

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra kybernetiky

Diplomová práce

**PDA home-care support**

Bc. Jiří Doležel

Vedoucí práce : Ing. Jaromír Doležal

Květen 2009

## **Acknowledgements**

I would like to thank to Ing. Jaromír Doležal for leading my diploma thesis, to Ing. Petr Aubrecht Ph.D for valuable technical advices, Ing. Pavel Šťastný for help with initial application and technical advices, and to medical experts Prof. MUDr. Martin Haluzík, DrSc, MUDr. Lenka Bošanská, and MUDr. Tomáš Roubíček for their collaboration in the design and development of the PDA application.

## **Manifest**

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady ( literaturu, projekty, SW atd.) uvedené v příloženém seznamu.

**V Praze dne** .....

.....

**Podpis**

České vysoké učení technické v Praze  
Fakulta elektrotechnická

Katedra kybernetiky

## ZADÁNÍ DIPLOMOVÉ PRÁCE

**Student:** Bc. Jiří Doležal  
**Studijní program:** Elektrotechnika a informatika (magisterský), strukturovaný  
**Obor:** Biomedicínské inženýrství  
**Název tématu:** Podpora home-care pomocí PDA

### Pokyny pro vypracování:

Navrhnete a implementujete aplikaci pro PDA, která usnadní sběr lékařských dat pro starší pacienty vyšetřované v místě bydliště.

Program musí poskytovat rychlé a přehledné rozhraní pro dokumentaci vyšetření sestrou mimo nemocnici v rámci home-care.

Aplikace bude implementována v jazyku JAVA a musí umožňovat následující funkcionalitu:

- získání požadavků na vyšetření specifikované lékařem z existujícího webového serveru,
- uložení požadavků a protokolů k vyšetřením lokálně off-line a odesílání po opětovném připojení k síti,
- zobrazení a vyplnění obecných formulářů pro vyšetření,
- zobrazení historických hodnot,
- přehled a kontrolu úplnosti vyšetření,
- správu fotodokumentace.


**Seznam odborné literatury:** Dodá vedoucí práce.

**Vedoucí diplomové práce:** Ing. Jaromír Doležal

**Platnost zadání:** do konce letního semestru 2008/2009

  
prof. Ing. Vladimír Mařík, DrSc.  
vedoucí katedry



  
doc. Ing. Boris Šimák, CSc.  
děkan

V Praze dne 16.12.2008

## **Abstrakt**

Tato práce realizuje podpůrné prostředky pro služby domácí péče (home care) na mobilních zařízeních typu PDA. Práce je součástí pilotní studie EU, která je zaměřena na vývoj systému pro mobilní elektronickou podporu home care. Pacientům se dostává poměrně široké spektrum služeb, jako třeba návštěvy zdravotnického personálu, lékařů a sociálních pracovníků. Problémem bývá předávání informací, které se týkají zdravotního stavu, předepsaných vyšetření a jejich změn, mezi těmito pracovníky. K4CARE je webová, elektronicky přístupná, platforma, která integruje zdravotní záznamy pacientů, plánovaná vyšetření a služby. K záznamům lze přistupovat přes webový prohlížeč, či bezdrátově pomocí mobilních PDA zařízení, či mobilních zařízení obecně.

## **Abstract**

The thesis is focused on project of a pilot study, which is implementing services supporting work of care professionals in the area of home care and running on PDA device. This work is part of K4CARE EU project, which is focused to develop system for home care support, using mobile electronics devices, like PDA. Patients are receiving various services, for example nurse visits, physicians or social workers. The issue might be the flow of information among them, which involves patient's health status, prescribed examinations and its changes. K4CARE is web based platform, which integrates patient's health record, planned visit and services, which can be accessed with web browsers or with PDA, or mobile devices generally, connected to the Internet.

## Table of Contents

1 Introduction.....	10
2 Aim of the project.....	11
3 Current solutions – State of the art.....	12
4 Application design (architecture).....	14
4.1 Requested and desired functionalities.....	14
4.2 Hardware (PDA device) requirements.....	15
4.3 Software requirements.....	16
4.4 Used devices.....	17
5 Implementation.....	18
5.1 Application description – programmers view.....	18
5.1.1 XML Document description.....	18
5.1.2 Packages and classes description.....	21
5.1.3 GUI structure.....	30
5.1.4 Localization.....	31
5.1.5 Communication.....	31
5.1.6 Storage model.....	32
5.2 Application description – end user view.....	32
5.2.1 Screens description.....	33
5.2.2 Scenario – Logging in, filling the document, saving.....	35
5.3 Implementation issues, constructions to mention.....	41
5.3.1 Data processing.....	41
5.3.1.1 Parser.....	41
5.3.1.2 DocHandler class.....	43
5.3.1.3 Element (Tag) class.....	43
5.3.1.4 Document model class.....	43
5.3.2 GUI.....	48
5.3.3 Photo documentation.....	50
5.4 Portability.....	50
6 Conclusion.....	52
7 Citation and references.....	53

## Abbreviation list

ICT	Information and Communication Technology
HCP	Home Care Professional
EHR	Electronic Health Record
PDA	Personal Digital Assistant
IZIP	Internetový přístup ke Zdravotním Informacím Pacienta
CFH	Connection For Health
OS	Operating System
Java ME	Java Micro Edition
J2SE	Java2 Standard Edition
SHA	Secure Hashing Algorithm
DOM	Document Object Model
DoM	Document Model

## Illustration Index

Illustration 1: Sequence diagram – typical use case.....	15
Illustration 2: Part of XML document representing info about visit and patient and piece of history.....	20
Illustration 3: Part of XML document with pieces of document, that are showing measure form.....	21
Illustration 4: GUI schema.....	30
Illustration 5: Login screen.....	33
Illustration 6: Patients screen.....	33
Illustration 7: Patient photo screen.....	34
Illustration 8: List of measurements screen. Currently is focused Blood pressure measurement.....	34
Illustration 9: Single measurement screen – input fields.....	35
Illustration 10: Single measurement screen – radiobuttons, 1st page.....	35
Illustration 11: Single measurement screen – radiobuttons, 2nd page.....	35
Illustration 12: Patients listing, clicking on “Patient card” of Will Timer.....	36
Illustration 13: Patient card. Clicking on “Measure modification”, to select measurements to be performed.....	36
Illustration 14: Set of possible measurements. Checking it will select it to be performed. Clicking the view button displays the measurement(Consciousness).....	37
Illustration 15: Displaying measurement (Consciousness).....	37
Illustration 16: Clicking “Submit” icon after setting the login info.....	38
Illustration 17: Data processing.....	38
Illustration 18: Downloaded list of patients.....	38
Illustration 19: Photo panel. Shows after pressing the examination button. This one is empty, with no photo.....	38
Illustration 20: Clicking on “start” (top left) and selecting the camera to take a new picture. ....	38
Illustration 21: Taking a picture (of the computer mouse, as an example). ....	38
Illustration 22: Clicking the “Load picture” button to load the photo recently taken.....	38
Illustration 23: Clicking the “Continue” button after loading the picture.....	38
Illustration 24: Group of measurements, presented by buttons.....	39
Illustration 25: Filling the measurement. Pop up window is showing to enter numbers.....	39
Illustration 26: Saving the filled measurement.....	39
Illustration 27: Group of measurements with first measurement filled (green color). ....	39
Illustration 28: Looking through the second measurement, 1st page of two.....	39
Illustration 29: Looking through the 2nd page of measurement and saving empty measurement.....	39
Illustration 30: Clicking on 3. measurement. 2. measurement is not filled, signaling with red color of button. h) Saving the filled, third measurement.....	39
Illustration 31: Saving the filled, third measurement.....	39
Illustration 32: Returning to the second (not filled) measurement.....	40
Illustration 33: Selecting an option on 1st page.....	40
Illustration 34: Selected an option on 2nd page and saving.....	40
Illustration 35: Group of measurements, all filled. Saving all for the current patient.....	40
Illustration 36: List of patient. For Will Timer indicating, that the document is filled.....	40
Illustration 37: Saving (and updating) data of filled patients (only) to the server.....	40



Illustration 38: Saving in progress.....	40
Illustration 39: Login screen after saving process is finished.....	40
Illustration 40: Example algorithm of searching children elements, in Pseudo-code.....	45
Illustration 41: Example document, hierarchy tree when parsing, resulting array in memory, subsidiary arrays.....	47
Illustration 42: searching a child of a element on position 2. The result are elements with indexes 3 and 7.....	48
Illustration 43: Comparing benchmarks of parsing and building document using an old (org.w3c.dom) and new tools (average of 3 test, varying about 0.5 second).....	48

# 1 Introduction

The K4CARE project brings together 13 academic and industrial institutions from seven countries for a period of three years starting March 2006.

In modern societies, the care of chronic disabled and elder patients at home involves life long treatment under continuous expert supervision that saturate European national health services and increase related costs.

K4CARE aims to combine the healthcare and the ICT (Information and Communication Technology) experiences of several western and eastern EU countries to create, implement, and validate a knowledge-based healthcare model for the professional assistance to senior patients at home.

This new healthcare Model for home care will contribute to achieve a European standard supported by the new technologies that improves the efficiency of the care services for all the citizens in the enlarged Europe [1].

The goal is to create a new ICT Sanitary Model (K4CARE model) for assisting HCPs (Home Care Professionals) in Europe, which will be provided through an intelligent web platform and provide e-services to health professionals, patients and citizens in general.

CTU responsibility in this project is to create a web interface for health professionals which enables them to operate with and evaluates patient's EHR (Electronic Health Record) and to collect patient data taken through PDA (Personal Digital Assistant) from patients at home.

PDA Application, which is goal of this work, is aimed for HCPs to examine patients in their home, collect their EHR and cooperate with web platform, which delivers tasks for HCPs to PDA and upload data taken from patients.

## **2 Aim of the project**

Target of this work is to create an application for PDA, which is going to be used by home care personnel. The main scenario is that a doctor, through a web module, checks the forthcoming visit data, which contains patients and what kinds of examinations are needed to be performed. A home care personnel, like nurse, download pre-selected data to a PDA device. In a PDA device, program automatically parse and process the data and create GUI (Graphical User Interface) for nurse. It contains all needed informations about patients and their examinations, that needs to be performed and represents the examination forms itself. When examinations are done, the nurse sends the collected data back to a server, where it is further processed.

The final application should be intuitive, quick and possibly portable to another platforms, not only the one, where it is developed. It should minimize the amount of input actions to PDA device for a nurse, since it is not very comfortable, due to a limited input choices and small display size. Also, receiving, processing and sending data should not require any action from personnel, except the initial login and “single click“.

### 3 Current solutions – State of the art

Currently, there are a few projects implementing and using EHR. In the Czech Republic, there is the IZIP system [2]. It stores complex patient health documentation. It is shared over the Internet and it represents some sort of electronic book, where are all medical related data of patient stored. There are, so far, about 1 million patients registered. Quite similar project CFH (Connection For Health) was launched in Great Britain, to unify three existing solutions into the one [3]. There is also implementation of EHR in the USA. It is called the Veterans Health Information Systems and Technology Architecture system (VistA). Part of this is VistA Imaging system [4], which integrates clinical images, scanned documents, and other non-textual data into the patient's electronic medical record.

Except EHR implementing systems, there are quite many commercial companies doing business in a home care service. These are offering various of services, like sanitary services, rehabilitation, domiciliary services, etc.

