Design and Development of a Prototype Android App for a KNX home

Richard Isaacs and Friedrich Praus

Department of Embedded Systems
University of Applied Sciences Technikum Wien

Höchstädtplatz 5, 1200 Vienna, Austria
reg.isaacs@gmail.com praus@technikum-wien.at

Smartphone Apps provide personal and customizable access to the Internet and immediate surroundings. With the introduction of KNXnet/IP, an IP communication to KNX networks became possible enabling KNX installations to be connected to those Apps easily. This paper describes the relevant tasks for the design and development of a prototype Android App that provides location independent access for supervising a home network. The App imports a static configuration model of the home network and provides an intuitive user interface. This network configuration is automatically created by exported standard information from the KNX project database with an additional custom parsing. The open source Java library, Calimero, is used for the communication to a KNXnet/IP server for management and transport layer telegrams. The Android App implements a client/service architecture persisting all network data and events to a local database. The graphical user interface (GUI) provides a basic set of features for monitoring, import, diagnostic and control functions. The App is tested using the Siemens Gamma Training Kit (GTK) with a typical home configuration.

1 Introduction

There are a variety of smartphone Apps available for monitoring and supervising a KNX network [1]. Since there is neither a KNX specification nor a certification process for Apps, the functionality and GUI are variable, sometimes confusing, and frequently require a better understanding of KNX and the commissioned network than is normally required for end users of such Apps.

The purpose of this paper is to discuss the design and implementation of a prototype Android App for a KNX network. Fig. 1 shows a simple KNX network consisting of a light, a switch, a display, a motion detector and a router. The network is configured and commissioned using ETS 4. The App connects to the KNX network over a wireless IP connection to a KNXnet/IP router.

The main goal of this project is to develop a KNX App that does not need manual configuration, requires only a basic knowledge of KNX by the user and requires no additional changes to the commissioned network.

This paper covers the following tasks needed for the successful development and operation of the App:

- Accurate planning and commissioning of the network by a certified technician (c.f. Section 2).
- Creating a detailed machine readable description of the network using an off-line tool (c.f. Section 3).

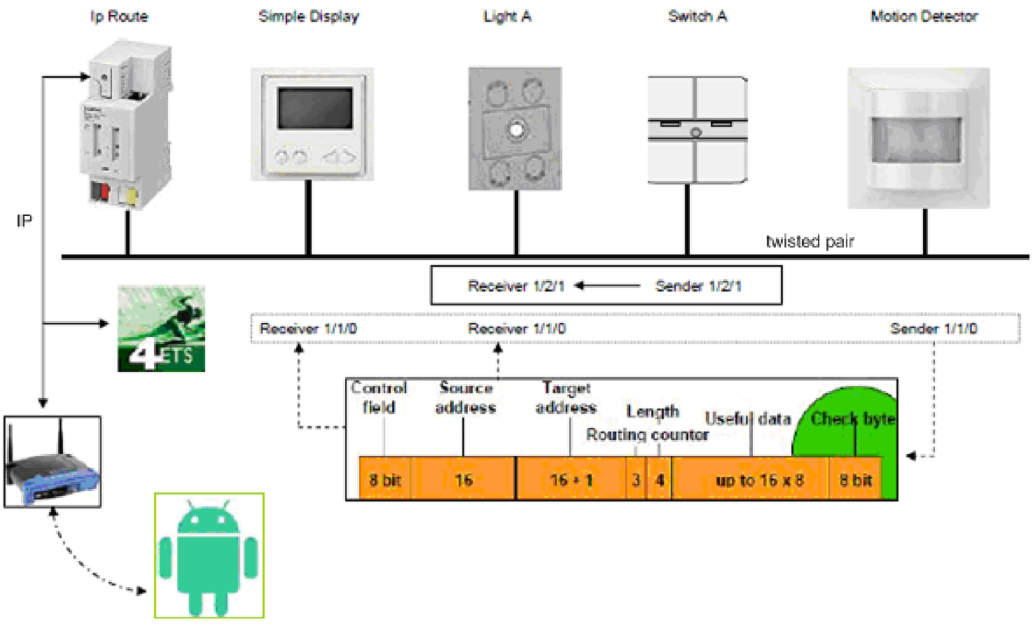


Figure 1: Simple KNX Network

- Using a communication library that implements all the KNXnet/IP core and tunneling functions as well as providing interfaces for group and management network operations (c.f. Section 4).
- Designing an App following the standard Android guidelines for operation, user interface and security (c.f. Section 5).
- Providing an intuitive, easy to use GUI with functions for importing, monitoring, control and troubleshooting (c.f. Section 6).

The completed App will be released as an open source “Android Library Project” containing user elements (UI), activities, utility functions, database tables and communication library.

2 Planning and Commissioning

Accurate planning and configuration of the KNX network is necessary to ensure the proper operation of an App targeting automatic import of ETS data. During the planning stage, not only the hardware and group addresses can be freely chosen by the technician, but also the location information, group address name, description and datapoint type are required to accurately describe the end device and its configuration.

Fig. 2 shows an example floor plan of a typical house. The building has three rooms: office, stairs and living room. The network consists of one switch in the staircase that controls the lights in the office and living room. Fig. 3 shows a possible group address configuration according to the floor plan using the ETS 4.

The group addresses are given user friendly names and datapoint types. The group address name should clearly reflect the function of the end device. A name such as “Light A” is a typical technical name used during installation, while a user friendly name such as “Light over the sofa" is more specific. In addition the technician can also add a “description” to add more detail.

When a datapoint type is missing, it defaults to a type that fits the communication object (CO) size. For example, a CO with the size of 1 bit defaults to `1.001` - a `DPT_Switch`. An incorrect datapoint type results in a misleading visual display. `1.001`, for instance, is shown as an on/off switch, while `1.002` is an enable/disable toggle switch.

The CO operational flags must be correctly set and whenever possible, enabled for reading. The configuration flag `(c)` connects the object to the bus. The read flag `(r)` indicates that a response telegram is sent after a group read telegram. The write flag `(w)` allows the shared value to be modified, and the transmit flag `(t)` sends an indication telegram when there is an external (e.g. button push) or internal (e.g. application timeout) trigger.

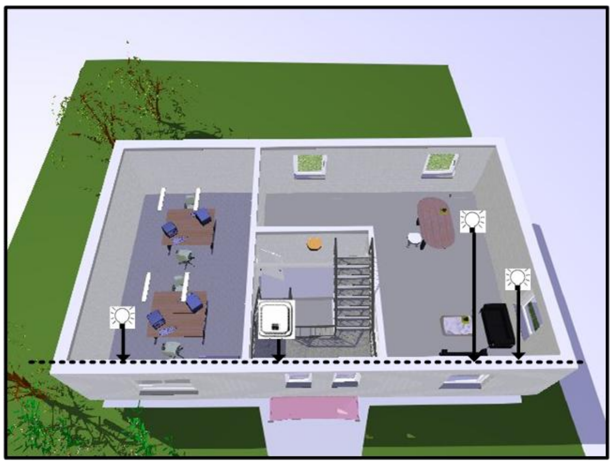
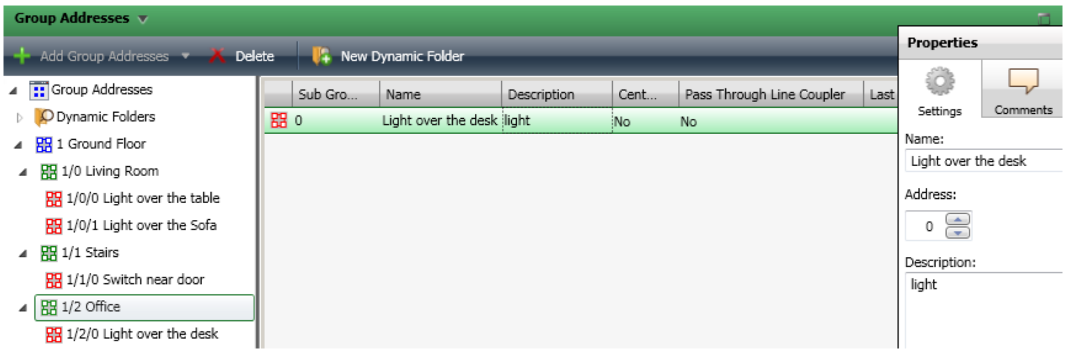


Figure 2: Simple Home Floor Plan [2]



Group Addresses

Add Group Addresses | Delete | New Dynamic Folder

- Group Addresses
  - Dynamic Folders
  - 1 Ground Floor
    - 1/0 Living Room
      - 1/0/0 Light over the table
      - 1/0/1 Light over the Sofa
    - 1/1 Stairs
      - 1/1/0 Switch near door
    - 1/2 Office
      - 1/2/0 Light over the desk

Sub Gro...	Name	Description	Cent...	Pass Through Line Coupler	Last
0	Light over the desk	light	No	No	

Properties

Settings | Comments

Name: Light over the desk

Address: 0

Description: light

Figure 3: ETS 4 Group Addresses Configuration with Description

Fig. 4 shows all the necessary configuration data needed by the App. The names are descriptive and indicate the function and the description provides additional detail. The flags and datapoint data types are defined for all the COs. The switch can only send, while the status of a light is readable. When a light value is changed, the light and dimmer both send an indication: "on" and "100%" respectively.

