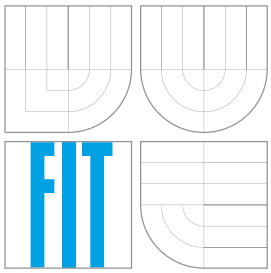
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

HONEYWELL: UNIVERZÁLNÍ GRAFICKÝ EDITOR OBRAZOVEK PRO LCD ZAŘÍZENÍ

HONEYWELL: UNIVERSAL GRAPHIC EDITOR OF SCREENS FOR LCD DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

RADEK ŠMEJDÍŘ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. FILIP ORSÁG, Ph.D.

BRNO 2009

Abstrakt

Cílem této bakalářské práce bylo vytvořit pro firmu Honeywell univerzální grafický editor obrazovek pro LCD zařízení. Tento editor je jedním modulem z celé aplikace, která slouží k automatizovanému testování termostatů a podobných zařízení. Editor umožňuje definování layoutu displeje a jednotlivých obrazovek a výstup ukládá do XML souboru. Program je určen pro platformu Microsoft Windows. Implementace byla provedena v jazyce C# s využitím prostředí .NET framework.

Abstract

This thesis aims at development of a universal graphic screen editor for LCD devices made by Honeywell. The editor is one module of a complex application, which serves automated testing of thermostats and similar devices. In the editor a display layout can be designed, as well as the individual screens. The output is stored to an XML file. The application is written in C# and must be executable in the Microsoft .NET Framework.

Klíčová slova

Honeywell, XML, C#, Microsoft Visual Studio .NET, .NET framework, displej

Keywords

Honeywell, XML, C#, Microsoft Visual Studio .NET, .NET framework, display

Citace

Radek Šmejdíř: Honeywell: Univerzální grafický editor obrazovek pro LCD zařízení, bakalářská práce, Brno, FIT VUT v Brně, 2009

Honeywell: Univerzální grafický editor obrazovek pro LCD zařízení

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením pana Ing. Filipa Orsága, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radek Šmejdíř
19. května 2009

Poděkování

Tímto bych chtěl poděkovat lidem z Honeywellu, se kterými jsem svou práci průběžně konzultoval, především pak panu Ing. Jiřímu Crhonkovi za odbornou pomoc, cenné rady a celkový dohled na mou prací.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Dnes se testování provádí ručně podle předem sepsaných pravidel, kde je napsáno, co by mělo být na vstupu a jaká je očekávaná odpověď testovaného zařízení. Při změně jakéhokoli detailu v programu, musí testeři provést veškeré testy znovu. Automatizováním těchto testů by se ušetřilo mnoho zdrojů, jako jsou například čas, finance, pracovníci, atd. Prvotní napsání vhodných testů pro nové zařízení by sice zabralo více času, ale opakované testy by byli pak už jen otázkou chvilky. Jedná se o experimentální projekt, protože není jisté, zda-li dokáží automatické testy spolehlivě odhalit chyby a zda-li nebudeme přeci jen postrádat lidský faktor při testování.

Cílem této práce je implementovat aplikaci pro vizuální popis grafických obrazovek LCD zařízení. Výstupem programu má být XML dokument, který bude popisovat jednotlivé segmenty, objekty, grupy a obrazovky zařízení. Návrh výstupního XML dokumentu je také součástí této práce. XML dokument s popisem určité obrazovky LCD zařízení, který bude vygenerován programem, jež je předmětem této práce, bude následně potřebný v testovacím skriptu.

V kapitole 2 je podrobněji popsána problematika a současný stav automatického testování. Jsou zde také uvedeny přínosy a rizika, které s sebou tento typ testování přináší. Kapitola 3 stručně popisuje technologie použité při řešení, seznamuje čtenáře s používaným názvoslovím a hlavně se zabývá návrhem struktury výstupního XML dokumentu. Kapitola 4 shrnuje implementaci celé aplikace, zejména její důležité části. Také tato kapitola stručně popisuje vytvořený program a jeho ovládání. V kapitole 5 naleznete zhodnocení tohoto projektu a návrhy na budoucí rozšíření.

Příloha A pak obsahuje stručný popis obsahu přiloženého CD. Na přiloženém CD také naleznete programovou dokumentaci, vygenerovanou programem Doxygen[1] a vytvořený vzorový projekt.

[1] www.doxygen.org

Kapitola 2

Teoretický rozbor

Tato kapitola pojednává o současném stavu testování, principu automatického testování, jeho očekávaných přínosů a rizik a o plánech pro budoucí rozšíření.

2.1 Automatické testování

Výstupem programu této práce bude XML soubor (na obrázku 2.1 označen jako blok `DisplayData`), který využije analyzátor displeje (`DisplayAnalyzer`) při zpracovávání video dat získaných od testovaného zařízení. Testovací skript se může v průběhu testů dotazovat analyzátoru displeje na stavy definovaných objektů displeje, od jednoduchých požadavků, jestli daný segment svítí, přes hodnotu objektu (konkrétní variantu), až po určení která sekvence objektu právě probíhá. U grup se dá dotazovat přímo na hodnotu, bude vrácena hodnota grupy jako řetězec složený z názvů aktuálních variant jednotlivých objektů.

Obrázek 2.1 znázorňuje rozmístění bloků a jejich úlohu na straně počítače. Na obrázku 2.2 je pak blokově znázorněna hardwarová strana včetně testovaného zařízení (označeno jako `DUT` – Device Under Tests).

2.1.1 Současný stav

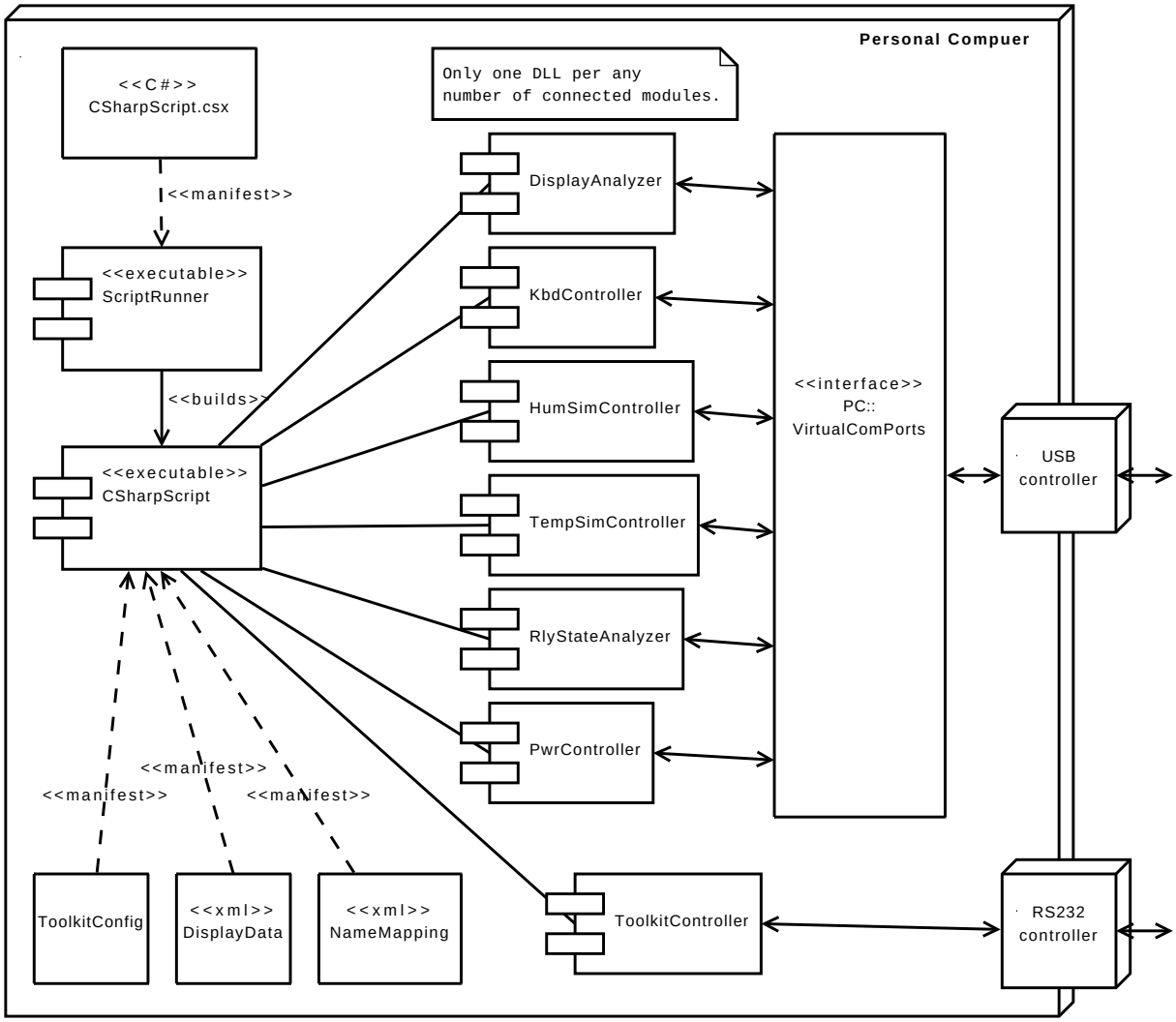
V současné době se testování probíhá ručně s podporou nástrojů pro sledování komunikace (Toolkit monitor). Některé testy jsou poloautomatizovány pomocí skriptů RF Toolkit monitoru, speciálních programů pro testované zařízení (switching testy) a close loop simulátorů pro testování kontrolních algoritmů termostatu. I přesto však probíhá 99% testů ručně.

Nasazení automatických testů by bylo výhodné zejména při aktualizaci software, odstranění chyby (vyžaduje opakované provedení skupiny testů) a při zátěžových testech.

