technologies

MDPI

Article

Model-Based Design of Energy-Efficient Human Activity Recognition Systems with Wearable Sensors

Florian Grützmacher [1,*], Albert Hein [2], Thomas Kirste [2] and Christian Haubelt [1]

[1] Institute of Applied Microelectronics and Computer Engineering, University of Rostock, 18051 Rostock, Germany; christian.haubelt@uni-rostock.de
[2] Institute of Computer Science, University of Rostock, 18051 Rostock, Germany; albert.hein@uni-rostock.de (A.H.); thomas.kirste@uni-rostock.de (T.K.)
* Correspondence: florian.gruetzmacher@uni-rostock.de; Tel.: +49-381-498-7289

Received: 31 August 2018; Accepted: 26 September 2018; Published: 29 September 2018

Abstract: The advances in MEMS technology development allow for small and thus unobtrusive designs of wearable sensor platforms for human activity recognition. Multiple such sensors attached to the human body for gathering, processing, and transmitting sensor data connected to platforms for classification form a heterogeneous distributed cyber-physical system (CPS). Several processing steps are necessary to perform human activity recognition, which have to be mapped to the distributed computing platform. However, the software mapping is decisive for the CPS's processing load and communication effort. Thus, the mapping influences the energy consumption of the CPS, and its energy-efficient design is crucial to prolong battery lifetimes and allow long-term usage of the system. As a consequence, there is a demand for system-level energy estimation methods in order to substantiate design decisions even in early design stages. In this article, we propose to combine well-known dataflow-based modeling and analysis techniques with energy models of wearable sensor devices, in order to estimate energy consumption of wireless sensor nodes for online activity recognition at design time. Our experiments show that a reasonable system-level average accuracy above 97% can be achieved by our proposed approach.

Keywords: human activity recognition; model-based design; energy efficiency; wearable sensors; dataflow graphs; software mapping

1. Introduction

Cyber-physical systems (CPS) couple embedded processing tightly with its environment by incorporating I/O, sensors, and actuators to support interaction between smart objects and users. They enable context-aware services and applications that have become a desired feature in smart homes, smart industries, and advanced e-health solutions.

One of the research topics that has recently gained increasing attention is the automatic detection of a user's currently performed activities, called *online activity recognition*. While machine learning has become a dominant approach, the classification of activities with statistical, signal-based, and frequency-domain-specific features, calculated from body-worn sensor signals, still allows for desirable fine-tuning towards resource-efficient configurations by domain knowledge. To enable unobtrusive systems, the use of wireless, body-worn sensors is unavoidable. However, like all other classes of CPS, their energy-efficient design is still of major concern, as it directly influences the battery lifetime and thus the usability of the system. General statements about the energy efficiency of such systems are difficult to make, as sensor modalities, feature sets, and classifiers vary among different classification goals. In order to address these difficulties, methods to estimate the energy consumption of a particular activity recognition configuration are necessary at design time.

In this article, a method to estimate the energy consumption of wireless sensor nodes for application-specific configurations is introduced. The estimation is based on dataflow models and analysis techniques, which have already become a de facto standard for signal processing, streaming, and multimedia applications. Based on these methods, we propose a novel approach for the estimation of energy consumption of wireless sensor nodes within the context of online activity recognition. In our experiments, we verified that our methods offer enough accuracy to substantiate early design decisions at the system level by comparing their results to energy consumption measurements of various widely used activity recognition configurations. The contribution of this article is thus twofold: (1) a formal modeling approach for activity recognition systems is introduced and (2) dataflow-based analysis techniques are extended to support energy estimation in early design stages.

The remainder of the article is structured as follows. In Section 2, related work is discussed, which is followed by an introduction to the target application domain in Section 3. The formal modeling techniques on which the presented concept is based is introduced in Section 4. This is followed by Section 5, which describes how activity recognition software can be modeled at different levels of abstraction by the proposed formalisms. The extension of the model-based analysis techniques to enable energy estimations at design time are explained in Section 6. The experiments to evaluate the proposed concept are presented in Section 7.

2. Related Work

Many works exist that have studied the model-based design of streaming applications with dataflow graphs. Important properties of the modeled application that can be analyzed from its graph representations involve throughput [1], throughput-buffering trade-offs [2], average and worst-case performance [3,4], latency and response time [5,6], and many others. In [7], dataflow graphs were used for modeling and analyzing the performance of online gesture recognition.

Related to energy efficiency, there exist works that reduced the energy consumption of processor cores through dynamic voltage and frequency scaling (DVFS) while guaranteeing certain throughput constraints by using dataflow-based analysis techniques. Furthermore, in [8], energy-aware task-mapping and scheduling optimization techniques for heterogeneous multi-processor system on chips (MPSoC) was studied. Instead of multi-objective optimization, our article is concerned with the modeling of activity recognition systems and their mapping to the distributed hardware platform of wireless sensor nodes and their data-aggregating hubs, such as smartphones. Furthermore, an energy consumption estimation approach for application-specific configurations of the activity recognition systems is shown.