3 Network Configuration

The configuration of a KNX network is stored in a MS SQL database on the technician's PC, which is not accessible during runtime of a smartphone App. At least the following information is required:

- Group address (e.g. `1/1/200`)
- Flags ORed with all shared group addresses (e.g. `crw-`)
- Group address description (e.g. `light, blind`)
- Group address name (e.g. `light over desk`)
- Room location (e.g. `ground floor`)
- Datapoint type (e.g. `DPT_switch 1.001`)

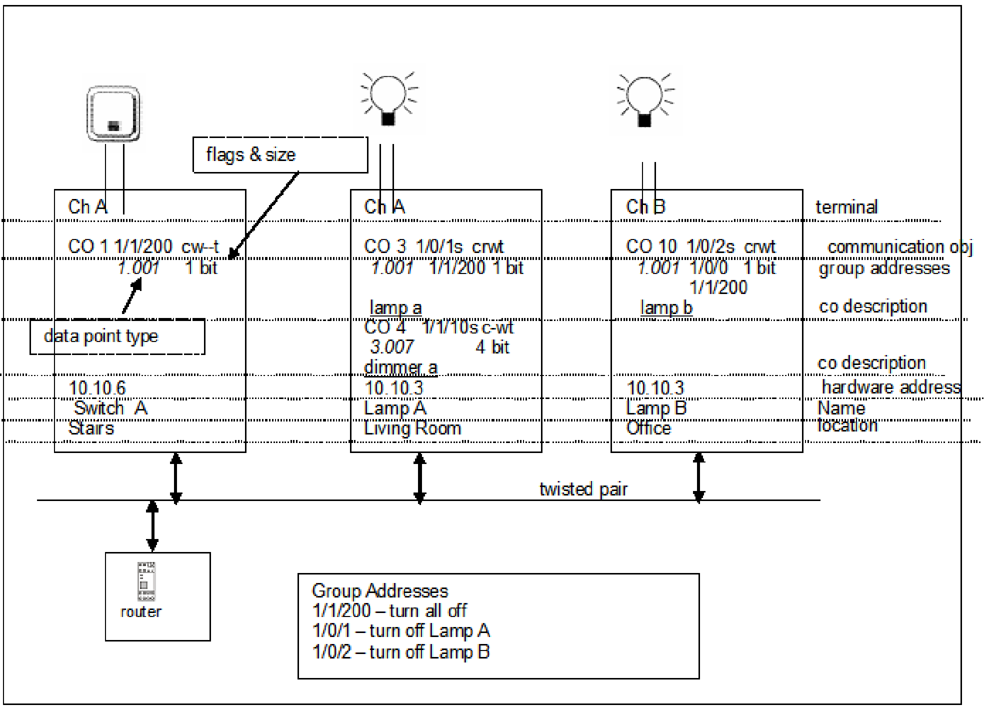
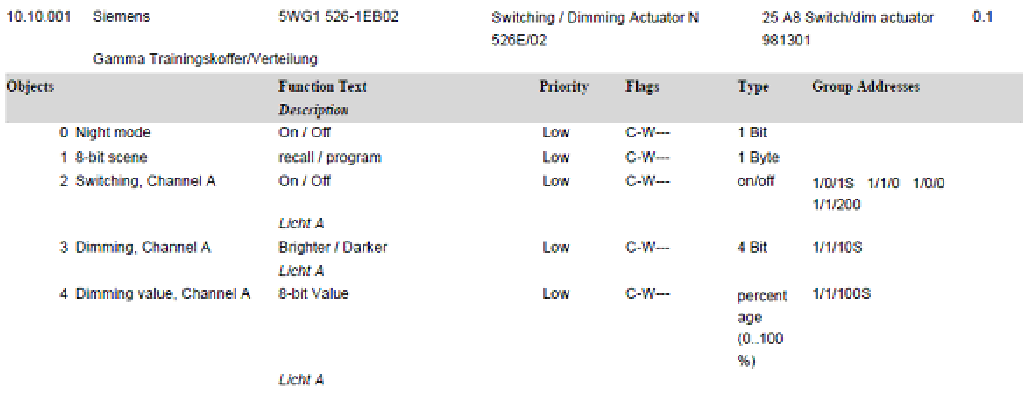


Figure 4: Overview of a Simple Home Network

- Optional hardware device (e.g. `10.10.3 switching/dimming actuator`)

Some Apps require the user to manually enter a subset of the above information. Other Apps read an ETS 4 exported group address or custom XML file to determine the network configuration. The exported PDF documents show various views of the project database. The most useful documents are: buildings, group addresses and topology. The device and group addresses in these documents are grouped together according to the building, group or hardware device.

Fig. 5 shows the topology details for a switching and dimming actuator. For example, CO #2 is an actuator for turning on/off a device connected to channel A. The CO is writable with a payload of 1 bit. The meaning of the bit (e.g. 1 = ON) may be determined by a parameter setting. When sending a telegram the group address 1/0/1 (S denotes sender) is used. The CO will read telegrams containing the sender address or any of the three associated group addresses.



10.10.001 Siemens 5WG1 526-1EB02 Switching / Dimming Actuator N 526E/02 25 A8 Switch/dim actuator 981301 0.1

Gamma Trainingskoffer/Verteilung

Objects	Function Text / *Description*	Priority	Flags	Type	Group Addresses
0 Night mode	On / Off	Low	C-W---	1 Bit	
1 8-bit scene	recall / program	Low	C-W---	1 Byte	
2 Switching, Channel A	On / Off *Licht A*	Low	C-W---	on/off	1/0/1S 1/1/0 1/0/0 1/1/200
3 Dimming, Channel A	Brighter / Darker *Licht A*	Low	C-W---	4 Bit	1/1/10S
4 Dimming value, Channel A	8-bit Value *Licht A*	Low	C-W---	percent age (0..100 %)	1/1/100S

Figure 5: Detailed Topology PDF [2]

The ETS 4 can also export the following `TXT` and `XML` files: device info, OPC, group address and project. The device info file shown below is a memory dump of the downloaded application program and commissioned group addresses for the hardware device. The group address name, location, description and datapoint type are missing.

```
DEVICE INFO

Device Info of Device 10.10.3

General
      ItemValueResource nameUnformatted value
      Mask version$0025 37
      Individual Address 10.10.3 IndividualAddress 43523
      ...
Group Communication
      Obj#0 (1 bit,--CT--,Low)1/1/0
      Obj#1 (4 bit,--CT--,Low)1/1/10
      Obj#2 (1 bit,--CT--,Low)1/1/1
      Obj#3 (4 bit,--CT--,Low)1/1/11
      Obj#4 (1 bit,--CT--,Low)1/1/2
      Obj#5 (4 bit,--CT--,Low)1/1/12
      Obj#6 (1 bit,--CT--,Low)1/2/2
      Obj#7 (1 bit,--CT--,Low)1/2/0
      Obj#8 (1 bit,RWC---,Low)1/1/90
      Obj#9 (1 bit,RWC---,Low)1/1/91
      Obj#10 (1 bit,RWC---,Low)1/1/92
...
```