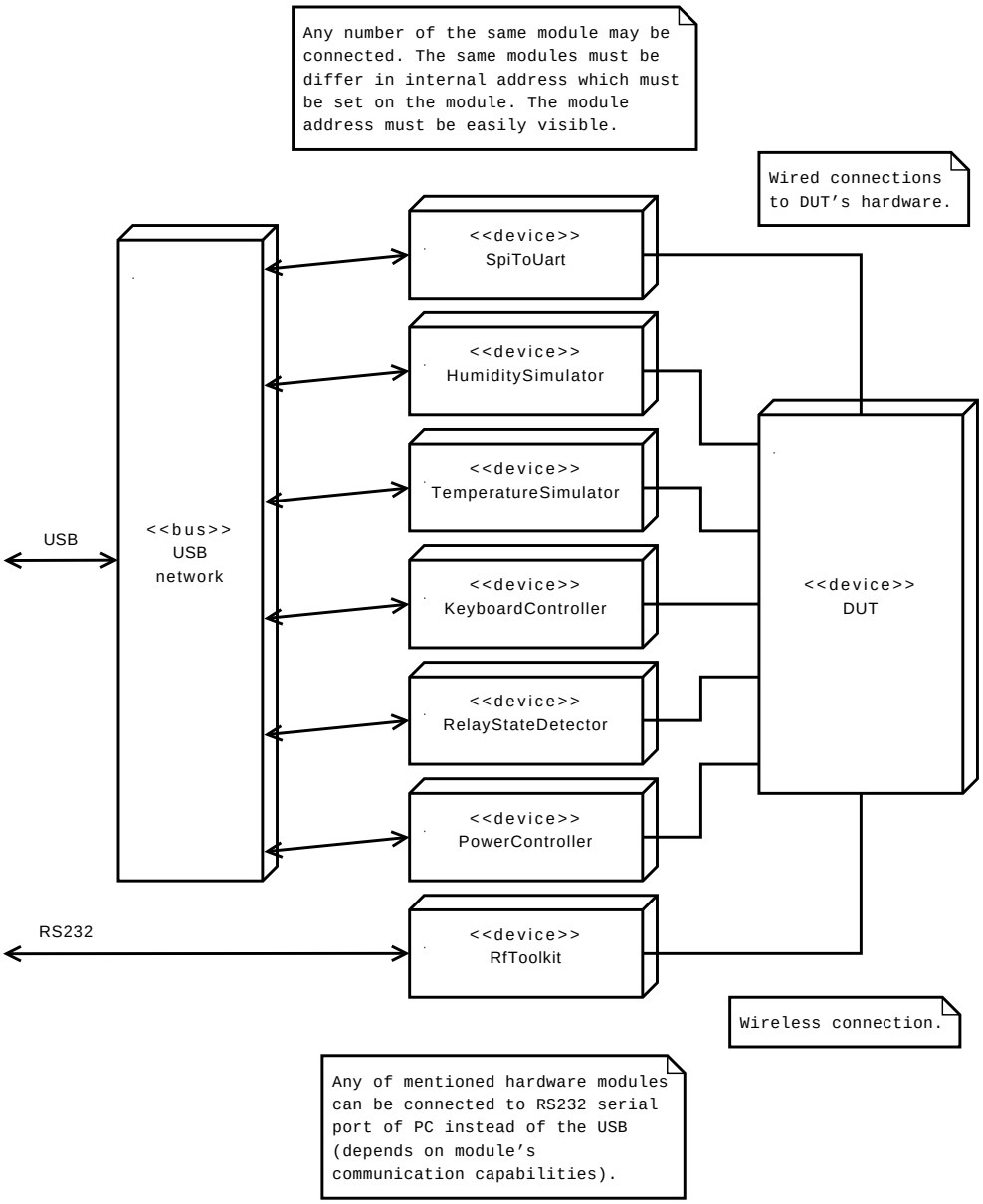
2.1.2 Koncept

Podle typu testovaného zařízení bude nutné během provádění automatických testů zpracovat některé z těchto úloh:

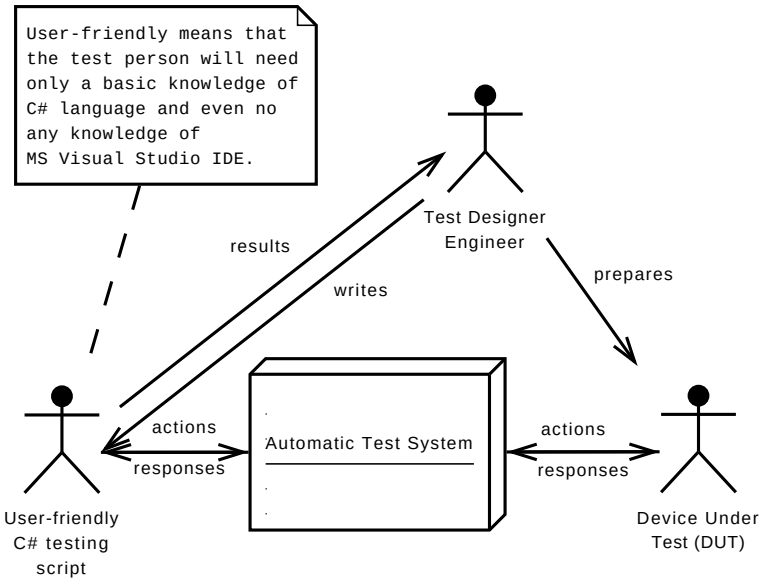
- Vyhodnocení displeje (video dat) pomocí sériové komunikace nebo kamerovým systémem.
- Ovládání klávesnice.



Obrázek 2.1: Rozvržení v PC



Obrázek 2.2: Rozvržení HW části



Obrázek 2.3: Systém

- Vyhodnocení sériové / rádiové komunikace – interface k ovládání dosud používaných testovacích aplikací (RF toolkit monitor).
- Monitorování stavu relátek (pro relátkové zařízení).
- Měření a monitorování časových vztahů a závislostí (např. mezi stimuly a reakcí testovaného zařízení, doba svitu touchscreenu, atd.).
- Ovládání napájecího zdroje (volitelně).
- Ovládání rezistorové dekády (volitelně).

2.1.3 Výhody a nevýhody

Výhody

- Významná časová úspora – souběžné provádění skupiny testů.
- Usnadnění prvotního ladění a odhalení největšího množství chyb v první fázi projektu (prvotní design) – menší množství chyb pro finální fázi projektu.
- Rychlejší nalezení zásadních chyb.
- Automatické vyhodnocení testu, efektivní generování záznamu prováděných kroků a dokumentování situace výskytu chyby, pro její snadné odstranění.
- Možnost provedení plně komplexních testů (např. test všech stavů a přechodů pro dané situace, test pro celý rozsah hodnot).
- Opakovatelnost testů – při každé odhalené a opravené chybě je nutné provést všechny testy znovu.
- Omezení lidského faktoru.

Nevýhody

- Odlišná filozofie testování – je zde riziko, že se nakonec automatické testy neprojeví jako dostatečně spolehlivé.
- Lidský faktor zůstává nepostradatelný.
- Pro sofistikované testy nelze aplikovat.
- Časová náročnost pro vývoj, prvotní rozběhnutí a nastavení automatických testů.
- Náročnější provádění automatických testů pro skupiny testovaných zařízení v závěrečné fázi projektu (není možný zásah do HW + SW testovaného zařízení).
- Možnost neodhalení chyby v programu během testu v důsledku nedokonalého testovacího skriptu.
- Nemožnost nasazení automatických testů pro všechny testované případy.

2.2 Plány do budoucna

V závěrečných fázích projektu se uvažuje o nasazení kamerového systému pro snímání obrazu. Informace o zobrazených segmentech se převede do stejné podoby jako v případě sériové komunikace dat z Video RAM (sériový modul bude nahrazen kamerovým modulem se stejným výstupem). Dalším nápadem bylo použití robotické ruky ke stimulaci tlačítek klávesnice testovaného zařízení.

Kapitola 3

Návrh řešení

Pro automatické testování se použije XML dokument z výstupu tohoto programu. Tento XML dokument je tedy hlavním pilířem celé aplikace, od něj se bude odvíjet návrh grafického uživatelského rozhraní (GUI) a schopností programu. Proto je tato kapitola zaměřena hlavně na detailní návrh struktury tohoto dokumentu.

3.1 Použité technologie

Zde uvedu a krátce popíši technologie, kterých využiji při řešení tohoto projektu.

3.1.1 Jazyk C# a .NET framework

C# (někdy též psáno jako C Sharp) je vysokoúrovňový objektově orientovaný jazyk vyvinutý firmou Microsoft jako součást platformy .NET framework. Jazyk byl později schválen standardizačními komisemi jako standard ECMA (ECMA-334) a ISO (ISO/IEC 23270). [4]

C# je jednoduchý, moderní, objektově orientovaný programovací jazyk s obecným zaměřením. Syntaxe vychází z jazyka C++ ale je hodně ovlivněna jinými programovacími jazyky jako je Delphi a Java. Typový systém jazyka C# používá automatickou správu paměti (podobně jako například jazyk Java). V jazyku C# je vše objektem (i instance typu `int` je objektem). V současné době je poslední verzí jazyka verze 3.0, který vyšel společně s .NET frameworkem 3.5 koncem roku 2007. [1]

3.1.2 Extensible Markup Language (XML)

XML je obecný značkovací jazyk, vyvinutý a standardizovaný konsorciem W3C[1]. XML je klasifikováno jako rozšiřitelný jazyk, protože dovoluje uživatelům definovat nové značky (tagy).

Hlavním účelem jazyka XML je sdílení strukturovaných dat mezi aplikacemi, zejména pomocí internetu, zakódování dokumentů a serializování dat (seřazení do bloku bajtů). Jazykem popisujeme strukturu dokumentu a obsah jednotlivých částí. Připojeným stylem se pak definuje vzhled dokumentu. V [5] najdete i podrobnější informace ohledně tohoto jazyka.

[1] World Wide Web Consortium (W3C), jejich cílem je „Rozvíjet World Wide Web do jeho plného potenciálu vývojem protokolů a směrnic které zajistí dlouhodobý růst Webu“

3.2 Názvosloví

V textu budu dále používat následující terminologie:

Entita Entita je nejmenším objektem na displeji. Entitou je například každé písmeno z nápisu `Control`, viz obrázek 3.1.

