DEVELOPMENT OF A LOW COST INTELLIGENT TOOL FOR LITERACY TEACHING AND LEARNING

Sorin Nicola, Radu Solea, Ionut Dinulescu,
Dan Dobriceanu, Romeo Tonca

University of Craiova,
Faculty of Automation, Computers and Electronics

Abstract: The tool described in this paper consist in an embedded system based on a cost effective thin-client PC (the e-Box) and running a software developed in Windows CE environment, using .NET compact framework and C# as programming language. External hardware used with e-Box is: a graphic tablet as pointing device, a VGA to TV converter for using a standard TV set as display, an USB memory stick and speakers. The application is developed for Romanian language but there is also provided necessary building blocks and interfaces to help other developers extend the platform for their own language and culture.

Keywords: embedded systems, educational aids, teaching, learning, character recognition.

1. INTRODUCTION AND BACKGROUND

Illiteracy is still a black hole in modern-day society, with around one billion people all over the world unable to read or write. In advanced societies, illiteracy is closely linked to social exclusion, to shame, social immobility and poverty (Blake and Blake, 2002).
As starting point was the Romanian example of Roma people minority, whose illiteracy level is notorious, illiteracy that, among other causes, is feeding a real poverty cycle. A World Bank study (Ringold *et al.*, 2003) states that compared to the estimated national illiteracy rate of 2 to 4 percent (Romanian Ministry of Education, 1998), 44 percent of Roma men, and 59 percent of Roma women were illiterate. This is also true for other more developed countries. Another example in the same study, referring to Spain states that, even if Spain is EU member for a long time, illiteracy levels for adult Roma are high, with rates approaching 70 percent. For the population over the age of 55, illiteracy rates for men and women are around 75 percent and 90 percent, respectively. In Portugal, almost half of the population can barely read and write. Even in Germany, 14 per cent of the country's population is stated to have severe difficulties with reading and writing (Deutsche Welle, 2002).
Due to lack of resources and limited knowledge of available technology, adult literacy programs in Romania have only recently begun to use IT for instruction. The types of technology that can be applied in adult literacy programs range from computer-assisted instruction to the use of audio and video recordings and the Internet (Withrow, 2003). All these technologies will enhance adult instruction and learning by benefits as:

- removing or diminishing barriers to participation, including time and distance
- extending instruction into life contexts, like home and workplace
- providing access to a wide range of resources and materials for instruction
- facilitate learning for minorities with special needs and differences.

The poverty cycle can be broken and by literacy learning at early ages but also at adult age (Thomas,

1989). A practical IT solution proposed by the authors is as an intelligent Literacy Tool (called by the authors iLIT).
The tool consist in a workspace based on a cheap thin-client (e-Box) and a software developed in a Windows CE environment using .NET framework and C# as programming language (Sharp and Jagger, 2002; Muench C., 2000; Microsoft, 2005). External hardware used is: a graphic tablet as pointing device, a VGA to TV converter for using a standard TV set as display, an USB memory stick and audio speakers. Main features are:

- Friendly user interface based on voice messages and easy to read cultural suggestive icons and menus
- Multi-lingual support (for minority learner)

The tool was thought as an affordable and robust alternative to expensive Tablet or Slate PCs, keeping in mind that hundred of years the ancient slate tablet was used to learn reading and writing.

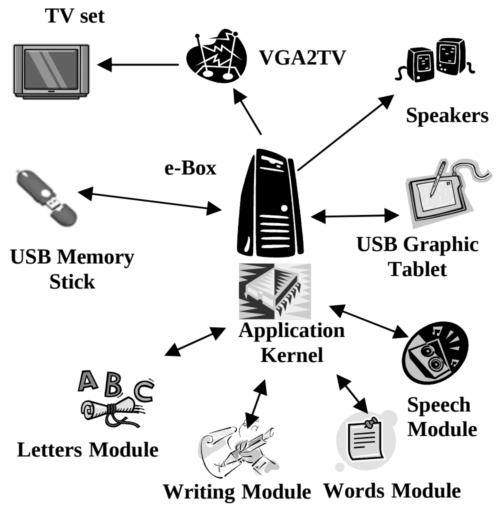

Fig. 1 System overview

The application was developed for Romanian language, but in order for the project to achieve its overall goal of creating a stand-alone platform for fighting illiteracy, an effort was made to provide also the necessary building blocks and interfaces to help other developers extend the platform for their own language and culture (University of Craiova, 2005).