Several works in the literature have studied the energy-efficient design of online activity recognition with wearable devices. Some of the approaches include disabling hardware sensors using prediction of activities [9,10], feature selection strategies to reduce the computational load [11], classifier selection [12], or fixed-point arithmetic [13]. Other approaches, orthogonal to those aforementioned, study the on-board calculation of the feature extraction stage of activity recognition systems. In [14–19], it was shown that calculating features on the wireless sensor nodes could reduce the energy consumption of their wireless transceivers or flash memories due to the reduced amounts of data.

While [20] already indicated a need for a trade-off between communication reduction and added computational effort, we have shown in [21] how a trade-off for an application-specific configuration can be estimated at design time. This was done by using energy models of the hardware components.

In this article, we combine the recent findings of energy consumption estimation and state-of-the-art formal model-based design and analysis methods. Our contributions are a model-based design methodology for energy-efficient wireless sensor nodes in the context of online activity recognition and an approach for estimating energy consumption from dataflow models in early design stages.

3. Application Domain

In many approaches to online activity recognition, multiple sensors are attached to the human body which collect multidimensional signals from different sensor modalities, such as accelerometers, gyroscopes, magnetometers, and many others. From these signals, activity recognition software attempts to detect the currently performed activity of the human.

Since the first attempts towards recognizing activities with wearable sensors, a common processing chain very similar to a general pattern recognition pipeline has been established. This workflow, referred to as the activity recognition chain (ARC) [22], is shown in Figure 1 and contains five processing stages: data acquisition, preprocessing, segmentation, feature extraction, and, finally, classification.

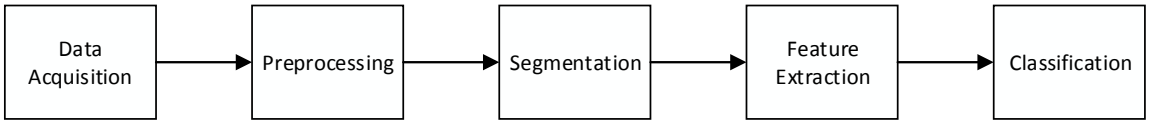


Figure 1. Activity recognition chain (ARC), according to [22].

While there are some approaches using adaptive segmenting of the sensor data stream, sliding window methods, most commonly with partial overlapping, is still the prevalent method of the segmentation stage [23–27]. Based on these segments, a set of descriptive features for the anticipated activities is calculated at the feature extraction stage. Time-domain features are used very often, as they require low processing and usually show a high performance in terms of latency and throughput. The most common are mean, variance, skewness, kurtosis, zero/threshold crossing rate, or frequency-domain features like energy, frequency bands, etc. [28–30]. Since this stage often drastically reduces the data rate, as a relatively small number of features is calculated from a large number of samples of a window, it was subject to a lot of research to perform this stage as near to the sensor as possible, i.e., on board of a wireless sensor node [14–16,18,19], or even on-sensor [17,21]. However, each application-specific setup varies in sensor sampling frequency, sliding window size, sliding window overlap, and number of features and their computational effort in calculating them. Thus, the amount of reduced communications and added computational effort changes for each setup and has to be traded off for each particular configuration [19,21].

In order to substantiate early design decisions, methods are necessary to estimate the resulting energy consumption at design time at high abstraction levels. Since activity recognition systems include many dataflow-oriented algorithms with little control flow, we propose to use design techniques based on dataflow graphs. For this purpose, we combined dataflow graph analysis methods with energy models [21] for energy estimation of wireless sensor nodes.

By considering modern sensors which already include microcontrollers performing sensor preprocessing like low- or high-pass filters, sensor fusion, or sensor correction algorithms, the preprocessing stage of most ARCs is usually already performed on the sensor itself. Moreover, for multi-sensor setups, the classification stage is usually performed on data-aggregating hubs, since the information of multiple sensors has to be processed together to perform activity classification. Although parts of the classification algorithms could also be considered for shifting their processing near to the data source, i.e., wireless sensor nodes, or the sensor device itself, we present our model-based design approach, with a focus on the segmentation and feature extraction stage of typical ARCs.

4. Synchronous Dataflow Graphs

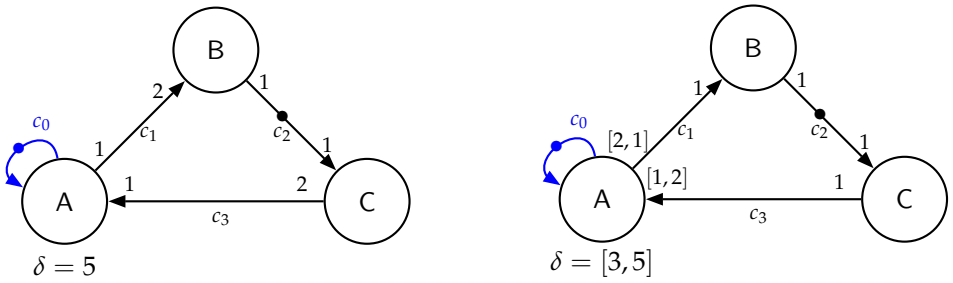
The concept of the presented modeling approach is based on *synchronous dataflow* (*SDF*) graphs, as described by [31].