Segment Segment je složen z jedné nebo více entit. Po elektrické stránce odpovídá segment jednomu pinu displeje. Například nápis `Control` představuje jeden segment (obrázek 3.1).

Objekt Objekt vznikne seskupením jednoho nebo více segmentů do jednoho logického celku. Příkladem může být sedmisegmentové zobrazení číslice (viz obrázek 3.2) nebo segmenty obsahující dny v týdnu.

Grupa Grupa se skládá z jednoho nebo více objektů. Vhodným příkladem využití je sloučení dvou a více sedmisegmentových zobrazovačů (obrázek 3.2).

Displej Obrázek představující segmenty, které jsou umístěny a rozsvíceny shodně, jako by uživatel sledoval skutečný displej.

Layout displeje Vzorový obrázek displeje (obrázek 3.3) se všemi zobrazenými segmenty.

Video RAM Posloupnost bajtů, jejichž bitové vyjádření značí stav rozsvícení jednotlivých segmentů na displeji.

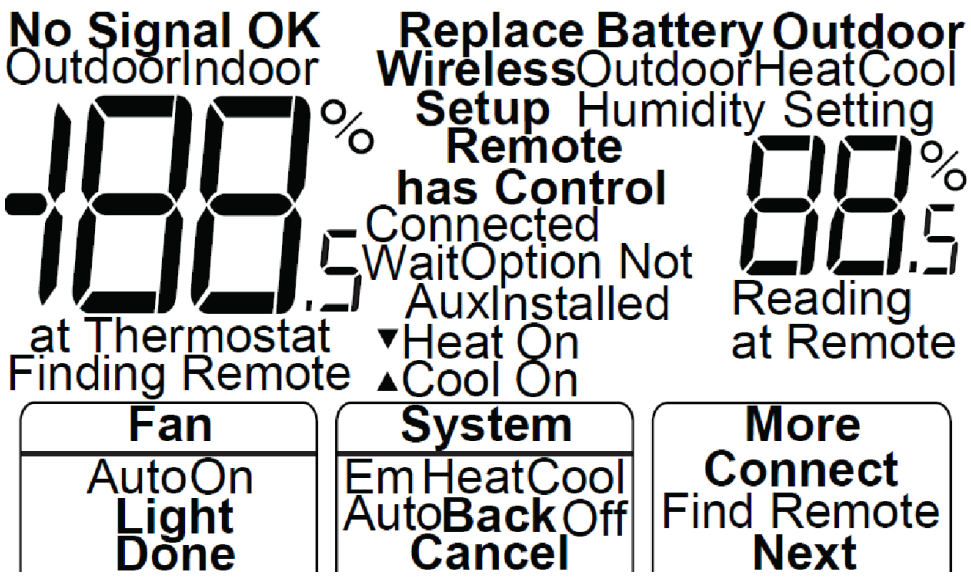
Počátek souřadného systému v bitmapě Počátek souřadného systému je definován v levém horním rohu bitmapy. Je to nejtypičtější řešení v počítačové 2D grafice a tudíž to bude jednodušší i z hlediska implementace. Na obrázku 3.4 vidíme, jak to bude vypadat v praxi.

Obrázek 3.1: Entita a Segment

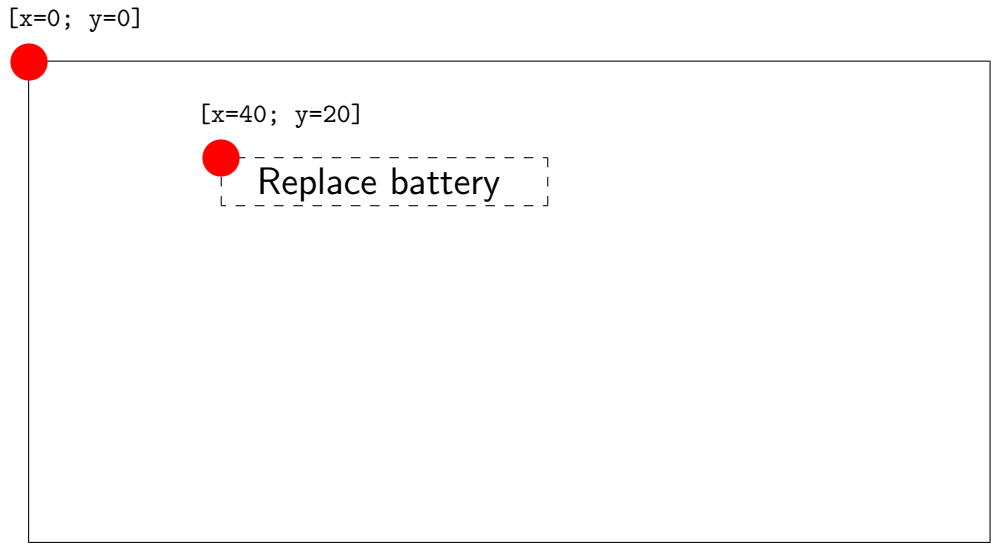
Obrázek 3.2: Objekt a Grupa

3.3 Struktura XML souboru

XML soubor vyhovuje standardu XML verze 1.0 a je kódován dle standardu UTF-8.



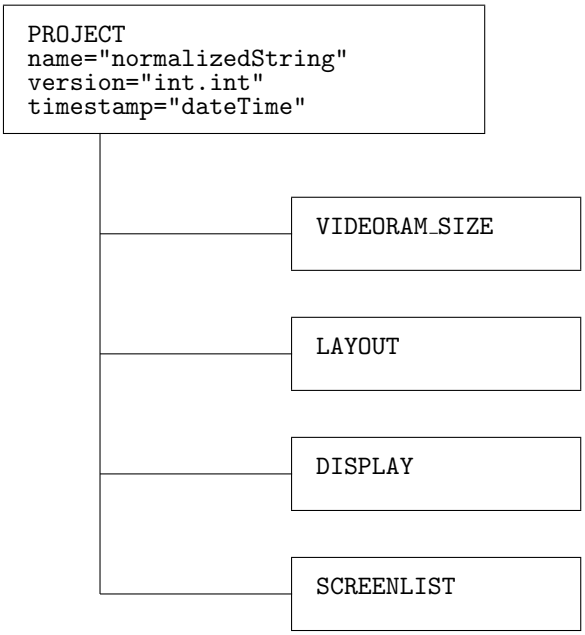
Obrázek 3.3: Vzorový layout displeje



Obrázek 3.4: Použitý souřadný systém

3.3.1 Kořen

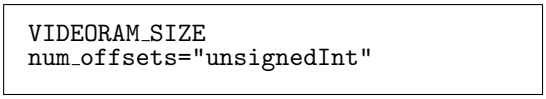
Kořenový element ⟨project⟩ obsahuje tyto atributy: jméno projektu, verze XML souboru a časové razítko poslední úpravy XML souboru. Verze XML souboru je uložena ve formátu X.Y, kde X je číslo hlavní verze XML souboru a je inkrementováno při změně layoutu displeje; Y je číslo vedlejší verze a je inkrementováno se změnou definic segmentů, objektů, skupin a obrazovek provedené na aktuálním layoutu displeje. Časové razítko je uchováno ve formátu YYYY-MM-DD_hh:mm:ss. Na obrázku 3.5 je graficky znázorněno hierarchické schéma kořenu.



Obrázek 3.5: Kořenový prvek

Velikost Video RAM

Zde jsou uloženy vlastnosti paměti Video RAM. Je zde uložena její velikost v podobě počtu offsetů (=bajtů). Tento tag je znázorněn obrázkem 3.6.

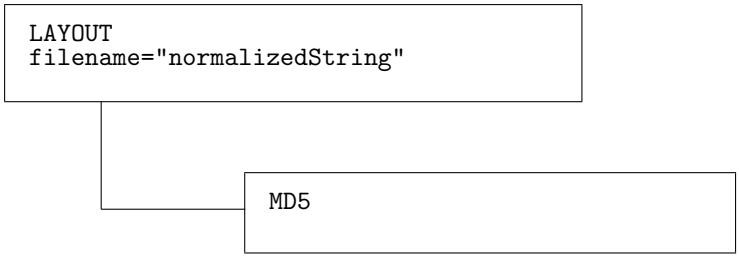


Obrázek 3.6: Struktura Video RAM

Bitmapa layoutu displeje

V párové značce ⟨layout⟩, umístěné hierarchicky pod kořenovou značkou ⟨project⟩ a následující bezprostředně za značkou ⟨videoram_size⟩, je uchován název souboru s designem displeje a kontrolní součet tohoto souboru. Kontrolní součet je typu MD5. Před

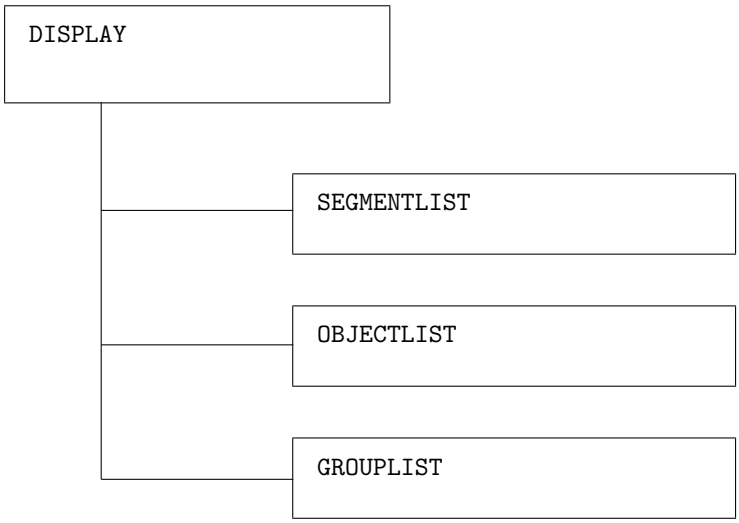
spuštěním programu se pomocí kontrolního součtu ověří, zda daný XML soubor náleží k aktuálnímu layoutu displeje. Pokud kontrolní součty souhlasí, program pro skládání obrazovek může použít nacachované objekty pro jednotlivé segmenty. V případě rozdílného kontrolního součtu je uživatel upozorněn na nesouhlasné verze souborů a kontrola správnosti segmentů a jejich případná úprava je jen na něm. Viz obrázek 3.7



Obrázek 3.7: Bitmapa displeje

Hierarchické schéma definice displeje

V této části je popsán layout displeje, viz obrázek 3.8. Definice obsahuje segmenty, objekty a skupiny.