The home care is considered as one of the most emerging medical technology, that increases the patient's comfort and allows more financial effective care (since it is cheaper, than placing patient to a hospital). There are several specialized projects, that is being currently developed. Some of these projects are targeted to work with PDA devices as well. One has been developed on AGH University of Science and Technology in Poland. It is called "PDA-based system for cardiology home care and pregnancy monitoring" [5]. It is focusing on remote cardiac monitoring and pregnancy related health issues. PDA device is used for its computational power and storage space. It communicates with interpretation center (which interprets many various signals) and is can modify the communication protocol and processing routines online. Then, the device may be used for various monitoring tasks including ECG tracking, sleep monitoring of patients with apnea of sleeplessness, muscle fatigue assessment during training or physical exercise, uterine contraction detection based on abdominal potentials in patients at risk of premature delivery. The project shows, that it can deliver monitoring of various body signals with quite cheap and powerful device, like a PDA.

Another interesting project has been launched by Danish company CSC Scandihealth, which is provider of healthcare IT products and services in Scandinavia. The system is called VITAE Care PDA [6]. It provides care workers with access to relevant client informations before or during visit to the client's home. Care workers has overview of all kinds of patient relevant informations, like health information, accessibility of the patient home, current medication, latest notes, and total list

of the services the client receives. The care worker can see current medication registrations, notes, focus areas, next of kin, own doctor, pharmacy, etc. The care worker is able to send notifications, register medication or document in notes and focus areas.

Another project, which is part of work of HealthForceOntario [7], is to achieve the goal of ensuring right mix of healthcare providers [8]. It provides nurses with PDA devices with instant access to a range of clinical tools that allow them to quickly diagnose and counsel patients at the point of care with a few simple clicks. PDAs are loaded with information about drug databases, medical dictionaries, best practice guidelines and reference material related to care and service.

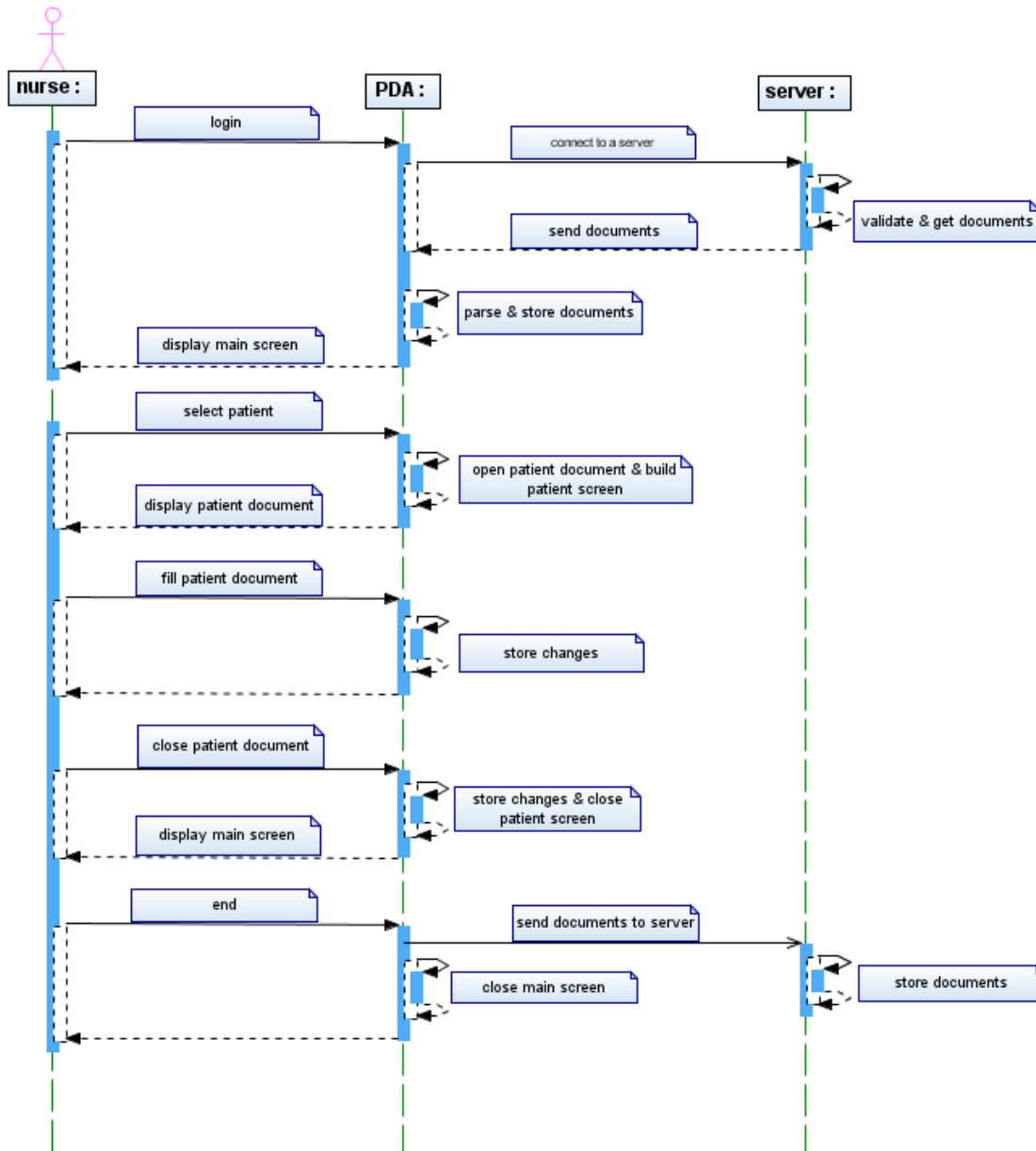
So, there exists several solutions, some are implementing EHR, then, there are companies ensuring home care services and then, there are several specialized projects. K4CARE is integrating EHR, home care services and basics examinations.

## **4 Application design (architecture)**

### **4.1 Requested and desired functionalities**

The application has to provide graphical interface for home care personnel. It needs to be intuitive, transparent and quick. It must realize the typical use case Illustration 1. Desired functionalities are:

- Obtaining requested (by a doctor) examinations plan from existing web server.
- Protecting the application and data access from non-authorized personnel access.
- Parsing and storing the data from server to be able to do the examinations offline. Sending the data (after examination done) back to server, when the connection is available.
- Creating the interface for listing patients and displaying measure forms and managing examinations.
- Allows doctor to send custom message to a personnel managing an examination and vice versa.
- Displaying historical values from past examinations.
- Checking, whether the document is filled completely.
- Managing the photo documentation.



## 4.2 Hardware (PDA device) requirements

There are several important requirements for the device:

- Wireless adapter – This is suppose to be the main way to access the Internet and server respectively, to download and upload data. It is possible to connect to the Internet over another PC, but it requires PC with the Internet connection, installed software and set the PC to enable connection for PDA. And, of course, there is needed presence in the PC area during the work.

- Hardware keyboard – This feature can be fully substituted with software keyboard (as it is always presented and the most common), but during the filling of the document, there will be some text writing as well and hardware keyboard could be very handful. Also, software keyboard occupies some space on the screen, thus, covering part of document.
- Camera – To be able to take images of patient.
- CPU speed – Since there is interpreted language used (Java), it is needed to have as fast CPU as possible. Though, it is not the key feature. CPU speed of currently selling devices are about the same and fast enough.
- RAM memory – As in CPU speed feature, it is needed, though not the key feature. It is always good to have a lot of available RAM, but several MBs is needed for the application to run.
- LCD display – The greater display, the better. Since the forms are going to be presented in the GUI form, like Labels, Buttons, Check Boxes, etc., then the small screen could be limiting for final comfort, impression and experience.
- GSM – Mobile phone could be useful, when concerning health issues.
- GPS (Global Positioning System) – For future use. Could be implementing some sort of navigational software and visit organization.

### **4.3 Software requirements**

The device with OS (Operating System) based on Windows Mobile platform was chosen, since it belongs to one of the most common environments.

There were few possibilities for software platform to be used for the development. First, and obvious, is C based language, like high level C# language. This was just considered and rejected, since open source platform is desired for such an academic project. Other possibility was Java, or Java ME (Micro Edition) to be more specific. It is a limited version of Java, to be used on mobile devices like mobile phone, PDA, etc. Since the hardware equipment and advancement of destined devices varies a lot, there are several versions of Java ME, but still, all versions are quite restricted, since each versions has to fit for quite wide group of devices with various equipment. There was first substructure of an application created using Java ME. Yet, it showed, that user interface was



not as desired and the interaction possibilities as well.

Finally, Mysaifu [9] software was found. It is a Java Virtual Machine, which runs on Windows Mobile, yet, working with J2SE (Java2 Standard Edition). It is an open source project under the GPLv2 license, which has an objective to make a Java Virtual Machine which conforms to the J2SE. It's not implementing all functionalities of J2SE, yet, it implements most of them and fits for this project.

#### **4.4 Used devices**

First device obtained was **HTC P3300**. Main hardware specifications are [10]:

- Processor – TI's OMAP™ 850, 201 MHz
- Memory – ROM: 128 MB, RAM: 64 MB, expansion slot for microSD memory card
- Display – 2.8" TFT-LCD, resolution 240x320, 65,536 colors
- Connectivity – GSM/EDGE, GPS, Wi-Fi, Bluetooth, mini-USB
- Camera – 2 Megapixels

Initial application was created and tested with the device and some findings showed up (summarized in chapter **4.1.2 Hardware (PDA device) requirements**). The most important equipment, among other, is hardware keyboard, Wi-Fi, camera, overall speed and Windows Mobile OS. There are not many such a devices. Final chosen was **HTC Touch Pro**. The key specifications are [11]:

- Processor – Qualcomm® MSM7201A™ 528 MHz
- Memory – ROM: 512 MB, RAM: 288 MB, expansion slot for microSD memory card
- Display – 2.8" TFT-LCD, resolution 480x640
- Connectivity – GSM/GPRS/EDGE, GPS, Wi-Fi, Bluetooth, mini-USB
- Camera – 3.2 Megapixels
- Other – Hardware keyboard

## 5 Implementation

### 5.1 Application description – programmers view

The data format for storing and interchanging information with the application had to be chosen. Finally, the XML format was chosen for all data and communication as well, since it has several advantages. First, it is widely accepted, portable and universal format. The other is, that it is used in the K4CARE project as well.

There are existing several standards, that should application implementing and working with her follow, to ensure interoperability of the application. One of the most important standards, that is K4CARE to follow, is HL7 standard. HL7 has goal to create the best and most widely used standards in healthcare [17]. And HL7 standard is working with a XML technology for documents representing and exchanging.

#### 5.1.1 XML Document description

Part of XML document related to user and patient can be seen on Illustration 2 and another part, related to a measure itself on Illustration 3. The root element of the document is *visit* and contains parameters *loginName* and *loginPass*, which is encrypted with SHA (Secure Hashing Algorithm). Children of *visit* element tag are:

- *userinfo* – Contains series of elements with info about application user.
  - *id* – Numerical ID of user.
  - *loginName* – Login name, which is used to log in a application and to a server to download data.
  - *loginPassword* – Password, for application and server.
  - *entityType* – Role of user.
  - *fullName* – Full name of user.
- *root* – Root element of single patient examination. Has an attribute *patientid* which hold ID

(numerical value) of examined patient. Is presented as many times as many patients are set for the Visit.

- *patientinfo* – Contains series of elements with info about a patient.
  - *PublicIdentifierXML* – Some common used ID, birth number in our case.
  - *contactPhoneNumber* – Contact phone number of patient.
  - *electronicAddressXML* – Email address.
  - *fullName* – Full name.
  - *contactAddress* – Contact address.
- *history\_documents* – Contains historical values from previous measurements.
  - *history* – This is a root element that contains all measurements of single patient from a single visit. It may be presented repeatedly. It has an attribute *date*. Children nodes has the same structure as a regular measurement, which will be explained later.
- *group* – Represents an elemental measurement. Is contained as many times as many different elemental measurement is required. It has attribute *label*, which is sign displaying at the top of the screen with a form, attribute *name*, that is used by system to distinguish between different group measurements and attribute *checkDate*, which is date, when is this elemental measurement finished – it is captured, when the save button on form screen is pressed.
  - *info* – This element is just one for single patient measurement. It has an attribute value, which contains info from doctor about a patient.
  - *radiobuttons* – Contains items to select from (just one option). Has an attributes *label* and *name*, which has the same meaning, as in *group* element.
    - *item* – Represent a single item – a single choice. Has attributes *label*, *name* (same as previous) and *selected*, which is boolean value representing, whether the option is checked or not.
  - *checkboxes* – Exactly the same (also the child-elements) as *radiobuttons* element, except multiple-choice is allowed.
  - *num\_radio* – The same as *radiobuttons* element, except it contains in addition attribute value, which is displaying a value of selected child-element.