Definition 1. *SDF Graph*

An SDF graph $G = (V, E, cons, prod, D, \delta)$ *consists of a set of vertices* V*, a set of edges* $E \subseteq V \times V$*, token consumption rates* $cons : E \to \mathbb{N}$*, token production rates* $prod : E \to \mathbb{N}$*, and an initial token distribution* $D : E \to \mathbb{N}_0$*.*

Furthermore, in line with [1,2], we use timed SDF graphs, where an execution time δ, also referred to as *delay*, is associated with each actor $\delta : V \to \mathbb{N}_0$.

An example SDF graph can be seen in Figure 2a. The vertices are called *actors* (named A, B, and C in Figure 2a) and represent function computations, which communicate data *tokens* over unbounded *channels*, with FIFO semantics represented by edges (c_0, c_1, c_2, and c_2 in Figure 2a). Hence, the presence and number of data packets are represented by tokens. With each channel, a number of *initial tokens* $D(e)$ is associated, marked with bullets in Figure 2a on channel c_0 and c_2. The consumption and production rates are annotated to the channel inputs and outputs and need to be fixed in SDF. Note that for the sake of simplicity, consumption and production rates not attached to the edges are assumed to equal 1, like channel c_0 in Figure 2a, and delays not attached are assumed to be $\delta = 0$. The currently available number of tokens on a channel is denoted by $d(e)$.



(**a**) SDF (**b**) CSDF

Figure 2. Examples of synchronous dataflow (SDF) and cyclo-static dataflow (CSDF) graphs.

Definition 2. *Actor Firing*

An actor $v \in V$ *can be* fired *if the number of available tokens* $d(e)$ *is greater or equal to the consumption rates* $cons(e)$ *of all its incoming channels* $e = (\tilde{v}, v)$*, i.e.,* $\forall e = (\tilde{v}, v) \in E : d(e) \geq cons(e)$*. If actor* v *fires, it consumes* $cons(e)$ *token from each incoming edge* $e = (\tilde{v}, v) \in E$*, and after its specified delay* $\delta(v)$ *it produces* $prod(e)$ *token on each outgoing edge* $e = (v, \tilde{v}) \in E$*.*

The execution of a consistent SDF graph forms fixed repetitive firing sequences caused by the constant production and consumption rates. Every such sequence is called an *iteration* and can be described by a *repetition vector* γ.

Definition 3. *Repetition Vector*

The repetition vector γ *of an SDF graph* G *assigns a number of firings to each actor* γ*:* $V \to \mathbb{N}$*. It is defined as the unique smallest non-trivial vector that satisfies the following balance equation:* $prod(e) \cdot \gamma(\tilde{v}) = cons(e) \cdot \gamma(v)$ *for each channel* $e = (\tilde{v}, v) \in E$ *from actor* $\tilde{v} \in V$ *to* $v \in V$*. In this sense, non-trivial means that* $\gamma(v) > 0$ *for all* $v \in V$*.*

The repetition vector describes how often each actor $v \in V$ fires during one iteration of the graph. *Iff* an SDF graph has a unique smallest non-trivial repetition vector, it is called *consistent*. This implies that the number of actor firings specified by the repetition vector has no net effect on the distribution of tokens on the channels, i.e., after a full iteration, the graph reaches a recurring state.

The execution of an SDF graph, where each actor fires as soon as its firing condition is met, is called a *self-timed execution* [2]. This furthermore allows multiple instances of an actor to fire simultaneously in parallel, allowing for maximum parallelization. This property is called *auto-concurrency*. However, to limit or prevent auto-concurrency, self-edges to each actor can be added with as much initial tokens on it, as simultaneous firings of that actor are allowed. Thus, by adding only a single initial token on a self-edge, the corresponding actor can only fire in a non-overlapping fashion, and fires as soon as the delay of its previous firing has expired, at the earliest [32]. This is shown in Figure 2a for actor A in blue.

Another important property that can be calculated from an SDF graph is its throughput: [1].

Definition 4. *Throughput*

The throughput TH is defined as the number of iterations of a time period divided by the duration of that period, i.e., the reciprocal of the average time duration of an iteration of the graph.

For a more detailed introduction to SDF graphs and their throughput analysis, we refer to [1,31].

The repetition vector of the example SDF graph in Figure 2a is $\gamma = [2, 1, 1]$ and the execution time for one iteration is 10.

Cyclo-Static Dataflow

In [33], the semantics of SDF was extended to *cyclo-static dataflow* (*CSDF*), with the concept of cyclic changing execution parameters of actors. In CSDF, following the more generalized definition from [2], the delays δ and production and consumption rates *prod* and *cons* do not necessarily need to be fixed anymore, but can change in fixed repetitive sequences, called *cycles*. Furthermore, following the definition from [2], the production rates $prod(e) : E \to \mathbb{N}^n$ of channels connected to an actor v, the consumption rates $cons(e) : E \to \mathbb{N}^n$ of channels connected to an actor v, and the delay $\delta(v) : V \to \mathbb{N}^n$ of actors v are sequences having the same length $n \in \mathbb{N}$, i.e., $\forall v \in V \wedge \forall e_o = (v, \tilde{v}) \in E \wedge \forall e_i = (\tilde{v}, v) \in E : |\delta(v)| = |prod(e_o)| = |cons(e_i)| = n$. During the ith firing, the actor v consumes $cons(e_i)[i \bmod n]$ tokens from each incoming channel e_i, produces $prod(e_o)[i \bmod n]$ tokens on each outgoing edge e_o, and has a delay of $\delta(v)[i \bmod n]$.

