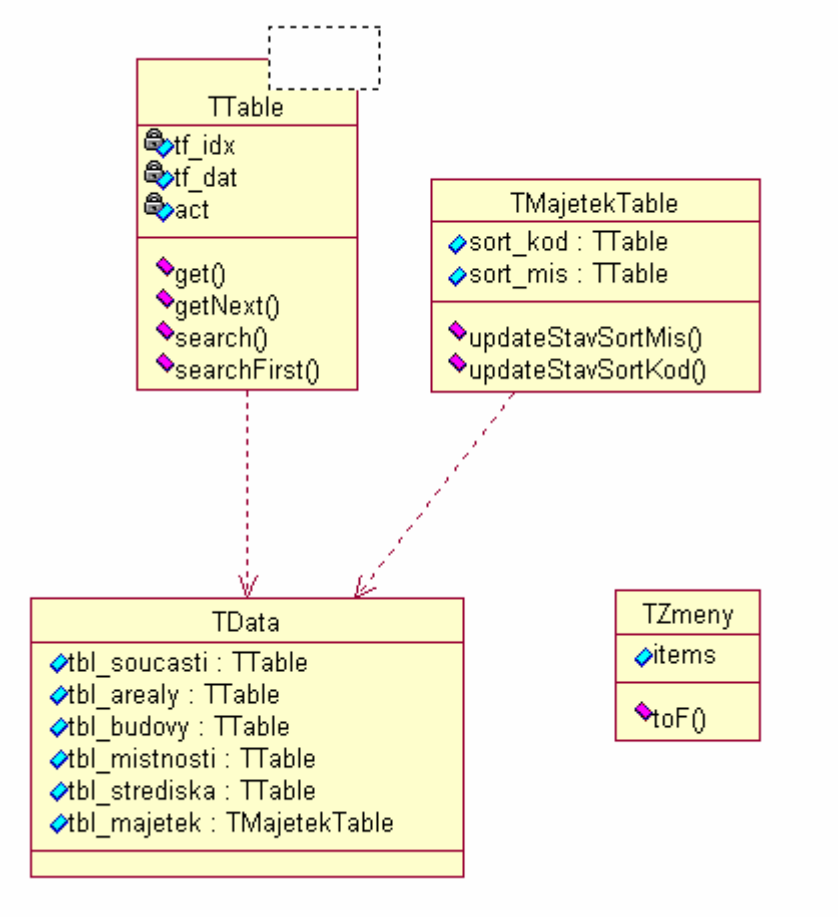
TZmeny

Třída realizuje změny položek. Definice této třídy je nutností, protože do seřazených tabulek nelze vkládat informace o změně umístění. Přepsáním pozice položky by nebylo zajištěné seřazení tabulky a nemohli by jsme binárně vyhledávat.

Hlavní částí této třídy je *vector*, který je součástí STL C++ a je definován v souboru *vector.hpp*. Prvkem vectoru je struktura *TItemPoz*:

- kod – č. kód položky
- pozice – nová pozice položky
- fidx – index do tabulky majetku seřazeného podle kódu

Třída pracuje nad souborem změn a to tak, že v konstruktoru si načítá data ze souboru změn a implementuje metodu pro uložení změn do souboru *toF*. Metoda *toF* zde plní funkci zálohy dat. Je možné kdykoliv provést uložení dat do souboru, i když instance stále existuje.



Obrázek 4.2: *Přehled definovaných tříd a jejich závislostí*

4.2.2 Dialogy aplikace

Aplikace je založená na dialogách odvozených ze základního dialogu knihovny MFC *CDialog*. V projektu jsem si definoval tyto dialogy:

- CMainDlg – hlavní dialog aplikace
- CInvDlg – dialog inventury
- CDetailDlg – dialog detailu položky
- CPromptDlg – dialog ručního zadávání kódů

CMainDlg

Hlavní dialog je spuštěn po startu aplikace a je určen pro zadání místnosti, pro kterou chceme inventuru provést. V konstruktoru dialogu je vytvořeny objekty třídy *TData* a *TZmeny*. Do ostatních dialogů jsou objekty předávány pomocí reference. Tento přístup zlepšuje čitelnost kódu a potlačuje tvorbu globálních objektů.

Dále je zde zaregistrovaná zpráva WM_SCANNER, která je zaslána dialogu při čtení scannerem. Funkce pro ovládání scanneru a zaregistrováni zprávy jsou součásti knihovny *USI.DLL*. Knihovna je dodávána společně s PDA firmou Unitech.

