The Application Architecture of Smartwatch Apps – Analysis, Principles of Design and Organization

Rainer Lutze¹, Klemens Waldhör²

Abstract: We present an innovative approach for the systematic construction of smartwatch apps starting from an analysis of the requested functionality. Based on the resulting functional requirements, we recommend to derive a suitable smartwatch app architecture based on the following five principles: 1) the need for dialogue management, 2) the need for a distributed architecture, 3) the advantages of a layered app architecture, 4) software engineering implications for the maintainability of the app with economic costs, and 5) the technical boundary conditions on the causal network of the app. The approach taken results from - and is backed by - our two year experiences on building smartwatch apps in the healthcare area.

Keywords: software architecture for smartwatch apps, principles of software construction for smartwatch apps, software engineering of smartwatch apps

1 Known Architectural Approaches

Apple watch apps in principal follow the IOS™ MVC architectural pattern, where the “view” part is dislocated on the watch and the “model” and “controller” portion of the app are running on a coupled iPhone™. The MVC (model view controller) is an approved app design pattern for separating the graphical user interface (GUI) of the app, view, from the domain and task specific logic of data manipulation, controller, and the persistent storage of data, the model. For alleviating the GUI implementation for their distributed apps for Apple watch, watchOS™ features three design patterns for presenting information on the Apple Watch screen ([Ap16]): (readonly) glances, actionable notifications and complications. For now, Apple does not support smartwatch apps running independently on the smartwatch.

In contrast, Samsung’s Tizen OST™ and Gear™ smartwatches feature standalone apps. For Apple’s watchOS™ the communication between smartwatch and smartphone works fully automatically (via Bluetooth). For Tizen, the Samsung Accessory Pack (SAP) (cf. [Sa16]) supports a software distribution between smartphone and smartwatch and the corresponding communicating between both devices (via Bluetooth), but it has to be explicitly programmed.

¹ Dr.-Ing. Rainer Lutze Consulting, Wachtlerhof, Wilhermsdorfer Str. 14, D90579 Langenzenn, Germany rainer.lutze@lustcon.eu
² FOM University of Applied Sciences, D45127 Essen / Nuremberg, Germany klemens.waldhoer@fom.de
Android Wear™ OS (cf. [An16a]) supports similar functionality. Programmable notifications based on the Bluetooth protocol are used to exchange data between the smartwatch and the mobile phone. This includes the option to distinguish between actions which are identical both on the mobile phone and the smartwatch and actions where the functionality may differ. In addition, Android Wear supports starting actions of other smartwatch and mobile phone apps using Intents. From a GUI point Android distinguishes three types, so called cards: 1) Cards for displaying information from notifications, typically sent from the mobile phone, 2) single action controls (play, pause) and 3) expandable stacks of cards, combing the information of several notifications. Cards themselves may be composed of several pages. The design of the cards and the app itself should follow the Android Wear design principles (cf. [An16b]).

For handling background services Android uses the Java thread concept. This in turn is mostly realized through some listener functionality. Thus Android provides the developer for the construction of an app basically with the same functionality – with some minor limitations – as for an app on a mobile phone. This is one of the advantages of Android compared to other approaches, in addition its well proven and stable wearable OS.

2 Engineering of Smartwatch Apps

Following a well-established workflow in the engineering process (cf. [FG13]), in the first step the function of a smartwatch app has to be determined. From this, secondly, the architectural design, form of the product, can be derived. Finally, the relevant technical standards and regulations for optimizing the app will be considered and applied.

![Product Development Process (PDP)](image_url)

Fig. 1 Product Development Process (PDP) proposed by Pahl/Beitz
2.1 Functional Aspects of Smartwatch Apps

The typical smartwatch app will include one – or a combination – of the following four basic functional aspects:

- **Presenting** – the function of the app is the presentation of foreign content (not created on the smartwatch) on the body-worn device. The presentation may be controlled by manual input, typically affirmations or selections, done by the wearer on the smartwatch.

- **Controlling** – the function of the app is to manipulate resp. command foreign activities via control values originating from the smartwatch software itself or issued by the wearer of the watch.

- **Monitoring** – the function of the app is to detect activity patterns or recognize information about the health conditions of the smartwatch wearer by fusioning the measured sensor data of the smartwatch. Monitoring does include a local display of such derived information on the watch and may include remote alerting in cases of detected emergency situations by deviant activity patterns.

- **Communicating** – the function of the app is to provide a real-time (chat, speech) communication between the smartwatch wearer and a distant person (or machine).