Note that, similar to SDF graphs, consumption and production rates not attached to the edges are assumed to equal 1 (e.g., channel c_0 in Figure 2b), delays not attached are assumed to be $\delta = 0$, and single rates and delays of CSDF actors with multiple cycles are assumed to be equal in each cycle. The cyclic changing consumption and production rates, as well as the cyclic changing delays, can be seen on channel c_1, c_3, and actor A of the example CSDF graph in Figure 2b, respectively. Its repetition vector is $\gamma = [2, 3, 3]$. The execution time for one iteration is 8.

All the properties of SDF and CSDF graphs important for our modeling approach are implemented and publicly available as part of the SDF3 tools from [34], which we used for analyzing the CSDF graphs in Section 7.

5. Modeling of Activity Recognition

In this section, we show how online activity recognition systems can be modeled with (C)SDF semantics at different levels of abstraction. For this purpose, we modeled an ARC with example sampling periods, sensor modalities, sliding window parameters, and feature sets. Examining the activity recognition chain in Figure 1, its different stages can be directly represented by SDF actors. In Figure 3, an example SDF graph of an ARC is shown. The first actor, named DA, models the data acquisition stage as an example for an accelerometer and a gyroscope sampled at 200 Hz. The sampling frequency is modeled by a delay of 5000 µs and the self-edge c_0 to prevent multiple simultaneous firings of the actor. The production rate on edge c_1 is 2, modeling a 3D sample from each accelerometer and gyroscope. In the preprocessing (PP) stage, a sensor fusion takes both accelerometer and gyroscope samples and produces a single 3D orientation vector. This is modeled by the actor named PP, consuming two tokens from channel c_1 and producing a single token on channel c_2 each

time PP fires. A sliding window of 128 samples is implemented at the segmentation stage, modeled by the actor named SW, consuming 128 tokens from channel c_2 and producing 128 tokens on channel c_3 at once. The actor named FE models the feature extraction stage. It consumes a full window of 128 tokens from channel c_3, calculates two features on its data, and produces the two features on channel c_4. Finally, the classification stage implements a classifier, which is modeled by the actor named CL, which consumes the two features from channel c_4 and performs the classification. We omitted the output of this stage for the sake of simplicity. The repetition vector of the SDF graph shown in Figure 3 is $\gamma = [128, 128, 1, 1, 1]^T$.

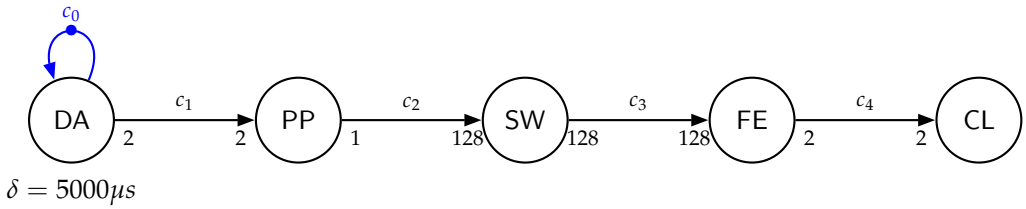
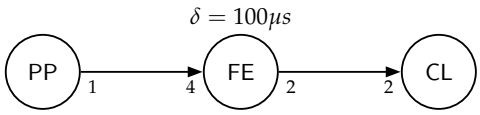


Figure 3. SDF graph of an example ARC.

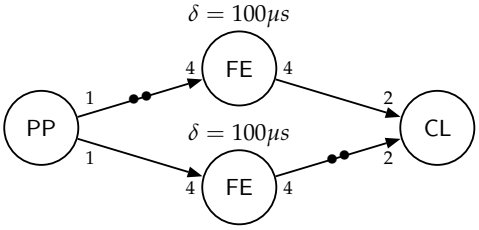
For the sake of readability, actors SW and FE can be merged to actor FE, modeling the sliding window-based feature extraction. By doing so, their delays will be added, resulting in the same behavior of the graph.

5.1. Detailed Application Modeling

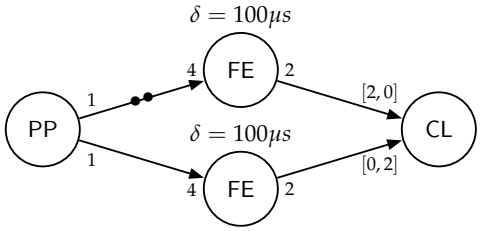
An example of the resulting sliding window-based feature extraction is shown in Figure 4a. Note that for the sake of readability the sliding window size $size_{SW}$ has been changed to 4 samples instead of 128. The actor FE is fired every four samples (after a new window is filled) and extracts two features that are produced on its outgoing edge. If a sliding window with an overlap is used, there are actually multiple sliding windows in parallel with a relative displacement defined by the overlap $O_{SW} \in [0, 1)$. To model overlapping sliding windows, as many actors as parallel processed windows ($\lceil \frac{1}{1-O_{SW}} \rceil$) are included, with an additional initial token to interleave the sliding windows with a displacement of $size_{SW} * O_{SW}$ samples. See Figure 4b for an example of the sliding window with $size_{SW} = 4$ and $O_{SW} = 0.5$. However, modeling the overlapping sliding windows in SDF semantics requires also initial tokens on the output edges of the FE actors in order to synchronize the firing of the last actor CL. This results in a token production of the actors FE and a token consumption of actor CL that is actually higher than it would be implemented in software. To overcome this, CSDF semantics can be used to cycle the input rates of CL, matching the actual token consumption rate and reducing the token production rate of the actors FE to its real amount of calculated features. The principle is shown in Figure 4c.