Obrázek 3.8: Hierarchické schéma definice displeje

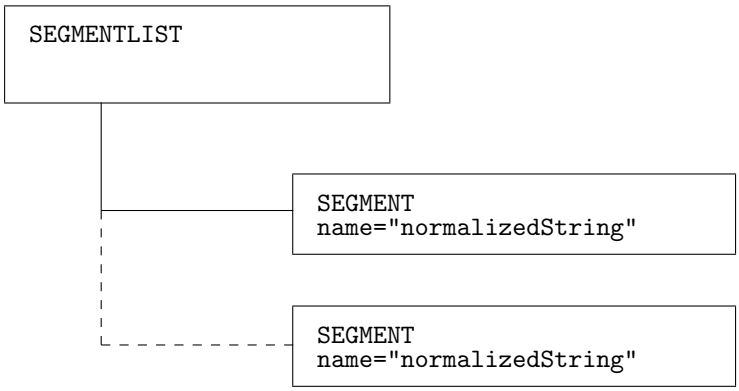
Seznam obrazovek

Seznam obrazovek a obrazovky samotné jsou v textu popsány dále. Konkrétně se jimi zabývá podkapitola na straně 19.

3.3.2 Segmenty

Seznam segmentů

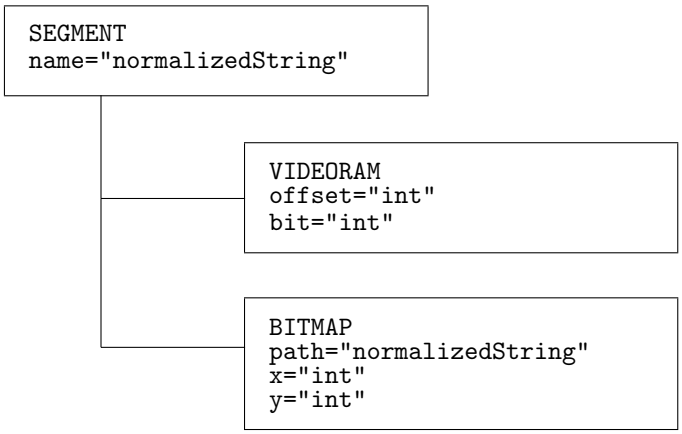
V této sekci jsou definovány všechny segmenty, které displej obsahuje. Viz obrázek 3.9.



Obrázek 3.9: Seznam segmentů

Segment

Obrázek 3.10 znázorňuje hierarchii segmentu. Značka segmentu je `⟨segment⟩`. Segment je definován svým jménem. Jméno segmentu musí být v celém projektu jedinečné, smí být utvořeno z malých a velkých písmen bez diakritiky, číslic, podtržítek a pomlček. Uvnitř definice segmentu je sekce `⟨videoram⟩`, kde je definována jeho fyzická pozice ve video paměti. Dále segment má sekci `⟨bitmap⟩`, kde je určena cesta k souboru s cache (bitmapou) daného segmentu a souřadnice s umístěním segmentu na displeji. Cesta musí být zadána v relativním formátu vzhledem k adresáři projektu.

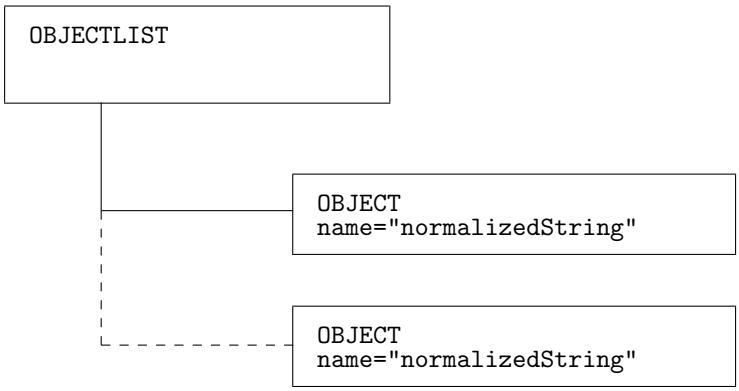


Obrázek 3.10: Segment

3.3.3 Objekty

Seznam objektů

V této sekci, značené tagem ⟨objectlist⟩, jsou definovány objekty. Tato sekce není povinná, definici objektů lze vynechat. Obrázek 3.11 graficky znázorňuje tuto sekci.



Obrázek 3.11: Seznam objektů

Objekt

Objekt se skládá z jednoho a více segmentů (⟨segmentlist⟩), mají definované možné stavy (⟨variantlist⟩) a možné animace (⟨sequencelist⟩). Je definován svým jménem, které musí být v celém projektu jedinečné, podobně jako u segmentu. Hierarchie objektu je znázorněna na obrázku 3.12.

Výčet variant

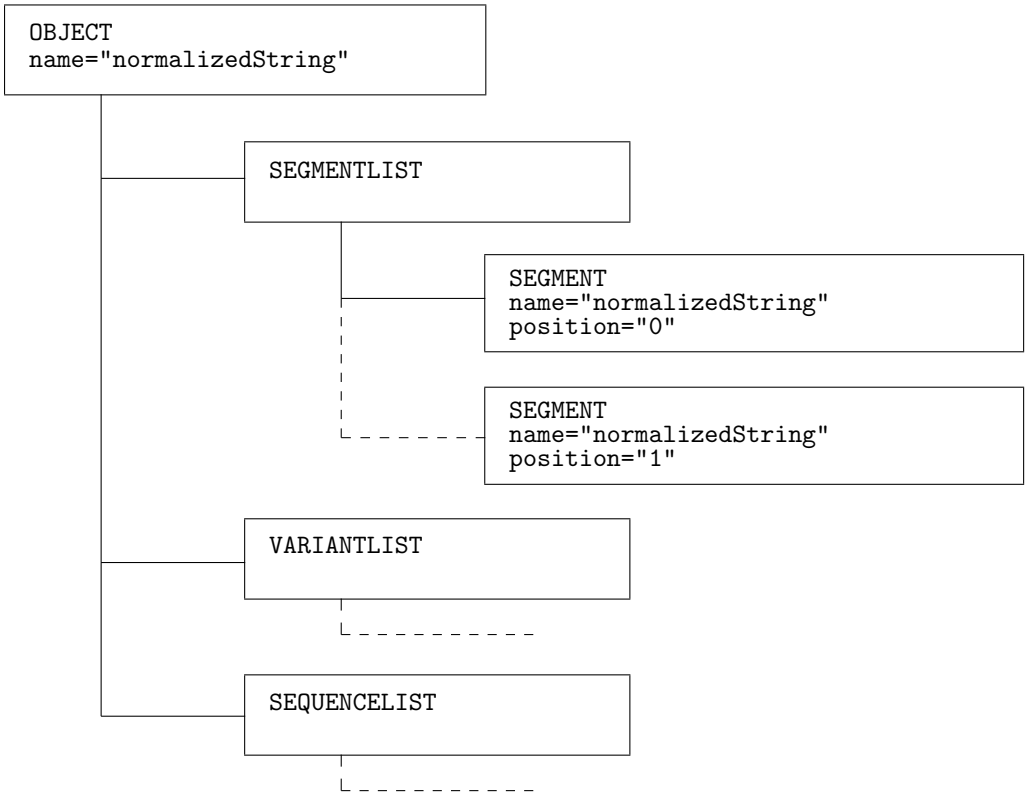
Značka výčtu ⟨variant⟩ je ve struktuře umístěna uvnitř definice objektu. Na obrázku 3.13 vidíme strukturu seznamu variant. Definuje seznam možných kombinací segmentů, které představují daný objekt. Příkladem může být sedmisegmentové zobrazení číslice. Jednotlivé segmenty sedmisegmentového zobrazovače je vhodné definovat v zažitém pořadí, tj. nultý segment (číslováno od 0) bude představovat segment a, první segment bude představovat segment b, atd. Rozestavení jednotlivých segmentů naleznete na obrázku 3.14. V dalších verzích programu se počítá se šablonami u variant, u těchto šablon bude záležet na pořadí jednotlivých segmentů, tak by bylo dobré, si na toto značení zvykat už teď.

Sekvence

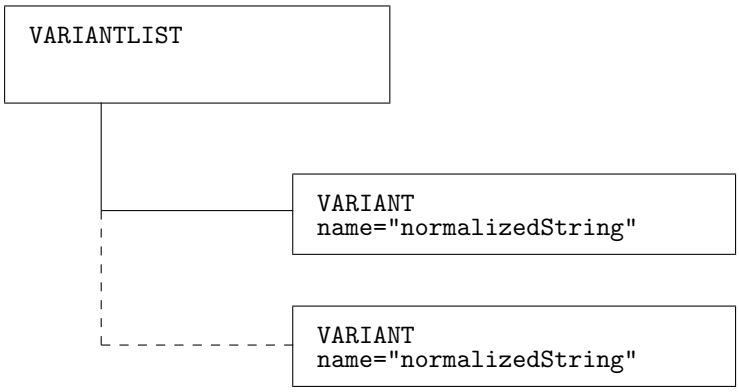
Sekvence (značeno tagem ⟨sequence⟩) jsou součástí objektu. Pomocí sekvencí se definují jednotlivé animace, jako je například cyklické blikání nápisu. Struktura sekvencí je uvedena na obrázku 3.15.