Many apps include a combination of those basic functional aspects. A popular combination is the “controlling-presenting” bundle, e.g. in a »news« app, where the smartwatch user selects his news categories of interest first, and then repeatedly gets latest news for the selected categories, which need to be presented in very compressed style. The “monitoring-presenting” bundle is present in »fitness tracker« apps, where the actual heart rate values, done steps and actual geographic position of the smartwatch wearer will be presented together in a map delivered from a foreign, central map service, e.g. Google, Here or Microsoft. Our assistance app for elderly people (cf. [LW16]) features a bundle of the “monitoring-communicating-controlling” aspects. A basic monitoring of the smartwatch wearer takes place and in case of detected emergency situations, health hazards, a speech connection to home emergency call center (HECC) will be established. Additionally, the assistance app controls a home automation (HA) system, in that the departure and arrival of the wearer from resp. at his/her home will be reported to the HA system for switching off/on critical electric appliances during his/her absence and effecting the lighting of the home.

2.2 Architectural Requirements from App Functionality - The Need for Dialogue Management

Even apps with pure presenting functionality will already run into a problem, if more than one piece of information shall be displayed simultaneously on the smartwatch screen. The same is true for monitoring apps, if multiple and different information derived from the smartwatch sensor analysis shall be presented to the smartwatch user at
the same time. A **dialogue management** will be necessary, which prioritizes the information to be displayed. The dialogue manager then exclusively assigns the smartwatch display and smartwatch input/output devices ( touchscreen, mic/speaker) for a defined time slice to that smartwatch app thread with highest actual priority. For example, in [LW15] a blackboard is used as a central synchronization mechanism between all different threads which desire to gain the user’s attention. A central scheduling algorithm determines the blackboard element with highest priority and passes a temporal exclusive privilege to this element for utilizing the smartwatch screen and I/O devices.

If the smartwatch app includes multiple aspects of presenting, monitoring and communicating, the dialogue manager will be inevitable for coordinating the smartwatch display as a scarce resource. Typically, outgoing emergency calls placed by the smartwatch user or incoming calls from other parties will have the highest priority for display and I/O utilization.

### 2.3 Architectural Requirements from App Functionality – The Need for a Distributed Software

For the current Apple Watch app architecture with its distributed MVC pattern, a distributed app implementation approach is mandatory. If for control aspects apps, monitoring or communication aspect apps complex configuration information will be necessary for the function of the app, for the input of the configuration information the smartwatch display is typically not well suited. A companion device (e.g. a coupled smartphone) will be needed, on which the configuration details can be visually presented in sufficient space and descriptiveness. Also, for the machine learning and data mining process of the sensor data captured and condensed on the smartwatch, a more powerful additional device will be needed to perform this analysis. For our assistance app for the elderly, data will be passed from the smartwatch to a powerful workstation or a cloud based service (e.g. RapidMiner), in order to perform these learning, data mining computations (cf. [Lu15], [WB15]). The model learned from the sensor data then will be retransmitted to the smartwatch app and applied there.

In addition, a highly available server may act on behalf of the smartwatch and continue and complete its operations, if, for example, the wearable accidentally should run out of power and cannot act by itself anymore. We utilize such a “long-term analysis server” (LTAS) in our assistance app for durably storing and processing data delivered from the smartwatch, especially for state-of-affairs covering a period of more than a day. With respect to the limited computational power, it might be also a reasonable decision to dislocate as much demanding computations as possible from the smartwatch to a more powerful companion device like a smartphone, server or cloud service. For example, also the needed dialogue manager may run on an accompanying smartphone, if the smartwatch is reliable coupled to the smartphone via Bluetooth in a sustainable way.
In conclusion, a software distribution of parts of the smartwatch app software to a companion device with larger screen, more powerful CPU and storage…. will be favorable for all but very simple mono-functional apps.

2.4 Architectural Requirements from App Functionality – The Advantages of a Layered App Software

We have already described the necessity of having an explicit dialogue management module for presentation control as part of multifunctional smartwatch apps. In addition to this GUI module a further modularization especially of monitoring apps is meaningful.

The monitoring task typically consist of:

1) the recognition of activity patterns and/or health conditions by sensor fusion (accelerometer, gyroscopes, magnetometer, barometer, GPS, heart rate monitor, …)

2) observing a specific sequencing or clustering of those patterns / conditions over time.

For subtask 1), sub symbolic reasoning techniques like neuronal networks or logistic regression have proven their validity (cf. [Lu15]). For subtask 2), the course of operations of the smartwatch wearer’s behavior over time can best be modeled and followed via finite automata (we have been using UML state machine, see [LW15]) or some variant of business process modeling (BPM). With respect to the strongly different implementation techniques for subtask 1) and 2), and for minimizing dependencies and interfaces, a layered architecture for subtasks 1) and 2) is to be recommended, where the results of subtask 1), the events and activities of daily living (cf. [LW16]), will be used as input, atomic elements, for processing the subtask 2). The resulting “uses” hierarchy (cf. [Pa72]) of smartwatch app software layers is depicted in fig. 2.