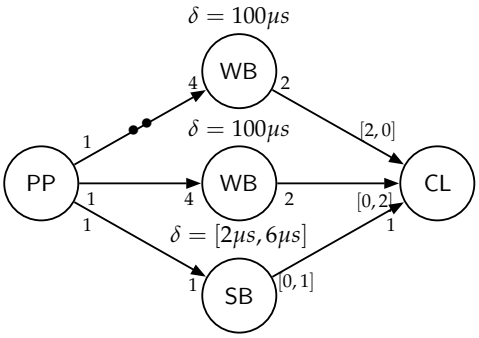
(**a**) Extraction of two features on a sliding window of four samples.



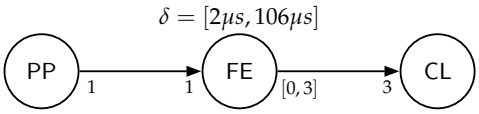
(**b**) Extraction of two features on a sliding window of four samples and 50% overlap.



(**c**) Extraction of two features on a sliding window of four samples and 50% overlap with CSDF.



(**d**) Two window-based and one sample-based implementation of feature extraction with CSDF.



(**e**) Modeling window- and sample-based feature extraction in a single CSDF actor.

Figure 4. Modeling sliding window-based feature extraction with different levels of abstraction.

Up to now, we included the modeling of feature calculations on a whole window when it is filled, e.g., for a fast Fourier transform (FFT) calculation. However, some more time-efficient feature implementations update a value for the whole window but in a sample-based fashion, and finalize the value with the sample that fills the window. Examples are one-pass computations of statistical moments of arbitrary order [35,36], such as mean, variance, skewness, kurtosis, etc., which are often-used features in activity recognition [23,24,28]. As an example, for computing the variance, with each new sample a sum is updated, and with the last sample belonging to that window the sum is updated and processed to calculate the actual variance from it for the whole window. In such sample-based feature extractions for whole windows, the actor should actually fire with each incoming sample but only produce an output, with the last sample finishing the window. This behavior can be captured with CSDF actors as well. If the updating of a sum takes 1 µs for each new sample, and the finalizing for the last sample takes 4 µs, the delay function of such an actor would be $[1\ \mu s, 1\ \mu s, 1\ \mu s, 5\ \mu s]$ and the corresponding production rate is $[0, 0, 0, 1]$ for a sliding window of $size_{SW} = 4$. Calculating two windows in parallel because of a window overlap of 50%, the actual delays and production sequences are superpositioned with a displacement of two samples, resulting in a periodical firing of two cycles with a delay of $[2\ \mu s, 6\ \mu s]$ and a production rate of $[0, 1]$. This is included in Figure 4d in actor SB together with the two window-based feature extractions WB, called FE previously.

To estimate energy consumption from the CSDF, it is necessary that the actual processing times of actors and the correct token consumption rates always match the implementation in order to extract the processor load and the transmission overhead on hardware communication channels, which we

show in Section 6. Furthermore, the unit of data, e.g., 3D floating point values, must kept constant when modeling it with tokens.

However, especially on wireless sensor nodes with integrated microcontrollers without task scheduling due to missing operating systems, feature extraction functions can be implemented in a sample-based fashion. This performs both the updating of the sample-based feature calculations with each new sample, and the buffering of samples for the last invocation for window-based processings, e.g., an FFT. These functions are invoked with each new sample and imply a static order scheduling of the different feature calculations inherently. Such implementation can be merged into a single actor summarizing the feature extraction calculations, which is shown in Figure 4e. The delay of actor FE summarizes the added processing times of all firing actors in each cycle, and their token production rate as well. This, on the one hand, simplifies the CSDF graph, but prevents the mapping of different feature calculations to different processing units and implies a certain scheduling of the actors. However, for the sake of simplicity, we continue with this level of abstraction in the following sections, without losing generality.

The full CSDF graph of the aforementioned example feature extraction, together with example delays for actor PP and actor CL, is shown in Figure 5.

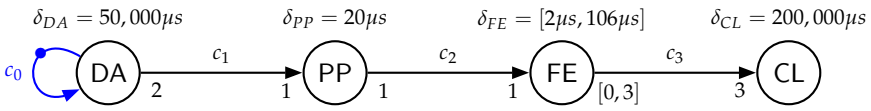


Figure 5. Example CSDF, including mapping and scheduling.