Dodatek k objektům

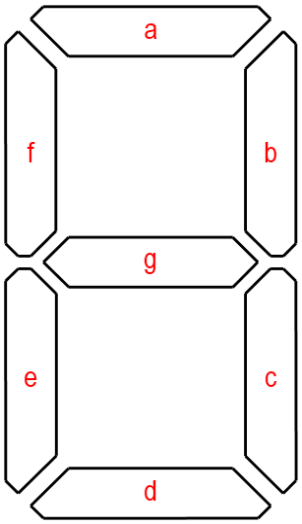
Do objektu sedmisegmentového zobrazovače není vhodné zahrnovat desetinnou čárku (ani žádné další znaky jako např. ° nebo %), protože při vkládání jednotlivých variant, kterých



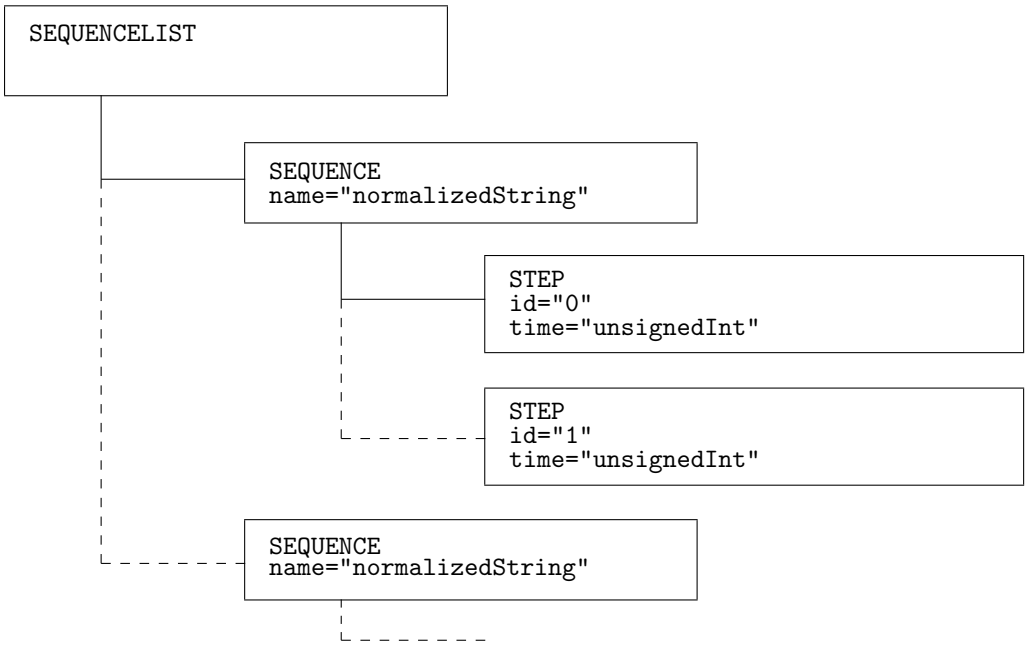
Obrázek 3.12: Objekt



Obrázek 3.13: Seznam variant



Obrázek 3.14: Sedmisegmentový zobrazovač



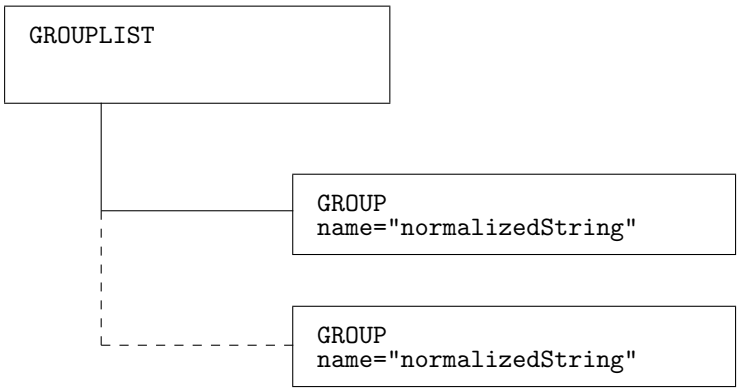
Obrázek 3.15: Seznam sekvencí

objekt může nabývat, by uživatel musel zadat celou sadu možností s desetinným oddělovačem a tutéž sadu možností bez desetinného oddělovače. Zdvojnásobil by se tak počet potřebných variant. Při testu se pak bude skript dotazovat na desetinnou čárku jako na samostatný segment, nebo na objekt (o jednom segmentu) jako součást grupy.

3.3.4 Grupy

Seznam grup

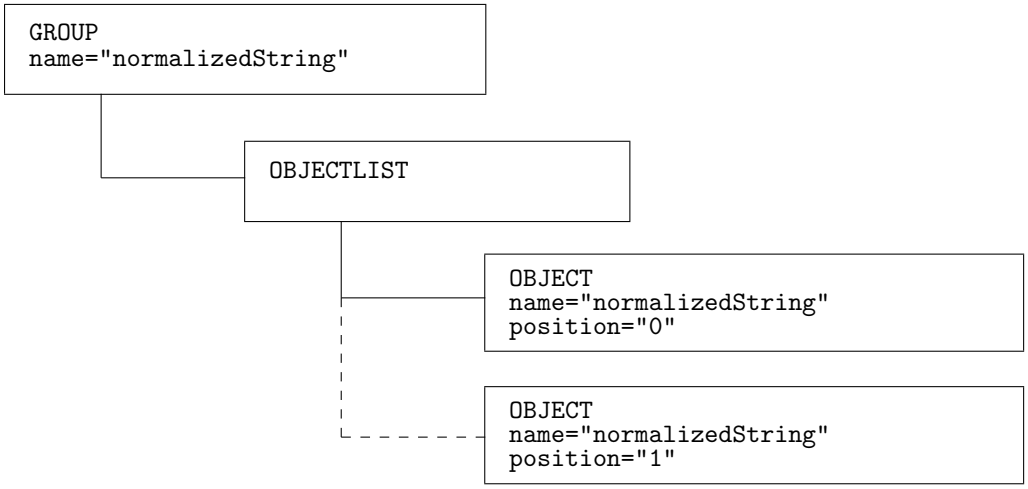
V sekci ⟨grouplist⟩ jsou definovány skupiny objektů (tzv. grupy). Tato sekce není povinná. Grafické znázornění naleznete v obrázku 3.16.



Obrázek 3.16: Seznam grup

Grupa

Na obrázku 3.17 je znázorněna struktura grupy. Grupa představuje skupinu objektů. Je definována svým jménem a pro definici jména platí stejné podmínky jako u jména segmentu. Vhodným příkladem grupy je skupina objektů sedmisegmentového zobrazovače.

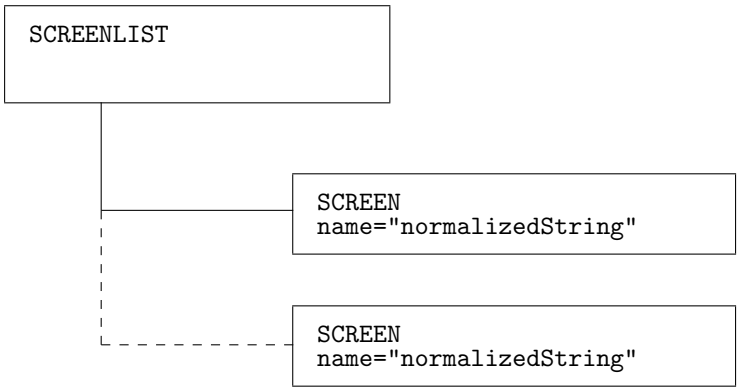


Obrázek 3.17: Grupa

3.3.5 Obrazovky

Seznam obrazovek

Seznam obrazovek `⟨screenlist⟩` obsahuje definice jednotlivých obrazovek. Díky využití definic těchto obrazovek je možné podstatně zjednodušit a zkrátit testovací skript.



Obrázek 3.18: Seznam obrazovek

Obrazovka

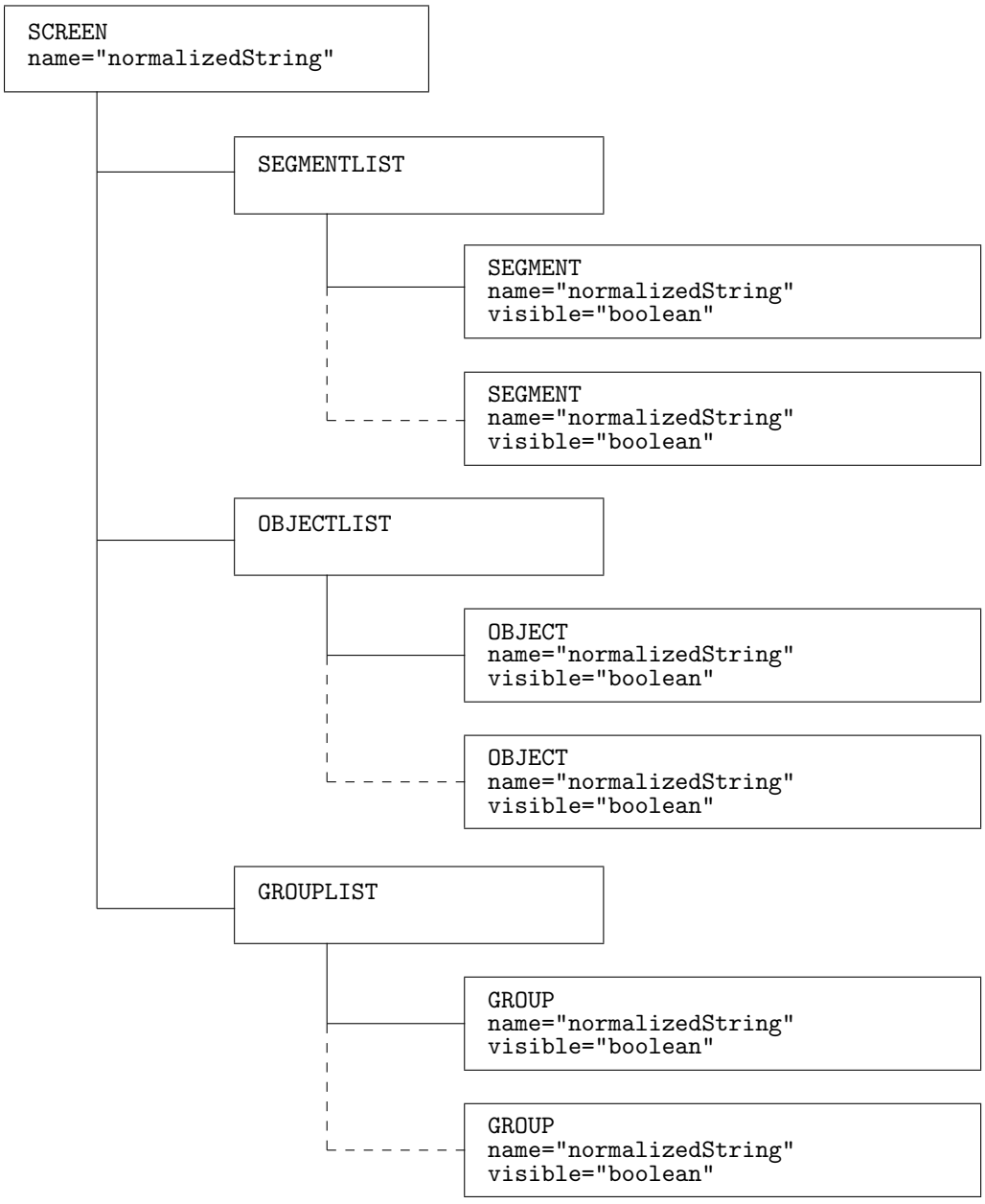
Struktura obrazovky je graficky znázorněna na obrázku 3.19. V obrazovce jsou uvedeny segmenty (i objekty a grupy), které v ní musí permanentně svítit a ty, které v ní za žádných okolnostech svítit nesmí. Ostatní segmenty můžou nabývat v dané obrazovce různých hodnot – na testování obrazovky nebudou mít vliv. Při testování zařízení se můžeme dotazovat, zda-li aktuální stav Video RAM odpovídá požadované obrazovce (jestli svítí všechny segmenty, které v dané obrazovce mají svítit a jestli nesvítí segmenty, které zde svítit nesmí).