The OPC export shown here contains more information, namely building location, name, and function type, but is missing the datapoint type and operational flags.

```
OPC EXPORT

Gamma Koffer Technikum
Gamma Trainingskoffer.Zentralfunktionen.1/0/0 Beleuchtung aus und Jalousie auf
  EIS 1 'Switching' (1 Bit) Low
Gamma Trainingskoffer.Zentralfunktionen.1/0/1 Wohnbereich verlassen
  EIS 1 'Switching' (1 Bit) Low 1/0/0 1/1/0 1/1/1 1/1/200
Gamma Trainingskoffer.Zentralfunktionen.1/0/2 Wind EIS 1 'Switching' (1 Bit)
  Low
Gamma Trainingskoffer.Beleuchtung.1/1/0 Licht A e/a EIS 1 'Switching' (1 Bit)
  Low
Gamma Trainingskoffer.Beleuchtung.1/1/1 Licht B e/a EIS 1 'Switching' (1 Bit)
  Low
Gamma Trainingskoffer.Beleuchtung.1/1/2 Licht C e/a EIS 1 'Switching' (1 Bit)
  Low 1/0/0 1/0/1 1/1/200
Gamma Trainingskoffer.Beleuchtung.1/1/3 Licht D e/a EIS 1 'Switching' (1 Bit)
  Low 1/0/0 1/1/200
Gamma Trainingskoffer.Beleuchtung.1/1/4 Licht E e/a EIS 1 'Switching' (1 Bit)
  Low 1/0/0 1/1/200 1/1/201
Gamma Trainingskoffer.Beleuchtung.1/1/5 Licht F e/a EIS 1 'Switching' (1 Bit)
  Low
Gamma Trainingskoffer.Beleuchtung.1/1/10 Licht A heller/dunkler
  EIS 2 'Dimming - control' (4 Bit) Low
...
```

The group address export is similar to the OPC export, but there are no datapoint types or operational flags.

```
GROUP ADDRESS

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<GroupAddress-Export xmlns="http://knx.org/xml/ga-export/01">
  <GroupRange Name="Gamma Trainingskoffer" RangeStart="2048" RangeEnd="4095">
    <GroupRange Name="Zentralfunktionen" RangeStart="2048" RangeEnd="2303">
      <GroupAddress Name="Beleuchtung aus und Jalousie auf" Address="1/0/0" />
      <GroupAddress Name="Wohnbereich verlassen" Address="1/0/1" />
      <GroupAddress Name="Wind" Address="1/0/2" />
      <GroupAddress Name="Ereignisprogramm 1/2" Address="1/0/3" />
    </GroupRange>
    <GroupRange Name="Beleuchtung" RangeStart="2304" RangeEnd="2559">
      <GroupAddress Name="Licht A e/a" Address="1/1/0" />
      <GroupAddress Name="Licht B e/a" Address="1/1/1" />
      <GroupAddress Name="Licht C e/a" Address="1/1/2" />
      <GroupAddress Name="Licht D e/a" Address="1/1/3" />
      <GroupAddress Name="Licht E e/a" Address="1/1/4" />
      <GroupAddress Name="Licht F e/a" Address="1/1/5" />
...
```

It is also possible to export the complete project database. For the default configuration of the Siemens GTK [2, 3] 20 files are exported that include among many other items: defined datapoint types, group addresses, commissioned device application programs and CO parameters. An overview of the XML layout is shown in Fig. 6. Since the specification [4] does not go into detail, parsing the various XML files is challenging. There is an XSLT parser [5] that converts the project files into a `datapoints.xml` file suitable for import by the Calimero library. The file contains the group address name, datapoint type, caching and expiry parameters, but the location and operational flags are missing.

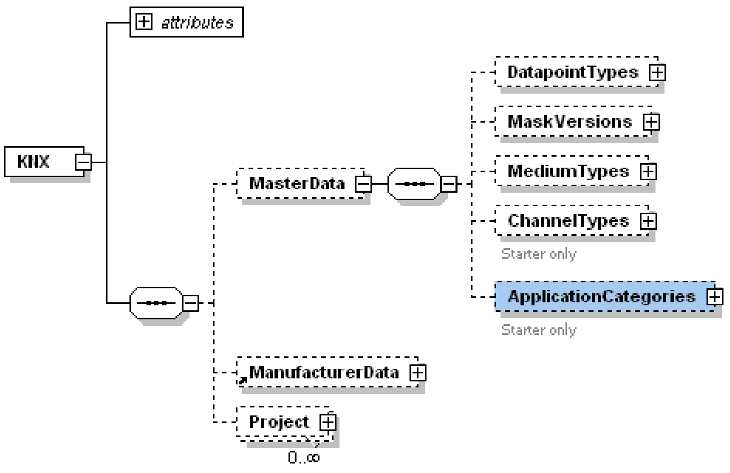


Figure 6: Project XML Schema [4]

```
DATAPOINTS.XML (XSLT)

<?xml version="1.0" encoding="iso-8859-1"?>
<datapoints>
  <datapoint stateBased="true" name="Wohnbereich verlassen" mainNumber="1"
             dptID="1.001" priority="Low">
    <knxAddress type="group">2049</knxAddress>
    <expiration timeout="0"/>
    <updatingAddresses>2504</updatingAddresses>
    <updatingAddresses>2305</updatingAddresses>
    <updatingAddresses>2048</updatingAddresses>
    <invalidatingAddresses> </invalidatingAddresses>
  </datapoint>
  <datapoint stateBased="true" name="Wind" mainNumber="1" dptID="1.017"
            priority="Low">
    <knxAddress type="group">2050</knxAddress>
    <expiration timeout="0"/>
    <updatingAddresses> </updatingAddresses>
    <invalidatingAddresses> </invalidatingAddresses>
  </datapoint>
...
```

Since no one file or combination of exported files provides all the significant information, an off-line parser has been developed. The tool parses the project data base into a collection of .NET objects that are used to output three xml files: `datapoints.xml`, `groupaddress.xml` and `devices.xml`. The `datapoints.xml` is read by the Calimero library and has the same format as the XLST transformed file. The `groupaddress.xml` contains additional information about the group addresses and the `devices.xml` contains a list of all the hardware devices with device address and product description. Since more than one CO may share the same group address, the tag `nrSharing` denotes how many COs could possibility be read or written with one telegram. The operational flags are ORed together for all the COs having the same group address. While parallel operation of several CO is a major feature of KNX networks, the status returned when reading parallel COs may be not be correct for all the devices.

```
GROUPADDRESS.XML

<?xml version="1.0" encoding="iso-8859-1"?>
<groupaddresses>
  <!-- groupName="Light over the table" -->
  <groupaddress address="2049" flags="crwt"
                areaName="Ground Floor" lineName="Living Room"
                description="light" nrSharing="2" >
  </groupaddress >

 <!-- groupName="Switch near door" ->
  <groupaddress address="2050" flags="c-w-"
                areaName="Ground Floor" lineName="Stairs"
                description="switch"
                description="switch" nrSharing="1" >
  </groupaddress >
<groupaddresses>
```

The `devices.xml` is used for diagnostic purposes to check if all the devices are operational.

```
DEVICES.XML

<?xml version="1.0" encoding="iso-8859-1"?>
<deviceaddresses>
  <deviceaddress address="4352"
                 description="IP-Router N 146"
  <deviceaddress address="4353"
                 description=" Switching / Dimming Actuator N 526E/02"
<deviceaddresses>
```

A `scenes.xml` file (see below) must be created by App user. It is used to change the values of a collection of devices - a scene. This is an optional file, because detailed information about KNX and the network is required.

SCENES.XML

```
<scenes>
  <script scenenr="0" title="Turn all lights on">
    <command groupAddress="1/1/0" cmd="on" comment="Light A - Living room" />
    <command groupAddress="1/1/1" cmd="on" comment="Light B - Living room" />
    <command groupAddress="1/1/2" cmd="on" comment="Light C - Living room" />
  </script>
</scenes>
```

4 Calimero Communication Library

The open source Java Library Calimero [6] provides not only IP communication to a KNX network, but also a high level abstraction for handling telegrams. The various functions are shown in Fig. 7.