2. SYSTEM OVERVIEW

Main system components are presented in Figure 1 with the hardware and software elements included in the iLIT System, and also the connections between them.
eBox-II (Icop Technology) provides one of the fastest Thin Client performance on the market, Vortex86 CPU at 200 MHz, 128Mb of SDRAM memory (8Mb dedicated for graphics controller), integrated accelerated 2D graphics controller, 32 MB of “Flash Disk” for storing embedded Windows CE image and applications, 10/100 BaseT Fast Ethernet connector, 2 PS/2 ports for PS2 Keyboard and Mouse, 2 USB ports, headphone and microphone connectors, SVGA monitor port, 1 parallel and 1 serial port (Icop Technology, 2005).
As display, a common TV set is used, connected to the e-Box through a VGA to TV Video Modulator. A TV set was used instead of a PC Monitor, firstly to lower the cost of the system, and secondly because a TV is found in most peoples’ houses.
The application also communicates with the user by speaking to him, by means of the attached speakers to the e-Box. As a pointing device it is used a Watcom Graphic Tablet attached to e-Box through USB, tablet that allows pointing and writing in a more familiar way consistent with every day life.
The Application Kernel is the coordinator for all software modules and thus it is Flashed on the e-Box. To store the additional modules and their data an USB Memory Stick is used, that offers enough space for storing a great variety of data.
A Speech Module is in a direct communication with the Application Kernel that exports its features to other modules. The Application Kernel also displays a graphical user interface, allowing the selection of individual modules.

3. SYSTEM IMPLEMENTATION

A platform Windows CE image was built for this application. Apart from the e-Box II BSP drivers, specific drivers were added for USB Mass Storage Devices and FAT file system to allow future expansions; a hard disk drive was not included from obvious reasons. The image includes the standard Windows CE GUI because future efforts will be made towards creating Learning Modules that will allow students to achieve basic Computer Skills. Graphics and multimedia support is extensive due to the audio and visual content of the Learning Modules. Actual image size is roughly 22 MB out of e-Box total of 64MB, without the application.
An important fact is that user interface was geared for elderly people. Layout, colours and colour combinations, font types and sizes were chosen according literature recommendations and guidelines as (Thomas *et. al*, 1999).
Another fact is that all algorithms used have to be highly computational effective due to processor low processing power and the lost of speed by using managed code (.NET C#).
The system modular architecture built around a kernel module is described in the following.

3.1 Kernel module

The kernel provides an easy-to use, dynamic interface used to launch any module that might be added to the learning platform. Its main goal is to achieve an intuitive and complete launch platform for all the future additions to the system. This is achieved by independently loading each module in its own *AppDomain* thus allowing freedom to the module developer, while the user is provided with enough information about the purpose of the module. The concept is described in Figure 2. The loader and remote loader classes provide basic loading capabilities to all the modules, while the main communication method between modules is ensured through the windows messaging system.

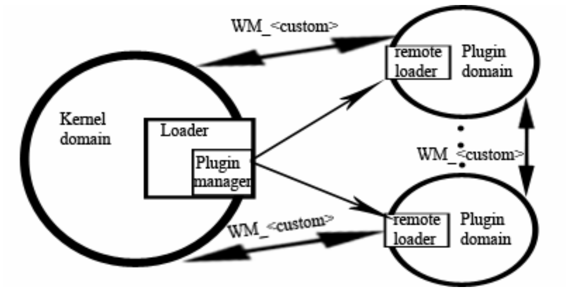

Fig. 2 Kernel module concept

Due to the need for flexibility, the project relies on USB *MassStorage Devices* to provide the physical support for incoming modules. This in turn, created the need for an insertion/removal detection mechanism, and while the PnP capabilities of the platform do not provide the means for this, a watcher thread was implemented to address this problem. This thread will watch for any changes in the *\Hard Disk<#>* directories and will raise an event every time such a directory is created or deleted.
The *UserInterface* class manages the intuitive interface, which the user actually sees, and interacts with. Built with simplicity in mind it only has 6 pictures that also act as buttons.