5.2. Hardware Model

The next step in designing the activity recognition system is to map the different software actors and communication channels to a hardware architecture. A *hardware model* $H = (P, T, S)$ is a set of heterogeneous processing cores P, a set of sensors S, and set of directly connected wired or wireless communication channels $T \subseteq S \times P \cup P \times P$ from the sensors $s \in S$ and to and between the processor cores $p \in P$. An example platform for online activity recognition can be seen in Figure 6, consisting of a wireless sensor board with an inertial sensor, an integrated microcontroller for sensor fusion (*FuserCore*), and a wireless link, connecting the node with a smartphone with a more powerful application processing unit (APU). The corresponding hardware model consists of a sensor named SE, the FuserCore named FU, a hardware communication link (e.g., I^2C) between them t_0, the smartphone processor named AP, and the wireless link (e.g., Bluetooth Low Energy) between them t_1. The hardware model can be seen in Figure 7.

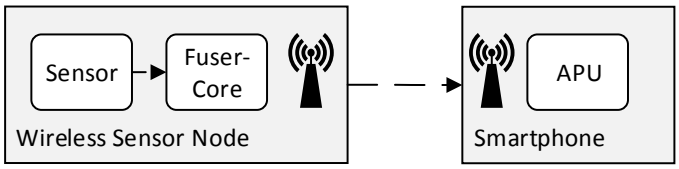


Figure 6. Hardware platform of a smartphone and a wireless sensor node.

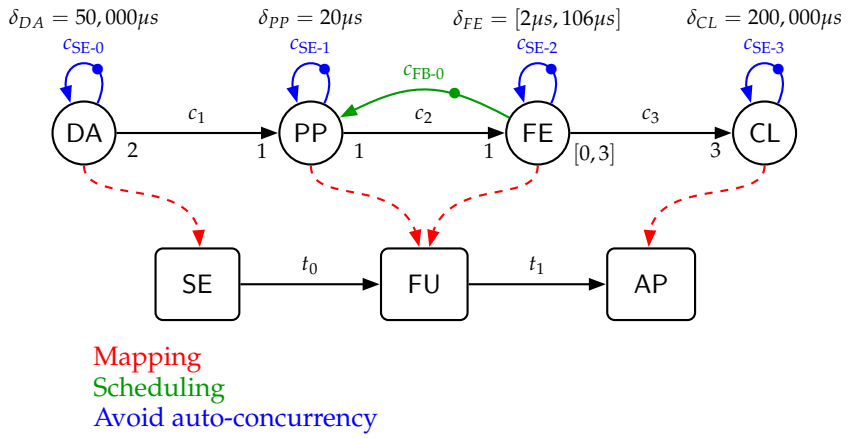


Figure 7. Example CSDF, including mapping and scheduling.

A *mapping* $M = (M_V, M_E)$ consists of a set of mapping relations M_V between all actors V of a given dataflow graph G, and the union of processor cores P and sensors S of a hardware model H, with each actor mapped exactly once, i.e., $M_V \subseteq V \times (S \cup P)$ with $\forall v \in V : |\{m_V = (\tilde{v}, p) \in M_V \mid v = \tilde{v}\}| = 1$, and a set of mapping relations M_E between all edges E of G and the hardware channels T of H, with each edge mapped exactly once, i.e., $M_E \subseteq E \times T$ with $\forall e \in E : |\{m_E = (\tilde{e}, t) \in M_E \mid e = \tilde{e}\}| = 1$. Furthermore, a mapping is only valid if the pair of actors connected by an edge $(v, \tilde{v}) \in E$ are either mapped to the same processor/sensor or to processors/sensors which are directly connected by a hardware channel $t \in T$, i.e., $\forall e = (v, \tilde{v}) \in E : \exists (v, p), (\tilde{v}, \tilde{p}) \in M_E \mid \exists (p, \tilde{p}) \in T \vee p = \tilde{p}$, and the number of actors mapped to each sensor $s \in S$ is maximal 1, i.e., $\forall s \in S : |\{m_V = (v, \tilde{s}) \in M_V \mid s = \tilde{s}\}| \leq 1$.

After mapping the dataflow graph to the hardware, scheduling has to be found to serialize the executions of all actors mapped to the same processor in a non-overlapping way. There exist several works in the literature on the scheduling of SDF, CSDF, and dataflow graphs in general [32,37–41]. For our approach, scheduling needs to be integrated into the dataflow graph itself by adding additional actors and edges, as in [32,37,38,41], to allow performance evaluation with dataflow graph analysis techniques afterwards. To prevent auto-concurrency, we first introduce self-edges ($C_{SE\text{-}0}, C_{SE\text{-}1}, C_{SE\text{-}2}, C_{SE\text{-}3}$) to each actor. Secondly, a scheduling scheme has to be selected for different actors mapped to the same processor core. For our model in Figure 7, a simple self-timed schedule is enabled by adding a feedback edge $C_{FB\text{-}0}$ from actor FE to PP. This prevents simultaneous firings of FE and PP. For detailed introductions on scheduling dataflow graphs we refer to the aforementioned literature. Note that the actors and edges added to G for scheduling purposes are not contained in the mapping M of graph G.

