VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

CREATION OF MULTIMEDIA CONTROL SYSTEM IN GNU/LINUX

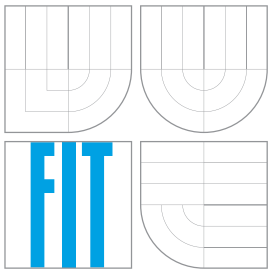
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL MINÁŘ

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VYTVOŘENÍ MULTIMEDIÁLNÍHO OVLÁDACÍHO SYSTÉMU V LINUXU

CREATION OF MULTIMEDIA CONTROL SYSTEM IN GNU/LINUX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MICHAL MINÁŘ
AUTHOR

VEDOUCÍ PRÁCE Doc. Ing., Dipl.-Ing. MARTIN DRAHANSKÝ, Ph.D.
SUPERVISOR

BRNO 2010

Abstrakt

Práce poukazuje na problémy týkající se vytvoření multimediálního systému, výběru jeho hardwarových a softwarových komponent, jeho zapojení do většího celku a vede k jejich řešení. Současně popisuje kroky zvolené v implementaci reálného systému.

Abstract

This thesis summarizes problems concerning the creation of multimedia system, selection of its hardware and software components, its inclusion into bigger system and provides advices leading to their solution. At the same time it describes implementation of such existing system.

Klíčová slova

dotykový panel, elektroinstalace, kalibrace, multimédia, linux

Keywords

touch panel, electroinstallation, calibration, multimedia, linux

Citace

Michal Minář: Creation of multimedia control system in GNU/Linux, bakalářská práce, Brno, FIT VUT v Brně, 2010

Creation of multimedia control system in GNU/Linux

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. Martina Drahanského

.....
Michal Minář
May 18, 2010

Poděkování

Tímto bych chtěl poděkovat řediteli společnosti Elko ep, Jiřímu Konečnému, za možnost podílení se na vývoji *imm* systému, za poskytnutí hardwarových komponent a možnosti práce přímo ve firmě. Dále bych chtěl poděkovat bc. Jiřímu Stýskalíkovi, Danielu Smičkovi a dalším z této společnosti za odbornou asistenci. Zvláště bych chtěl poděkovat ing. Miroslavu Kubízckovi za jeho ochotu, při zodpovídání mých dotazů, a pomoc. A v neposlední řadě mému vedoucímu doc. Martinu Drahanskému za jeho rady, co se týče této práce, a trpělivost.

Contents

Chapter 1

Preface

1.1 About

This bachelor thesis focuses on making touch panels part of bigger system such as *intelligent home* 1.2. It summarizes requirements on such devices, software components and user interface. Thesis contains description of existing functional system for *intelligent homes*, and concrete touch panels. It guides through the process of selecting suitable software components 3, installing them to target platform, designing and implementing user interface 4.

After reading this work, reader should be acquainted with possible problems, he/she might face, when trying to implement interactive user interface to integrated automated system. Thesis should give him/her hints, how to solve or evade these problems. Reader should also get notion on how real system for home automation works and what components is composed of.

Note that this is a truncated version of thesis. The content published here is made publicly available with permission from Elko Ep, ltd. The full version includes content, that is not meant for public reading, as it contains company's secrets.

1.2 Intelligent homes

Over decades electronic gadgets are providing us comfort of simplifying tedious daily tasks. They are steadily evolving to bring us new functionality, which we need or want. *Intelligent home* is a neat colligation of household appliances, embedded systems, mobile devices etc. forming an integrated system with artificial intelligence automatting [16] tasks connected with house keeping (for example heating, ventilation, security, houseplant watering, energy saving), enterteinment etc. Bringing this automation to homes is the next step in making electronic devices useful.

Among devices used in home automation belongs: time relays, time switches, staircase switches, dimmers, twilight switches, power and auxiliary relays, power supplies, controlling and signalling devices, thermostats, gsm gateways and much more. As amount of different appliances is ceaselessly increasing, there is necessity for centered, systematic control, providing user friendly interfaces to manipulate these devices on smaller or larger scale. One of systems mediating these features is INELS 2.1.

1.3 Interfaces

Interfaces to home automated systems provid users (in this case residents of home) ways to configure or to set state of particular devices. For example turn on and off the light bulb, set its intension, open a garage door or set currently playing FM radio to other station of player in other room. This process involves translation of user's presses of keys/screen to desired command, which is send across some kind of system bus to desired device, containing a microprocessor, setting or reading it's register's state.

Commonly used interfaces to *intelligent home*'s control systems are televisions, flat panels, touch panels, or just switches. Current direction of development aims mobile devices (smart phones) and web services for remote control, which allows for instance setting the house's room temperature to desired value for expected time of arrival of system user, or just controlling security measures, house state, when nobody's home. In the future we may also expect more systems supplied with voice recognition and probably another different ways of interacting with humans.

Chapter 2

System components

Touch panels, on which this thesis is focused on 2.5 are to be part of existing system of intelligent electroinstallation 2.1. This chapter describes what this system takes care of, what components are part of it and how can be configured to suit the needs of its users.

2.1 INELS

Means Intelligent electroinstallation. It's a system developed by Elko Ep, ltd. It's designed to control runnings of all kinds of structres from small family houses, apartments across administration spaces to large complex buildings. From ordinary electroinstallations it differs[3] mainly in configuration of sensors and actors (*units* later on), which is done after their connection with central unit by 2-wired bus, that is shared by all devices supporting INELS protocol (EPSNET 2.2). This greatly simplifies installation. *Units* can just be plugged anywhere to this bus and later be configured. They can also be plugged or removed anytime, even at runtime, which also applies to their configuration.

This system has a central unit 2.1.1. It acts as a mediator between higher artificial intelligence and *units*. Can be connected with pc, or other facility with ethernet cabel. Configuration is done in software application *INELS designer and Manager* 2.3 or through web interface available by built-in webserver.

2.1.1 Central unit

Will be refered to as *CPU* from now on. Currently there are two supported central units by *INELS*. It's CU2-1M or FOXTROT PLC from Teco corp. Both support communication by *EPSNET* 2.2. Both of them support communication over serial channels in multiple modes. For example FOXROT PLC supports:

EIO connection of additional framework modules

PC attaching of superior system[1].

PLC connection of multiple systems of TECOMAT/TECOREG for the purpose of quick data exchange.

MPC connection of multiple systems [2] for purpose of data exchange

UNI user channel for universal usage

MDB connection of superial system by protocol MODBUS

[1] PC or superior TECOMAT, TECOREG or EPSNET system
[2] *EPSNET multimaster* 2.2.1

PFB connection of PROFIBUS DP slave station

UDP handling of special submodules

DPS realization of station PROFIBUS DP slave

CAN attaching of station CANopen to PLC

CAS realization of station CANOPEN

CAB attaching of bus CAN

CSJ attaching of bus CAN

Modes *PC*, *PLC* and *MPC* are used with EPSNET 2.2 network. These modes are also used in communication over ethernet interface, which uses *EPSNET* udp packets. This interface can operate in multiple modes simultaneously. Modes *PC* and *MDB* are permanently active.

There is one more communication mode supported by this interface. It's *UNI*, which serves for transmission and reception of arbitrary data over network protocols TCP and UDP.

Last interface to mention is USB. Data exchange conforms to *PC* communication mode.