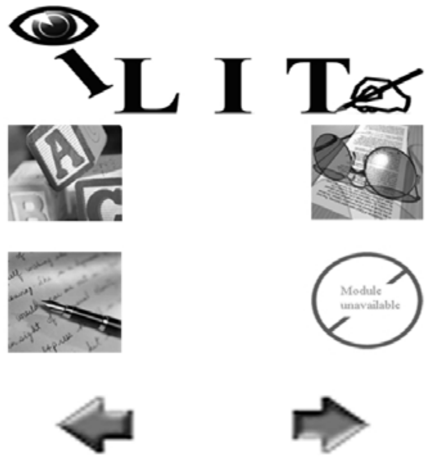

Fig. 3 Kernel user interface screen

The four images in the middle are the actual module buttons the images are loaded from the module directory, which must contain a bmp image with the same name as the assembly.

The 2 lower buttons provide means to navigate through the modules, 4 at a time; when there are less than 4 modules available the correspondent image will become *<module unavailable>*.
OnMouseOver the interface will play a sound, which will explain the purpose of the module to the user. When modules are inserted or removed the interface is notified in order to prevent loading inexistent modules or not showing the newly - arrived modules.
The *PluginList* is a collection, which holds the absolute paths to all the available modules at a time. The modules are indexed by the drive name and module number. These are all the required parameters for the *UserInterface* to work properly.
The *PluginManager* class will keep track of all the loaded modules and the drives they were loaded from. The Loader and *remoteLoader* classes provide basic loading and unloading means as well as probing for a particular interface in a module, and initial context passing.

3.2 Speech Module

In order to achieve the goal of proper learning for targeted people, accurate speech modules are needed, which give proper help to the user, mainly to accommodate and navigate within the application interface, and secondly to send voice messages about current tasks, answers to user commands and accurate reading of text messages.
There are different speech needs for iLIT modules, some imply explaining the interface usage, others to reproduce a text who can be in played in plain words or spelled by letters, or to read the message (words or letters) written by the user on the graphic tablet, also spelled by letters. In order to accomplish all these, the speech engine has methods to play the whole message or to split messages in letters and play them in corresponding order, or to combine the two methods one after another.
These tasks can be accomplished using two basic design approaches, one using a Text-To-Speech (TTS) technology to convert text commands/messages to spoken words and seconds using prerecorded audio message (commands, words, sentences and letters). The TTS technology represents a better storage space saver and implies a lesser work for the module creator, reducing the time needed for recording audio samples, but the problem with this implementation is that it uses grammar rules different for every language, and there are a lot of languages without a proper TTS engine implemented.
The communication between the command module and speech module is ensured through the windows messaging system when a proper message is sent to speech module. The message contains information about the audio file to be played, the desired method of speech, whole word or spelled by letters.
To reduce the needed storage space, audio samples are normally compressed using MPEG Audio Layer 3 codec. But there is also a list of common audio

samples to be played, for example letters that are frequently used, and in order not to waste processing power to decode them, they are stored as uncompressed PCM audio files.

The main class is *SpeechEngine* who manages the sound playing requests. This class translates the received message from any other module, parses and processes the message in the appropriate way. For example, if the message contains a valid path to a mp3 file, it plays that audio, and then it will or will not spell by letters the sent message. The *SpeechEngine* knows that the text representation for every message that will be spelled is the name (without path and extension) of the requested audio file to be played. If the string from the message is not a valid path, the string will be spelled.

For playing the whole audio file, it chooses the *SpeechEngine.Play()* function that uses *MP3Player* class to play that file, and for spelling *SpeechEngine.Spell()* that splits the message in letters and uses the *WavePlayer* class to play the files corresponding to each letter. The *WavePlayer* class has a list of audio streams corresponding to each letter that is loaded in memory at first usage and that can be played by the *Play(),* function which splits the message in letters.