6. Energy Consumption Estimation

To estimate energy consumption of a CSDF graph G, a given mapping M, and integrated scheduling, we first need to assign energy models to the hardware components we are interested in. We will use the example graph in Figure 7. The most decisive hardware components whose energy consumption changes for different ARC setups are the sensor SE, the integrated FuserCore FU, and the wireless Bluetooth Low Energy (BTLE) link t_1. The energy consumption of the sensor will change among different sampling rates and sensor modalities, like the accelerometer and gyroscope [42]. The energy consumption of the integrated processing core on the wireless sensor node is dependent on the computational load introduced by the on-board feature extraction, while the energy consumption of the wireless transmissions depend on the transmission rate between the wireless sensor node and the smartphone.

For each energy dependency case of a hardware component, we need to integrate a measure in the dataflow analysis which allows us to estimate the energy consumption of the wireless node. Similar to the penalty functions in weakly consistent scenario-aware dataflow graphs in [43], we define an effort function for each component that we are interested in.

As the energy consumption of the sensor depends on its sampling frequency, the effort function $ef_{freq}(s)$ of a particular sensor $s \in S$ counts the number of firings during one iteration of the graph of that actor, which is mapped to hardware component $s \in S$, i.e., $v \in V \mid p = s, \exists(v,p) \in M_V$, divided by the average iteration duration or multiplied by the throughput of the graph, respectively. The number of firings within one iteration can be acquired by calculating the repetition vector of the graph and using the corresponding entry $\gamma(v)$. The effort function $ef_{freq}(\mathsf{SE})$ for the example graph in Figure 7 is calculated by $ef_{freq}(\mathsf{SE}) = \gamma(\mathsf{DA}) \cdot TH$.

For a particular processing core $p \in P$, the effort function $ef_{load}(p)$ captures its computational load. The load is defined as the accumulated time the core is actively processing within an observation period, divided by the duration of the period. It is intuitive to define the observation period as one graph iteration. The active processing time within one iteration can be calculated by summing up all delays of the actor firings mapped and scheduled on the processor of interest. This sum is then divided by the iteration period or multiplied by the throughput of the graph, respectively. Note that this is possible due to the integrated scheduling, as it prevent the simultaneous firings of actors mapped to the same processor. Thus, simply adding up all the delays of all occurring actor firings within one iteration on a processor core is sufficient to acquire the active time of the processor. The pseudo-code in Algorithm 1 shows how to calculate the effort function $ef_{load}(p)$ of the load of a particular processor core $p \in P$.

Algorithm 1 Calculating effort function $ef_{load}(p)$ capturing the load of a processor core. p.

```
1:  procedure CALCULATE_LOAD_EFFORT(p,G,H, M,γ(G), TH(G))
2:      Ṽ = getMappedActors(p, G, H, M)
3:      sum = 0
4:      for all ṽ_i in Ṽ do
5:          n = γ(ṽ_i)
6:          for all j in [0, 1, ..., n − 1] do
7:              sum += δ(ṽ_i)[j mod n]
8:          end
9:      end
10:     return sum ∗ TH(G)
```

The algorithm gets the target processor p as its first parameter, together with the CSDF graph G, the hardware model H, a mapping M, the repetition vector $\gamma(G)$ of G, and the throughput $TH(G)$ of G. The function $getMappedActors()$ returns the set of actors that are mapped to the processor core p. In line 4, it is iterated over this set, saving the repetition vector entry of each actor a_i to variable n in line 5. In line 7, the delays of all cycles within the n firings of actor a_i are summed up in variable sum. This is repeated for all actors a_i by the loop in line 4. At the end in line 10, sum is multiplied with the throughput $TH(G)$ of graph G and returned.

To acquire the effort function $ef_{trans}(t_1)$ accounting for the transmission rate over the wireless BTLE link, we need to identify all channels mapped to it, and count the number of token productions from all actors producing tokens on those channels during one graph iteration. This can be done by iterating through all cycles of each actor connected to those channels as source actors, and accumulating all the production rates within one graph iteration. The repetition vector entries of those actors define how often we have to iterate through the cycles. The accumulated number of produced tokens is then divided by the iteration period of the graph or multiplied by its throughput, respectively. The principle is shown in Algorithm 2.

Algorithm 2 Calculating effort function $ef_{trans}(t)$, capturing the amount of transmission on hardware channel t.

```
1:  procedure CALCULATE_TRANSMISSION_EFFORT(t,G,H, M,γ(G), TH(G))
2:      Ẽ = getMappedChannels(t, G, H, M)
3:      sum = 0
4:      for all ẽ_i in Ẽ do
5:          ṽ = getSourceActor(ẽ, G)
6:          n = γ(ṽ_i)
7:          for all j in [0, 1, ..., n − 1] do
8:              sum += prod(ẽ_i)[j mod n]
9:          end
10:     end
11:     return sum ∗ TH(G)
```

The algorithm gets the target hardware channel t as its first parameter, together with the CSDF graph G, the hardware model H, a mapping M, the repetition vector $\gamma(G)$ of G, and the throughput $TH(G)$ of G. The function $getMappedChannels()$ returns the set of channels that are mapped to the hardware channel t. Over this set it is iterated in line 4. In each iteration, the source actor $\tilde{v}$ of channel e_i is returned by function $getSourceActor()$ in line 5, and its repetition vector entry is saved to variable n in line 5. In line 7, over all cycles actor $\tilde{v}$'s n firings are iterated, its production rates to channel e_i are summed in variable sum in line 8. This is repeated for all channels in $\tilde{E}$ by the loop in line 4. At the end in line 11, sum will be multiplied with the throughput $TH(G)$ of graph G and returned.