3.4 Hierarchie projektu

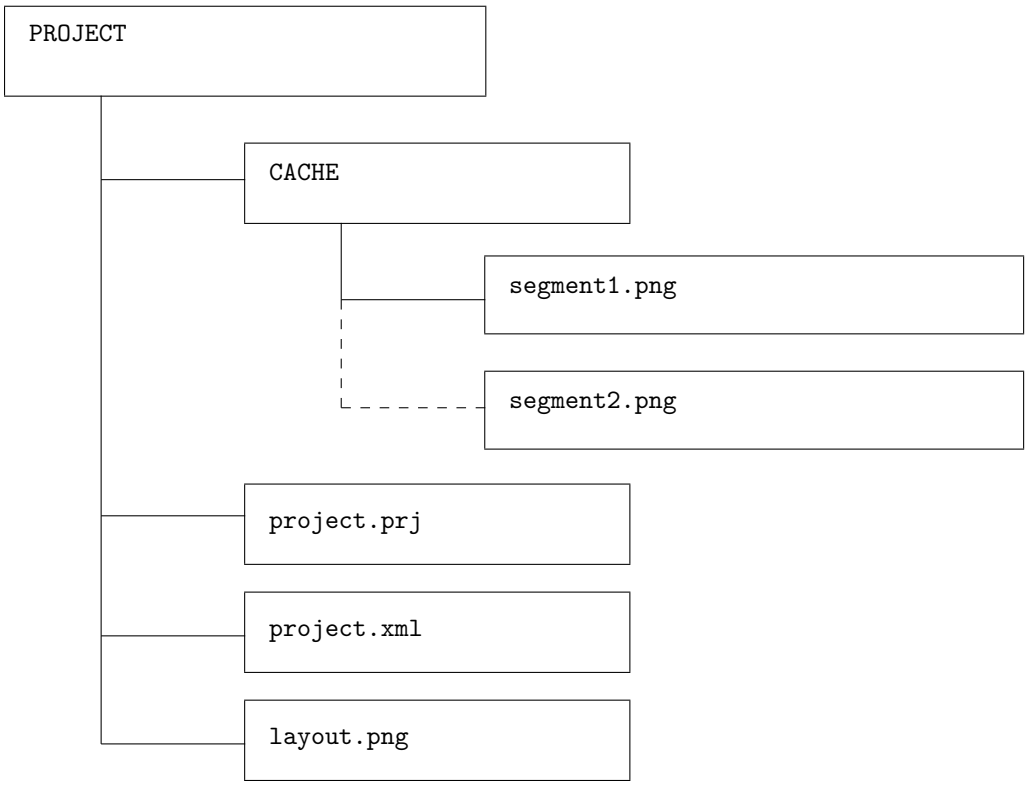
Každý projekt musí obsahovat XML popis layoutu displeje (ten posléze bude sloužit jako vstup pro modul analyzátoru displeje) a obrázek layoutu displeje. Soubor s koncovkou `.prj` bude obsahovat jméno XML souboru a bude sloužit primárně jako soubor projektu, na který se budeme odkazovat při otevírání již existujícího projektu. Do budoucna tento soubor může uchovávat další informace, jako je například specifické nastavení aplikace, pozice a velikost oken a mnoho dalších. K projektu budou ukládány i jednotlivé obrázky segmentů, které budou uloženy v podadresáři `cache`. Proto jsem zvolil jednoduchou adresářovou strukturu pro ukládání projektu. Obrázek 3.20 tuto hierarchii znázorňuje.

3.5 Převod obrázku na černo-bílý

V programu potřebujeme pracovat s dvoubarevnou bitmapou. Co když ale uživatel programu předá obrázek s vyhlazenými okraji? Podobné situace se mohou vyskytovat a tak si obrázek převedeme pomocí matice do odstínů šedi. Tato operace je velice rychlá, takže uživatel nebude zdržován zbytečným čekáním, jako by tomu bylo kdybychom obrázek



Obrázek 3.19: Obrazovka



Obrázek 3.20: Hierarchie projektu

převáděli bod po bodu. Matice pro převod obrázku do odstínů šedi má následující tvar 3.1:

$$M = \begin{pmatrix} 0.299 & 0.299 & 0.299 & 0 & 0 \\ 0.587 & 0.587 & 0.587 & 0 & 0 \\ 0.114 & 0.114 & 0.114 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

Hodnoty v matici jsou získány ze vztahu 3.2 [2]:

$$I \ = 0.299R + 0.587G + 0.114B \tag{3.2}$$

Aplikací matice 3.1 na obrázek dostaneme obrázek ve stupních šedi. K získání požadovaného černo-bílého obrázku využijeme metodu prahování. Prahování 3.3 upravuje hodnoty jasu jednotlivých obrazových bodů (vzorec převzatý z [2]):

$$G(x,y) = \{ \begin{matrix} 0 & \text{pro} & I(x,y) < T \\ 1 & \text{pro} & I(x,y) \geq T \end{matrix} \tag{3.3}$$

Kde G je výstupní bod, I vstupní bod a T prahová hodnota.

3.6 Označování segmentů

Segmenty můžou mít různé tvary a rozměry, proto bude vhodné k jejich označování použít metodu semínkového vyplňování. Pro šikmé čáry o šířce jednoho pixelu bude nutné použít osmi-cestné semínkové vyplňování. Může se také stát, že se budou dotékat dva spolu nesouvisející segmenty a je nutné je označit každý zvlášť. Tento případ vyřeším pomocí vyplňování pixelů segmentu jen v oblasti výběru obdélníkem.

3.6.1 Sloupcové semínkové vyplňování se zásobníkem

Sloupcové semínkové vyplňování, inspirované z [3], budu využívat pro 4-cestné a jeho upravenou variantu pro 8-cestné vyplňování (označování) segmentů. Také ho využiji při Area FloodFill (plošném 4-cestném semínkovém vyplňování), kde uživatel označí obdélníkem oblast, pro kterou se provede v jednotlivých entitách označení právě tímto typem semínkového vyplňování.

4-cestná varianta

Tato implementace využívá dvou booleovských proměnných `spanLeft` a `spanRight` k zapamatování, jestli testované pixely na levé nebo pravé straně jsou součástí nového řezu nebo jsou-li již uloženy v zásobníku. Do zásobníku se tedy přidávají jen potřebné body pro další řezy. Následující pseudokód s komentáři tento algoritmus dostatečně popisuje.

```
void FloodFill4Scanline(int x, int y, int newColor, int oldColor)
{
    // nemáme co obarvovat?
    if (newColor == oldColor) return;

    // do zásobníku vložíme první bod
```

```
    Stack.Clear();
    Stack.Push(x, y);

    // pomocné proměnné
    int y1;
    bool spanLeft, spanRight;

    // zpracováváme body ze zásobníku, dokud není prázdný
    while (Stack.Pop(x, y))
    {
        spanLeft = spanRight = false;

        // nalezneme nejvyšší bod ve zpracovávaném sloupci
        y1 = y;
        while (y1 >= 0 && Image[x][y1] == oldColor)
            --y1;
        ++y1;

        // zpracováváme body sloupce od vrchu až dokud nenarazíme na
        // spodní okraj obrázku nebo na pixel jiné barvy než oldColor
        while (y1 < h && Image[x][y1] == oldColor)
        {
            Image[x][y1] = newColor;    // přebarvíme aktuální bod

            // přidáme jeden bod levého řezu do zásobníku
            if (!spanLeft && x > 0 && Image[x-1][y1] == oldColor)
            {
                Stack.Push(x-1, y1);
                spanLeft = true;
            }
            // řez skončil, v dalším kroku budeme hledat další řez
            else if (spanLeft && x > 0 && Image[x-1][y1] != oldColor)
                spanLeft = false;

            // přidáme jeden bod pravého řezu do zásobníku
            if (!spanRight && x < w-1 && Image[x+1][y1] == oldColor)
            {
                Stack.Push(x+1, y1);
                spanRight = true;
            }
            // řez skončil, v dalším kroku budeme hledat další řez
            else if (spanRight && x < w-1 && Image[x+1][y1] != oldColor)
                spanRight = false;

            ++y1;    // přechod na následující bod sloupce
        }
    }
}
```

8-cestná varianta

8-cestná varianta semínkového vyplňování dokáže vyplnit i šikmé čáry, stačí když se dotýkají jednotlivé pixely alespoň svými rohy. Tento algoritmus vychází ze 4-cestné varianty, oproti ní však musí testovat u každého zpracovávaného sloupce další 4 body (levý a pravý horní a levý a pravý spodní).