- *item* – The same element as in *radiobuttons*, except it contains an attribute, which represents a numerical value of this choice.
- *input* – Allows to input some written (text or numerical) value. Attributes and child elements are the same as in *radiobuttons* element.
- *image\_paint* – Represents an image to be displayed. It is used for home care personnel to draw some points related to a patient. For example, on image with human body, a home care personnel would mark several points, where patient suffer pain. Element has attributes *name*, which identify the image to be displayed and *coords*, which contains coordinates of points drawn into that image.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<visit loginName="kovacova" loginPass="mEgW/TKWIoduFJB2NCZObzMun7M=">
<userinfo>
  <publicIdentifierXML>584856/5555</publicIdentifierXML>
  <contactPhoneNumber>123123123</contactPhoneNumber>
  <entityType>ProfessionalActor</entityType>
  <id>4429</id>
  <electronicAddressXML>test@te.czrtf</electronicAddressXML>
  <loginName>kovacova</loginName>
  <addressXML></addressXML>
  <fullName>Petra Kovacova</fullName>
  <contactAddress>test</contactAddress>
</userinfo>
<root patientid="4434">
  <patientinfo>
    <publicIdentifierXML>745478/7895</publicIdentifierXML>
    <contactPhoneNumber>777810458</contactPhoneNumber>
    <electronicAddressXML>john.taylor@yahoo.com</electronicAddressXML>
    <fullName>John Taylor</fullName>
    <contactAddress>Prague 1293</contactAddress>
  </patientinfo>
  <history_documents date="2009.03.12">
    <history>
      <group label="Consciousness" name="Consciousness">
        <radiobuttons label="hearing" name="hearing">
          <item label="normal" name="normal" selected="false">
          </item>
          <item label="hearing aid is needed"
            name="hearing aid is needed" selected="true">
          </item>
          ...
        </radiobuttons>
      </group>
    </history>
  </history_documents>
</root>

```