CInvDlg

Dialog CinvDlg je spuštěn po vybrání místnosti. V konstruktoru jsou předávány reference na objekty dat a změn. Po vyvolání dialogu se provede načtení aktuální místnosti do paměti. Seznam položek je realizován *vectorem*. Prvkem vectoru je struktura *TItemIdx*:

- kod – čárový kód položky
- stav – stav položky
- fidx – index do tabulky majetku seřazené podle místností
- lidx – index do prvku *CListCtrl* který zobrazuje položky místnosti.

Načtení prvků do místnosti nám urychlí vyhledávání předpokládaných položek místnosti. Vyhledávání položek je realizováno funkcí *find*, která je součástí STL a je definovaná v souboru *algorithm.hpp*.

Po zadání položky je zahájeno vyhledávání v aktuální místnosti. Není-li položka nalezena v aktuální místnosti provádí se vyhledávání v souboru majetku. Po nalezení položky je spuštěn parametrický dialog *CDetailDlg*. Parametr určuje zda se jedná o položku místnosti, nebo o položku, která v místnosti není.

Po ukončení dialogu se provede změna stavu a případný přesun do této místnosti. Je-li proveden přesun uloží se tato informace to objektu změn.

Změny jsou zálohovány po každém odchodu z místnosti. V paměti stále zůstávají, ale provede se záloha těchto dat do souboru voláním metody *toF*.

CDetailDlg

Je parametrický dialog. V konstruktoru jsou předávány reference na objekty dat a změn a dále parametr dialogu: 0 – položka byla nalezena v místnosti, 1 – položka není v této místnosti. Na základě parametrů jsou uživateli nabídnuty operace s touto položkou:

- 0 – položka byla nalezena
  - kontrola na stávající pozici
- 1 – položka nebyla nalezena
  - přesun položky do aktuální místnosti – položka je přesunuta do aktuální místnosti
  - kontrola na stávající pozici – položka je zkontrolována na své původní pozici

V konstruktoru dialogu se také kontroluje stav položky. Kontrola již zkontrolované položky není uživateli dovolena a je o této skutečnosti informován. Poté přechází dialog do módu prohlížení a stav položky nemůže být změněn.

CPromptDlg

Tento dialog slouží k manuálnímu zadávání kódů. Dialog se skládá z nadpisu, který může být změněn v konstruktoru a polem pro zadávání dat. Nadpis je realizován prvkem *CStatic* a pole pro zadáváni dat prvkem *CEdit*. Data z dialogu jsou předávána přes referenci v konstruktoru.

4.3 Problémy řešené při implementaci

Při řešení projektu jsem se setkal z mnoha problémy. Většinu problému jsem vyřešil, ale za cenu velkého zdržení.

Největším problémem byla nekompatibilita standardního řetězce v C++ a řetězce, který je podporován knihovnou MFC. Pro účely konverze těchto datových typů jsem si vytvořil sadu funkcí, které jsou definované v souboru *Toolkit.h*:

- *toBSTR* – převede BSTR na standardní *string*
- *fromBSTR* – převede standardní *string* na *BSTR*

Dále bylo nutné po každé práci s typem *BSTR* provádět manuálně dealokaci paměti. Tuto skutečnost jsem vyřešil tvorbou třídy *BSTRWrap*, která v destruktoru dealokuje typ *BSTR*. Použití jde vidět na následujícím příkladě: *BSTRWrap(toBSTR(„prevod stringu na BSTR“))*.

Dalším problémem byla automatická komunikaci mezi PDA a pracovní stanicí. Microsoft ActiveSync definuje rozhraní, které je nutné implementovat. Po implementaci tohoto rozhraní lze k programu přistoupit přes COM rozhraní. Tuto část se mi nepodařil realizovat.

5 Závěr

5.1 Zhodnocení dosažených výsledků

V rámci projektu jsem se musel seznámit s evidencí majetku na VUT. Musel jsem prostudovat evidenci majetku v systému SAP a program pro inventarizaci majetku ProBase. Také jsem prostudoval modul evidence majetku, který s inventarizací úzce souvisí.

Po prostudování všech systému jsem provedl analýzu programu ProBase. Po domluvě s Ing. Musilem jsem se zaměřil na nový přístup sběru dat pomocí PDA.