Kapitola 4

Implementace a testování

Z hlediska implementace je program rozdělen do následujících částí:

- Třída s XML daty projektu.
- Rodičovské okno celé aplikace.
- Okno LCD designeru.
- Okno LCD monitoru.

Podrobný popis jednotlivých tříd a jejich závislostí naleznete v programové dokumentaci na přiloženém CD.

4.1 Třída XML projektu

Třída `XML_Project` je pilířem celé aplikace, jednotlivé části aplikace v podstatě jen zprostředkovávají grafické uživatelské rozhraní (GUI) k instanci této třídy. Třída se mimo jiné stará i o vygenerování potřebné adresářové struktury projektu. Většina podtříd byla implementována s ohledem na použití v komponentě `PropertyGrid` pro snadnou editaci jejich vlastností. Všechny třídy XML dokumentu jsou také připraveny pro serializaci dat, která se využívá například při ukládání dat do souboru.

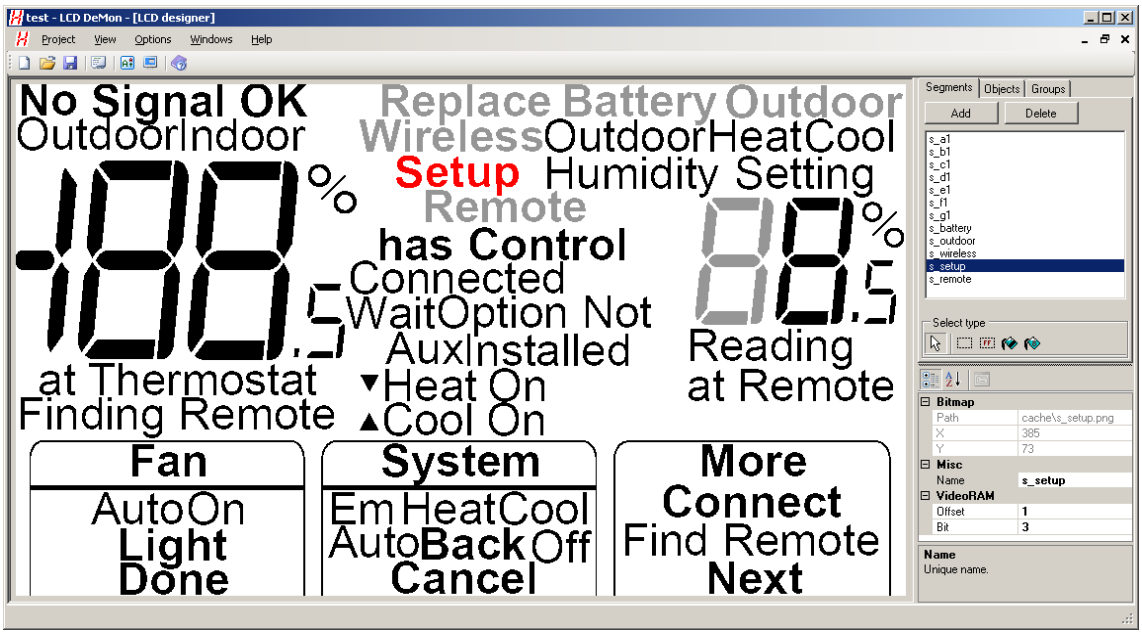
4.2 Rodičovské okno aplikace

Toto okno uchovává aktuální instanci třídy `XML_Project` a obsluhuje její základní metody, jako například založení nového projektu, otevření již existujícího projektu, atd. Také se stará o zobrazování podoken LCD designeru a LCD monitoru a o dialogové okno, které zobrazuje informace o aktuálním projektu (datum poslední změny, verze projektu, informace o obrázku layoutu displeje, počty definovaných segmentů, objektů, grup, velikost aktuální zabrané Video RAM, apod.), Z tohoto rodičovského okna je také voláno okno se stručným uživatelským manuálem k této aplikaci.

4.3 LCD designer

LCD designer slouží k definici prvků layoutu displeje, tj. definici jednotlivých segmentů, objektů a grup (skupin objektů). Na pravé straně okna je panel se záložkami, každá záložka

slouží k editaci jiného prvku displeje. U definice segmentů je nutné označit v obrázku displeje daný segment a přiřadit mu správnou hodnotu ve Video RAM. Všechny neúplně zadané segmenty jsou vykresleny tučným řezem písma, aby bylo na první pohled vidět, kde něco chybí. Tento styl zvýraznění je použit i u definice objektů i grup. Objekty se nenaklikávají přímo v obrázku displeje, ale pomocí dvou seznamů. Levý seznam představuje dostupné segmenty a pravý segmenty v editovaném objektu. Potom je možné definovat platné hodnoty stavu objektu pomocí variant editoru, viz. následující podkapitola Variant editor. Pokud má mít objekt nějakou animaci, například blikání, definujeme tuto animaci pomocí sekvence editoru popsaného v podkapitole Sekvence editor. Varianty ani sekvence nejsou povinné, ale pokud budou definovány, musí být dodrženo pořadí postupu, v jakém to zde bylo uvedeno. Takže nejdříve přidání segmentů do objektu, pak varianty a nakonec sekvence. Toto pořadí je důležité proto, protože varianty jsou přímo závislé na segmentech v objektu a sekvence jsou zase přímo závislé na variantách. Program si pořadí definice ohlídá. Definice grup se provádí také pomocí dvou seznamů, podobně jako u definice objektů, s tím rozdílem, že grupy nemají varianty ani sekvence.



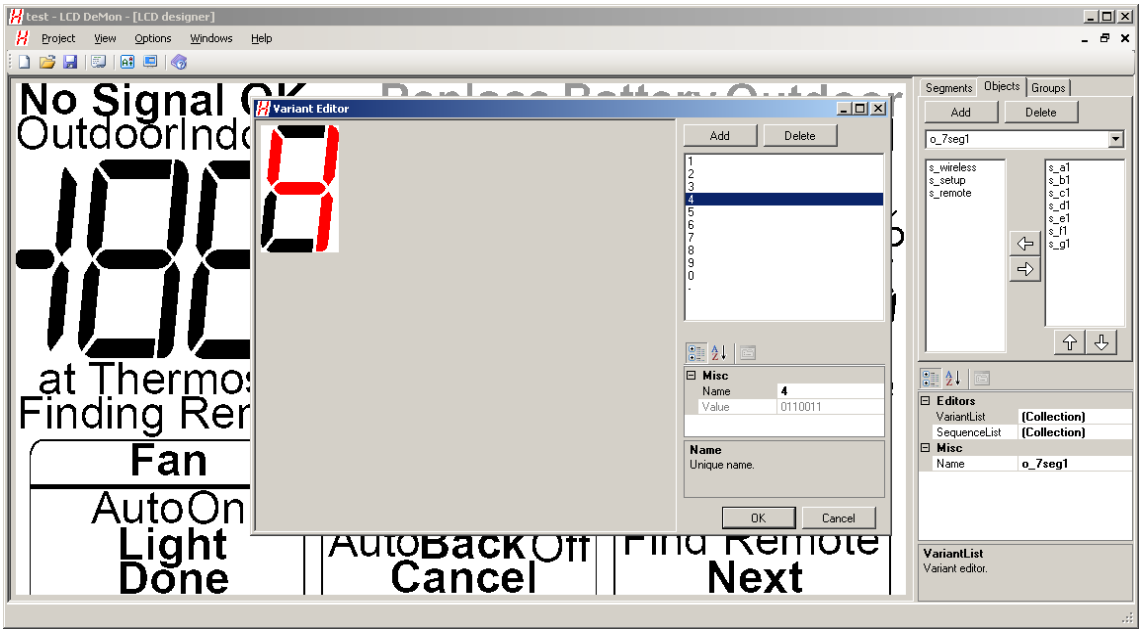
Obrázek 4.1: Snímek LCD Designeru

Variant editor

V tomto editoru se definuje seznam přípustných stavů editovaného objektu, tj. všech hodnot, kterých může daný objekt nabývat.

Výběr segmentů pro definovanou variantu se provádí přímo klikáním myší na jednotlivé segmenty.

Na obrázku 4.2 je vzorový snímek tohoto editoru při definování sedmisegmentového zobrazovače.



Obrázek 4.2: Snímek editoru variant

Sequence editor

Tento editor slouží k definici animací (blikání, . . .). Animace je posloupnost po sobě jdoucích variant. Jeden objekt může mít definováno více animací, analyzátor displeje v testovacím skriptu pak podle posledních několika stavů Video RAM určí, o kterou konkrétní sekvenci (animaci) se jedná.

Definování segmentů se provádí přesunem vybraných variant ze seznamu variant do seznamu aktuální animace. Grafické okno slouží pro názornost při definovaní a hlavně pro přehrávání definované animace (lze tak zkontrolovat výslednou animaci).