2.2 EPSNET

Is an industrial network used among other things in *INELS* system. There are two kinds of stations in EPSNET network:

master active participant, controls the communication

slave passive participant, it answers the questions of superior station

Communication between stations is based on "challenge-response" principle, where challenge can be generated only by *master* station. This allows connection of larger amount of participants through half-duplex interface RS-485.

2.2.1 Configuration

Two basic configurations are supported:

monomaster There is only one superior station in network.

multimaster There are more superior stations with several subsidiary.

monomaster

Most common configuration. If we need to exchange data among subsidiary stations, we must do so with help of *master* station, which can be PC with driver providing support for communication with systems TECOMAT/TECOREG or station TECOMAT/TECOREG with serial channel in MAS or *MPC* [3] modes.

Subsidiary stations are systems TECOMAT/TECOREG with serial channel in *PC* mode. There may be 126 of them at most.

[3] transmittion of token message (paremeter MT) is Off

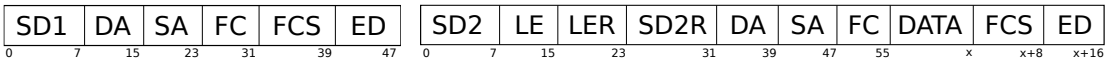
multimaster

The bus is controlled by a single *master* unit at once. After it settles all it's requests, it passes on the control to other superior station. All of the other stations, which are not controling the bus at a moment, act as a slave units. As in previous mode, the subsidiary stations can communicate with each other with the help of *master* station. But if we reconfigure one of them to be a superior station, we can exchange data directly.

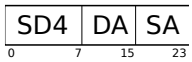
Superior units can be PC with driver supporting communication with systems TECOMAT/TECOREG with multimaster mode, or systems TECOMAT/TECOREG with serial channel in *MPC* [4] mode.

There may be up to 127 stations overall.

2.2.2 Common sctructure of protocol

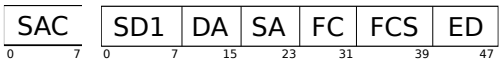


(a) message without data array (b) message with data array

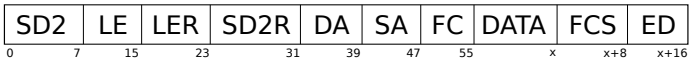


(c) message token

Figure 2.1: communication *master* $\Longrightarrow$ *slave*



(a) short ack. (b) long acknowledge



(c) answer with data array

Figure 2.2: communication *slave* $\Longrightarrow$ *master*

Figures 2.1 and 2.2 shows datagram structures used for communication over *EPSNET* protocol. Particular datafields are described in table 2.1.

2.2.3 Communication requirements

transmittion from *master* station

- The delay between sending single bytes must be shorter, than $3 * t$, where t is a time period necessary for transmittion of single byte. [5]

[4] transmittion of token message (parameter MT) is On

[5] there is an exception for mode *PC*, for details please refer to [4]

field	description	value
SD1	*start delimiter*[1]1	`$10`
SD2	*start delimiter* 2	`$68`
SD4	*start delimiter* 4	`$DC`
LE	data length, count of bytes in fields DA, SA, FC, DATA	3 ... 249
LER	*length repeat*, same value as LE	3 ... 249
SD2R	opening character reteated	`$68`
DA	*destination address*	0 ... 126
SA	*source address*	0 ...126
FC	*frame control* - to distinguish message type	0 ...126
DATA	message data, max. 246 bytes	—
FCS	*frame check sum* over fields: DA, SA, FC, DATA	$\sum_{i=\mathrm{DA}}^{END^2} \bmod 256$
ED	*end delimiter*	`$16`
SAC	*short acknowledge*	`$E5`

[1] constant opening character
[2] END is FC for SD1 and FCS $-$ 1 for SD2

Table 2.1: Datafields descriptions

- The delay between received answer and transmittion of next message must be longer, then t.

common principles

- There must always be an answer to message send by *master*. If not, it is a serious failure in communication.
- The delay between transittion of last byte and reception of the first byte of answer must be at least t. The maximum time is chosen by programmer of *master* station.
- The adresses of all paricipants of communcation must be unique.

principles for *multimaster* mode

- If there is no traffic, *master* station can begin transmittion without a challenge after a certain amount of time, which must be longer, then the longest timeout [6]. This amount must depend on address of superior station to avoid possible collisions, when more stations begin transmittion at once. The formula used to compute it is folowing:

$$\text{wait\_time} = timeout + 500 + 10 \cdot \text{address} \qquad [ms] \tag{2.1}$$

- Upon reception of token message 2.1(c), the superior station waits for time period longer, than t 2.2.3 and then can begin transmittion as *master*.
- After settling down all of its requests, the *master* sends the token message to the other superior station.

[6] time between the query and answer from *slave* station

2.2.4 Data format

All of the messages have given format:

- 1 start bit
- 8 data bits
- even parity
- 1 stop bit

*CPU*s supports switching-off the parity usage.

2.2.5 Data security

Data in message are protected with odd parity, checksum (FCS field) and by the correct sequence of values in frame. If check of any of these arrangements fail, the message will be thrown away.

CRC polynom

When dealing with modems not supporting parity check, that is for this reason switched off, we face the danger of data corruption. To evade it, *epsnet* protocol supports optional security measure. The CRC polynom. The length of field with it is 16 bits.

As this is not subject of this thesis, the detailed description is omitted, you may refer for example to [12].

2.2.6 Comunication services

There are two groups of services for communication with systems TECOMAT/TECOREG. Those are:

1. system communication services
2. public communication services

system communication services

Are available on all communication channels in *PC* mode. But there is a limitation, that only one channel can use these services at once.

They are used for configuration of devices and debugging.

public communication services

Available on all communication channels in modes *PC* and *MPC*. Overview of available services is listed in table 2.2.

For detailed description of above commands, please refer to [4]. Because our main application 4 will not be sending these requests directly, as will be noted later on [7] , we will omit the details.

[7] reference was removed due to protection of corporate secrets

service code [1]	name	description
$08	SETTID	set time
$0A	GETSW	read state word
$0B	READN	read from memory
$0C	WRITEN	write to data memory
$0D	WANDRN	write and read to/from data memory
$0E	GETERR	read error stack
$0F	READB	read a single bit from data memory
$10	WRITEB	write a single bit to data memory
$90	READBD	destructive read of bits from data memory
$91	READND	descructive read from data memory
$93	WANDRND	write to and descructive read from data memory

[1] FCS field 2.1

Table 2.2: Overview of supported public services

2.3 IDM

Stands for INELS Designer & Manager. It's a graphical application designed for configuring INELS system [14]. It keeps notion about system as a project. These projects contains common informations about building along with plans of floors stored as images. These can be further divided into rooms. Upon these floor plans, user can place widgets with associated devices and specify them actions performed upon user's interfactions with widgets. By interaction is meant clicking on the widget shown on floorplan.

Different actions can be specified for changes of states of different sensors. This can be for example excessing minimal or maximal value of thermal sensor.

It supports following modes of configuration:

administrator Provides complete management of system and project. Connection to *CPU* is optional, as in this mode, user can manage the project off-line and the later upload it to *CPU*.

configurator Provides basic configuration of system. This mode needs active connection to *CPU*.

user No configuration of system is allowed here. This mode serves purely for interaction with system. Connection to *CPU* is also needed here.

Access to these modes is allowed after authorization by password.