Před samotnou implementací jsem se musel seznámit s novým vývojovým prostředím a to MS Visual Studio C++. Práce s tímto vývojovým prostředím nebyla zpočátku jednoduchá, ale po delší práci mi začalo vyhovovat.

Integrace inventarizace umožní každému uživateli systému Apollo provést inventuru svého majetku. Zpřehlední kontrolu a zlepší evidenci majetku. Vzhledem k vývoji modulu evidence majetku v tomto akademickém roce není inventarizace plně integrována do modulu, ale vhodný návrh lokální databáze umožní jednoduchou integraci v další etapě projektu.

5.2 Další vývoj projektu

Další vývoj projektu by spočíval ve vylepšení inventury, plné integrace do modulu evidence majetku a tím i do systému Apollo. Také by bylo vhodné přenést aplikace na cílové zařízení, které se bude při inventuře používat. V současné době je aplikace spouštěna na zapůjčeném PDA od firmy **Eprin spol. s r. o.**. Rozšířením by také mohl být přihlašování uživatelů na PDA pomocí čipových karet a šifrování dat. Také by bylo vhodné vyřešit problém s komunikací, který se mi nepodařilo vyřešit.

5.3 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.

Dále mě velice obohatilo seznámení s novými vývojovými prostředími. Při řešení projektu jsem pracoval současně ve dvou různých programovacích jazycích což mi umožnilo provést srovnání těchto jazyků. Dříve jsem byl zvyklí programovat pouze v jazyce C++, ale při řešení projektu jsem byl donucen hledat alternativy v jazyce Pascal.

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

Literatura

[1] Václav Bezděk: Osobní evidence majetku VUT v Brně [Bakalářská práce]. FIT VUT Brno, 2006

[2] Kolektiv autorů: Příručka produktu Microsoft Windows eMbedded. 1.5.2006. URL: http://www.microsoft.com/cze/windows/embedded/pruvodce/

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

[4] Andrei Alexandrescu: Moderní programování v C++. Šablony generické komponenty a návrhové vzory. Praha, Computer Press 2004

[5] ICS a.s. Hoňatecká 19/1772, Praha 8. ProBase manuál VUT.

Příloha A Uživatelská příručka

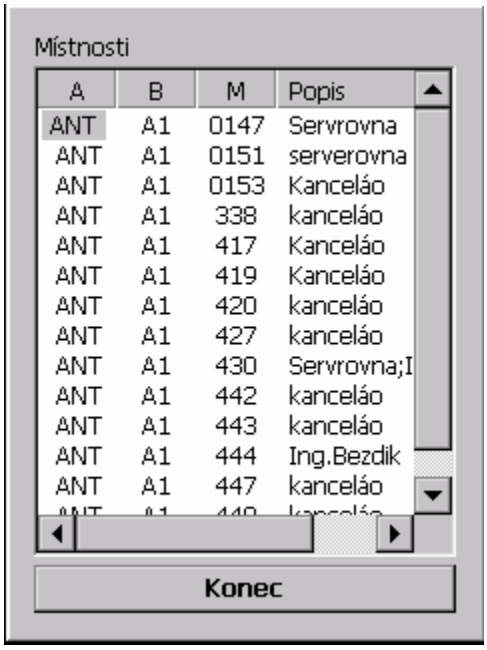
Modul pro komunikaci

Obsluha modul pro komunikaci je jednoduchá. Spočívá v zahájení inventury a v jejím ukončení. Obsluze je zobrazena tabulka z majetkem a aktuálním stavem.

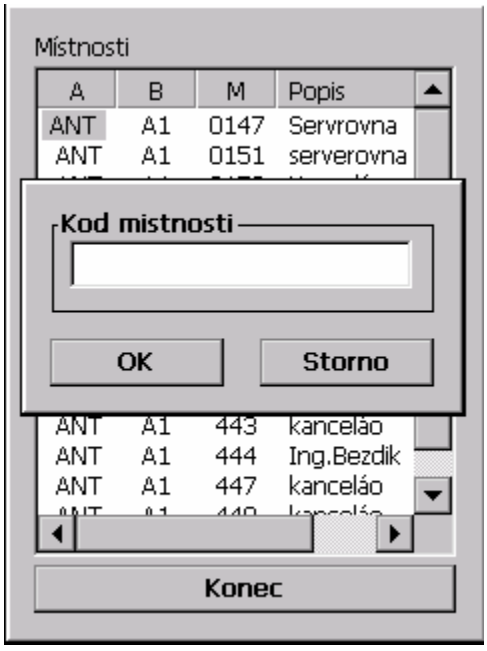
Aplikace pro PDA

Po spuštění aplikace je obsluze nabídnut seznam místností, který je v PDA nahrán. Uživatel může místnost zadat různými způsoby:

- Vybrání ze seznamu místností – Obsluha může vybrat místnost ze seznamu, který jí je nabídnut.
- Zadáním kódu pomocí klávesnice – Při stisku klávesové zkratky FUNC+1 může obsluha zadat kód místnosti přímo pomocí klávesnice.