```

...
</history_documents date="2009.03.12">
</history>
<group label="Consciousness" name="Consciousness">
  <radiobuttons label="hearing" name="hearing">
    <item label="normal" name="normal" selected="false">
    </item>
    <item label="hearing aid is needed" name="hearing aid is needed"
      selected="true">
    </item>
    <item label="deaf" name="deaf" selected="false">
    </item>
  </radiobuttons>
  <radiobuttons label="vision" name="vision">
    <item label="normal" name="normal" selected="true">
    </item>
    <item label="eyeglasses are needed" name="eyeglasses are needed"
      selected="false">
    </item>
    <item label="blind" name="blind" selected="false">
    </item>
  </radiobuttons>
</group>
<group label="Blood pressure (Hgmm)" name="Blood pressure (Hgmm)">
  <input checkDate="" label=" " name="zvlastni_upozorneni">
    <item label="Systole" name="Systole" value="">
    </item>
    <item label="Diastole" name="Diastole" value="">
    </item>
  </input>
</group>
...

```

*Illustration 3: Part of XML document with pieces of document, that are showing measure form.*

## 5.1.2 Packages and classes description

1. **ESIADoM** (Element Stored In Array Document Model) – Contains classes for manipulation with XML documents. As told earlier, original **org.w3c.dom** package performance was unacceptably slow. Therefore, custom DoM (Document Model) was created, that uses for XML file parsing an event based QDParser [12], which is distributed under GPLv2 license. Classes in this package are:
  - `ESIADocument` – Represents parsed XML document, contains basic methods and

various subsidiary arrays for boosting the speed of working with the document.

- `QDParser` – Simple parser, calls class implementing `DocHandler` interface.
- `DocHandler` – Interface, that specifies methods for `QDParser` to “talk to” document. The methods are `startElement(String tag, Hashtable h)`, `endElement(String tag)`, `startDocument()`, `endDocument()`, `text(String str)`.
- `SSPDocHandler` – This class is building the document model. Implements `DocHandler` interface.
- `SSPElement` – Represents the XML tag in `ESIADocument`. It contains info about tag type (element, attribute, text, value), position of element in a model and info related to a text value and attributes, all based on primitive type `int` addressing.

2. **client** – Contains interfaces and classes that defines fundamental functionalities that are required for certain data types, or their roles respectively. Classes are:

- `ClientObject` – Represents the client object which is parsed from servlet response, contains methods `init(Element el)` and `store()`.
- `CommunicationSupport` – Interface, that represents utility class which allows to communicate with server. Contains methods `getVisit` and `postVisit`.
- `CommunicationSupportFactory` – Singleton class, that specifies some other possibilities for communication, or data source respectively. It is communication over the HTTPS with server, or offline mode, when stored data is used.
- `CommunicationSupportListener` – Interface, that declares methods used to listen about connecting status.
- `History` – Interface to declare methods working with historical values (from past measures/visits). Methods specify ID of related patient, date of measure and historical data itself. Extends `ClientObject`.
- `HistoryImpl` – Class implementing `History` interface and `ClientObject` interface. Historical data are stored in a `HashMap` with key created form name of measurement.
- `NoSuchVisitException` – Class inherits from `Exception` class. It is fired, when there are problems downloading data for the visit, like when the visit is not defined for the current login info, or if the authentication fails.
- `Patient` – Interface, extends `ClientObject`, represents one patient, thus declaring

functionalities related to name, id, patients images, history, state of document (whether it is filled, not visited, half filled,...) and some subsidiary variables to DoM.

- `User` – Interface, extends `ClientObject` interface, representing an application user. Contains functions relevant to login name, password and ID.
- `Visit` – Interface, extends `ClientObject` interface. Represent one visit, which contains multiple patients and some related info to a visit. Contains patients involved in the current visit, and their histories of measurements (both stored in an array). Also contains a user currently logged in and currently examined patient.

### 3. **client.impl** – Implementing previously declared interfaces for roles and communication.

Classes:

- `PatientImpl` – Implements the `Patient` interface (and therefore `ClientObject` interface).
- `UserImpl` – Implements the `User` interface (and, therefore `ClientObject` interface).
- `VisitImpl` – Implements the `Visit` interface (and, therefore `ClientObject` interface).
- `HTTPCommunicationSupportImpl` – Implements the `CommunicationSupport` interface. Holds info needed to connect to a server for downloading a `Visit` data and handles connection errors. Also contains algorithm to encrypt text, which is, currently, used for password encoding.
- `OfflineCommunicationSupport` – Implements the `CommunicationSupport` interface. Used to work with an offline mode. It loads a `Visit` data already downloaded and parsed, from a local storage. Local storage is set to a user's personal folder.

### 4. **client.utils** – Contains just a utility classes.

- `DOMUtils` – Used for manipulation with a XML document, and its memory representation. It is used to print prolog of XML document, set a document to a buffer for data transfers. Then, it contains methods for history processing and methods for digging the text value of an element.
- `FileUtils` – Works with storage system. Checks, if the folder structure exists, and creates it, if it doesn't. It deletes files after uploading it to a server, overwrites and stores saved documents. Stores downloaded and parsed data from server. Moves images taken from embedded camera and moves them to an appropriate folder. The `FileUtils` Also

contains methods to work with file names and extensions to ensure no images and documents will overwrites other.

- `MemoryUtils` – Used only to determine free, total memory (the total amount of memory in the Java virtual machine) and maximum memory (the maximum amount of memory that the Java virtual machine will attempt to use).
- `SizeOf` – It evaluates the size of data structures in a running application.

5. **forms** – This package contains classes that represents the GUI forms. Contains methods, that are parsing such parts of a XML document, which are representing measure forms, and builds the GUI based upon its content. Package also contains logic to display patient list, traverse between patients and their measure forms and saves data inserted by home care personnel and stores it to a solid memory. Classes:

- `AbstractComponentPane` – Abstract class. extends `Panel` class. Declares abstract methods to initialize (create component and fill it with context) and stores data collected during a Visit. Contains component's name, label, getters and setters and patient ID. There is a list of components that belongs to this component, an `ArrayList` of components for focus traversing. Then, there are some variables, arrays and methods to point to a document model for easy and fast storing and manipulating with the document. It contains methods, that are filling the document with historical values (from previous measures). This class also defines a type, that is used to keep track of the state of filling of all parts of the document. It's being used from the smallest (elemental) parts of GUI form to the whole document.
- `MainGenericContainerPane` – Extends `Panel` class. Contains all the components of a single patient represented by a group element and list them as a buttons.
- `SimpleComponentFactory` – Singleton class, which is used for creating components from parsed XML document. Contain method `mainPanels`, which is called by an instance of `MainGenericContainerPane` class and returns an instances of a `GroupComponent` class, that are listed over there in `MainGenericContainerPane`. Also contains method `createPane` called from instance of `GroupComponent` class and returns parsed components which are placed in a `GroupComponent` panel.
- `GroupComponent` – Extends `AbstractComponentPane`. Holds the structures and



logic of a single page displayed and relevant functions. It represents the group element from the XML document and parses related parts of the document (subtree of group element). A group element represents a single measurement, or a few measurements that are related. It contains a Button, which is displayed in a different context (in the MainGenericContainerPane), when displaying all types of measures represented by a group element. The button brings up the content of this panel. The panel contains top sign with page count and arrows to roll over pages (if more pages – measurements – are presented in the group element). Arrows are represented by inner class, which has method paint overridden and has 2 different images, depending on whether the arrow is pressed or not. Then, it contains the main (elemental) measure form itself. On the last page, there is a button to save all the measurements. Also, switching to and from a full screen mode is implemented through a PopupEnabledWindow interface, which contains method addToPopup. It also implements mouse listeners and key listeners to control the this part of application with stylus (over the touch screen) or hardware keys. Also, the state of filling of the document within a group element is evaluated and indicated through a color of the group button. Then, it contains method to find out the size of panel representing single measurement and whether there is need for horizontal and/or vertical scrollbar.

- InputComponent – Extends AbstractComponentPane, one of components, that represents the measure itself. It is used to input text. This class also implements PopupEnabledWindow through addToPopup method, which is used to display enumerator to input numbers quickly and easily.
- LabelComponent – Extends AbstractComponentPane, just to display signs and infos. Leaves store method empty.
- CheckboxesComponent – Extends AbstractComponentPane, represents multiple selection choice.
- RadioComponent – Extends AbstractComponentPane, represents single selection choice.
- NumRadioComponent – Extends AbstractComponentPane. It is pretty much similar to RadioComponent class, but every item has a numerical value. Also, more of these components are usually presented in a one group component and sum of all is required (which is implemented in a GroupComponent class).
- FirstNote – Extends AbstractComponentPane. Contains a message from

doctor for a home care personnel, who is examining a patient. Doesn't contains any functionality involving history, storing and determining the state of filling of the document.

- `ImagePaint` – Extends `AbstractComponentPane`. It is used for parts of the document, where are attached images, that requires some painting. For example, marking an area on image of body, where patient suffer pain. It contains functionalities to mark a point, delete the closest point (to a point on screen marked by personnel) and delete all points. It stores the coordinates of points marked within the document and sends these to a server (and not the whole image).
- `LastPanel` – Extends `AbstractComponentPane`. Has bigger input window for Home care personnel to write down a note for a doctor, if needed or required.

6. **localization** – Package provides application with a logic and resources to select and use various language subsets. Implemented are Czech and English language. Classes:

- `TranslateSingleton` – Extends `PropertyResourceBundle` class, manages resources for a language variations. It is a singleton class with a static method for creating an instance of class, if none has been created yet. During initialization, it sets the language, that is set for OS environment. During the application run, user can switch the language to another subset.
- `Vocabulary` – Property files, containing resources for provided languages.

7. **painting** – It contains the application logic relevant to work with patients, application context, GUI screens down to displaying list of patients (which is login screen, list of patients, saving data screen). Classes:

- `ApplicationContext` – Singleton class, in which is an application context and some logic for switching screens. Context and methods are:
  - Screen resolution. Covers width and height changes, when switching to portrait and landscape modes (on non-square displays), when it needs to calculate with Windows Taskbar height as well.
  - Screen DPI (Dots Per Inch). Placing and sizing of graphical components is related with DPI. Placing tend to be relative, though, it was not always possible. Then, absolute (using coordinates in pixels) placing had to be used. Common DPI ranges from 72 up to 192. Our devices has DPI values 96 and 192. That means, that

graphical components (for example images) has half size on 192 DPI screen, than the same on 96 DPI screen. And also, the distance from up left corner (which is the point, from what are the positions counted) varies, depending on DPI. So, there is hold default DPI value (for which components are placed) and ratio of actual and default DPI.

- Fonts for entire application, evaluates DPI so the size of a certain font is DPI dependent.
- Colors – For indication of state (usually on/off state) of components. Used for arrows released/pressed state and for indicating un/checked state of checkboxes and radio buttons within forms. Color of screen's background.
- Image icons for arrows and logic to load it.
- Paths of destination, where embedded camera stores taken pictures.
- Boolean value, which stands for debug mode, in which case a richer and detailed output to console is turned on.
- GUI components, like frames, used to determine the size of panel, or as a container for screens. To display patient's photo before performing the examination of patient.
- Logic for creating screens with relevant context.
- Logic for determining size of measure forms to find, whether a scrollbars are needed.
- Logic for determining state of application during events, which are pressing login button and state of login process and data processing after successful login.
- Setting traversal system for entire application.
- Closing application with finishing and saving appropriate data.
- `FocusChange` – It is used, because focus system is nor working with all of the components. It contains a list of components to work with, focus traversal rules and actions to perform, when focus is gained or lost.
- `LoginPane` – Extends `JPanel` class and implements `CommunicationSupportListener` interface. Represents login screen. Contains methods to start communication and loading data from storage. Graphical components are placed absolutely, so it overrides `paint` method and computes coordinates of inner components.
- `PatientComponent` – Extends `JComponent` class, represents list of patients set for the Visit. The list is represented by `awt.List` class, `Paint` method is overridden and absolute positioning of graphics components implemented. Other methods are to

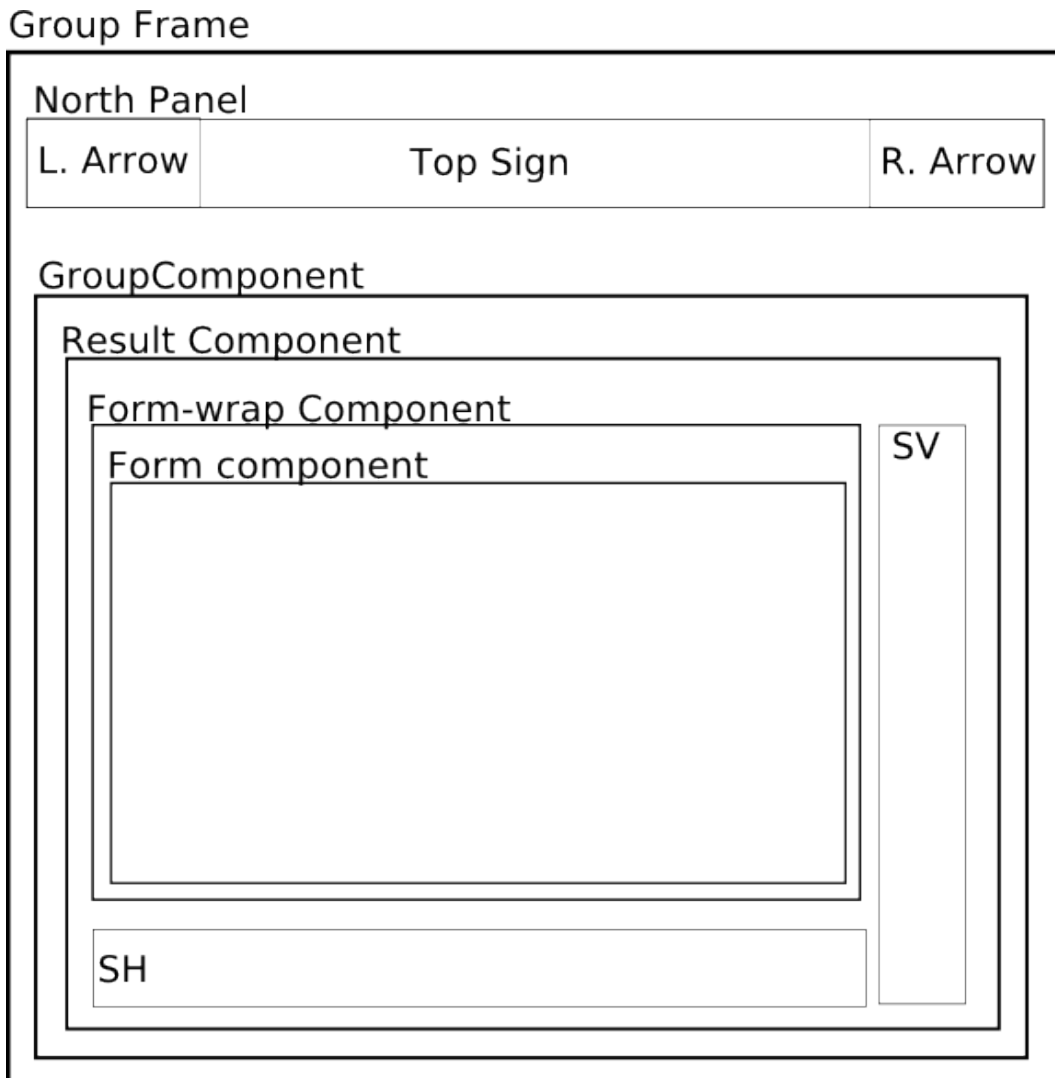
determine what patient to set focus on, who is a currently selected patient and to set the status of filling the document for every patient. That is represented with text form, since `awt.List` is able to work just with string types.

- `PatientPhotoPane` – Extends `JPanel` class, displays bigger patient picture and contains buttons to return to a list of patient, continue to examine the patient or loading new picture, which was recently taken by an embedded camera.
- `PopupEnabledWindow` – Interface, contains method `addToPopup(PopupMenu popupMenu)`. For components to add context to be displayed in a Pop-up window.
- `PopupCreate` – Extends `PopupMenu` class. Realize the Pop-up menu for classes, that implements `PopupEnabledWindow` interface. Used in measure forms to toggle full screen mode and to input numbers into an input form.
- `SavingComponent` – Extends `JComponent` class, overrides `paint` method. Makes a screen that is being displayed while sending filled data back to the server.
- `Main` – Contains method `main`, which starts the application. Create a `Frame` class instance for `LoginPane` screen, sets `MenuBar` for it and creates instance of the `ApplicationContext` class and calls its method to initialize login screen.

8. **painting.rsc** – Contains locale non-dependent images. Just an arrows to switch between pages of a measure form.
9. **painting.rsc.cz** – Contains images for Czech language subset (Images containing text info, like icons with connecting or saving process state).
10. **painting.rsc.en** – The same as above, but for English language subset.
11. **painting.threads** – Updates screen during process of sending data back to the server.  
Classes:
  - `SavingRefreshThread` – Extends `Thread` class. Periodically sleeps and wakes up and calls `paint` method of `SavingComponent` instance to refresh current status of saving process on screen. Also checks, if the saving process doesn't take too long and after some time, it forces to end the saving process.
  - `ExecuteThread` – Extends `Thread` class. Initialize the saving process (posting the Visit to the server). When it is over, stops the `SavingRefreshThread` instance.

### 5.1.3 GUI structure

The schema of GUI is displayed on Illustration 4 and described below.



**Group Frame** is frame declared in a `GroupComponent` class. `GroupComponent` class contains a list of related measurements (which are displayed as a **Form Component**). **North Panel** displays group measurement label (name) and also arrows (**L. Arrow**, **R. Arrow**), if more measurements are involved. **GroupComponent** is `GroupComponent` class, that extends `Panel` class, or `AbstractComponentPane` class respectively. Before showing the form, method from `ApplicationContext` class is called and as a parameter gets **Form Component** and does some adjustments related to sizing. This method checks the size of a **Form Component**. If it overlaps the parent components borders in width or height (or both), it creates horizontal scrollbar

**SH**, or vertical scrollbar **SV**, or both. It implements listeners of these scrollbars, track its changes and sets bounds of **Form Component**, thus scrolling the panel. To avoid counting “pixel exact” size of panel on a new position, the **Form Component** is placed into another panel called **Form-wrap Component**. Otherwise, the panel would be covering scrollbars (or components placed on the same or lower level generally) when moving up or right direction. But this way, the scrollbars are always above the **Form Component**, which is sliding below scrollbars and not above. And all this structure is put into another single **Result Component** panel and returned to a **GroupComponent**.

If the **Form Component** fits into its parent, no panels and modification are made and **Form Component** is returned.

#### 5.1.4 Localization

Localization is solved with a Singleton class `TranslateSingleton`, which extends `PropertyResourceBundle`, and contains one method `getInstance`. The language is set the same, as the environment language. The default language is set to English. So, when placing text, or image (that is locale dependent), it looks like this:

```
TranslateSingleton.getInstance().getString(msg);
```

In the case when placing an image, there are more images (one for each language subset), that differ by name for each language subset, so name of appropriate language variation is returned.

So far, English and Czech language are supported.

#### 5.1.5 Communication

Communication is made by calling servlet with parameters:

- `loginName` – User login name.
- `loginPass` – Password, encrypted with hashing algorithm.
- `type` – Upon its string value is decided, what action is to be performed. Available choices are to download the visit data, upload filled visit data, upload taken images.

Response codes are processed in the case of error. The error cases distinguished are *wrong login info*, *connection error*.

### **5.1.6 Storage model**

The application checks, if folder structure exists during the start and creates if it is not. It is located in user's home folder. In a future, it will be probably moved into the application folder. Since function `mkdirs()` is not working, each folder has to be checked and created from top down. The top folder name is *k4care* and contains sub-folders:

- *images* – Contains sub-folder *doc\_images*, which is for images involved in forms. Used, for example, to paint on image of a body, where patient suffers pain. Then, there is the *patient\_images* sub-folder, which contains images, that involves a photo documentation – images of a patient taken during the examination.
- *measurement* – Contains downloaded and parsed XML files, that contains info for patient examination. One patient is stored in a single file. Then, files with historical values are presented. Each historical examination is in a single history XML file.
- *service* – Contains an application related informations. Like encrypted password to login, user info, etc.

## **5.2 Application description – end user view**

Most of the application can be handled with keyboard keys *up*, *down*, *enter*. There are a few swing components, because of its advanced functionality, though it doesn't support focus subsystem. On solution is being worked on (probably by creating generic components).



## 5.2.1 Screens description

**Login screen:** Contains input fields for user name and password, to log into the application and authorize to the server as well. There is also check box, that is to set the offline mode, which is used, when the data are already downloaded and patient is examined. Then, there is an icon in the bottom right corner to log in. The data for offline mode is stored in a XML files on solid storage. Illustration 5.

**Patients screen:** Contains the list of downloaded patients with photo of selected patient in the upper right corner. There is an icon in bottom right corner to start the examination of a patient. Also, there is menu defined, that contains items *finish* to quit the application and *Save result ...* to upload already performed examinations to the server. Illustration 6

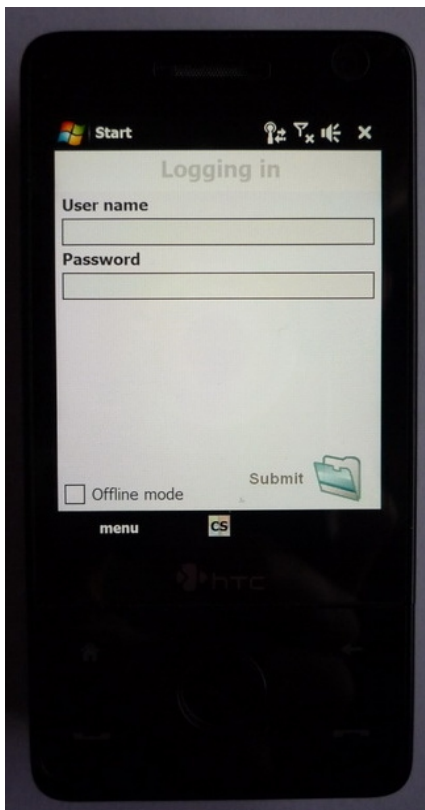


Illustration 5: Login screen.

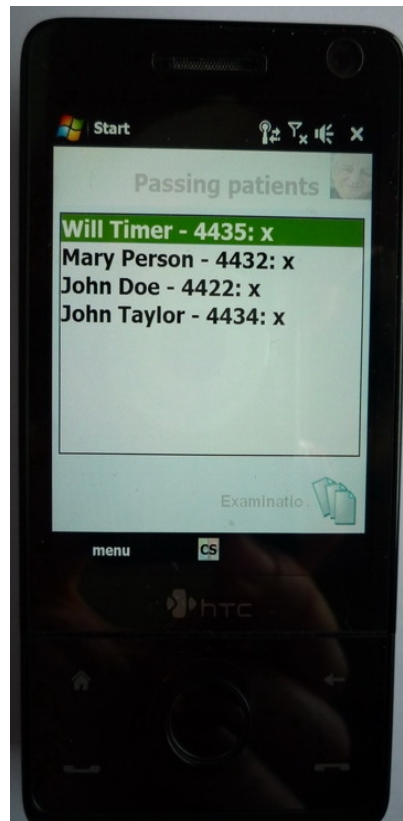
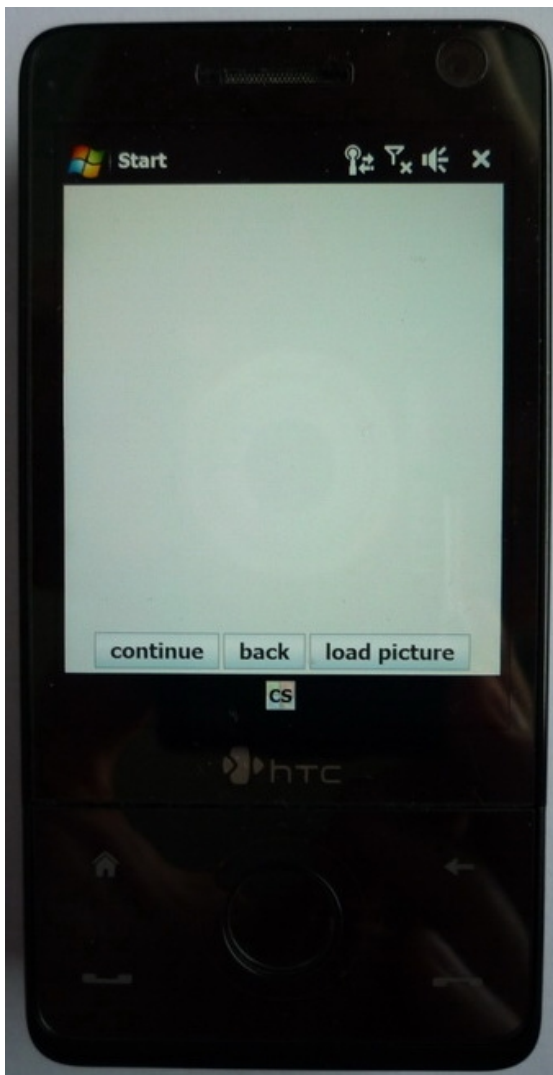


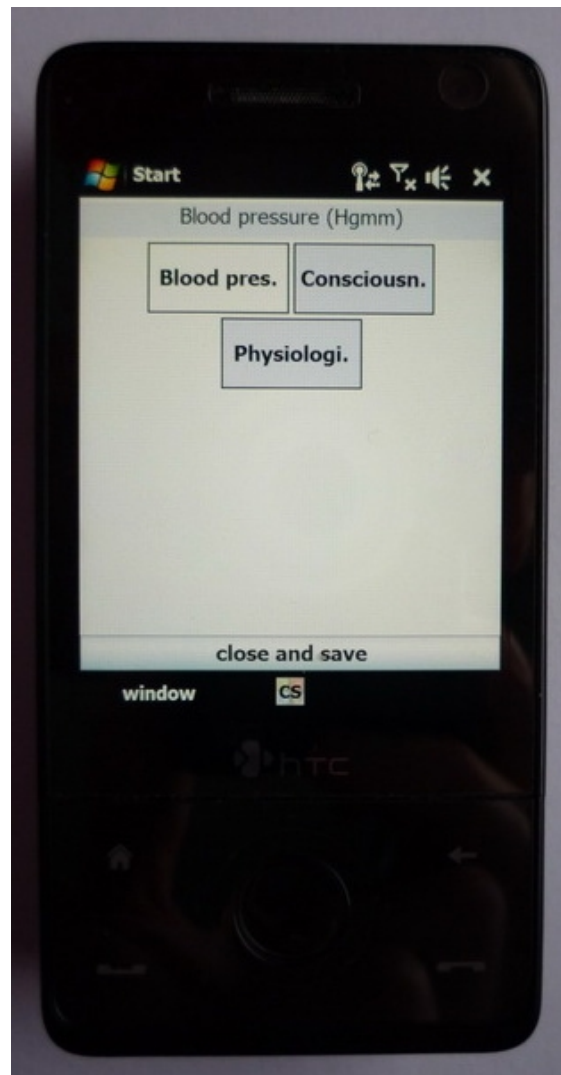
Illustration 6: Patients screen.

**Patient photo screen:** Screen with big patient photo over the whole screen. Also contains buttons *continue*, to continue to examination, *back* to return to a patients list and *load picture*, to load picture recently taken from embedded camera. Illustration 7

**List of measurements:** On the top of the screen, there is label, which contains the name of currently focused measurement, the main (and the most) space occupies buttons, that represents measurement. The state of filling the single measurement is indicated with a background color of each button separately. On the bottom of the screen, there is the *save* button to save current work and move one level up, into the list of patients. Illustration 8



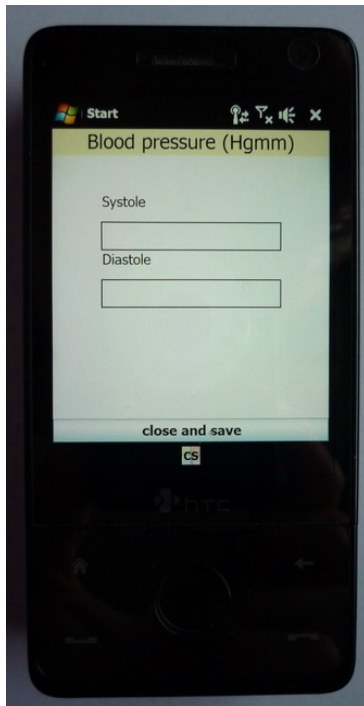
*Illustration 7: Patient photo screen.*



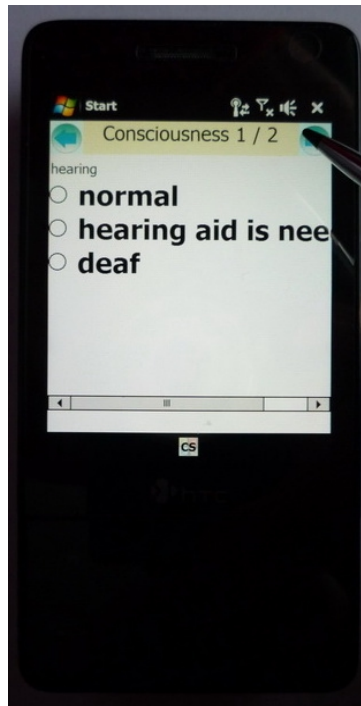
*Illustration 8: List of measurements screen. Currently is focused Blood pressure measurement.*

**Single measurement screen** – Contains name of measurement in the middle top of the screen and page number, if the measurement had split into more pages respectively. If more pages presented, there are arrows to switch between pages. The most space occupies the measurement

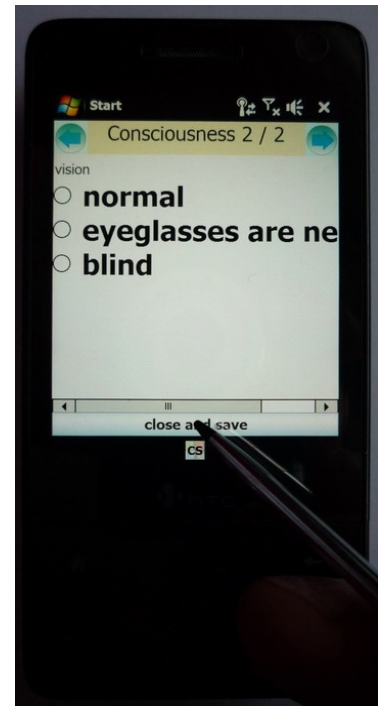
itself. On the last page bottom, there is the *save* button, that saves changes and move one level higher, displaying measurements screen. Illustration 9, Illustration 10, Illustration 11



*Illustration 9: Single measurement screen – input fields.*



*Illustration 10: Single measurement screen – radiobuttons, 1st page.*



*Illustration 11: Single measurement screen – radiobuttons, 2nd page.*

## 5.2.2 Scenario – Logging in, filling the document, saving

Typical scenario of the project is going to be described in this chapter. On the server side [3], a doctor selects measurements for each patient. A home care personnel selects some patients from the list of patients to be examined (since there are suppose to be more HC personnel, splitting all the patients among them).



Search patient:  Search

All patients  My patients

	Surname	First name	Identification number	Doctor in charge	
	Timer	Will	784587/1245	Karel Kováč	Patient card
	Taylor	John	745478/7895	Karel Kováč	Patient card
	Stastny	Pavel	784587/1245	Karel Kováč	Patient card
	Smith	John	687445/7983	Karel Kováč	Patient card
	Person	Mary	821605/1647	Karel Kováč	Patient card

Illustration 12: Patients listing, clicking on "Patient card" of Will Timer.



**Full name:**  
Will Timer

**PublicID:**  
784587/1245

**Contact address:**  
Canterbury 1245

**Birth date:**  
Mon Mar 02 00:00:00 CET 2009

**patient ID:**  
4435

Show records graphs

Measure modification

Disable this patient

visit summary

Fri Mar 13 09:56:25 CET 2009
Thu Mar 12 22:50:59 CET 2009
Thu Mar 12 20:31:05 CET 2009
Thu Mar 12 20:26:46 CET 2009
Thu Mar 12 20:24:21 CET 2009
Thu Mar 12 20:21:45 CET 2009
Thu Mar 05 13:18:13 CET 2009
Thu Mar 05 12:34:12 CET 2009
Thu Mar 05 12:12:08 CET 2009
Thu Mar 05 11:20:42 CET 2009

Illustration 13: Patient card. Clicking on "Measure modification", to select measurements to be performed.



patient name: Will Timer patient ID: 784587/1245

measure type			is active	frequency	seriousness	
OA1	Hodnoty životních funkcí	Hodnoty životních funkcí	<input type="checkbox"/>	Daily	Very high	View
OD_GS	Glasgowská stupnice	Glasgowská stupnice	<input type="checkbox"/>	Daily	Very high	View
OA10	Stav kůže	Stav kůže	<input type="checkbox"/>	Daily	Very high	View
OA12	Omezení schopností	Omezení schopností	<input type="checkbox"/>	Daily	Very high	View
OA11	Bolest	Bolest	<input type="checkbox"/>	Daily	Very high	View
D11g	NortonScale	NortonScale	<input type="checkbox"/>	Daily	Very high	View
OA3	Vědomí, komunikace	Vědomí, komunikace	<input type="checkbox"/>	Daily	Very high	View
OA2	Psychický stav	Psychický stav	<input type="checkbox"/>	Daily	Very high	View
D13_3	Consciousness	Consciousness	<input checked="" type="checkbox"/>	Daily	Very high	View
OA5	Riziko pádu	Riziko pádu	<input type="checkbox"/>	Daily	Very high	View
D13_2	Physiological Functions	Physiological Functions	<input checked="" type="checkbox"/>	Daily	Very high	View
D13_1	Blood pressure (Hgmm)	Blood pressure (Hgmm)	<input checked="" type="checkbox"/>	Daily	Very high	View

Illustration 14: Set of possible measurements. Checking it will select it to be performed. Clicking the view button displays the measurement(Consciousness).



Consciousness

hearing

- normal
- hearing aid is needed
- deaf

vision

- normal
- eyeglasses are needed
- blind

Illustration 15: Displaying measurement (Consciousness).

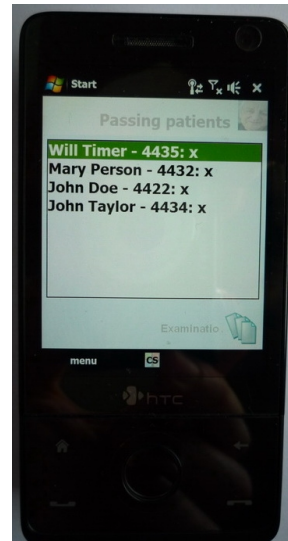
Now, the HC personnel can start using PDA, which is to log in, download data and then do the examinations:



*Illustration 16: Clicking “Submit” icon after setting the login info.*

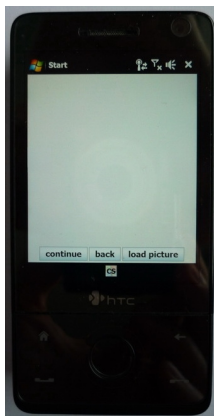


*Illustration 17: Data processing.*

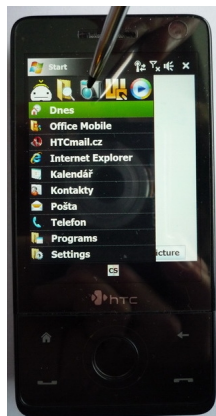


*Illustration 18: Downloaded list of patients.*

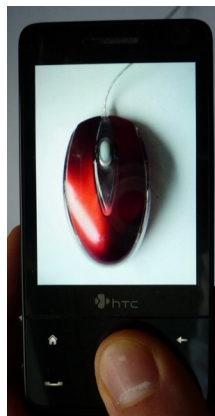
The data is downloaded, lets process to an examination. It start with big patient photo. If it is not presented, it can be taken on the place.



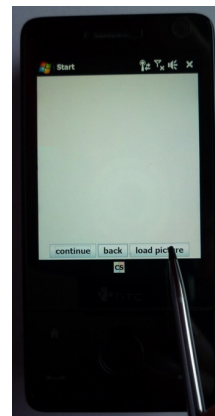
*Illustration 19: Photo panel. Shows after pressing the examination button. This one is empty, with no photo.*



*Illustration 20: Clicking on “start” (top left) and selecting the camera to take a new picture.*



*Illustration 21: Taking a picture (of the computer mouse, as an example).*

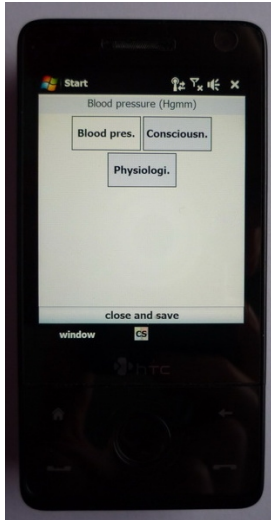


*Illustration 22: Clicking the “Load picture” button to load the photo recently taken.*

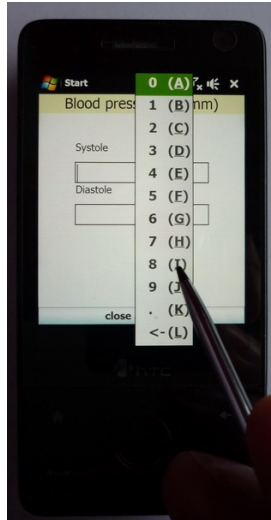


*Illustration 23: Clicking the “Continue” button after loading the picture.*

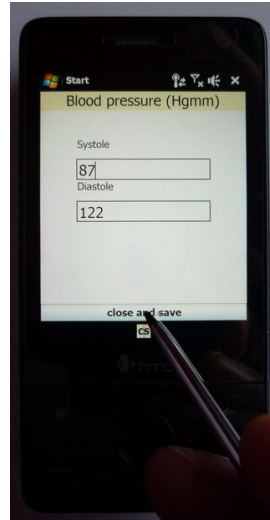