2.3.1 Configuration

Configuration of system is divided into several sections:

Inputs are further divided to sections *digital*, *analog*, *thermometers*, *cardreaders*

Outputs again, these are divided to *digital* (on-off actors) and *analog*

Heating let us control house heating systems. Menus of this section allows us to assign different scenarios to operation modes.

Alarms allows to define settings for electronic signalling system. We specify "secured zones/entries" and assign them detectors connected to *INELS* system by their digital inputs.

System provides us with configuration of counters and timers, together with settings of special events in system

The above list is not complete as it lists only items, we are currently interested in.

2.3.2 Actions

IDM lets us assign actions to various events in *INELS* system. These can be divided to:

- system actions
- alarm actions
- *Sophy* actions
- time actions
- input actions
- user actions
- output actions
- heating actions

There are two kinds of actions we can define:

1. build-in action
2. user action

First kind of actions operate upon unit itself meanwhile the second lets us set the commands sent, upon triggering, to other units.

build-in action

Menu of commands to set here differs based on whether we deal with on-off actors or "dimmers"[8].

user action

With this kind of action we can change state of another units.

user action we can trigger already defined user action

(un)set system bit change the system bit

send *sms* to phone number defined in *GSM*

drop-call means to dial a phone number and let it ring for 20 seconds

alarm change states of alarm groups

counter/timer various commands to alter their states, running and stopping them

lightning groups commands for predefined groups of lights

relay groups commands for groups of units with digital outputs

heating lets us change operation modes

[8] these refer to units providing analog output as noted in

2.3.3 Export

Application also supports export of configured devices. There are several different options. Common feature of these options is that output is one of more plain text files containing list of devices with their properties. One device per line. Follows list of export options.

1. Inputs and Outputs
2. Time program
3. Time events
4. Counters and Timers
5. Events for vizualization

In terms of our main application 4, we will focus only on the first option. This export produces output in format shown by table 2.3 Where particular fields, divided by spaces

inels_item	REG	CF	ADDR	[.B]	TYPE	PUB_INOUT

Table 2.3: Format of `export.pub`

mean:

`inels_item` name generated by *IDM* 2.3, which is a composition of user defined names for *unit* and input/output

`REG` can be `X`, `Y` or `R`, that stand for: input, output and user data register

`CF` this is a compability field, in which we are not interested

`ADDR` address of register mapped in address space of *CPU*

`[.B]` present only if type of value is `BOOL`. The `B` stands for position[9] of bit in register.

`TYPE` type of variable. Possible values for this field are:

REAL floating point number
BOOL single bit in register
BYTE 8-bit value

`PUB_INOUT` compability field

The snippet of generated export file by *IDM* is shown in listing 2.1.

Listing 2.1: export.pub

```
1  room_3_1_IN_1 X B 4 .0 BOOL PUB_INOUT
2  room_3_1_IN_1_Counter R B 17673 BYTE PUB_INOUT
3  room_3_1_IN_2 X B 4 .1 BOOL PUB_INOUT
4  room_3_1_IN_2_Counter R B 17701 BYTE PUB_INOUT
5  room_3_2_UP1 X B 6 .0 BOOL PUB_INOUT
6  room_3_2_UP1_Counter R B 17729 BYTE PUB_INOUT
7  room_3_2_DOWN1 X B 6 .1 BOOL PUB_INOUT
8  room_3_2_DOWN1_Counter R B 17757 BYTE PUB_INOUT
9  room_3_2_GREEN1_OFF R B 22207 .1 BOOL PUB_INOUT
10 room_3_2_GREEN1_TRIG R B 22207 .2 BOOL PUB_INOUT
11 room_3_2_GREEN1 R B 22217 .0 BOOL PUB_INOUT
12 room_3_3_TERM X F 12 REAL PUB_INOUT
```

[9] possible values are 0 ... 7

2.4 IMM

Content was removed due to protection of corporate secrets.

2.5 Touch panels

Follows detailed description of two touch panels, which were selected as platforms for main application:

- SofCon's TOUCH11/PC
- SofCon's TOUCH51/PC

Specifications for both panels is listed by table 2.4.

	Touch11/PC	Touch51/PC
Display	LCD 5,7„ touch	LCD 10,4“ touch
Resolution	640x480	800x600
Colors	262144	
Processor	AMD Geode LX800/500MHz	VIA EDEN, INTEL ATOM 667Mhz
Ram	256MB	
Storage	Compact Flash 2-8GB	
Interfaces	Ethernet 1Gbit/s, 2 x RS232, 4 x USB	
	1 x PS2 (keyboard + mouse), Centronics (LTP)	
	—	VGA 1024x768, RS232/RS422/RS485, PC104Bus

Table 2.4: Touch panels specifications

2.5.1 Touch panel communication

Operating system communicates[9] with panel through serial port RS232. Folows parameters of communication:

- 8 bits data
- even parity
- 1 stop bit
- speed 9600 Baud/s

Chapter 3

Bring it to life

Making touch panel part of *INELS* 2.1 is a task composed of few non-trivial tasks.

1. Choosing an appropriate operating system according to requirements. 3.2
2. Make sure, that operating system correctly handles all peripherals and subdevices of chosen hardware platform. 3.4
3. Implement a user interface not necessarily dependend on selected operating system.
4. Make it easy to install all needed software components. 3.7

3.1 Objective

Create user interface for controlling intelligent home devives through epsnet protocol with touch panel as targeted platform. This includes choosing suitable operating system, make necessary changes to it to provide access to *units* via *EPSNET* 2.2 protocol. Application is to be controled by interacting with device's touchscreen.

3.2 Operating system

3.2.1 Requirements

OS must be stable, easy to configure and it must support needed software programming enviroment and tools for running main application. Also it should support natively all hardware of targeted platform.

3.2.2 Previous installation

Producer of targeted touch panels 2.5, company SofCon, distributes with them one of default installations of operating systems, based on customer's preferences:

- Microsoft Windows XP Embedded 3.2.2
- Debian GNU/Linux 4.0 voyage 3.2.2

MS Windows XP Embedded

With regard to operating system, which is targeted by main application IMM 2.4, and to the fact, that application for these touch panels is offering subset of features of the main application, and so it can reuse the already done code of it, and considering, that main application's target operating system is linux, then MS Windows it not a good choice.

Debian GNU/Linux 4.0 voyage

The problem with second default installation of Debian Linux, is that company is providing support only for this particular version of distribution, this includes provided source code of drivers needed to communicate with touch screen peripheral. This distribution is outdated and does not meet the requirement of supporting necessary programming tools. In particular at least the version 2.6 of python is needed. This distribution provides package with version 2.4.4 at most, which is not acceptable. Absence of other minimal versions of the rest of needed packages need not to be mentioned. This leaves us with few options:

1. compile desired packages by ourselfs
2. select another more appropriate distribution

First option does not meet requirement of easy configuration. Compiling packages from sources is difficult task by itself. But most difficult would be managing needed dependencies, when package A depends on p. C, which on the other hand depends on packages D and E. Compiling single software component (for instance python) could result in need to compile other n packages. This would soon leave us with our own distribution of GNU/linux, which is certainly not, what we want.

Based on previous statements, the second option was selected. Downside of this decision is that linux driver for touch screen, since it's not part of official kernel source tree, will need to be managed to fit with paricular versions of linux kernel used by selected distribution, if producer of touch panels does not decide to support this distribution later on.