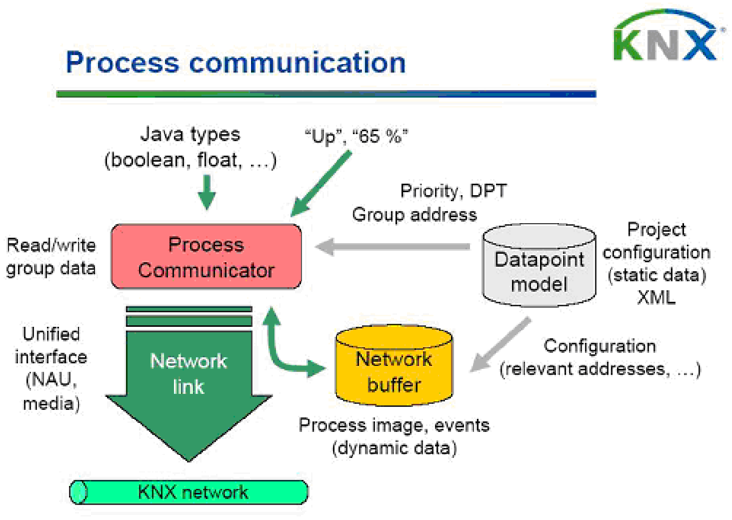

Process communication
KNX
Java types (boolean, float, ...)
"Up", "65 %"
Priority, DPT
Group address
Read/write group data
Process Communicator
Datapoint model
Project configuration (static data) XML
Unified interface (NAU, media)
Network link
Network buffer
Configuration (relevant addresses, ...)
Process image, events (dynamic data)
KNX network

Figure 7: Calimero process communication [7]

The library provides the following KNXnet/IP functionalities: core service, tunneling service, routing service, and device management service. The low level APIs provide point-to-point functions for management of the device as well as point-to-multi-point group functions. Both blocking and non-blocking transmission is possible. The high level functions hide (abstract) all the underlying details of the telegram and provide a simple interface using a datapoint object. A datapoint consists of a group address, a datapoint type, a name, caching and timer parameters. The telegram payloads (shared value) are translated from strings to binary values when writing and translated from binary values to strings for confirmations and notifications. For example, a light (1 bit) value is translated to the string `ON` or `OFF` for the datapoint value `1.001`. The multi-threaded library provides an event call back handler for signaling confirmations and indications from the KNX network. An XML file consisting of a list of datapoints can be imported into the library. This list is used by the group functions, for caching of values and setting expiration timers.

The App uses the group functions for monitoring and changing the value of an end device. The telegrams are packed into an IP/UDP frame (see Fig. 8) for communication to the KNXnet/IP router. There are additional functions for setting up and tearing down the connection to the router. For the group functions, there are 3 scenarios to consider:

The read function Fig. 9 (a) sends (tunnels) a `GroupRead` telegram with a group address to the router, which writes the telegram to the bus. One or more devices may read the telegram and respond with a confirmation telegram. The confirmation is no guarantee that the read operation was successful, only proof that a device read the telegram. After the device's application has decoded the telegram and read

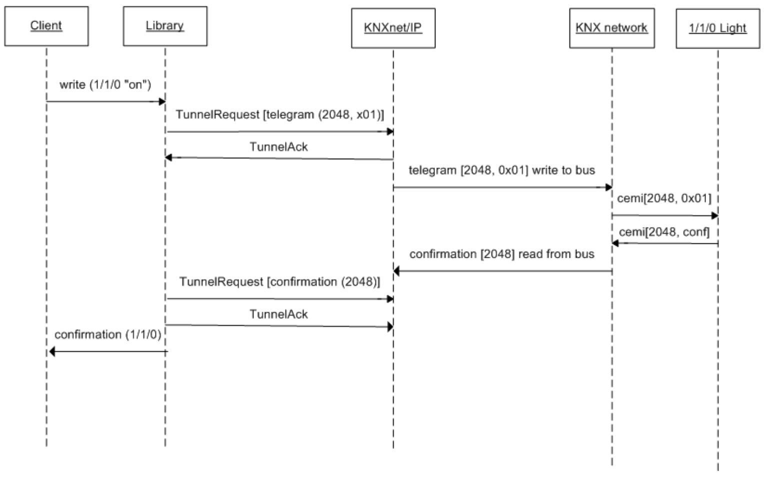

Client
Library
KNXnet/IP
KNX network
1/1/0 Light
write (1/1/0 "on")
TunnelRequest [telegram (2048, x01)]
TunnelAck
telegram [2048, 0x01] write to bus
cemi[2048, 0x01]
cemi[2048, conf]
confirmation [2048] read from bus
TunnelRequest [confirmation (2048)]
TunnelAck
confirmation (1/1/0)

Figure 8: Tunneling Scenarios

the corresponding shared value, an indication telegram with the status of the read CO is returned. If the group address (GA) is configured for more than one CO, then additional indications may be returned for the same GA.

The write function Fig. 9 (b) sends a telegram with a payload and group address. After the write is confirmed, there may be none, one or more indications returned. For example, turning on a light may result in an indication that the light for the GA is ON. A second indication with a different GA may indicate that the dimmer is at 100%.

Finally, indications Fig. 9 (c) can be received at anytime if a CO value changes. For example, pressing a light switch writes an indication to the bus. Also, indications can be triggered internally by a device polling another device. For example, a display may poll the temperature sensor.

The following code sample scenario illustrates the group level commands.

```
main()
{
  // Define datapoint

  GroupAddress gaLightA = new GroupAddress(0, 0, 2); // group address
  StateDP dpLightA = new StateDP(gaLightA, "Light A", 1, "1.001"); // see xml below

  pc = new ProcessCommunicatorImpl(netLinkIp); // high level connection

  String status = m_pc.read(dpLightA);          // = ON or OFF

  pc.write(dpLightA, "on");                     // turn on light

  // Monitor any network changes

  myNetworkLinkListener = new MyNetworkLinkListener();  // add event handler
  netLinkIp.addLinkListener(m_myNetworkLinkListener);   // recv confirmations
                                                        // and indications
}


class MyNetworkLinkListener
{
  public void confirmation(FrameEvent arg0) // response to a group read/write

  public void indication(FrameEvent arg0)   // changed shared value
```

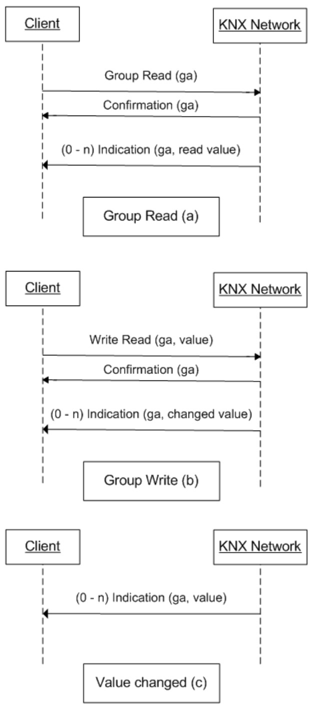


Figure 9: Telegram Scenarios

```
  public void linkClosed(CloseEvent arg0)   // network connection closed

}
```

To import the above declared datapoint, the xml layout is:

```
<datapoint stateBased="true" name="Light A" mainNumber="1" dptID="1.001"
           priority="Low">
 <knxAddress type="group">2</knxAddress>  <!-- 0/0/2 -->
 <expiration timeout="0" />
 <updatingAddresses> </updatingAddresses>
 <invalidatingAddresses> </invalidatingAddresses>
</datapoint>
```

The Calimero library comes complete with source code, JavaDoc, unit tests and sample code. Since the library was developed for J2SE, it can be used with Java SE programs and Android Apps. The library is modular, concise and fast. It can be extended and modified, and unused modules can be removed.

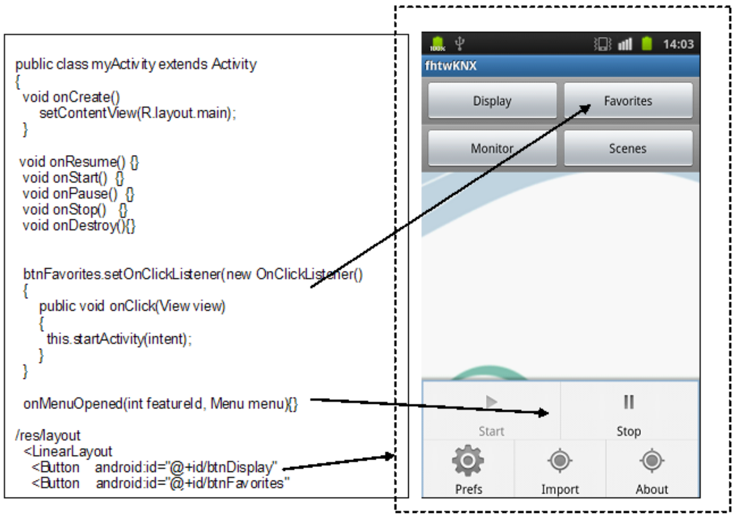
5 Android Overview

Android is an open source platform for development of applications for mobile devices. The Android stack consists of the Linux kernel, hardware drivers, libraries and runtime, application framework and other applications. For App development this stack is hidden and not of immediate importance. Access to the hardware, file system, Internet, and other Apps is provided by the application framework (mainly mangers) and the runtime libraries. The main building blocks for an App are the Activity, Intent, Service and Content Provider.