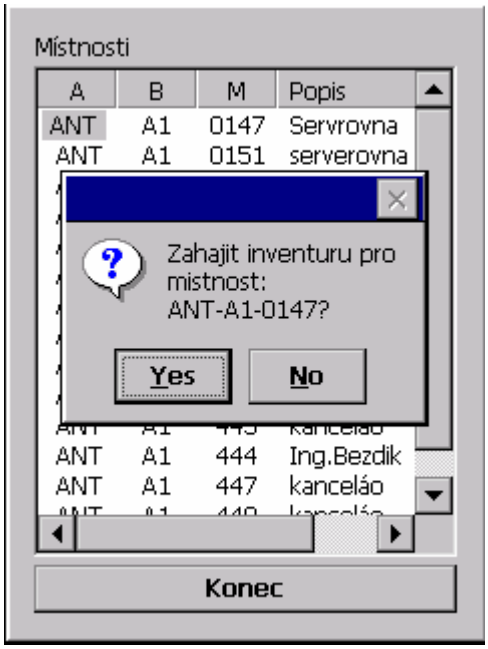
Obrázek 1: *Okno aplikace po spuštění*



Obrázek 2: *Manuální zadání kódu místnosti*

- Přečtením kódu místnosti – Tato volba je asi nejčastější.

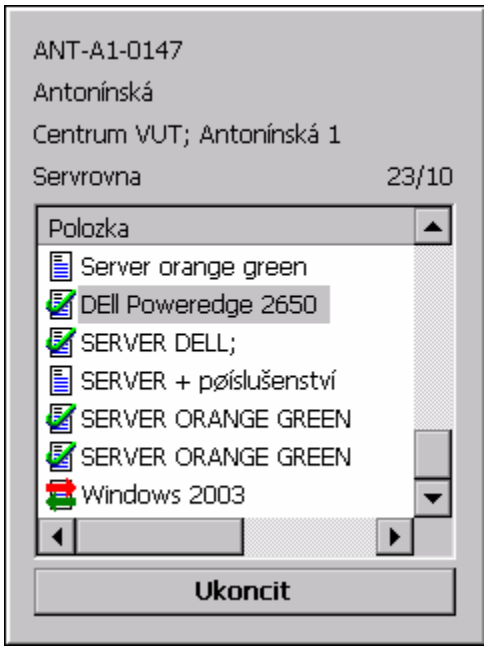
Po zadání kódu místnosti a kontrole tohoto kódu na existenci je obsluha dotázána zda chce zahájit inventuru pro tuto místnost.



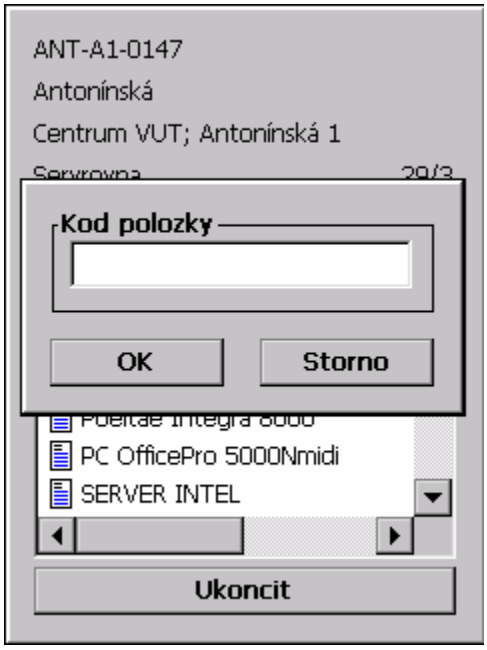
Obrázek 3: *Vstup do místnosti*

Po zahájení inventury je uživateli nabídnut seznam položek v místnosti a jejich aktuální stav. Položku pro kontrolu lze zadat stejně jako u místnosti několik způsoby:

- Přečtení kódu položky
- Ručním zadáním kódu položky
- Vybráním ze seznamu



Obrázek 4: *Zobrazení položek místnosti*

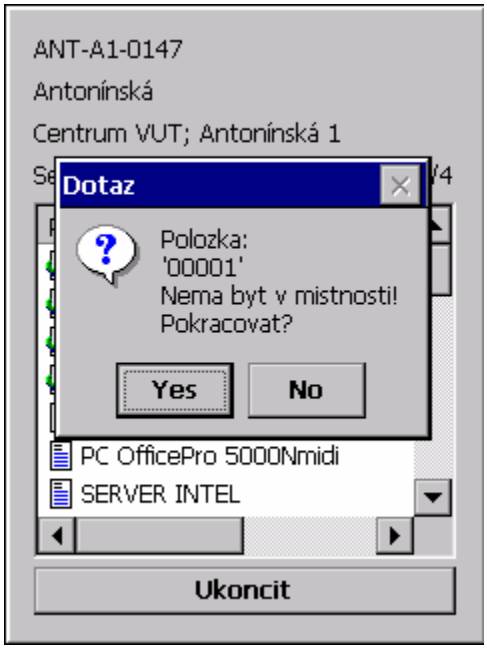


Obrázek 5: *Manuální zadání kódu položky*

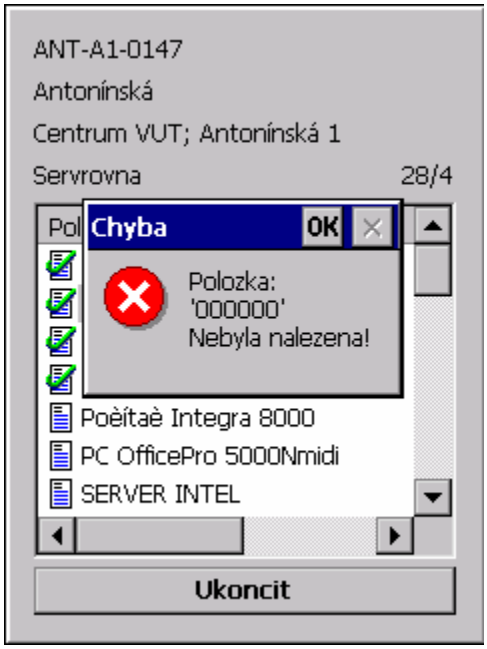
Položka v seznamu může nabývat těchto stavů:

- Nezkontrolovaná
- Zkontrolovaná
- Přesunutá

Je-li položka nalezena v aktuální místnosti lze jí přiřadit stav zkontrolováno. Není-li položka nenalezena v místnosti je obsluha dotázána, zda se má tato položka hledat v celé databázi. Při neúspěšném vyhledávání je obsluha upozorněna na neexistující položku.