3.3 Ubuntu 9.10 "Karmic Koala"

Is a chosen distribution for touch panels 2.5. Current version of it's kernel installed is 2.6.31, which is 13 versions newer than one provided with default installation 3.2.2. This means that many changes were made to linux kernel since the touch screen driver were made. And slight changes were made even to linux kernel driver interface, which made the driver not compilable.

3.3.1 Driver changes

Follows few examples of such changes in c code.

- Removing proc root entry of device from proc file system had been done calling this function:

```
1         remove_proc_entry( const char *name
2                          , struct proc_dir_entry *parent
3                          );
```

passing as the second argument macro `proc_root_driver`, which expanded to `NULL`. This macro was removed in later versions. Thus it's occurencies needed to be replaced by `NULL` directly.

- Also from `struct proc_dir_entry` was removed attribute `owner` due to security reasons. This attribute specified owner module, which was usually set to macro `THIS_MODULE`, as in our case. Occurences of this assignment had to be removed.

- `struct input_dev` representing input device had attribute `event` of type

```
1         void (*) ( struct input_dev *dev
2                  , unsigned int type
3                  , unsigned int code
4                  , int value
5                  );
```

that is used to specify event handler for particular input device. It's type was later changed to

```
1         int (*) ( struct input_dev *dev
2                 , unsigned int type
3                 , unsigned int code
4                 , int value
5                 );
```

Thus event handler needed to be modified to return 0, if event is handled and -1 otherwise.

As can be seen, needed changes to the driver, due to kernel api changes, are rather cosmetic. All changes made were supplied with conditional compilation statements depending on version of kernel, to support earlier ones. Especially to preserve the compability with version of kernel of mentioned Debian GNU/linux distribution 3.2.2. To allow producer of touch panels 2.5 to merge the changes with their base source code without side effects, and thus support wider range of different linux kernel versions.

3.4 Touch screen input

As noted before, touch screen is connected to system bus via serial port RS232. X server is not alone able to communicate with touch screen through this port. Protocol for communication with it is special to touch panels from SofCon.

Allowing input from touch screen to be handled by X server, can be achieved by multiple ways:

write driver for X This driver would have to handle the communication with touch screen directly through serial port.

write driver for linux kernel In this case, driver would create a translation layer between any user space application, that would want to communicate with device through standard linux kernel input system api.

First method is not recommended as noted in [2]. When input device is a "common" one (touchscreen, mouse, keyboard, etc.), as in this case, a better solution is to make proper

linux kernel driver and let the X server pick up its general driver for it automatically. Also second solution is more general one, because it lets multiple user space applications to access the device. In this case second option was selected.

Touch screen driver registers itself in kernel after loading its module, to handle connection, disconnection and event processing. But loading this module and connecting the touch screen to serial port is not enough for kernel to attach them together. First the settings for the serial port as noted in 2.5.1 and protocol number (described below) must be set by calling `ioctl` function on particular file descriptor of openned serial port device.

3.4.1 `inputattach`

This is actually a common process of attaching input devices connected via serial port to their respective linux kernel driver. Standalone application was made to handle this process. It's name is `inputattach`. It does a simple job, taking device and connection type name as parameters and attaching it to kernel input system. Internally it has predefined protocol numbers for each device it supports (actually these numbers are defined in `linux/serio.h` headers distributed with vanilla kernel). One of which is assigned to device, when making connection, by calling this program. It then runs as a daemon. Sequence of events triggered by attaching device is illustrated on figure 3.1.

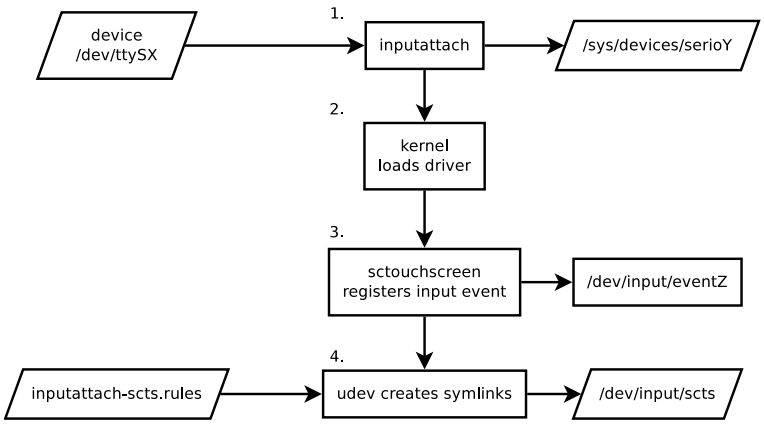


Figure 3.1: Attaching serial device to input system

Since our kernel driver is not part of official kernel source tree, it's not even one of recognized devices of this application. But company SofCon modified it to include support for their touch panels. Protocol number assigned to this device by company is 0x36. Which is the same number as of driver for Zhen Hua PPM-4CH RC transmitter, assigned to it by linux kernel's developpers. This makes a problem, since protocol numbers for different devices must by unique. When device is attached, kernel defautly loads *zhenhua* module, because protocol number matches the one assigned to device by `inputattach` [1]. To circumvent this, *zhenhua* module needs to be blacklisted, and then our driver takes its place, when `inputattach` is called.

[1] `inputattach` has in our case the same protocol id number for two different devices

3.4.2 Touch screen driver

When driver is loaded, it registers two input devices in kernel input system (the third step in figure 3.1). One provides access to one way motion events with absolute coodinates yielded from driver to user space. The second is two-way control event.

Virtual devices (in this case input events) are by *udev* daemon added to `/dev/input` directory with name `eventX`, where `X` is a number dynamicly assigned to this input event by *udev*. Through the virtual device (the one providing motion events), X server's driver is handling all user input from touch screen. But because X server is not smart enough to assign this virtual device driver we want, this driver needs to be assign to it explicitly. This can be done statically by editing X server's configuration file typically located in `/usr/X11/xorg.conf` or using hal (hardware abstraction layer), but more general way is to create symlink to this virtual device with constant name pointing to specific event in `/dev/input`. And then configure *X* server to assign this symlink a driver we prefer. Creation of this symlink can be achieved by writing *udev* rules file.

udev

udev [10] is a daemon running in a user space, which listens to the events reported to user space from kernel typically through *netlink*, which is the "prefered interface between user space and kernel for IP networking configuration" [6]. It keeps state of files located in `/dev` directory actual, based on currently connected devices supported by running kernel. What epsecially insterests us is the fact, that it allows administrator of system to specify rules of naming schemes for individual devices or for set of them with common attributes. And also create symbolic links refering to real virtual devices.

This means, that we can create a symlink for virtual device in form `/dev/input/scts`, which points to one of `/dev/input/eventX` virtual device and from now on, refer only to this symlink, whose name does not change.

Rules are kept in text files typically located in `/lib/udev/rules.d` (files provided directly with `udev` package or by other distribution's packages) and in `/etc/udev/rules.d` (local system rules). For exact format of rules file see [8]. Rules used to create a symlink for our input event devices are these:

Listing 3.1: Udev rules file

```
1 SUBSYSTEM=="input", ATTRS{name}=="SofCon Serial TouchScreen", \
2       SYMLINK+="input/scts"
3 SUBSYSTEM=="input", ATTRS{name}=="SofCon Serial TouchScreen Control", \
4       SYMLINK+="input/scts-control"
```

Rules needed to be divided to multiple lines for the sake of readibility, but in *udev* rule files, one rule must occupy one line at most.