After the initial photo screen, the measurement itself follows.



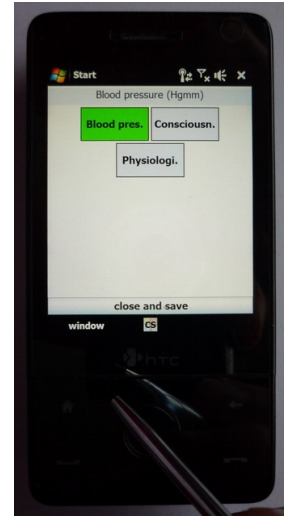
*Illustration 24:*  
Group of measurements, presented by buttons.



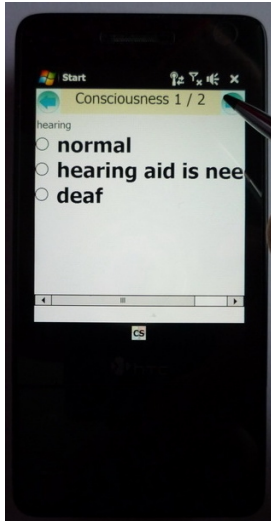
*Illustration 25:*  
Filling the measurement. Pop up window is showing to enter numbers.



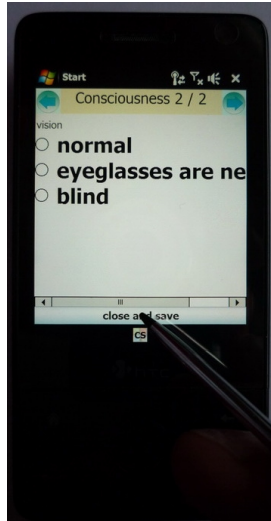
*Illustration 26:*  
Saving the filled measurement.



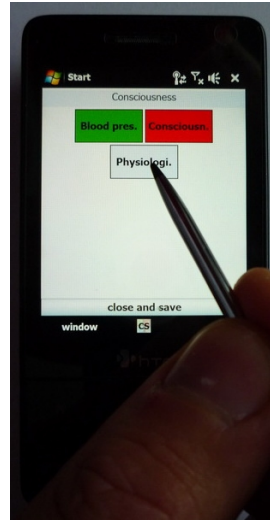
*Illustration 27:*  
Group of measurements with first measurement filled (green color).



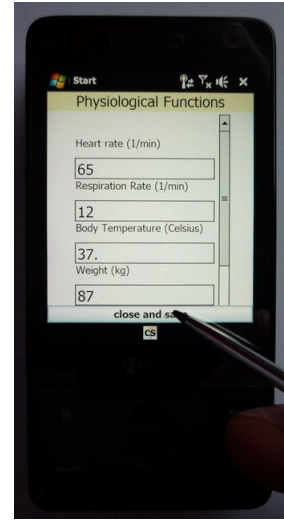
*Illustration 28:*  
Looking through the second measurement, 1st page of two.



*Illustration 29:*  
Looking through the 2nd page of measurement and saving empty measurement.

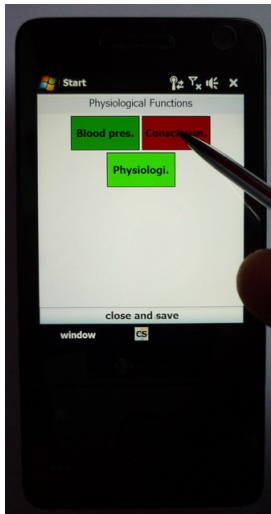


*Illustration 30:*  
Clicking on 3. measurement. 2. measurement is not filled, signaling with red color of button. h) Saving the filled, third measurement.

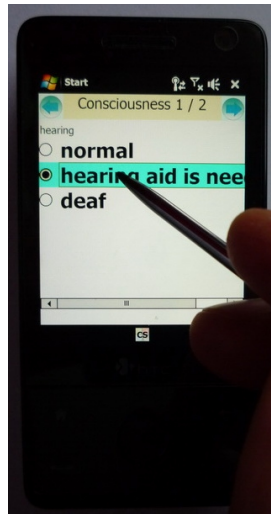


*Illustration 31:*  
Saving the filled, third measurement.

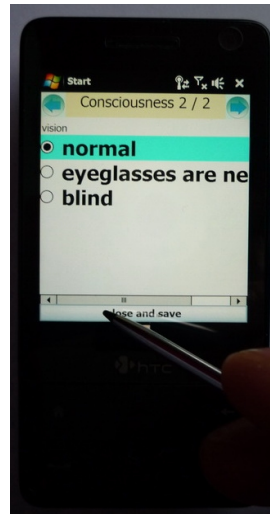
Now, lets finish the measurements not yet done and save it and upload to the server.



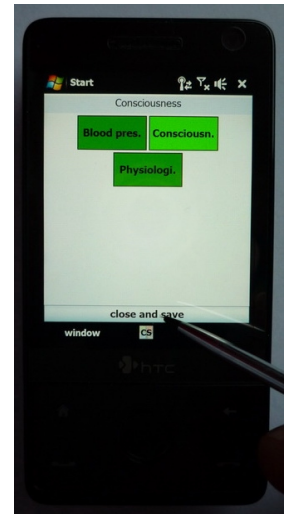
*Illustration 32:  
Returning to the  
second (not filled)  
measurement.*



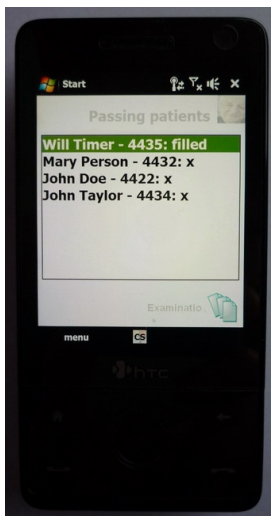
*Illustration 33:  
Selecting an option  
on 1st page.*



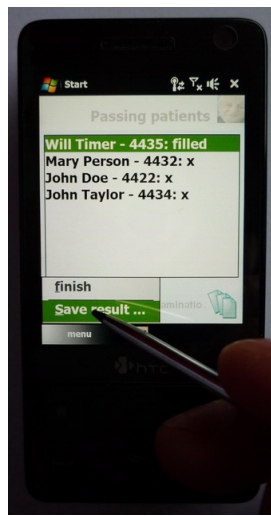
*Illustration 34:  
Selected an option  
on 2nd page and  
saving.*



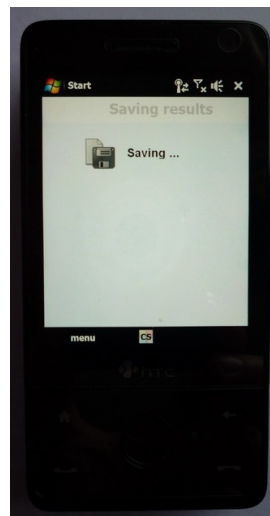
*Illustration 35:  
Group of  
measurements, all  
filled. Saving all for  
the current patient.*



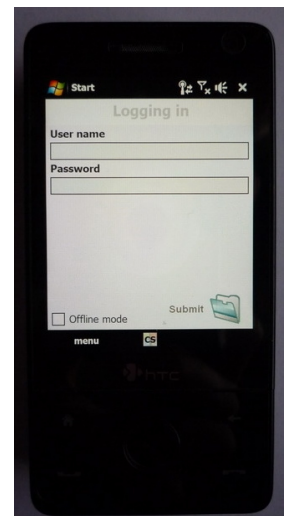
*Illustration 36: List  
of patient. For Will  
Timer indicating,  
that the document is  
filled.*



*Illustration 37:  
Saving (and  
updating) data of  
filled patients (only)  
to the server.*



*Illustration 38:  
Saving in progress.*



*Illustration 39:  
Login screen after  
saving process is  
finished.*



## 5.3 *Implementation issues, constructions to mention*

Some issues and their solutions are going to be mentioned in this chapter. These are result of used hardware and software combination. Usually, it is when standard object-oriented approach couldn't be used, classes and methods, that are defined in J2SE didn't work properly on Mysaifu [9], or at all. So, the solution was to program these functionalities from scratch.

### 5.3.1 **Data processing**

Method *parse* from class *javax.xml.parsers.DocumentBuilder* was originally used for processing the XML data. It returns DOM (Document Object Model) as an instance of *org.w3c.dom.Document* class. But performance on PDA device was too low. Parsing approximately 100kB XML file, which had contained very little data of 9 patients, lasted around 60 seconds. Such a low speed is a result of that Mysaifu, as a JVM for PDA, is designed to work with limited resources (especially little RAM, and CPU speed as well). Thus, missing a lot of techniques and robustness to speed up its run, which are implemented in J2SE.

So, the idea to speed up parsing and building DOM, is to restrict creating or using objects, restrict the size of objects to work with, by removing unnecessary methods and unneeded functionalities.

#### 5.3.1.1 **Parser**

For parsing, already done solution was used. It's called QDParser [12], which is fast, event driven, non-validating parser. The source code, with comments, is just above 300 lines. It has some restricted functionalities, like:

- cannot read custom entity definitions. Only the standard ones available are: **&(&amp;);**, **<(&lt;);**, **>(&gt;);**, **'(&apos;);**, and **“(&quot;);**.
- doesn't support conditional sections, **<![INCLUDE[ ... ]]>** or **<![IGNORE[ ... ]]>**.
- since it doesn't process any attribute declarations, all attribute types to be CDATA. Thus, a

simpler *java.util.Hashtable* storing key/value information is used to hold an element's attribute list, instead of *org.xml.sax.AttributeList*.

But all the missing features is possible to handle with document preprocessing. So, the parser itself is able to:

- It recognizes all the elements' start and end tags.
- It lists attributes, where attribute values can be enclosed in single or double quotes.
- It recognizes the `<[CDATA[ . . . ]]>` construct.
- It recognizes the standard entities: `&(&amp;);`, `< (&lt;)`, `> (&gt;)`, `' (&apos;)`, and `“ (&quot;)` as well as numeric entities.
- It maps lines ending in `\r\n` and `\r` to `\n` on input, in accordance with the XML Specification, Section 2.11 [13].

The source code contains just one class with one public static method `parse (Reader r)` and one interface `DocHandler`, that declares 5 methods, called by parser. The methods are:

- `startElement (String tag, Hashtable attributes)`
- `endElement (String tag)`
- `startDocument ()`
- `endDocument ()`
- `text (string elementValue)`

The names of methods are self-explaining.

So, to process a XML document, it's required to create a class, that implements `DocHandler` interface and builds a memory representation of a document.

### 5.3.1.2 *DocHandler class*

Most of its logic is in method `startElement`. For each tag it always creates a class instance with an appropriate info (tag type, depth, attributes (for element type tag), indexes to subsidiary arrays – will be explained later).

### 5.3.1.3 *Element (Tag) class*

The document itself is stored in a `java.util.ArrayList` (from now on, I will refer to it as “`ArrayList<tags>`”) class, tags are represented by a generic class, which contains following data types and their getters only:

- `int type` – Tag type (element, text, attribute, attribute value).
- `String value` – Value/name of a tag.
- `int attributesCount` – Number of attributes.
- `int depth` – Depth in a document hierarchy.
- `int address` – Index this tag in an subsidiary array of element indexes (which are indexes to `ArrayList<tags>`). If the tag is not an element, then it contains an address of the element it belongs to.
- `int addressInvTXT` – Index this tag in an subsidiary array of element indexes and text values indexes.

### 5.3.1.4 *Document model class*

The document is represented by `ESIADocument` class. It contains the `ArrayList<tags>` with stored tags and some subsidiary arrays, that holds indexes of elements for easy access (Illustration 41):

- `int[] elementOpenArray, elementAndTXTOpenArray` – Addressing

elements only and elements with its text info in an `ArrayList<SSPElement>`.

- `int[] elementDepthArray, elementAndTXTDepthArray` – Holds the info, how deep in the document hierarchy these elements are.

After parsing is done, method `childrenIdent` is called to initialize arrays with indexes. The method goes through tags stored in the `ArrayList<tags>` and detecting element tags and text tags (passing over attributes and attribute values with no action) and copies its indexes in the `ArrayList<tags>` to an subsidiary arrays. Then, parsing and building the document model is done. Arrays are useful for methods, that works with the document, which are:

- `int[] getRootChildren()` – Returns indexes (addressing to the `ArrayList<tags>`) of children of the root element.
- `int[] getChildren(int parent)` – Returns indexes of children of parent element, `parent` is passed as a parameter.
- `int[] getChildrenWithTXT(int parent)` – Return indexes of children elements and also their text values indexes.
- `String getTexContent(int element)` – Returns text value of the element.
- `String getAttribute(int element, String attrName)` – Returns value of the attribute tag.
- `int getAddrAttrValue(int index, String attrName)` – Returns index to the attribute value.
- `int[] getAttributes(int element)` – Returns indexes of attributes of the element.
- `String getAttributeVal(int attribute)` – Returns value of the attribute tag.
- `SSPElement getSSPElement(int element)` – Returns element object (generally, it can be any tag – element, text value, attribute, attribute value. But it is used only for elements).
- `String getSSPElementName(int element)` – Derived from previous method, return only name of the element.
- `void setSSPElementName(int element, String value)` – Stores value of

a tag generally. Used to store element text value, or attribute value.

- `int getSSPElementType(int element)` – Returns the type of the tag. It can be element, text value, attribute, attribute value.

The main idea of such a organization is, that when getting data out of it (children, attributes, etc.), all needed to do, is to go through the subsidiary arrays, or the `ArrayList<tags>` and reading types of tags and checking theirs depth. Most of it are `int` based operations.

This can be done, because when parsing the document, passing through a document has order of DFS (Depth First Search) algorithm, in terms of graph theory. And the same order has the resulting `ArrayList<tags>`. So, the algorithm needed to return children of a parent looks something like this:

```
1. int parentIndex = ArrayList<tags>.get(parent).getAddress()
2. int parentDepth = elementDepthArray[parentIndex]
3. int nextElementIndex = elementOpenArray[parentIndex+1]
4. int childDepthTemp = elementDepthArray[nextElementIndex]
5. while(parentDepth < childDepthTemp) {
6.     if (childDepthTemp == parentDepth+1)
7.         add_nextElementIndex_to_result
8.         nextElementIndex = nextElementIndex + 1;
```

The above algorithm works well, because elements following the parent are all members of its subtree, until it reaches a tag, that has the same depth, or lower depth (thus, its on the same level or higher in the document hierarchy). This is because of DFS order of the document in an `ArrayList<tags>`, and other subsidiary arrays has the same order, as well. So, the algorithm Illustration 40 does the following:

- line 1 – Int address of parent is used to get element from `ArrayList<tags>` and its index (address) in subsidiary arrays `elementOpenArray` and `elementDepthArray`.
- Line 2 – Remembers the parent's depth.
- Line 3 – Reads the first element following the parent.
- Line 4 – Reads its depth.
- Line 5 – Starts a cycle, which runs for all elements in parent's subtree (for all consecutive

elements with depths lower than parent). From this point, the algorithm searches through `elementOpenArray` until it reaches the end depth condition.

- Line 6 – Checks, if the depth is bigger just by 1 from the parent's depth. If it is so, then it is a child element, so will add it to a result array (line 7).
- Line 7 – Adds element to a result.
- Line 8 – Moves to next element, just by moving to next index in `elementOpenArray`.

There is no need to check a tag type, because all tags are elements, since `elementOpenArray` contains element indexes only. It's fast, because only indexes from `int[]` are read and compared. Only disadvantage is, that during searching, it needs to go through the whole subtree, and not only children. So, the worst case performance is linear (when the parent is the root element of a document, because it needs to go through all elements) dependent on elements (not tags), the best case performance is constant (hitting the leaf).

Getting text values, attributes, attribute values works on a similar principle. Only, there is not subsidiary `int[]` array available (like for element type tags), so `ArrayList<tags>` is searched and tag type validation needs to be performed for each one. But, it's searched only a few following consecutive tags, not the whole subtree, because attributes and values are just behind the element.

The representation of the document and getting children elements is showed on Illustration 41 and Illustration 42.

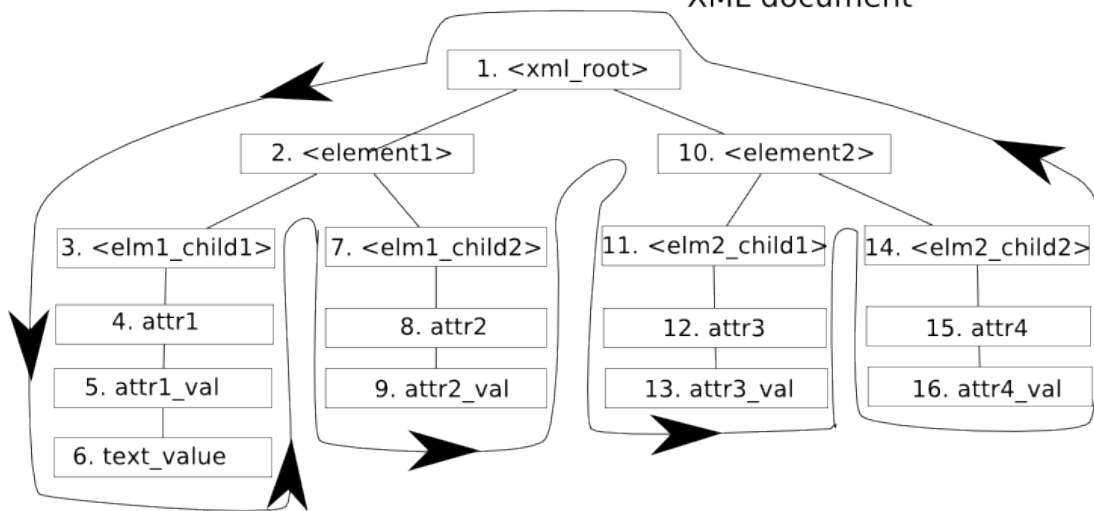
### Example XML document