Another advantage of a layered, modular architecture will be the possibility to execute certain layers (in our case only the bottom layer) always in the background in a guaranteed way, whereas other layers will be executed only on demand, when the smartwatch wearer interacts with his/her watch. Otherwise the execution of those layers usually will be hibernated or may be even ceased (see section 2.4) for energy saving reasons by many smartwatch operating systems.
Fig. 2: Layered Smartwatch App Software Architecture for Monitoring Functional Aspects
2.5 Software Engineering Implication for the App Architecture

Programmable smartwatches worn on the wrist are a relatively new gadget, for which use cases and application scenarios are still under development. This is especially true for business applications of smartwatches in areas as telemedicine or healthcare. As a consequence, the application logic implemented in today’s smartwatch apps primarily is best practice, empirical knowledge, which grows and changes from day to day. The situation is complicated by the fact that also the smartwatch hardware typically is of the first or at most second generation and substantial changes, hopefully improvements of the hardware and/or system software have to be considered. It is therefore a real challenge, how this smartwatch application software can be maintained with economic effort. Another aspect concerns the decomposition of the empirical knowledge in independent chunks which can be considered and supported separately and in isolation from each other.

Based on the long-term experiences in other software areas, declarative model and description techniques are well suited for addressing this maintenance problem. For example, in our assistance app for the elderly, each individual health hazard potentially affecting an elderly person will be described by a separate UML state machine (USM). The multitude of all finite state machines will be executed simultaneously on the smartwatch and synchronized by the dialogue management in form of a central blackboard (cf. [LW15]). Fig. 5 depicts the finite state machines for monitoring the absence from home with respect to the hazards of running away from an (agreed) vicinity of the home or staying away from home for an (agreed) excess period of time. In our application domain, the discussion of the app’s functionality with the care-giving domain experts takes place using the graphic state machines diagrams. From these depictions the state machine transition tables will be compiled automatically and transferred into the smartwatch. The simultaneous execution of the state machines will be done by a (static) interpreter for these (frequently changing) state transition tables.

2.6 Technical Boundary Conditions for the Causal Network of the App

A decisive factor for successful smartwatch programming is energy conservation of the limited capacity of the smartwatch battery. In business applications, the typical requirement is that the smartwatch battery has to last at least a full, extreme working day (let us say: 18 hours) and may have to be recharged each night. The implication from such a harsh requirement is that energy consuming sensors, communication acts and the display illumination of the smartwatch have been activated and used only as little as possible and in those limitations specified by the concise functionality the app solution has to achieve.

It is common practice for most smartwatches to turn on the display and activate the input devices only when the user turns up and rotates his wrist so that he/she can actually view the smartwatch display. Awaking the smartwatch app from a hibernation mode and
executing the smartwatch app code only in those “viewed” situations is also a standard procedure for many app operation systems. This may be well sufficient for many presentation and control functional aspects.

But, for monitoring and communication aspects an always ongoing, **sustainable background operation** of the smartwatch apps is essential and indispensable. Up to now, this is typically only guaranteed for incoming calls, incoming notifications with “breaking news” urgency and the synthetic “step counter” most wearables will provide as a synthetic sensor. Although, the demand for such reliable and steady background operations for parts of the smartwatch apps meanwhile has been acknowledged by many operation system designers. For example, the Tizen OS™ (rel. 2.3 and up) offers the concept of a “service application”, which will be immediately started at boot time of the smartwatch and will run continuously in the background.

### 3 Experiences with Applying the Principles

We have built our Smartwatch assistance app for elderly persons, targeting at a safe and self-determining living at home until the very high age (cf. [LW16], [LW15] for a detailed description) for the Samsung Gear™ S and S2 smartwatches following the engineering principles described above. The app focuses on the monitoring functional aspect, but also includes control and communication aspects (see fig. 3). The task to be solved by the app requires a standalone app without any dependency on a smartphone carried along (which might have been forgotten by the members of the intended user group).

Fig. 3: The implemented assistance app for Samsung Gear™ S smartwatch

### 3.1 The App Architecture in Detail

**Distributed Architecture.** Due to the fact that the Gear S features a large 2 inch, 360x480 pixel AMOLED display it was possible to integrate also the **configuration**
editor for the individual data of the wearer and his/her family data into the smartwatch app and to renounce a distributed architecture. For the follow-up model S2 with its only 1.2 inch 360x360 pixel circular display, an app external configuration editor is mandatory, realized either by a companion app running on a coupled Android smartphone or a web browser based solution. This is currently being implemented.