First two fields (`SUBSYSTEM` and `ATTRS`) represents matches, by which desired device is searched for. There can be arbitrary number of them, but in this case, two are sufficient. First filters only those devices belonging to kernel's input subsystem, while the other matches attribute name assigned to input device by our kernel driver.

3.4.3 *X* server configuration

Since `/usr/X11/xorg.cong` is not implicitly present in distribution 3.3 and prefered method of its configuration is to use *hal* [2]. We will obey this convention.

hal

Stands for "hardware abstraction layer". It's another process in user space dealing with connecting, disconnecting and configuring devices in linux and in various forms is present at majority of operating systems. While *udev* manages virtual device files, *hal* takes care of additional tasks concerning it's administration. For example [10]:

- Loading driver modules.
- Managing virtual model of hardware and different views attached to it.
- Communication with processes concerning changes in device states.
- Creation of mount points for filesystems.

As in case with *udev* 3.4.2, *hal* collects information about what to do with particular device from its configuration files. These files are written us *xml* files according to dtd definition. To tell *X* server to assign desired driver (in this case *evtouch*3.4.4) to particular device, rules like listing 3.2 shows,

Listing 3.2: Hal policy file

```
1 <match key="@info.parent:info.subsystem" string="serio">
2   <match key="@info.parent:info.linux.driver" string="scts">
3     <match key="info.capabilities" contains="input.touchpad">
4       <merge key="input.x11_driver" type="string">evtouch</merge>
5     </match>
6   </match>
7 </match>
```

needs to be written to hal policy file.

Also this configuration composes of matching part and rule part. This time we first search for parent device of one, we are focused on. Concretely line 2 requires, that its parent device belongs to subsystem `serio`. While line 3 filters from matched parents those, with *scts* linux driver, which is the identification name of our driver. And because we are interested only in input device providing us with user input and not configuration, we apply last filter rule asking for device with touchpad capabilities. And to this device is assigned an attribute `input.x11_driver` with value `evtouch`, which tells the *X* server, that this is the driver we want to use.

3.4.4 *evtouch*

evtouch is a linux-touchscreen driver for *X*. From *evdev*, which is a generic input driver for *X*, it differs by providing few additional options concerned with touch screens [1]:

TapTimer This timer starts when the state MAYBETAPPED is entered. When this timer expires a tap-event is issued and the state changes to UNTOUCHED. Default value is 200ms.

[2] Currently *hal* is being marked as deprecated and all configuration is to be done via udev in future versions of *X*.

LongTouchTimer This timer is always started when the state TOUCHED is entered. When the timer expires before you untouch the screen again the state moves to LONGTOUCHED and on entering that state a longtouch-event is issued. Default values is 400ms.

MoveLimit If the pen moves out of this radius a "mouse-press"-event becomes impossible. Default value is 30 Pixels.

Rotate There are two valid values:

CW Rotate the screen clockwise

CCW Rotate the screen counter-clockwise

Everything else will be treated as "no rotation". Default values is "no rotation".

Calibrate This option starts driver in calibration mode. In this mode, driver opens a *FIFO* file 3.4.4 for writing and when another process opens this same file for reading, it writes there input events in this format (in *C* code):

```
struct Point {
    int x; int y;
};
```

The other process reading input events can obtain the `x` and `y` coordinates this way:

```
if (read( m_fd       //file descriptor of fifo file
        , &p         /* instance of above struct
                      * representing point */
        , sizeof(p) //size of this sctruct
        ) == -1) {
    if (errno == EAGAIN) {
        /* file was openned in nonblocking mode
         * and reading would block */
        break;
    }
    //error processing
}
```

Events are generated only upon pressing touch screen and moving pressing object around.

The use of this option will be covered later on [3] .

It also provides calibration tool, which utilises calibration mode noted above. This tool is in more detail described in [4] .

***FIFO* file** Stands for first in, first out queue. In operating system such a file represents named pipe [10]. It's an interprocess communication object, which has its place in OS's file system, even if no process actually use it. As such, it can have its own permission access settings as any other regular file. It can be even packed to archiver and later be restored.

But let's be concerned with communication possibilities. From the application's point of view, sending data to other process via pipe is the same as writing to file or socket. Pipe

[3] reference was removed due to protection of corporate secrets

[4] reference was removed due to protection of corporate secrets

has two ends. Both ends can be openned by multiple processes or even threads of the same process, although it's rarely used.

It can be described as an interface to memory buffer, which is limited by its size, so in order not to fill it up, pipe must be read at the same time as it is written to. When buffer is filled up, the next attempt to write data to pipe will block, until the same size of data chunk is read from it. When openned in nonblocking mode, this same attempt would generate error `EAGAIN`. It should be noted, that when writing to pipe, which has the second end closed, process will receive the signal `SIGPIPE` from kernel and default action upon receiving this signal is to end the program. To prevent this, a handling function should be registred.

Before communication can begin, this pipe must be constructed:

```
1  if (mkfifo( pipe_path            //path (name) of pipe
2            , S_IRUSR | S_IWUSR    //permissions of produced file
3            ) != 0) {
4      //error handling
5  }
```

Then we can open it just as a regular file:

```
1  FILE * f = fopen(pipe_path, "r");
2  if (f == NULL) {
3      //error handling
4  }
5  ...
6  fclose(f);
```

This code opens the pipe for reading only. The other process writing data to it would supply the fopen function `"w"` as a second argument.

A special characteristic this pipe has is that it must be openned by both processes at once. So this open operation would block until second process opens up the other end of pipe. To evade this blocking, we can open the pipe in nonblocking mode. In that case, pipe would decline any attempt for communication until it's fully openned.

3.5 Calibration

Content was removed due to protection of corporate secrets.

3.6 Packaging

To simplify the process of installation of needed software components for mentioned distribution, packages were made. This simplifies not only installation process, but especially updating packages, when new version of software is released. Thanks to packaging system on package-based distribtions we don't need to worry about cleaning up the system after removal of certain applications [13].

To create a package for your distribution, please refer to the packaging guide of that distribution. Here we will be only scratching the surface of creating the packages for Ubuntu.

Packages the were up to now made, to make the touch panel functional, are following:

1. `sctouchscreen-dkms`
2. `inputattach-scts`

3. `evtouch-calibrate`

3.6.1 `sctouchscreen-dkms`

This package installs touch screen driver 3.4.2 to system. Actually it leaves all the work to *DKMS*.

DKMS

Stands for Dynamic Kernel Module Support [11]. It allows to automatically compile linux drivers, when the new version of kernel is installed to system. This is particularly helpful for proprietary drivers, because without *DKMS*, the user must wait for the release of particular driver compiled with the version of kernel, he is running, for him to use desired hardware. With *DKMS* the driver is compiled transparently without the need of user intervention.

Since we are expecting to update touch panel's installations, this greatly simplifies administration.

To make a package using this system, we may employ the `dkms` tool directly in this process with this command:

```
$ dkms mkdsc -m ${PKG_NAME} -v ${PKG_VERSION} --source-only
```

or use a standard packaging procedure covered by packaging guide [5]. In the end, we end up with `debian` directory with files to edit such as:

`control` defines compile time and runtime dependencies in form of packages for this distribution

`rules` this file is actually a makefile, which is run by *GNU* make and controls the build process together with installation to directory, which will be later packed

`postinst` a shell script taking care of postinstallation actions

We will focus on `postinst`, the snippet showing important lines is shown in listing 3.3. With these lines set, the driver is, after installation of package, added to kernel source tree and from that point on is actualized upon any update of kernel.

