was used to convert JSON representation into Java Objects using string mapping. When a JSON configuration file is loaded, the type field representing the module is checked and, if it matches to a map key, this module is instantiated by using the class definition that is coupled to the key. Reader for user annotations: To enable smartphone users annotating sensor data, CRNTC+ integrates an ACTLog component, which works as a reader for user input. ACTLog provides a configurable user interface (UI) within CRNTC+ to capture annotations in pre-configured categories. To annotate data, the phone user needs to tap and hold a category label and then select a sub-category label instance from a configured list displayed. Annotations can be directly processed in CRNTC+ or stored to a labels file for subsequent analysis. ACTLog resides in the smartphone-specific layer. Between-layer communication: Besides direct within layer communication, DirectInput and DirectOutput components are used as internal gateways to transfer data between framework layers. This design is needed to bridge between the different implementations of both layers: while the smartphone-specific layer uses native code of the Android platform, the generic processing is integrated as a library in the CRNTC+ application. A direct data communication between the layers is essential to minimise overhead and processing load compared to other communication forms between layers, such as RPC or TCP/UDP.

2.5 Framework characterization

To evaluate the CRNTC+ framework performance, we assessed extensibility, scalability, and energy consumption. Extensibility: We evaluated the ease of adding a new component and measured the steps necessary to create new Readers and Writers. Table I summarizes the extensibility evaluation results. Four steps were needed to add a new sensor Reader component and Writer, with 18 code lines and 22 code lines, respectively. Adding a new UI element requires five steps and 34 code lines. While the actual complexity of adding components depends on the required functionality, the evaluation indicates the basic framework-specific requirements for an extension. It can be observed that UI elements require the largest effort, since an icon is needed and the Android framework requires to handle life cycles of “Activities”. Overall, the result indicates that the framework does not imply complex steps for functionality extension. Scalability: We evaluated scalability by incrementally adding, recording, and visualizing calibrated accelerometer data from Shimmer sensors. To assess performance we measured CPU usage and measurement jitter. Up to three sensors could be simultaneously recorded without losing samples at a sampling rate of 200 Hz. When using four sensors, responsiveness of the UI reduced and CPU time for updating the UI decreased. This result suggests that three sensors could be safely recorded without losing samples. Energy consumption: For the Epilepsy case study described in paragraph 2.5, two applications have been created for gathering sensors data and for seizure event detection. During the execution of both applications, energy consumption of the smartphone was monitored. With the full sensor configuration, battery level discharged by 80% during 6 hours of sensor recording. This result can be explained by the continuous data writing onto the SD card, decoding of data sent via Bluetooth, and continuous screen use for annotating data. It can be expected that reducing sensors will reduce energy needs. Similarly, online processing without storing to the SD card could increase battery life.