The *MP3Player* and *WavePlayer* classes implement the *AudioPlayer* interface. The code is language independent, the only thing that needs to be modified for translations being the alphabet and its corresponding audio files. The module was tested for performance during the implementation, and that's why the sounds for letters (which are frequently used) are stored in uncompressed files and loaded in memory at first usage.

3.3 Learning the Alphabet - Letters Module

This module focuses on presenting the letters of the alphabet, one by one, to the user, and testing the user's abilities to learn each letter, by testing him twice for each letter. The tests consist of checking from a grid of random letters only the correct ones.

The main classes for this module are presented in the class diagram. The main application class is *LearnToReadModule1* that contains an instance of *TabletInterface* that represents the backbone for the user interface.

The *TabletInterface* class was designed to be easily used as the backbone for future implementations. It consists in a number of 28 buttons, each having an identification number from (0…27), arranged along a center drawing panel. The buttons of the *TabletInterface* were considered instances of *PictureBox* from S*ystem.Windows.Forms* so that pictures could be applied on each of them. Also behind the buttons and the drawing panel, we placed another *PictureBox* that represents the background of the user interface. The drawing panel placed in the center of the screen is an instance of *System.Windows.Forms.Panel* and represents the workspace for the *LearnToReadModule1* or another module. The *TabletInterface* can be applied on any given *Form* and its buttons can be easily customized once this class is instanced.

LearnToReadModule1 class represents the core of this learning module, guiding the user through the alphabet and testing him to decide whether he has learned the letter and can advance to the next one.

Previous Learned Letter Button allows the user to view the letters that he has learned so far. A pressing of this button switches between the current displayed letter and the previous letter of the alphabet. The *Next Letter Button* changes the displayed letter to the next letter of the alphabet. If the maximum letter was reached, that is the next letter to be learned by the user, this button hides itself. *Advance to Test Button* leads the user to tests for the current letter or to the next letter if the user passed test 2 for the current letter. *Exit Application Button* leaves the application and saves, using the *XML_Module1* class, the needed information for a later run of the module.

3.4 Words Module

After the user got familiarized with all of the letters, he/she can use this module to learn how the letters are linked together in words.

The words that will be presented refer to entities that the user is already familiarized with, this approach ensuring a fast learning process. A bunch of words that may not be part of the user's life are also included, increasing his/her general knowledge. Anyway, the words list can be easily customized by the software support personnel, based on each user's living environment, habits, etc.

This module consists of two sub-modules: first one is used for presenting the words to the user, and the former for allowing the user test his reading abilities acquired during the presentation.

Three more buttons are available, two for navigating through the words, and one for replaying the current word. The presentation of a word consists of displaying the image of the object that the word refers to, and then the word is spelled letter by letter both visual and audio.

Because the button that is implemented by .NET Compact Framework does not allow displaying an image on it (Wigley and Wheelwright, 2002), we created our own *CImageButton* class which is derived from the standard *Control* class. Any button can be instantiated by *CImageButton* in two ways: with an image on in and with text (as a standard button, but with a different look-and-feel).

The former *CImageButton* instances are used in the *Words Test* sub-module, for displaying the letters, but it is intended to put text on buttons in advanced modules, when the user will have enough reading proficiency. For easily changing the layout, the *CImageButton* class provides public properties that allow changing the caption, the image and the sound that will be played when the mouse is over the control.

