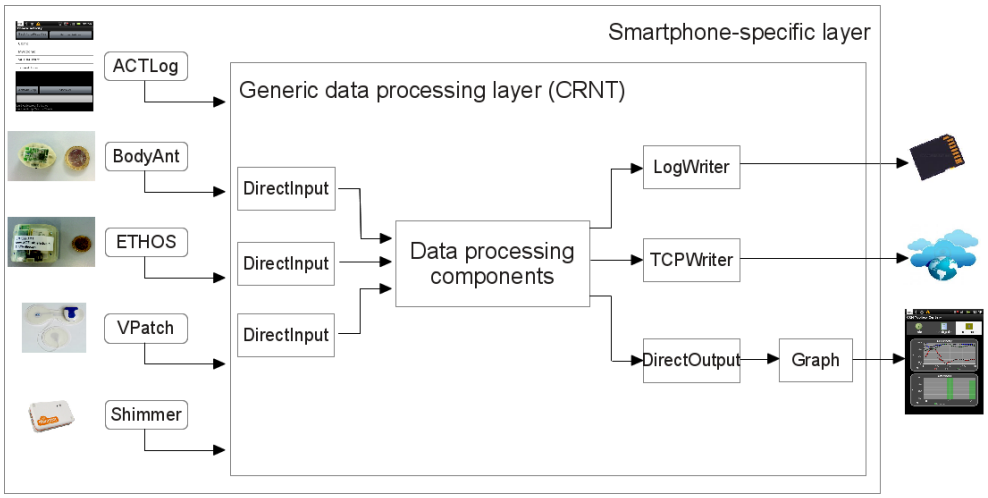

Figure 3 Functional overview of the CRNTC+ framework. Reader components are used to capture sensor data and user input, while Writer components serve to output information, e.g. through a Graph component or via a WLAN link. Our approach considers smartphone-specific (platform dependent) and generic data processing layers.

2.4 Implementation

In our framework design and implementation we targeted extensibility, and scalability through convenient interfaces to add and customise components. Moreover, design efficiency is fundamental to minimise energy consumption. Here, we detail the implementation of key component classes: Readers and Writers. Moreover, general implementation considerations for CRNTC+ on the Android platform are described. Readers: To interface with sensors and devices via different communication standards Readers are used. Through Readers, various smartphone-integrated sensors can be recorded. Examples for external device interfaces include BluetoothReader and ANTReader components. Readers connect to devices specified in the component configuration. Depending on the sensor device protocol, data streaming is subsequently started and readings are decoded for further processing in the framework. E.g., ANTReader uses the ANT+ protocol to connect to sports or custom devices, such as BodyANT, ETHOS, and Vpatch. The BluetoothReader can be used, e.g., to interface to a heart rate belt or to Bluetooth accelerometers. Due to the phone APIs, Readers reside in the smartphone-specific layer of CRNTC+. Writers: Writers encode data streams from CRNTC+ for further analysis and feedback. E.g., the Graph component provides a time series view on the phone's screen for reviewing sensor or feature data. Writers reside in smartphone-specific and generic data processing layers. E.g., data file logging to an SD card can be performed through the LogWriter component using generic POSIX calls, whereas the Graph component requires platform-dependent features. Component instantiation: In CRNTC+ a JSON-based configuration is used to describe component instances, their parameters, and communication links. A JSON configuration is instantiated at runtime by matching a class definition of the component. The GSON[4] library

[4] https://github.com/google/gson