```

<xml_root>
<element1>
  <elm1_child1 attr1="attr_val1">
    text_value
  </elm1_child1>
  <elm1_child2 attr2="attr_val2" />
</element1>
<element2>
  <elm2_child1 attr3="attr_val3" />
  <elm2_child2 attr4="attr_val4" />
</element2>
</xml_root>

```

### Order of parsing XML document



### Resulting order in an ArrayList<tags>

```

1. <xml_root>
2. <element1>
3. <elm1_child1>
4. attr1
5. attr1_val
6. text_value
7. <elm1_child2>
8. attr2
9. attr2_val
10. <element2>
11. <elm2_child1>
12. attr3
13. attr3_val
14. <elm2_child2>
15. attr4
16. attr4_val

```

### Subsidiary arrays, loaded after parsing

elementOpenArray

1	2	3	7	10	11	14
---	---	---	---	----	----	----

elementDepthArray

0	1	2	2	1	2	2
---	---	---	---	---	---	---

elementAndTXTOpenArr

1	2	3	6	7	10	11	14
---	---	---	---	---	----	----	----

elementAndTXTDepthArr

0	1	2	2	3	1	2	2
---	---	---	---	---	---	---	---

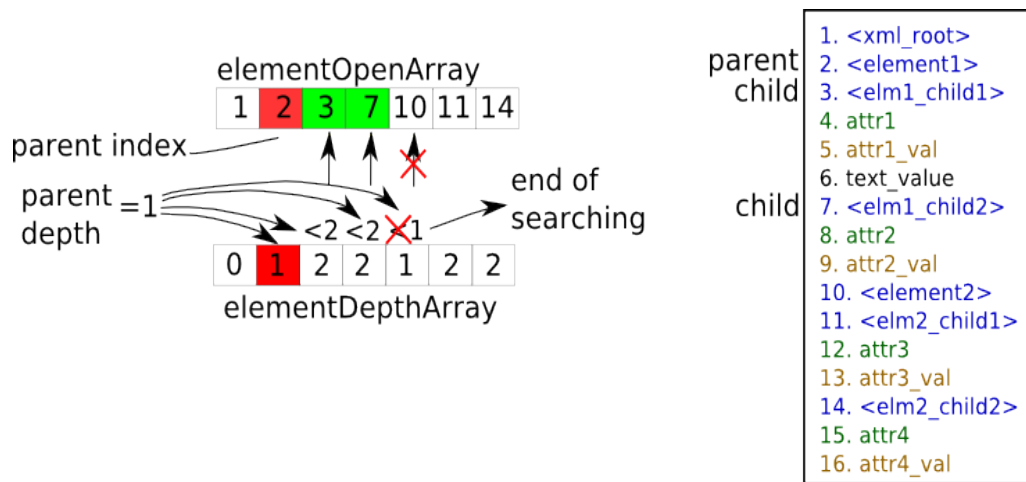


Illustration 42: searching a child of a element on position 2. The result are elements with indexes 3 and 7.

Some benchmark results with original and new solutions are in Illustration 43

	<i>org.w3c.dom</i>	<u><i>QDParser + ESIADocument</i></u>
Processing 48kB XML file	28s	6s
Displaying patient – parse patients document + build GUI from its context	10s	8.5s

Illustration 43: Comparing benchmarks of parsing and building document using an old (*org.w3c.dom*) and new tools (average of 3 test, varying about 0.5 second).

### 5.3.2 GUI

With Mysaifu, there were two main possibilities, what widget toolkit (graphical elements to be used for designing the application) to use, which are AWT or Swing. Swing has nicer look, allows advanced functionalities and effects, while AWT is simpler, faster, needs less memory. Several things were programmed using swing. It uncovered a few defects:

- Reaction time on click – Clicking on swing component, like a button, invokes a response after a split second, which is noticeable and annoying.
- `JScrollPane` - Component, which acts as `JPanel` class, but if components placed into it occupy more space, than the available space on `JScrollPane`, then scrollbars will



appear. The scrolling is very slow and with lagging.

- The focus subsystem is missing – it is possible to do it manually, like always remember, what button is selected and listen for input from keyboard and do the action. But there is still missing appearance of focused component.

So, after this experience, swing was thrown away. There were also some attempts with mixing both toolkits, but it bring up some troubles. When mixing swing-lightweight and AWT-heavyweight components [14], a lightweight components might have transparent pixels, while heavyweight is always opaque. On PC, some functional fragments of code worked, but it was not working on PDA.

After this, the application was switched to AWT, which is performing well enough. Though, there were also problems with the auto-scrolling component `ScrollPane` – similar functionality like `JScrollPane`. Scrolling is fast, but it is not updating new context during scrolling and old positions remains and are add to new ones, thus, resulting in indistinguishable mixture of all previous screens and a new screen. Therefore, scrolling is implemented via scrollbars, listening their events and moving with given component (`Panel`). The size of component movement is dependent on size of scrollbar movement. The need for scrollbars is determined during the setting of GUI for the patient and is determined for each `GroupComponent` separately.

Then, there are various problems concerning components aligning, components hiding, font changing, bad context overdrawing, etc. It doesn't really impact functionality, though it is annoying. It is result of combination such an experimental JVM and complex structure of the application. And the complex structure is result of demanded functionalities and some additive structures and code to speed up the application.

Another problems are resulting from varying DPI on different devices and various resolutions. The common DPI values are ranging from 72 and 192 (our two devices, that we used, has a 96 DPI and 192 DPI). For absolute positioning, the position of the first one is twice as far from the beginning, as in the higher value case. Also, the sizes of icons proportionally changes. It is resolved by choosing the icons size to look naturally on all common DPI values. Also, resolutions of the display vary among devices. On PC, it doesn't really matter, because any application size can be set. But on PDA, size setting doesn't work. Impossibility to set the the size is not that big problem, since the size would be chosen as big as possible anyway. So, it has the size of screen, restricted from top and bottom by TaskBars. But it needs to be considered when doing absolute

positioning. There is ahead known resolution of the display and sizes of icons, so icons are placed to have an absolute certain distance from corresponding border. Then, it looks good, because icon, which is, for example, in bottom right corner, will always be in bottom right corner (with the distance 10 pixels from bottom and right border of the screen).

### **5.3.3 Photo documentation**

The scenario is, that personnel turns on the camera anytime during patient examination and takes wanted photos. When the camera is closed, the application shows up. Loading is implemented as a part of measure form, in the last component (measure screen). It is a standalone Frame, which reads images from embedded camera storage, displays it and if it is confirmed (by pressing a button) it is moved to a specified folder with an appropriate name, (comprising from patient ID and incrementing number. Then, next image is read.

## **5.4 Portability**

The application is created in J2SE, thus, the code is runnable over many platforms, from PDA using the Windows Mobile 2003 higher, and on all platforms supporting J2SE. Though, there are a few limitations.

As mentioned in chapter 5.3.2 GUI, there are various problems with focus, aligning, covering and then problems with varying DPI and resolutions. None of these problems makes the application unusable, it is just unpleasant (and it is being worked on universal solution).

Only the limitations are speed of CPU, or overall speed respectively. It is related to a XML document parsing but that is processed right after the document for all patients downloaded, and it shouldn't take more, than a couple of minutes. And this can be done on the way to the first patient. Another possible issue is the size and resolution of the display. But as mentioned previously, the application was tested on device with the display size 2.8" and resolution 240x320 pixels. That is an acceptable minimum for this application. Though, there are not many devices with lower resolution or display size.

To sum this up, the application would run on any mobile device with screen size 2.8" and

resolution at least 240x320 pixels and then on any platform supporting J2SE, which are Windows OS, Unix based OS, Apple Macintosh, etc.

## 6 Conclusion

This work is part of ICT based system, which is integrating EHR and home care services for senior citizens (or generally any patient, that might need it) in a single system. That brings a great advantage, since health professionals can check patient's health status, drugs currently taken, their health status related images and all of that from past as well. Then, they can order and edit and request home care services to be done and simple examinations as well.

A mobile device, like a PDA, plays an important role in this project. It is an electronic tool, that acts at the point of care. Currently, nurses use paper forms, which can grow a lot during history, their accessibility and selectivity is limited and complicated, when searching through past forms. The paper forms contains a big group of measurements, though only few of these (or one measurement only) might be needed. So, the PDA device brings great advantages. It allows home care professionals to easily accept tasks and notes from a doctor at one time and perform an actual examinations, capture photo documentation, add general notes for a doctor. Then, upload the data back by a single button click anywhere, where is an Internet connection.

There were many problems during the application development with the experimental platform porting J2SE on PDA devices yet they were all solved successfully. Speed issues were addressed with custom solutions showing big performance grow. The look of the application and detail functionalities were consulted during the development with doctors from General Hospital in Prague in order to meet their needs and the desired changes were implemented. There is going to be a testing phase, when a few nurses from General Hospital in Prague are going to use this in a real environment, which should be launched during this summer.

Great advantage is, that it is developed in J2SE Java platform, therefore it is portable to many different operating systems and platforms.

## 7 Citation and references

- [1] K4CARE. [cit. 2009-05-17]. <http://www.k4care.net/>
- [2] IZIP. [cit. 2009-05-17]. <http://www.izip.cz/>
- [3] HORA, Ivo, K4Home care Web Interface, Praha, 2009. Diploma thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybernetics
- [4] VistA Imaging Overview. [cit. 2009-05-17]. <http://www1.va.gov/imaging/page.cfm?pg=3>
- [5] Augustyniak, Piotr. PDA-based system for cardiology home care and pregnancy monitoring. Krakov. AGH University of Science and Technology in Krakov, Institute of Automatics. [cit. 2009-05-17]. <http://home.agh.edu.pl/~august/pub/pdf/p66.pdf>
- [6] VITAE Care PDA. CSC Scandihealth A/S. Aarhus, Denmark. [cit. 2009-05-17]. [http://www.scandihealth.dk/Losninger/Losninger\\_produktml\\_pdf/VITAE\\_Suite\\_UK/UK-VITAE-Care-PDA.pdf](http://www.scandihealth.dk/Losninger/Losninger_produktml_pdf/VITAE_Suite_UK/UK-VITAE-Care-PDA.pdf)
- [7] HealthForceOntario. [cit. 2009-05-17]. <http://www.healthforceontario.ca/>
- [8] HealthForceOntario. Ontario, 2008. [cit. 2009-05-17]. <http://www.homecareontario.ca/public/docs/releases/2008/ohca-applauds-personal-digital-assistant-award-may-08.pdf>
- [9] Mysaifu JVM. [cit. 2009-05-17]. [http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index\\_en.html](http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html)
- [10] HTC Corporation. HTC P3300. [cit. 2009-05-17]. <http://www.htc.com/cz/product.aspx?id=11684>
- [11] HTC Corporation. HTC Touch Pro. [cit. 2009-05-17]. <http://www.htc.com/cz/product/touchpro/specification.html>
- [12] Brandt, Steven R. Java Tip 128: Create a quick-and-dirty XML parser. JavaWorld[online]. 31. May 2002. [cit. 2009-05-17]. <http://www.javaworld.com/javatips/jw-javatip128.html?page=1>
- [13] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation 26 Novemer 2008. [cit. 2009-05-17]. <http://www.w3.org/TR/REC-xml/>
- [14] Fowler, Amy. Mixing heavy and light components. [cit. 2009-05-17]. <http://java.sun.com/products/jfc/tsc/articles/mixing/>
- [15] Java™ 2 Platform, Standard Edition, v 1.4.2, API Specification. Sun Microsystems.[cit. 2009-05-17]. <http://java.sun.com/j2se/1.4.2/docs/api/>
- [16] Java™ 2 Platform Standard Edition 5.0, API Specification. Sun Microsystems. [cit. 2009-05-17]. <http://java.sun.com/j2se/1.5.0/docs/api/>

[17] Health Level Seven. [cit. 2009-05-19] <http://www.hl7.org/about/hl7about.htm>