Listing 3.3: `postinst` script

```
NAME=sctouchscreen
VERSION=1.4

case "$1" in
    configure)
        echo "Adding Module to DKMS build system"
        dkms add -m "$NAME" -v "$VERSION"
        echo "Doing initial module build"
        dkms build -m $NAME -v $VERSION
        echo "Installing initial module"
        dkms install -m $NAME -v $VERSION --force
        echo "Done."
    ;;
esac
```

To later remove driver from tree, these few lines of code are enough:

```
dkms remove -m $NAME -v $VERSION --all
depmod
rm -rf "/usr/src/$NAME-$VERSION"
```

These should be part of removal scripts for package (`prerm` and `postrm`).

3.6.2 `inputattach-scts`

This is the glue for other packages. It installs following configuration files:

`40-scts.fdi` the destination directory is

```
/usr/share/hal/fdi/policy/20thirdparty
```

this file is shown in listing 3.2. Thanks to it the *X* server assigns device the *evtouch* driver.

`blacklist-scts.conf` is installed to `/etc/modprobe.d`. It contains simple line forbidding loading of `zhenhua` module as mentioned before 3.4.1. Thanks to it our `sctouchscreen` driver will be assigned to serial device, after `inputattach` is run.

`inputattach-scts.rules` is installed to `/lib/udev/rules.d` Listing 3.1 shows the contents of this file. Thanks to it, we may refer to virtual input device file by name.

`inputattach-scts.conf` is installed to `etc`. It contains the path to serial port device, that is used to communicate with touch panel's screen and mode, in which to run the daemon (`scts` mode).

3.6.3 `evtouch-calibrate`

The implementation of application, this package provides, is covered in section 3.5. This package just installs executable to `/usr/bin` and initialization script to `/etc/init.d`. This script starts the application together with *X* server and after successful calibration, it stores configuration settings in `/etc/evtouch/config` for *evtouch* driver to load them upon *X* server's start. It also takes care of setting the correct environment for this application to work, sush as setting the calibration mode of *evouth* driver.

3.7 Installation

Compact flash disk present in touch panel has capacity ranging from 2GB to 8GB. This is enough for embedded device, as long as we deal with it economically.

Disk partitioning will be covered later. Otherwise, the process of installation is not much different from installation on desktop computer, except, that we use minimal configuration, leaving us with naked system, that we use as a base.

After installation of distribution to compact flash on touch panel device [5]. The steps needed to take are following:

1. install necessary packages from official repositories

[5] this is currently done by traditional way of installation from CD, which is inserted to CD-rom mechanics connected via SATA → IDE convertor

2. install our own packages 3.6
3. create new user
4. install main application

3.7.1 Disk partitioning

The partitioning we are going to do is for the sake of distribution of working system to other touch panels. It would be tearsome to install the distribution with all the required packages all over again to multimple disk drives, when most of the contents would be the same. This is dealt with by cloning of compact flash, which is covered later. Since we don't want to clone all of the contents of compact flash, when we'll be performing update of distribution, but we want to preserve user configurations, he made. It is wise do divide the disk to partitions, from which some can be overwritten, when update comes, and others will stay the same. Listing 3.4 shows selected partition scheme for 4GB compact flash disk.

Listing 3.4: `cfdisk` disk partition scheme

```
1  Name      Flags     Part Type  FS Type     [Label]      Size (MB)
2  ------------------------------------------------------------------
3  sdf1      Boot      Primary    Linux                      78.45
4  sdf2                Primary    Linux                    3501.20
5  sdf3                Primary    Linux                     437.65
```

The `sdf` means, that we are manipulating partition 1 of block device `sdf`, which represents our compact flash. On your system, you may confront different naming scheme. The first partition is bootable and will contain boot loader [6]. The second partition, which is the largest, will contain system installation files. Last parition will keep home directory of main application. This partition will hold data, that we want to preserve. Other partitions can be overwritten, when we'll perform update of system by disk cloning.

3.7.2 Disk cloning

Cloning of the contents of one compact flash is easy as long, as it has the same size. Before cloning, we have to recreate the same disk partition scheme on the destitation flash card. Then we can use these commands to do the copy:

```
dd if=/dev/sdf1 of=/dev/sdg1
dd if=/dev/sdf2 of=/dev/sdg2
```

This example assumes the `/dev/sdf` to be the same device as on listing 3.4. Second device (`/dev/sdg`) is the distination. These two commands copy first two entire partitions to the second disk.

Beware

Above commands work in case, that corresponding partitions on both disks are of the same size, or the partitions on the second disk are larger. Otherwise it would not work and you would have to select another method.

[6] GRUB in our case

Chapter 4

Main application

As noted before 2.4, this is an *IMM* client, which uses the *epsnet* server to communicate with *INELS* units. Currently it is not an *IMM* player suited for playing audio or video. This may change in future version. It will also depend on hardware capabilities of supported touchpanels, since playing video files is computationally expensive.

Thanks to abstraction provided by *epsnet* server from communication with *CPU*, this application becames very simple.

The application is not complete yet at the time, these lines are written. And a lot of things presented here can change in near future. So we won't go deep into implementation details, rather we will focus on main concepts, that should stay the same.

4.1 Screens

Functionality, or rather actions, the user can trigger, are divided to number of screens. Here is their list:

main screen figure 4.1

music screen figure 4.2(a)

movie screen figure 4.2(b)

tv screen figure 4.2(c)

web screen figure 4.3(a)

camera screen figure 4.3(b)

foto screen figure 4.2(d)

Main screen is the opening one. It is basically a crossroad to the other screens and at the same time it provides the quick access to important devices in the house, which can be instantly controled. As noted before [1] , devices listed on this screen depends on current *room* context.

4.1.1 Main screen

We will now describe the buttons user can interact with on the main screen. The buttons marked with red rectangles on picture of main screen 4.1, represents *INELS* items. As you can see, the button *Lamp* is highlighted. This marks, that lamp is turned on. When user presses this button, a slider may show up, depending on the settings in `rooms.cfg` file

[1] reference was removed due to protection of corporate secrets

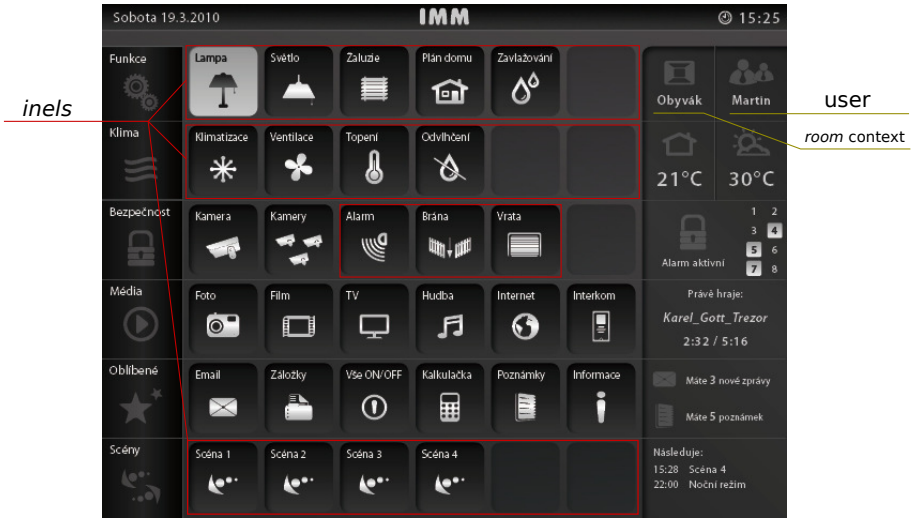


Figure 4.1: Main screen

shown in listing [2] and `export.pub` (listing 2.1). If the type for the device represented by this button [3] is *REAL*, the slider shows up and user can manipulate the intensity. Otherwise [4] this simply turns off the lamp.