Layered Architecture. We implemented the proposed 3-layer structure:

- The **bottom layer** contains in fact 2 sub-layers with a neuronal net (NN) structure for ADL, EDL recognition by sensor fusion (see fig. 4). The NN output is used by the upper sub-layer for:
  
  - aggregating ADLs which are defined by a starting and terminating EDL as endpoints (cf. /LW16/ for details),
  
  - calculating the **wellbeing function** for the smartwatch wearer, consisting of the three wellbeing aspects of inactivity between recognized ADLs, the excess duration of ADLs and the agility of the smartwatch wearer (cf. /LW16/ for a definition of this wellbeing function),
  
  - updating the **nominal values** (by time series analysis) which are necessary parameters of the wellbeing calculation (cf. /LW16/ for details),

Additionally, the communication module for handling 3G speech connections with the HECC and http communication with the LTAS (for reporting occurred ADLs) is located on this layer.

![Fig. 4 The implemented 3-layer app architecture for the assistance app](image-url)
• The **medium layer** contains a) the interpreter for the USMs and b) the state transition tables and current state of the corresponding USMs. Each USM models and describes the handling of a specific health hazards over the course of time (e.g. fig. 5). With each state of an USM typically an output to be displayed on the smartwatch screen is associated [as “do” action as long as the current persists], and 3G, http communication acts with the HECC and/or LTAS to be performed [as “entry”, “exit” actions of the state transitions]. USM states also have a priority attached (cf. fig 5 and /LW15/).

• The **upper layer** contains the dialogue manager implemented as a priority structured blackboard (BB) ([En89]). Each USM pushes its current state on the specific BB level associated with the state’s priority. The BB scheduling algorithm then selects a suitable USM state from those with highest priority on the BB, for which the associated output will actually show up on the smartwatch screen in a specific situation (cf. /LW15 for details).

### 3.2 Software Engineering Aspects

The implemented layered architecture proved its advantages especially in effectively organizing the implementation work. Following the layer and module structure, the work was assigned to and distributed between different persons / teams with clearly defined interfaces and team responsibilities.

The declarative approach of describing the course of operations for health hazard handling by USMs demonstrated its value in that the graphical depictions of the state machines ("state machine diagrams") could be utilized as a common basis for discussion with the healthcare domain experts about the required behavior of the app.

### 3.3 Technical Limitations and Aspects

Several limitations in the implementation of the required app features have been evident. The most urgently needed stable background operations are not available at all on the Gear S model, and are only available for native (i.e. “C” programmed) apps on the S2, not the wearable (i.e. Java Script programmed) app variant we have been utilizing so far. This is a severe drawback, in that the app temporarily ceases it operation due to hibernation in a way which cannot be influenced by the app. Only a manual input operation of the smartwatch wearer will reliably terminate a hibernation phase and will bring back the app into operation. But again, the layered architecture delivers a blueprint to convert exactly the bottom layer of the app into a TIZEN native service application, in order to get the EDL, ADL recognition dependably and uninterruptedly working on the S2 smartwatch.

The energy consumption control of special smartwatch sensors needs also to be improved. Especially for the high power consuming GPS sensor, this sensor can only be
influenced indirectly on the Gear devices by not using it, not by directly switching it off/on, which would be strongly preferred due to a much better efficiency.

As far as technical standards and regulations are concerned, for future speech communication a SIP protocol stack (and support for related IP protocols like RTSP) would be desirable besides HTTP, in order to realize standard VOIP communication and to support – in our domain of interest - the SCAIP protocol for social care alerts via IP (cf. [SI14]).

4 Conclusions / Future Work

The proposed engineering principles have proven their validity for our application case. But, in general, the systematic construction and the software engineering of smartwatch apps still is in \textit{status nascendi}. The current focus is on the principal realization of required functionalities and economic maintenance costs. For the future, as soon as these issues will have been sufficiently solved, the focus will shift and include the formal validation and verification of the smartwatch apps.

With their plenitude of current and future sensors (e.g. for blood pressure measurement, c.f. [LW16]), smartwatch apps will be suited for medical diagnosis and thus may turn into a regulated medicine product in the long-term. Even if they will run on a repeatedly, yearly updated smartwatch consumer hardware will have to cope also with non-reliable/imprecise casual sensor readings. This will put much more emphasis on the structure, principles and methods by which the app has been built, in order to verify its function, as requested for this product category.

Also privacy concerns and the potential use of smartwatch data in the scope of benefit plans for insurances issues in the long-term do require well established engineering principles and standards for constructing smartwatch apps. Only in this way, a auditable transparency of the smartwatch app software will be achieved, which allows to verify that such most sensitive personal data will be handled in a ethic and legally adequate manner.

5 References


Fig. 5: UML finite state machine for handling some health hazards during an absence from home.