While there are alternative development environments (e.g. PhoneGap, WebView, HTML5 [8], App Inventor, scripting languages, C/C++), the Android Java SDK provides the best integration for the Android

OS. The Java runtime is based on the open source Harmony Java and the Dalvik VM runtime, which is tailored to mobile devices. The programming model is similar to developing a Java servlet. There is no `main()` method. The application classes are installed in a framework (e.g. Tomcat). The `main` is started by an HTTP URL referencing the class. For Android the installation of the App, the main and related activities, resources and manifest are saved in a secure area of the Linux file system. The activity is launched via user inputs, intents (messages), alerts, notifications or internally by the OS. When the activity is started for the first time, `onCreate` method is called. Unlike a PC program, an activity has a life cycle that is controlled by the Android OS.

Fig. 10 shows some of the functions and resources used by an activity. The layout of the screen is declared in a separate layout file. In the Figure the linear layout results in the buttons being arranged from top to button and from left to right. The activity installs an button event handler. For example, whenever the button `Favorites` is clicked or tapped, the activities `onClick` function is called. When the `Menu` button on the smartphone is pressed, the menu items are inflated according to the menu layout. In the case of the menu, the activity is notified when a menu button is pressed.



```
public class myActivity extends Activity
{
 void onCreate()
     setContentView(R.layout.main);
 }

 void onResume() {}
 void onStart()  {}
 void onPause()  {}
 void onStop()   {}
 void onDestroy(){}


 btnFavorites.setOnClickListener(new OnClickListener()
 {
     public void onClick(View view)
     {
      this.startActivity(intent);
     }
 }

 onMenuOpened(int featureId, Menu menu){}

/res/layout
 <LinearLayout
   <Button   android:id="@+id/btnDisplay"
   <Button   android:id="@+id/btnFavorites"
```

Figure 10: Main Activity: Code and Screen layout

Fig. 11 shows the lifecycle of an activity. When the activity is started `onCreate`, `onStart` and `onResume` are called. Pressing the Home key destroys the activity: `onPause`, `onStop` and `onDestroy` are called. Pressing the Back key loses focus, but does not destroy the activity: `onPause` and `onStop` are called. Any state change also results in calling various activity functions.

Since the internal memory of the smartphone is limited, the Android OS performs garbage collection when needed and garbage collects any idle activities. Thus, the activity needs to save and restore any important data over the course of the activity's lifecycle. For example, rotating the smartphone causes a configuration change that destroys and restarts the activity in order to repaint the screen for the new orientation.

An App usually consists of several activities with different screens/views that provide a flow similar to a browser. When starting a new activity due to the user pressing a button or menu item, the current activity is paused, placed on task stack and a new activity is started. The previous activity is restarted by pressing the Back key. Intents are used to start a specific activity(class) or any activity that meets some filter requirement (e.g. action = view). Any activity can also dynamically register for system wide broadcast intents.

A service is used for longer running tasks in the background with no user interface. The service has a simple lifecycle: create, start, stop and destroy. Services are forced to stop only when memory is critical and can be restarted automatically. A service can be started using an intent or explicitly started.

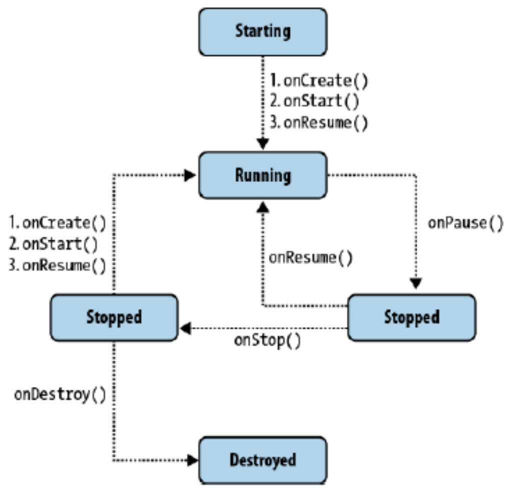


Figure 11: Android Activity Lifecycle [9]

The communication between the caller and service uses either an asynchronous intent or synchronous remote procedure calls (RPC).

The persistent storage uses the Linux file system either with explicit calls to the file system or indirectly using shared preferences or SQLite. The SQLite database is implemented as a file. Since it is stored under a unique user ID of the process that created the database, it is isolated from any other application. To permit sharing of an SQLite table, content providers are used. The content provider provides the SQL create, read, update and delete (CRUD) functions and cursors.

The App runs in its own process with its own unique user ID and a forked copy of the Dalvik VM (Zygote). The security is a combination of Linux OS security and explicitly defined permissions that a user must agree to before installing the App. The GUI is updated using a single thread. Should this thread block while executing a callback in an activity or service, the OS will notify the user to either forcibly close the activity or continue to wait. To avoid Application Not Responding (ANR) any function requiring more than 10 seconds must start a background thread, use the Android asynchronous functions (e.g. asynchronous, looper or loader) or call a service.

Android offers a wide variety of UI elements, called widgets. Fig. 12 shows a small sample of the most common widgets. The simple widgets are: `textview`, `edittext`, `button`, `checkbox`, `spinner`, `progressBar`, `slider` and `radiobutton`. The composite widgets are: `listview`, `gridview`, `scrollview`, `sliding drawer`, `webview` and `viewpager` and `tabhost`. For images there are the widgets: `imageview`, `gallery`, `time and date picker`. Animations are available to provide feedback when pressing a button, movement by displaying a group of images with short pause and focus change. Custom views can be developed using an existing view or completely new view using OpenGL. Gestures, such as short and long taps, swipe up/down or left/right provide a more App-like user input. In addition feedback using sounds or vibration is also possible. There are menus and context menus that can be replaced with newer action bar. Themes and styles can be defined for one activity or the application. There are a variety of layouts: `linear`, `relative`, `absolute` and `custom`.

On the Home screen, custom Android widgets can be developed to show the latest status or a summary of important information. The notification bar at the top of Home screen uses icons for notification of some important event. Dialogs and toasts can ask for user input or display a short notice, respectively. A newer UI element, fragments, can be embedded in an activity. They allow multi-pane UI and reuse in other activities. Finally, different screen orientation, screens size and 9-patch graphics are supported.

There are many tools available from standard Java development: static analysis, memory leak and profiling. Java is the primary development library using the IDE Eclipse. The Android Java is a subset of the Java SE. Most of the core Java classes are available while AWT and SWING are missing and other

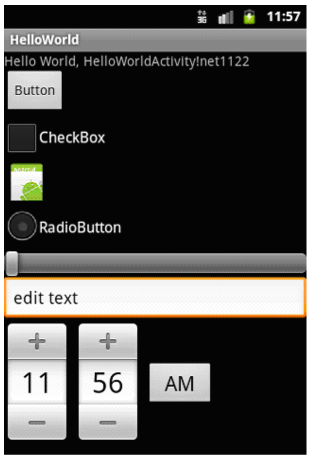


Figure 12: Some Common Android Widgets

Java classes have been changed. The Android Java is similar enough to the Standard Edition Java, that Java programs for a PC can be compiled using the `Android.jar` and they will execute on Oracle's runtime.

An App should be targeted for versions 2.2 - 4.1, currently version 2.3.3 being the most widely used. Apps for this version may use the newer 3.0 GUI function (e.g. action bar) with back ported libraries. The App must be responsive, intuitive and easy to use. The memory footprint should be as small as possible. The CPU usage should be kept to a minimum to reduce draining the battery. Although the Dalvik runtime provides efficient garbage collection and just-in-time (JIT), an App can still leak memory and waste CPU with poorly designed functions. For example, a common bug is maintaining a reference to an Activity class that is not longer visible. The garbage collector is prevented from reclaiming the class object, because of the static reference.

An user will no longer trust an App if it crashes requiring a "Force close", or blocks the GUI thread too causing ANR. The App should crash gracefully and write out the crash details to the log file. If possible the App can use Application Crash Report for Android (ACRA) [10] to report the crash cause to the web server.

6 Prototype Implementation

Fig. 13 presents the architecture of an App being connected to a KNX network – a Siemens Training Kit (GTK) over a wireless connection in this example. The App is divided into two packages: client and service.

The KNXService runs in the background in its own process. The service is responsible for the connection to the KNX network using the Calimero Library. A service is used because the KNXnet/IP router requires a periodic heartbeat message and the network is monitored for any asynchronous indications such as a button being pushed, even when no client is active.

The service provides an remote procedure call (RPC) interface for reading and writing datapoints as well as a call back function for signaling events from the network. A SQLite table indexed by the group address is used to save the latest value written, read or received via an indication along with a timestamp. The Calimero provides an internal caching and expiry function that would need to persist whenever the service is stopped. The database replaces the caching and the service can implement the expiry timeout function. The service also broadcasts the current state of the network connection. This intent is useful for indicating the status of the connection in case the user loses the wireless connection or the service is not running.