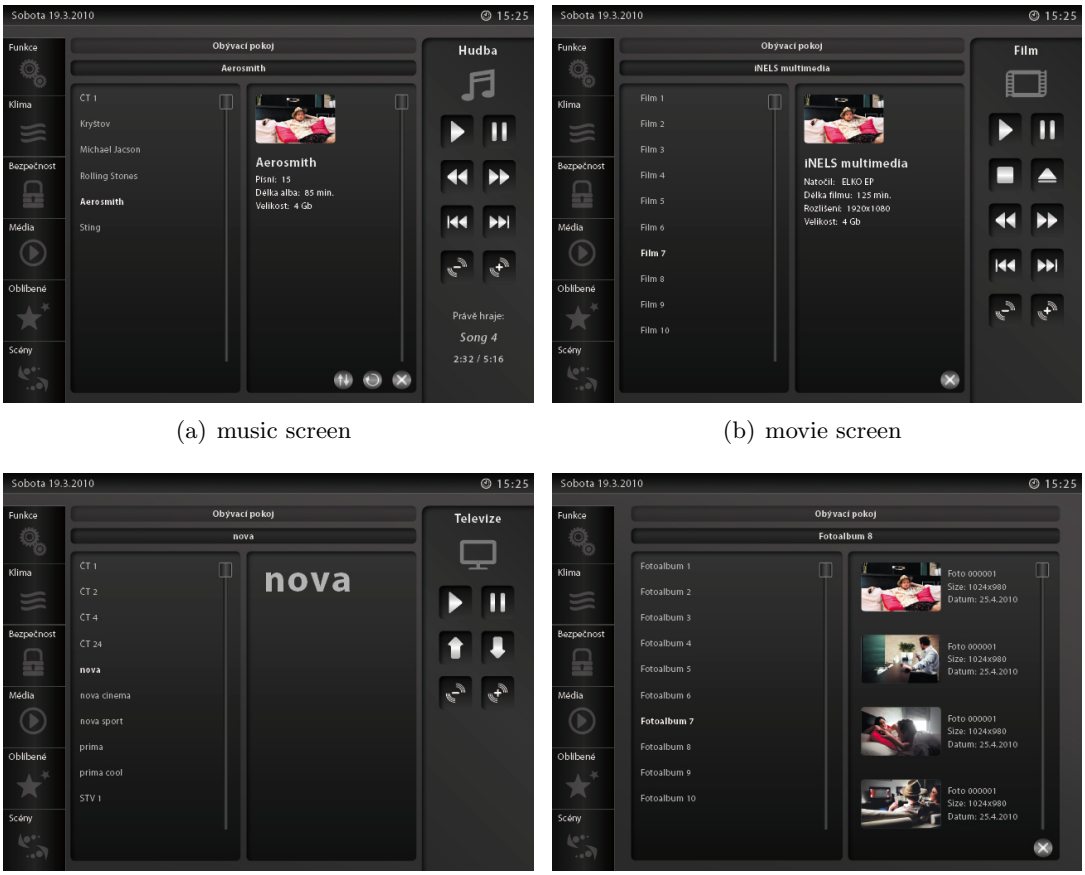
All of these items are shown only if they were defined for current room context, which is marked on the figure. Room context is one of rooms listed in file `rooms.cfg`. Implicitly it is set for the room, in which the touch panel is placed. When user presses this button, the next room context is shown. And all of the buttons represening *INELS* are reloaded. This applies also to the music bar showing currently played track on player defined for current room context and to temperature indicator displayed directly below shown room context.

The other buttons are self-explanatory, most of them trigger displaying of another screen. Some of those screens are shown on figures 4.2 and 4.3.

4.1.2 Multimedia

Figures 4.2 shows multimedia screens. These does not operate upon *INELS* items, but also needs *epsnet* server running. This is actually true for first three screens. The last needs only configured *NAS* storage or network shared storage on remote host [5] to show the photos.

Music, movie and tv screens use `player_man` [6] object to get status of player. Pressing the controls on right panel would trigger expected action on player of current room context. As noted before, database of music files, movies, tv stations and photos are stored on remote host or *NAS* storage in simple directories, which are mounted upon boot.



(a) music screen

(b) movie screen

(c) tv screen

(d) foto screen

Figure 4.2: Multimedia screens

4.1.3 Other screens

These provide various functionality, that can be discerned by simple look. Figure 4.3 does not list all of the available screens in application. But their number and functionality will be most certainly modified in near future, so they serve us as a bare outline of possible functionality.

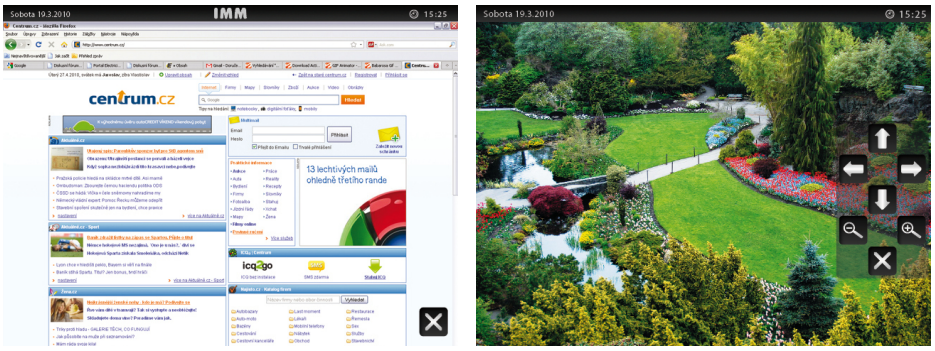
Most interesting is the camera screen. This uses embedded *mplayer* [7], which draws frames of stream from ip camera into provided widget.

The list of cameras is obtained from remote storage . The mounted directory contains files with similar line:

```
rtsp://{LOGIN}:{PASSWORD}@{IP_ADDRESS}:{PORT}/mpeg4/media.amp
```

After *mplayer* is started with this address, it logs in to the ip camera and starts playing

[2] reference was removed due to protection of corporate secrets
[3] which is stated in `export.pub`
[4] the type would be *BOOL*
[5] in very simple configuration, fotos could be present only on hard disk of touch panel
[6] reference was removed due to protection of corporate secrets
[7] a moview player for linux [15]



(a) web screen (b) camera screen

Figure 4.3: Various other screens

the stream. The previous address is different for various ip cameras. To access this stream, refer to official manual for you ip camera.

4.2 Implementation

This *IMM* client is also implemented in python. From *IMM* HD client it differs in used graphical toolkit. This uses *Qt*, "Qt is a C++ class library and GUI toolkit for Unix, Windows, and embedded systems (with the latter running on Linux)" [7].

4.2.1 Object design

Object design follows the separation of functionality to screens. The main classes used are following:

MainMenu is the main window of application. It actually does not render anything on screen by itself. It only changes panels (`QWidgets`) in its layout.