Obrázek 6: *Dotaz na pokročilé vyhledávání*

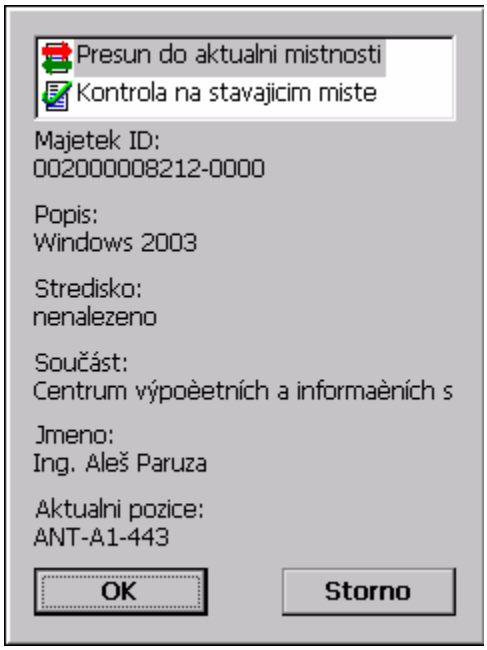


Obrázek 7: *Upozornění nenalezení zadané položky*

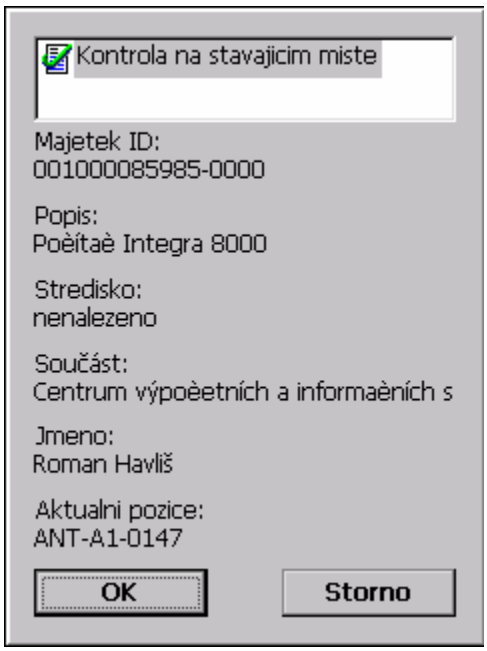
Je-li položka nalezena je zobrazen detail této položky. Na základě umístění položky jdou provádět různé operace:

- Položka je nalezena v aktuální místnosti a nebyla ještě zkontrolovaná
  - Kontrola na stávajícím místě
- Položka byla nalezena v jiné místnosti a nebyla ještě zkontrolovaná
  - Kontrola na stávajícím místě – Položka je zkontrolovaná na stávajícím místě a není přesunuta do aktuální místnosti.

- Přesun položky do aktuální místnosti – Položka je přesunuta do aktuální místnosti a získá stav přesunutá.

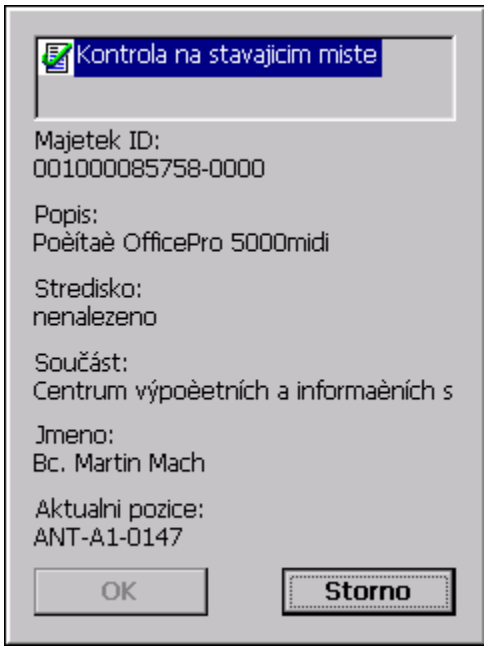


Obrázek 8: *Detail položky mimo aktuální místnost s možností změny stavu*

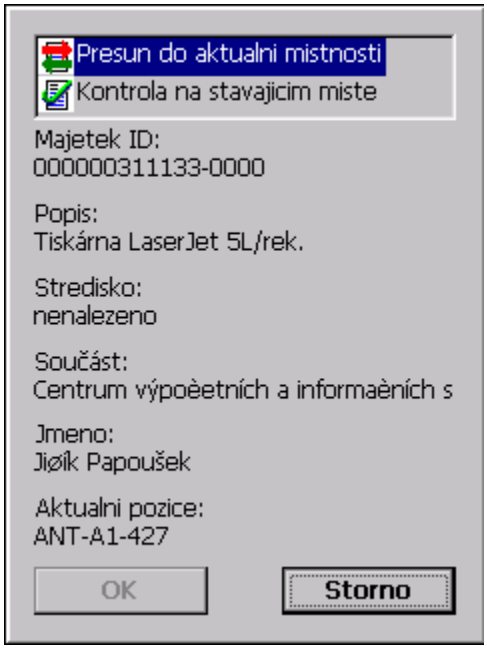


Obrázek 9: *Detail položky v aktuální místnosti s možností změny stavu*

- Položka už byla zkontrolována – Má-li položka jiný stav než nezkontrolováno, není obsluze umožněno měnit tento stav. Obsluze je zobrazen detail položky, ale jakákoli změna stavu položky není dovolena.



Obrázek 10: *Detail položky mimo aktuální místnost*



Obrázek 11: *Detail položky v aktuální místnosti*

Z místnosti se odchází stiskem tlačítka ukončit. Jsou-li v místnosti nezpracované položky obsluha je upozorněna a dotázána zda chce opravdu inventuru pro tuto místnost ukončit.

Obrázek 12: *Dotaz na odchod z místnosti*