Words Test; After the user gets familiarized with the words presented in the above sub-module, he can click on the next option in the upper left of the screen for taking a test. The test consists in displaying an image (from the ones used during the presentation),

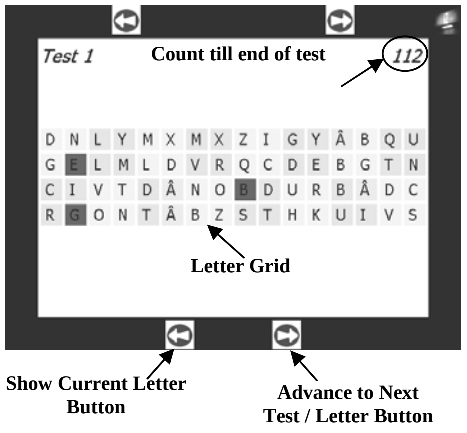
Test 1
Count till end of test
112
D N L Y M X M X Z I G Y Â B Q U
G E L M L D V R Q C D E B G T N
C I V T D Â N O B D U R B Â D C
R G O N T Â B Z S T H K U I V S
Letter Grid
Show Current Letter Button
Advance to Next Test / Letter Button

Fig. 4 Letters module screen

while the user tries to choose the correct letters to compose the corresponding word.
As seen in the picture, there is a button matrix at the bottom of the screen, each button containing a letter. In order to compose the word, the user must click on the button that contains the next letter in the word. The system waits until the user clicks on the correct button, each mistake raising an audio message. When the entire word was completed, it automatically passes on to the next word.
For enforcing the user to click on the correct letters, without automatically learning the column and row number of the button that must be pressed, the letters are generated in a random order. In addition, the buttons may contain symbols other than letters. A fast algorithm for this task is proposed. An index array initialized with letters, in the order they appear in the alphabet it is used. Then a random number N is generated, representing the number of permutations that will be made. This number should be big enough to ensure a good shuffle of the letters. Then *N* permutations are made, at each step two random indexes being chosen and the elements that are found at these positions being swapped.
The user interface related classes were kept apart from the classes that handle the back logic of the application, allowing easy change and debug of the functionality without changing the front interface.

3.5 Learning to Write - Writing Module

The aim of iLIT's Write Module is to create and improve the user's writing capabilities. The user must draw an approximate shape of the letter that is displayed to him, and can advance to the next letter if it's decided that what he drew is 70% similar to the displayed letter's shape.
Consider, for example, that the user is in the process of learning to write the letter "A". The user interface used for write training is shown in Figure 5.

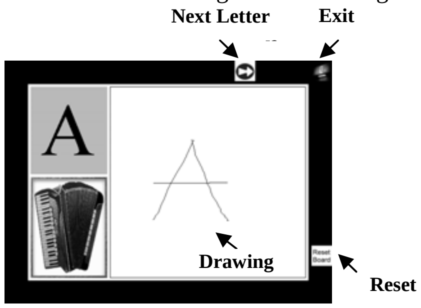

Fig. 5 Writing module main screen

The user interface updates with the letter and its corresponding picture, the user is told using voice messages about the letter that he must draw and the suggestive word that contains this letter. The *Drawing Board* represents the area where the user must draw the letter presented to him in the upper left corner. Below the presented letter is placed a suggestive picture of a word, e.g. "Acordeon" that contains the letter "A". If the user decides that what he drew is wrong, he can always clear the *Drawing Board* by pressing the *Reset Board Button*. If it's decided to leave this module, the *Exit Application Button* does it If the user considers that what he has drawn is correct and wants to advance to the next letter he has to press the *Next Letter Button*, which leads the user to the next letter only if the letter drawn represents at least a 70% approximation of letter "A".
As a skeleton for the user interface the *TabletInterface* class was used, presented earlier, in the iLIT's Letters Module. On the drawing panel of the *TabletInterface* for the shown letter's representation, the displayed picture, and drawing board *PictureBox* objects were used.
Writing recognition algorithm behind the user interface is the decision factor that lets the user advance to the next letter. Hand writing recognition algorithms involve training the software to recognize each letter. The training process is required, because it is almost impossible that the final user would draw the same letter in the same way, each time. What the user draws will be always an approximation the correct shape of the letter, the approximation degree depending on different factors, such as user's handiness (which should be very low for iLIT's intended users), precision of the pointing device, etc.
An additional tool called *CharTrainer* was created, that assists the process of training the characters. This has a simple interface that allows drawing a character, calculating a signature and saving this signature to a binary file. For example, the letter *A* is drawn and the signature saved in the file *A.dat*. When the final user is required to draw the letter *A* in the *Writing Module*, the signature for his drawing will be