MainPanel renders the main screen.

CamPanel renders the camera screen.

MusicPanel renders music screen.

MoviePanel renders movie screen.

Above classes except for MainMenu have in common, that they define a *Qt signal* `show_main` which is emitted, when user triggers a closing action. This action may be defined by various ways in different widgets. When this signal is emitted, it is catched in *Qt slot* `showMain` of `MainMenu` object, that hides currently set panel and shows instance of `MainPanel` instead. For description of *Qt signals/slots* mechanism, please refer to *Qt*'s documentation.

Simplified object design is illustrated on figure 4.4. Follows a description of selected classes.

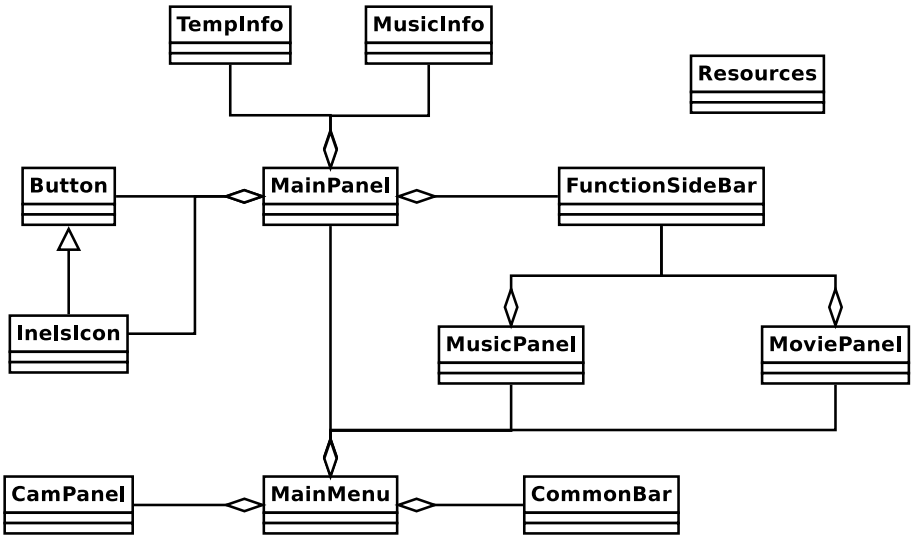


Figure 4.4: Object design

Resources

This class is modeled as singleton. It provides methods to access application settings, which are stored in `ini` configuration file. Settings contain address of *epsnet* server with its port, paths to some images and floorplans, path to skin directory and currently selected skin. Currently there is only one available skin, but it may change in the future. Configuration of *IMM* clients is stored in the same file `/etc/imm/imm.cfg` as the configuration of server. This is not true for this application, since there are many options different compared to *IMM* HD client.

Most important thing skin defines are path to icons, colors of fonts depending on context and dimensions of several widgets. There is no need to define background colors, because background is rendered from bitmap images.

CommonBar

Is a top bar common to most of the panels. It renders current date and time together with bitmap image in background. As you can see, it's owned by `MainMenu`, because there is no need to include this bar in every panel, since it's shared by them. When a request, to show a screen without this bar, comes, `MainMenu` must recognize it and hide it in order to let the child panel occupy all of the screen.

FunctionSideBar

This is a bar to the left of the screen and is shared by most of the panels. Currently it does not have assigned any special functionality apart from bringing user to the main panel. But this will probably change in the future versions.

4.3 Start of application

Application is started with optional argument `-c` as this:

```
./imm.py [-c CONF]
```

where `CONF` is configuration file loaded by `Resources` object.

It is started as full screen application occupying all available space of display.

Chapter 5

Conclusion

The creation of multimedia system is not a simple job. The reason for it is that it's composed of a lot of smaller tasks, of which some may prove as much more difficult to overcome, than the implementator would thought. Some of these tasks include crucial decisions, which may render the outcome unusable. This includes selection of hardware platform, operating system, the object design of main application. And we are not taking into account the other systems, that our application needs to communicate with.

Reader should especially note, that choosing a hardware platform, is not just a mere selection of hardware components. Software support for this platform is at least at the same level of importancy. User should be confident, that such support won't diminish in years to come, for that would left him with hardly mantainable piece of hardware. In this sense, the open source drivers may be a solution. But again, proper support from manufacturer will make developers of main application and in the end, the client, contented.

As with any other application, the developer should predict the future need for modifications, because of different client's preferences, that may vary accross time, as clients become more familiar with application. For it the application should be modular and configurable. Modularity does not necessarily bring complexity to project. It may, on the contrary, result in more synoptical design. The posibility for configuration is unfortunatelly a more demanding feature. Since it represents hidden dangers in form of not handled combinations of options. Application, especially for embedded devices, must be stable and must handle all possible combinations of configuration options correctly. For when it ends in undefined state, and operating system kills the application forcibly, the user may not be able to start it again or just fix the configuration, that caused it to fall.

From the developer's point of view, this area of development is challenging and at the same time rewarding. It's not an easy task to implement all desired features when dealing with a big range of users demanding various extensions to program. But this is actually an advantage, since there is always something to modify, improve or add to satisfy new demands. Simply put, there's always something to do, and developer does need to worry about shortage of work for years to come.

Bibliography

[1] Linux-touchscreen driver for x. http://www.conan.de/touchscreen/evtouch.html, November 2008.

[2] http://www.x.org/wiki/Development/Documentation/XorgInputHOWTO, October 2009.

[3] Přehled sortimentu, March 2009.

[4] Sériová komunikace programovatelných automatů tecomat - model 32 bitů. Technical Report TXV 004 03.01, Teco corp, Teco a.s. Havlíčkova 260, 280 58 Kolín 4, March 2009. version 15.

[5] Ubuntu: Packaging guide/complete. https://wiki.ubuntu.com/PackagingGuide/Complete, 2010.

[6] Christian Benvenuti. *Understanding Linux Network Internals*. O'Reilly Media, Inc., 2005. ISBN 0596002556.

[7] Matthias Kalle Dalheimer. *Programming with Qt*. O'Reilly Media, 2nd edition, January 2002. ISBN 978-0596000646.

[8] Daniel Drake. Writing udev rules. http://reactivated.net/writing_udev_rules.html, 2008.

[9] Jan Hvozdovič. Touch software, aplikace pro práci s dotykovou obrazovkou, příručka uživatele a programátora. Technical Report 4.00, SofCon, spol., s.r.o., Křenova 11, 162 00 Praha 6 Czech Republic, February 2006.

[10] Jelínek Lukáš. *Jádro systému Linux*. Computer Press, a.s., May 2008. ISBN 978-80-251-2084-2.

[11] Richard Petersen. *Ubuntu 9.04 Desktop Handbook*. May 2009. ISBN 978-0982099841.

[12] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, 3rd edition, May 2003. ISBN 978-1558608320.

[13] Ubuntu Documentation Project. *Ubuntu 9.04: Packaging Guide*. Fultus Corporation, 2009. ISBN 1-59682-153-1.

[14] Jiří Stýskalík. Příručka pro software inels designer & manager. Technical Report 2, 2008.

[15] MPlayer team. Mplayer - the movie player. http://www.mplayerhq.hu/DOCS/HTML/en/intro.html, 2009.

[16] Wikipedia. Home automation.
http://en.wikipedia.org/wiki/Intelligent_home, September 2009. [Online; accessed 10-April-2010].