The drain on the battery is a very important consideration. The service is manually started and

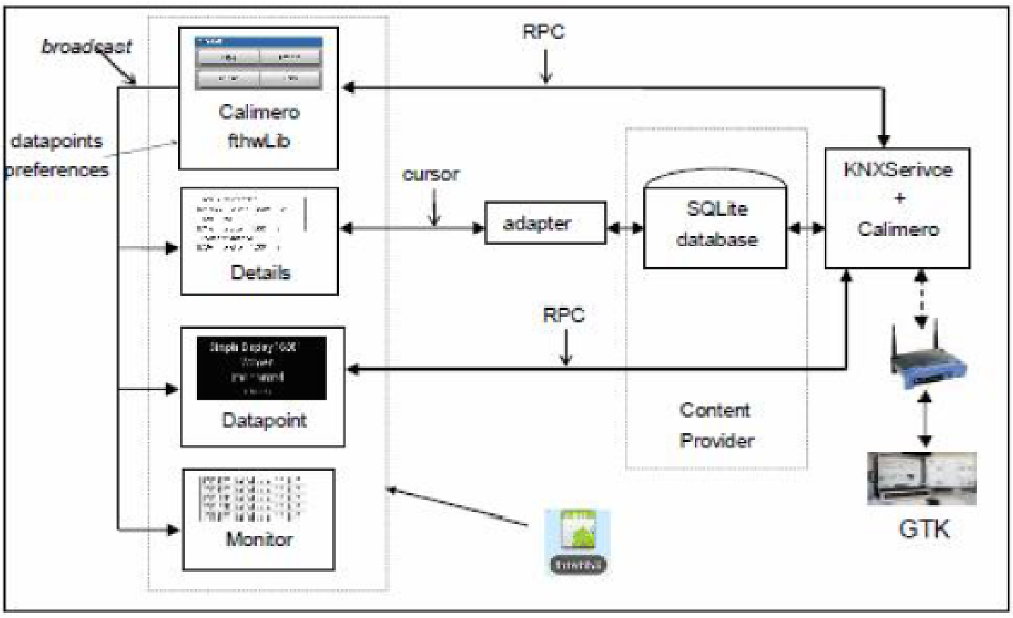


Figure 13: FHTW KNX App Overview with wireless access to GTK

stopped by the user. Calimero does not poll the network, but blocks listening for indications or the link closing. The client does not poll the database, but receives update notifications. For activities not using a managed database cursor, updates are received using broadcast intents. The SQLite database is accessed using a content provider, because the client and service execute in their own process. The Linux security prevents an App from accessing the private files, the SQLite tables, of another App.

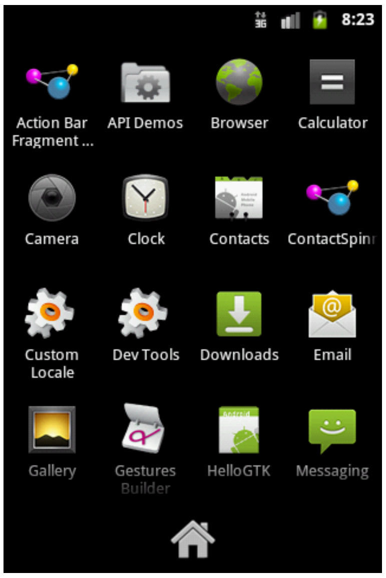
The client is a collection of activities using simple, composite widgets and menu items. The client is responsible for importing the network configuration data, starting and stopping the service, displaying the datapoints with their current values, changing the state of a datapoint and performing simple diagnostic tests.

The Home activity is started from the launcher screen (Fig. 14a) or using a home Android widget (Fig. 14b). The widget also displays the last indication status. The home view (Fig. 14c) provides buttons for the main functions and menu items for the auxiliary functions. The upper left icon in the notification status bar indicates the state of the connection to the network. The time that the connection was established can be read by sliding down the notification message window. Fig. 15 is an overview of the operational workflow.

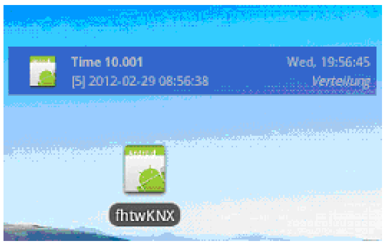
The display view is shown in Fig. 19a. All the datapoints along with their current state are shown in a list that can be sorted by various criteria. The group address name, value, location and group address are shown. The "rdsts" indicates that no value has been or can be read for this group address. During startup the service reads all the readable group addresses. The only possible value that could be displayed for a write only group address is the last written value, which is not done. The display will automatically update should any value change due to an indication.

A short tap on an item in the list brings up a dialog for putting the datapoint in a “Favorites” list (Fig. 16). A long tap on an item will bring up a specific view of the datapoint according to its datapoint type. Fig. 17a shows the view of an on/off light; Fig. 17b shows the view of dimmer; Fig. 17c changes a temperature value and Fig. 18 is changing the time.

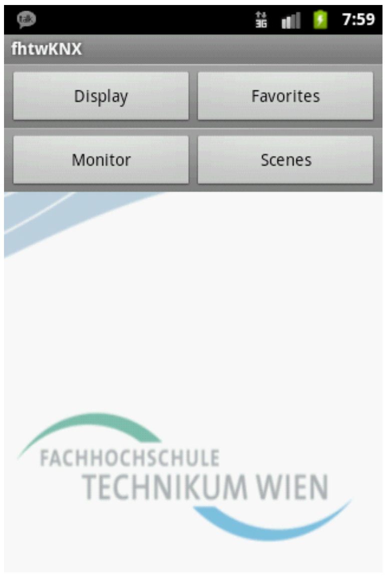
The Favorites view is just a shorter version of the display view containing the most frequently accessed group addresses. The monitor view (Fig. 19b) shows all the current activities on the network (i.e. the indications). Any indication not found in the datapoint table is ignored. Displaying telegrams in hex and guessing the size of payload would be confusing. A "learning" mode is not necessary as the complete project database has been imported.



(a) Android Launcher Screen



(b) Custom Android Widget on Home Screen



(c) Home Activity connected to GTK

Figure 14: Screenshots Launching

The scenes view (Fig. 19c) is simply a display of a pre-configured sequences of group write telegrams selected by scene name. The scenes are imported in XML format. When a scene is selected, the App will write all the commands to the network.

The Home activity menu items (Fig. 20a) are for starting/stopping the service, importing and setting preferences. When the service is started, the network configuration datapoints file is read and parsed into an hashmap, which is used to create the records for the datapoints table. After successfully connecting to the KNX network, all the readable group addresses are read and the table updated. Finally, the service broadcasts that the network is up and the service is ready. The client also imports the datapoints file to create a hashmap of the all the datapoints.

All the necessary user settings and IP addresses are set in the preferences activity as shown in Fig. 20b. The preferences are saved in an App local file. Various tests for the specified preferences are possible by selecting a test menu item. The possible tests are shown in Fig. 20c.

The "Check WiFi" tests the connection to the wireless access point. The "Check KNXnet/IP" performs a discovery function searching for all active "KNXnet/IP" routers. The "Check Datapoints" reads, parses and displays the network configuration file. The "Connect Test" uses the configured local and remote addresses to test connect to the KNX network.