calculated, and then compared with all of the signatures found in *A.dat*. If the new signature fits with one of the existing signatures, then the user has drawn the correct letter. For the writing algorithm to work properly, the software must be trained to recognize dozens of shapes for the same letter. As an improvement of the comparison process, only an average signature is stored in the binary file, the algorithm being reduced to only one comparison. When *CharTrainer* adds a new signature vector, it will increase the number of signatures, and update the average signature. *CharTrainer* was designed so it can be configured to work with different pattern recognition algorithms, by only changing the function that calculates the signature.

4. RESULTS AND FUTURE DEVELOPMENTS

First modules developed and tested proved that such a solution is a viable one and could be further developed to its end-goal of bringing education and knowledge closer to those who need it.
Such system through modularity, ruggedness and the ability to operate remotely on common and inexpensive hardware could prove to be a way for aiding remote or isolated communities in achieving at least a basic level of education.
More modules were planned, with current work focused on optimization, on improving the existing algorithms and the user interface. One algorithm that it is a candidate for improvement is the one used by the Writing module. Alternatives are now considered to the pattern recognition algorithm currently used, so a choice will be able made for the one that is faster and well suited to be used by persons that do not have good writing abilities yet.
A quick overview of other possible additions:
- User tracking and profiling, intended to provide a valuable tool for teachers and instructors, in assessing a user's knowledge, to identify knowledge gaps and areas where the student needs further assistance.
- Network connectivity, allowing a teacher to work with several students remotely, over the Internet and also allowing teachers to reach remote areas without the need of physical presence.
- A special encasing, which would allow the system to function almost independently, offering better protection for the device in harsh, untrained environments.

5. ACKNOWLEDGEMENTS

The work presented in this paper is based on the report presented by a team from University of Craiova at the final of the Microsoft's contest Windows Embedded Student Challenge 2005 (University of Craiova, 2005), see Figure 6. For this contest Microsoft Corp. provided the needed software, basic hardware (e-Box II) and support in order to achieve these results.

Fig.6 iLIT application on display at WESC2005 Final at Microsoft, Redmond, WA

REFERENCES

Blake, B. E. and Blake R. W. (2002) *Literacy and Learning: A Reference Handbook*. ABC-CLIO

Deutsche Welle (2002), *Trapped in Illiteracy*, 19.02.2002, *www.dw-world.de*

Icop Technology, Inc. (2005) eBox-150LS User Manual v 1.0, www.icop.com.tw

Microsoft, Inc. (2005), Window CE Home Page http://msdn.microsoft.com/embedded/windowsce/default.aspx

Muench C., (2000) *The Windows CE Technology Tutorial-Solutions for the developer*, Addison-Wesley

Ringold D., Orenstein M. A. and E. Wilkens (2003), *Roma in an expanding Europe. Breaking the poverty cycle*, World Bank Study, April 2003

Sharp, J. and Jagger, J. (2002) *Microsoft Visual C# .NET Step by Step*, Microsoft Press, 2002

Thomas A. M.,(1989) *Adult literacy volunteer tutor program evaluation kit*, Ministry of Advanced Education and the National Literacy Secretariat, Victoria, B.C., Canada, 1989

Thomas, K.G.F, Laurance, H.E., Luczak, S.E. and W.J. Jacobs.(1999) "Age-Related Changes in a Human Cognitive Mapping System: Data from a Computer-Generated Environment." *CyberPsychology & Behavior* 2(6), December 1999, pp 545-566.

University of Craiova (2005), iLIT An intelligent tool to assist literacy teaching and learning, *Final report*, Windows Embedded Student Challenge 2005, WESC2005, www.windowschallenge.com

Withrow, F. B. (2003) *Literacy in the Digital Age: Reading, Writing, Viewing, and Computing*, Scarecrow, 2003

Wigley, A. and Wheelwright, S.(2002) *Microsoft .NET Compact Framework (Core Reference)*, Microsoft Press, 2002