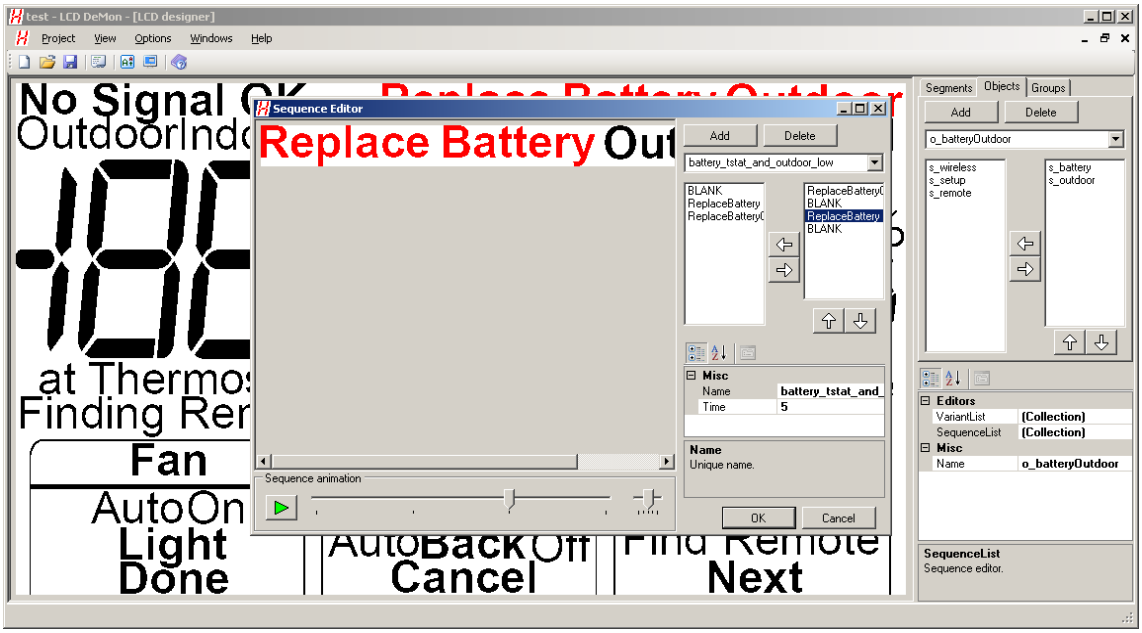
Obrázek 4.3 ukazuje editor sekvencí při práci.

4.4 LCD monitor

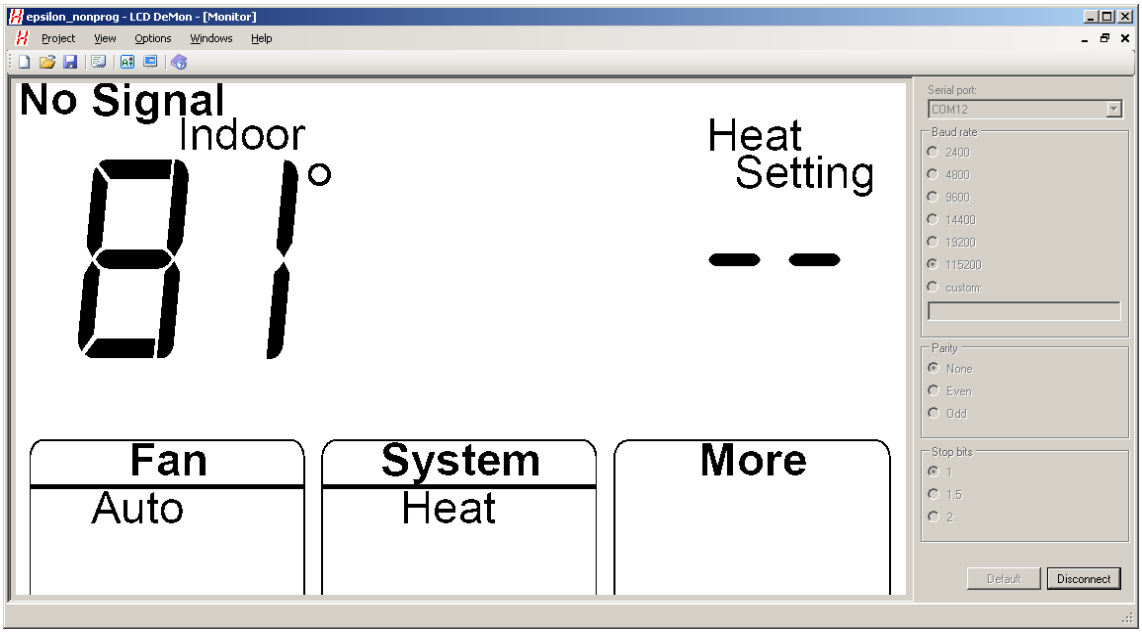
Toto okno slouží k zobrazování aktuálního stavu Video RAM zařízení na obrazovce počítače. Při otevření LCD monitoru se načte aktuální XML projekt z rodičovského okna. Před připojením k zařízení (přes sériový nebo virtuální sériový port) si okno vezme aktuální stav projektu rodičovského okna a po dobu připojení k testovanému zařízení bude zakázána editace v okně LCD designeru.

LCD monitor pracuje pouze se segmenty XML projektu a velikostí potřebné Video RAM, ostatní data jako objekty a grupy se zde nevyužívají. Jednotlivé bity v získané video paměti představují jednotlivé segmenty. LCD monitor tyto bity vyhodnotí a dle toho zobrazí patřičné segmenty v obrázku displeje. O nic víc se nestará, jen interpretuje přijatá data.

Na obrázku 4.4 je zobrazen náhled spuštěného LCD monitoru připojeného ke vzorovému testovanému zařízení.



Obrázek 4.3: Snímek editoru sekvencí



Obrázek 4.4: Snímek LCD monitoru

Kapitola 5

Závěr

Prvním krokem byl návrh struktury výstupního XML dokumentu. Návrh byl průběžně konzultován se zástupci firmy Honeywell, aby výsledná aplikace splňovala veškeré požadavky. Dle návrhu popsaného v kapitole 3 byli definovány třídy, z nichž každá popisuje jednotlivý element hierarchie XML dokumentu, jako například `Seznam segmentů`, `Segment`, `Objekt`, `Varianta`, `Sekvence`, atd. Tyto třídy jsou definovány tak, aby na vyžádání samy generovaly ze svého obsahu XML kód a aby je bylo možné modifikovat v komponentě `PropertyGrid`. Poté bylo navrženo grafické uživatelské rozhraní (GUI) jednotlivých oken aplikace. Byl zvolen takový styl, že je jedno rodičovské okno a to se stará o jednotlivá podokna (`LCD designer` a `LCD monitor`). Sloučením názvu `LCD designer` a `LCD monitor` vznikl název aplikace – `LCD DeMon`. V `LCD designeru` je pro editaci vlastností jednotlivých položek použita komponenta `PropertyGrid`. Data segmentů se označují přímo v obrázku displeje, jinak se povětšinou přesouvají položky mezi dvěma seznamy. `LCD monitor` byl přidán navíc mimo zadání. Jeho účel je prostý, vizualizovat přijatá obrazová data od připojeného zařízení. Podrobněji byla okna `LCD designeru` a okno `LCD monitoru` popsána v kapitole 4.

Tento program byl vyvíjen na požadavky společnosti Honeywell, kde bude také tento program použit v praxi.

V dalších verzích programu se počítá i přímo s definicí jednotlivých obrazovek, kde se definují segmenty, objekty a grupy, které v dané obrazovce nesmí nebo naopak musí být aktivní. Návrh i implementace třídy XML projektu s tím již počítala, proto jsou zde již potřebné metody a třídy implementovány a zdokumentovány. Při rozšiřování aplikace stačí tedy jen dodělat patřičné GUI pro definice těchto obrazovek. Analyzátor displeje se pak bude ptát, zda-li aktuální stav video paměti odpovídá dané obrazovce. Tím se ještě podstatně zjednoduší testovací skript. U Variant editoru přibudou šablony předdefinovaných variant objektů, protože některé opakující se objekty, jako např. sedmisegmentové displeje, je rychlejší a pohodlnější mít již předdefinované, než je pokaždé definovat znovu. Je zde také menší riziko chyby. Uživatel si bude moci vytvářet i vlastní šablony. Dalším plánovaným rozšířením by měla být podpora pro dot-matrix oblasti displeje. Dot-matrix oblast je oblast displeje tvořená maticí bodů, tato oblast displeje může zobrazovat téměř jakékoli tvary. Protože layout displeje prochází při vývoji mnoha změnami, je v plánu také podpora jednoduché změny obrázku displeje, kde bude pomocí průvodce provedena změna layoutu, modifikováno umístění segmentů, atd. V poslední době se začínají hodně používat dotykové displeje, tzv. touchscreeny. Podpora pro definice těchto oblastí je taky plánována v dalších verzích.

Literatura

[1] Archer, T.: *Myslíme v jazyku C#*. Grada Publishing, 2002, ISBN 80-247-0301-7.

[2] Kršek, P. a Španěl M.: Základy Počítačové Grafiky – Redukce barevného prostoru. Vysoké Učení Technické v Brně – Fakulta Informačních Technologií [slajdy], 2007.

[3] Vandevenne, L.: Lode's Computer Graphics Tutorial – FloodFill. [online], [cit. 2009-04-20].
URL http://student.kuleuven.be/~m0216922/CG/floodfill.html

[4] Wikipedia: C Sharp (programming language). [online], [cit. 2009-01-13].
URL http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[5] Wikipedia: XML. [online], [cit. 2009-04-05].
URL http://en.wikipedia.org/wiki/XML

Příloha A

Obsah CD

Přiložené CD obsahuje následující adresáře a soubory:

- `/bachelor_thesis`
  - `/bachelor_thesis_src` – zdrojové kódy technické zprávy
  - `bachelor_thesis.pdf` – elektronická verze textu bakalářské práce
  - `bachelor_thesis_print.pdf` – elektronická verze bakalářské práce pro tisk
- `/implementation`
  - `/_drivers` – ovladače a knihovny potřebné pro běh programu
  - `/doc` – programová dokumentace ve formátu HTML
  - `/epsilon_nonprog` – vzorový projekt pro zařízení Epsilon Non-programmable
  - `/layouts` – vzorové obrázky layoutu displeje
  - `/lcd_demon_src` – zdrojové kódy programu
  - `/manual_src` – zdrojové kódy uživatelského manuálu
  - `lcd_demon.exe` – spustitelná release verze programu
  - `lcd_demon.chm` – uživatelský manuál k programu
- `INSTALL.txt` – popis instalace
- `README.txt` – popis obsahu CD