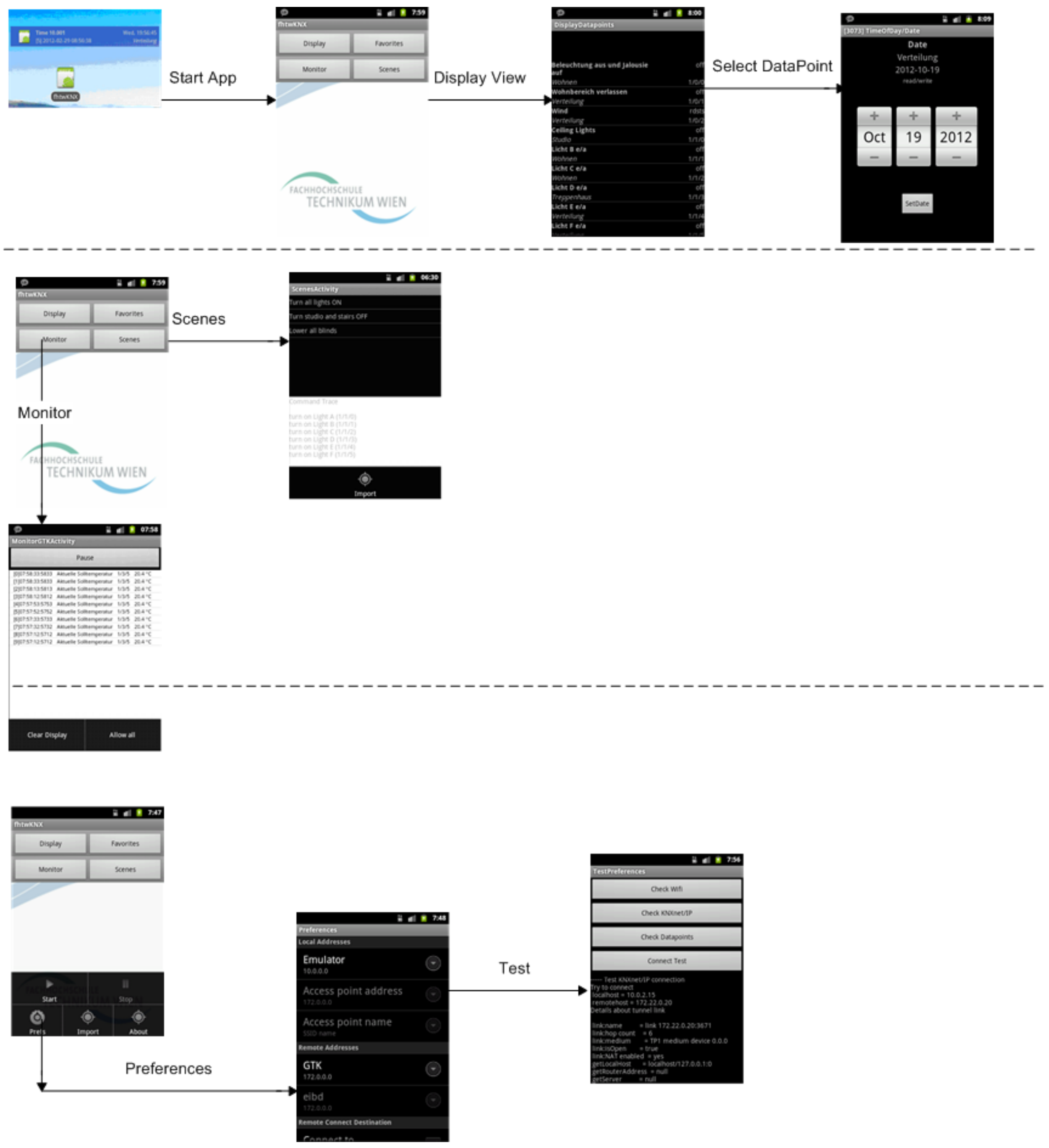


Figure 15: Overview of App Operational Workflow

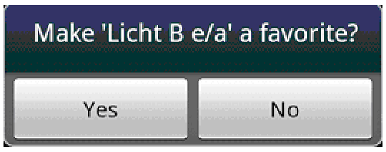
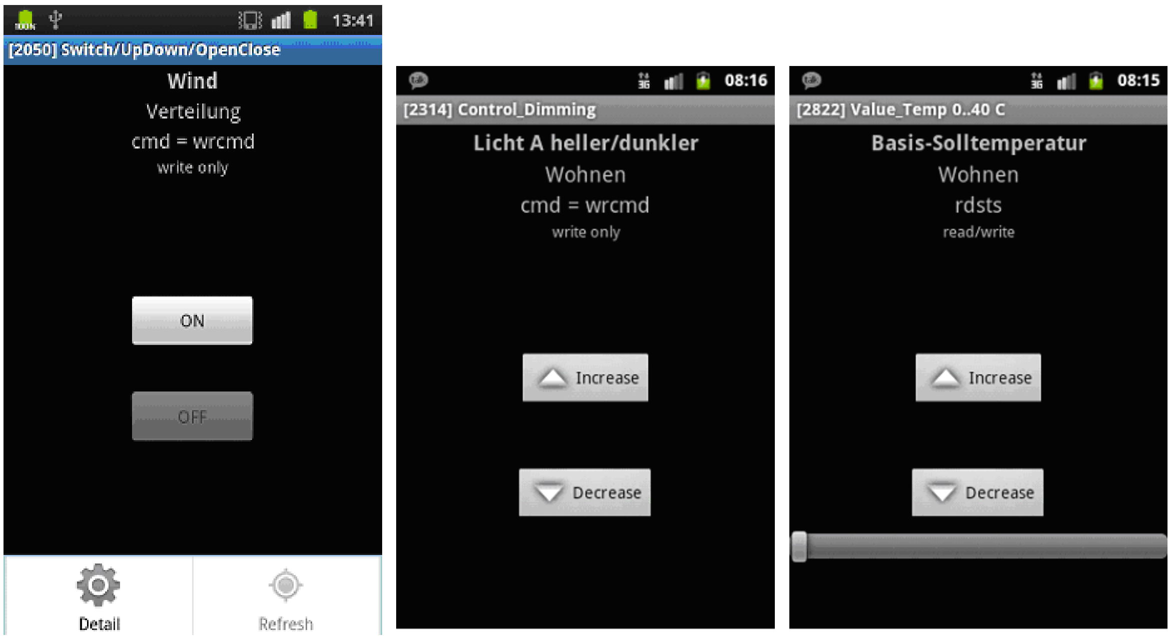


Figure 16: Favorites Dialog



(a) Datapoint Type 1.001 View **(b)** Datapoint Type 3.007 View **(c)** Datapoint Type 9.001 View

Figure 17: Screenshots Datapoint Types

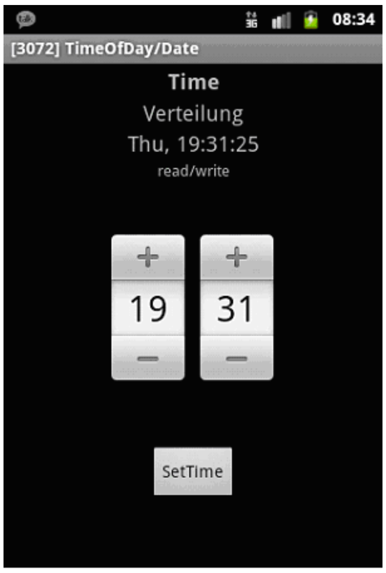
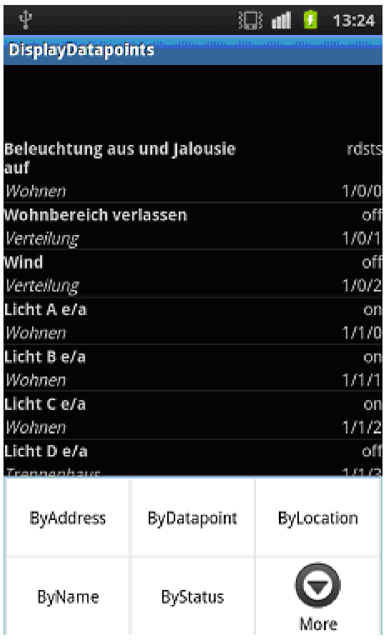
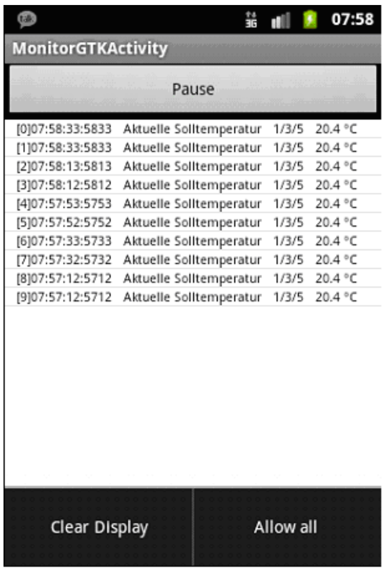


Figure 18: Datapoint Type 10.001 View


DisplayDatapoints

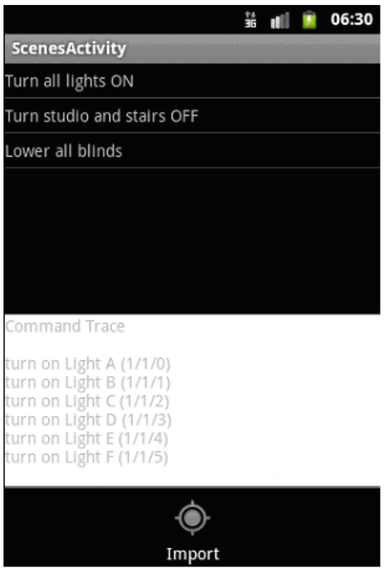
Beleuchtung aus und Jalousie auf	rdsts
Wohnen	1/0/0
Wohnbereich verlassen	off
Verteilung	1/0/1
Wind	off
Verteilung	1/0/2
Licht A e/a	on
Wohnen	1/1/0
Licht B e/a	on
Wohnen	1/1/1
Licht C e/a	on
Wohnen	1/1/2
Licht D e/a	off

(a) Display View with Sort Menu


MonitorGTKActivity

```
[0]07:58:33:5833  Aktuelle Solltemperatur  1/3/5  20.4 °C
[1]07:58:33:5833  Aktuelle Solltemperatur  1/3/5  20.4 °C
[2]07:58:13:5813  Aktuelle Solltemperatur  1/3/5  20.4 °C
[3]07:58:12:5812  Aktuelle Solltemperatur  1/3/5  20.4 °C
[4]07:57:53:5753  Aktuelle Solltemperatur  1/3/5  20.4 °C
[5]07:57:52:5752  Aktuelle Solltemperatur  1/3/5  20.4 °C
[6]07:57:33:5733  Aktuelle Solltemperatur  1/3/5  20.4 °C
[7]07:57:32:5732  Aktuelle Solltemperatur  1/3/5  20.4 °C
[8]07:57:12:5712  Aktuelle Solltemperatur  1/3/5  20.4 °C
[9]07:57:12:5712  Aktuelle Solltemperatur  1/3/5  20.4 °C
```

(b) Monitor View


ScenesActivity

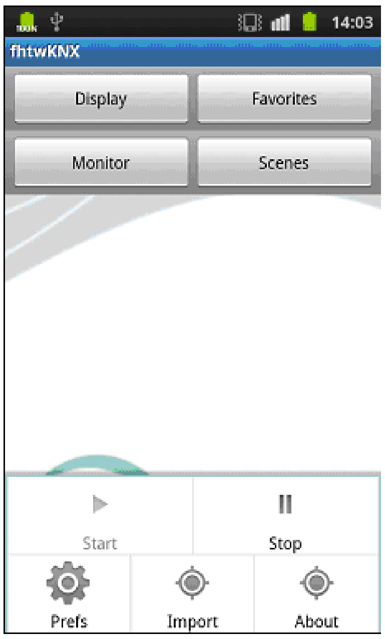
Turn all lights ON
Turn studio and stairs OFF
Lower all blinds

Command Trace

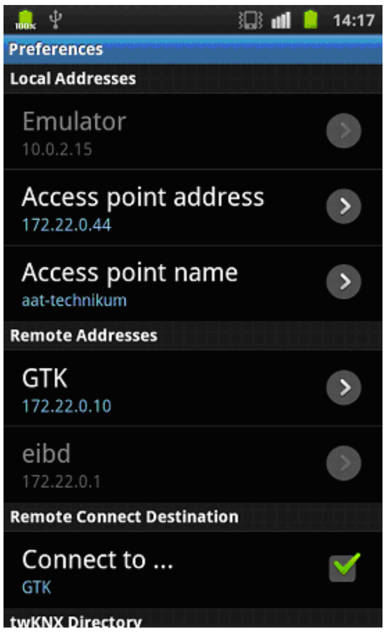
```
turn on Light A (1/1/0)
turn on Light B (1/1/1)
turn on Light C (1/1/2)
turn on Light D (1/1/3)
turn on Light E (1/1/4)
turn on Light F (1/1/5)
```

(c) Scenes View

Figure 19: Screenshots Views



(a) Home View with Menu


Preferences

Local Addresses

Emulator
10.0.2.15

Access point address
172.22.0.44

Access point name
aat-technikum

Remote Addresses

GTK
172.22.0.10

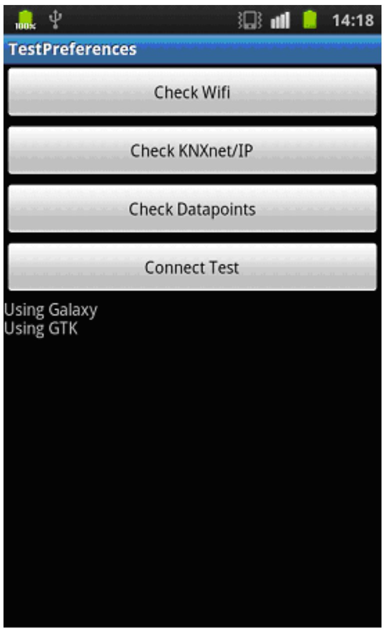
eibd
172.22.0.1

Remote Connect Destination

Connect to ... ✓
GTK

twKNX Directory

(b) Preferences View


TestPreferences

Using Galaxy
Using GTK

(c) Tests View

Figure 20: Screenshots Views

7 Evaluation

The prototype App demonstrates the important operational and development aspects. First, the technician must carefully configure the KNX home with meaningful names, correct datapoints, accurate building location, and useful descriptions. Second, using an offline program the project configuration is parsed into several XML files. The App user does not need to enter or change the configuration. Third, the App can assist with determining the KNXnet/IP addresses and with verifying the wireless and configuration files. Finally, after gaining some experience with the network, the App user can create a favorites list for easier operation.

The App works best with lights, blinds, displays and temperature devices. The following is a list of improvements:

- Move the service package into the client App package.
- Import hardware devices for simple diagnostic testing of the network.
- Change the Home view layout.
  - Use the new action bar
  - Use the view pager for swiping to screens composed of similar device types (e.g. light, blinds).
- Change the details Listview.
  - Display group name instead of group address.
  - Add an icon indicating the type of end device.
- Use gestures for use input.
  - Use swipe up/down for dimming.
  - Use swipe left/right for showing similar group of devices or more details about the datapoint.
- Use animation for lights (e.g. on or off) and blinds (movement up/down).
- Create activity fragments for reuse of common UI elements.
- Add Application Crash Report for Android (ACRA) for monitoring crashes.
- Optimize code and view handling use static analysis and Android viewer.
- Add automatic testing procedures.
  - Use junit, instrument and random (Monkey) user events.
  - Perform regression testing using software simulator and demo board.
- Add function to save and restore functions for preferences and configuration from local files and cloud.
- Convert App to an open source Android Library project.
- Add Android permissions specific for the App operation.
- Change any function or RPC that blocks for more than a second to an asynchronous operation.

8 Conclusion

While there are several Apps for KNX and iPhone, very little has been written about the actual development with the exception of [11].

Most of the KNX Apps have similar features, but provide different user experiences – visually and operationally. Performing tests with various (free, lite and purchased) Android and iPhone Apps, shows that KNX Apps are still in the development stage. Some of the user interfaces are confusing, the configuring of the network datapoints is time consuming and detailed KNX knowledge is required. The operational flow sometimes is misleading. The favorites and grouping of similar devices is not easy to understand or follow. Some of the Apps crash. Saving and restoring preferences and configuration information to a file or the cloud is not always possible. Finally, the configuration and operation of the App expects that the user has a better understanding of KNX and the home network than is normally the case.

This prototype App shows that with careful planning and detailed exporting of the network configuration, the user can check, monitor and control the KNX network for simpler devices. The user needs only a basic understanding of KNX. Operational problems due to lack of status from a group address (write only), confusing parallel operation, and actions resulting in no response can be filtered out using a favorites list.

References

[1] R. Isaacs, “fhtwKNX - Android App for a KNX Smart Home”, Master’s thesis, AAT-Lab@Department of Embedded Systems, Department of Telecommunications and Internet Technologies, FH Technikum Wien, Mar 2012.

[2] Siemens, “Standard GTK Installation”.

[3] Siemens Flyer, “GAMMA Training Kit”.

[4] KNX Association, *KNX Handbook: XML Data Encoding*, volume Volume 3: System specifications, part 8: KNXnet/IP, KNX Association, 2007.

[5] T. Wimmer, (August 2011), XLST transformation parser for KNX project [Online]. Available: http://sourceforge.net/projects/calimero/files/ETS4

[6] Boris Malinowsky, Georg Neugschwandtner, Wolfgang Kastner, “Calimero: Next Generation”, 2008.

[7] Boris Malinowsky, Georg Neugschwandtner, Wolfgang Kastner, “Calimero: Next Generation Design Feature Highlights”, *KNX Scientific Conference*, 2007.

[8] Reto Meier, Michael Mahemoff, “Google I/O 2001:HTML5 versus Android: Apps or Web for Mobile Development?”.

[9] M. Gargenta, *Learning Android*, O’Reilly, USA, 2011.

[10] ACRA open source, “Application Crash Report for Android”.

[11] Björn Bittinse, Jürgen Sieck, Michael Herzog, “Communication Systems for Building Automation and Control”, *UKSim Fourth European Modelling Symposium on Computer Modelling and Simulation*, 2010.