Note that for acquiring the effort functions, only actors and channels of the original dataflow graph are taken into account, excluding additional actors and channels introduced for mapping and scheduling.

Energy Models of Hardware Components

The major components in the presented energy estimation approach are energy models of the decisive parts of the target architecture, that model the energy consumption changes depending on a certain parameter. The sensor is the first component in the processing chain, whose power consumption depends on it's sampling frequency. As an example, the ultra low-power sensor hub BHI160 [42] has low power modes, in which duty cycles between the sampling times power down the sensing hardware to improve energy consumption. The data sheet provides the power consumption of sensor combinations at different sampling frequencies. This provides enough information for an energy estimation. The effort function $ef(\mathsf{SE})$ functions as the parameter for the energy model of the target sensor configuration. The power consumption of the sensor is given as a dependency on the sampling frequency, which is in line with the defined effort function $ef(\mathsf{SE})$.

The energy consumption of the integrated FuserCore depends on its computational load. In [21], we have shown that its energy consumption is linear to the processing load introduced to the integrated FuserCore, and how a simple energy model can be built from it if not available within the data sheet. The fitted energy model is accurate enough to estimate the energy consumption of different feature extraction configurations executed on it. The energy model takes the computational load, as defined in Section 6, as a parameter and can take the effort function $ef(\mathsf{FU})$ directly as its input.

The component with the highest influence on the wireless sensor node's energy consumption is the wireless link [17]. In [21], we have shown how to build a similar energy model of a BTLE transceiver which is directly dependent on the number of BTLE packets sent in a specified time interval, i.e., with a certain frequency. In general, such a model has to be converted into the appropriate data unit when using effort function $ef(t_1)$ as its input parameter. The effort function $ef(t_1)$ counts token productions on the hardware channel t_1 per time unit, i.e., token production frequency. In our dataflow model, a token models a 3D floating point value, which in this case is in line with the energy model from [21], as one BTLE packet contains one 3D floating value as well. However, for the general case, it needs to be mentioned that a conversion into the appropriate data unit might be necessary if the data unit of tokens does not match the energy model's input parameter.

The differences of each effort function of two configurations are used to calculate the difference in energy consumption of each component of the two configurations by using their energy models. In order to estimate an absolute energy consumption of a configuration, a measurement of a reference configuration is needed. This can be any configuration which is easy to deploy without additional engineering overhead, which will be shown in Section 7.

7. Experiments

For our experiments, we implemented the setup shown in Figure 6 consisting of a Samsung Galaxy S5 smartphone and a custom wireless sensor node. The sensor node ships with a Bosch BHI160 Ultra-Low-Power Sensor Hub [42] containing an accelerometer, an gyroscope, and an integrated 32-bit floating-point microcontroller, referred to as *FuserCore*. Furthermore, a DIALOG DA14583 system-on-chip (SoC) with an integrated BTLE radio transceiver and baseband processor is used to acquire the sensor data from the BHI160 over I^2C and send it via BTLE to the smartphone. The board is powered by a CR1225 coin cell battery at 3 V. The overall power consumption of sending raw accelerometer data at 100 Hz via BTLE is $\approx$ 4 mW. All implementations of on-board feature extraction were performed on the BHI160's integrated FuserCore. The DIALOG controller is mainly used for packing end sending BTLE packets.

7.1. Energy Measurements

For measuring the energy consumption of the wireless sensor node, we used the approach from [44], as it is easy to deploy and meets the necessary requirements to compare average energy consumption of different configurations. We used a 100 Ω shunt resistor in series with a constant 3 V power supply and a DSO-X 3034A oscilloscope from Agilent Technologies to measure the average voltage drop over the shunt over a time of at least 100*s* per measurement. Due to its proportional nature, the average energy consumption can be deduced from the measured average voltage drop with sufficient accuracy. The temperature dependency was considered to be neglectable in our setup, which could be substantiated with our acquired results. When comparing the average voltage drop of different configurations, the systematic error of the measuring apparatus and the tolerance of the shunt resistor are canceled out, and thus, propagation of uncertainty can be neglected for our evaluations. From all measurements of the voltage drop over the shunt, we calculated the power consumption presented in Section 7.4.

7.2. Model Calibration

Since our custom sensor node is not shipped with any energy models for the FuserCore or the BTLE transceiver, we followed the same approach as we did in [21] to acquire those models. We implemented nine calibration configurations, starting with a baseline of sending raw accelerometer data sampled at 100 Hz via Bluetooth. From this calibration configuration, we synthetically reduced the output data rate of the sensor by implementing four additional configurations in its firmware to only send every 2nd, 4th, 8th, or 16th sample, respectively. This only reduces the output data rate of the sensor, without changing its sampling frequency or power mode. The BTLE packets contained 3D

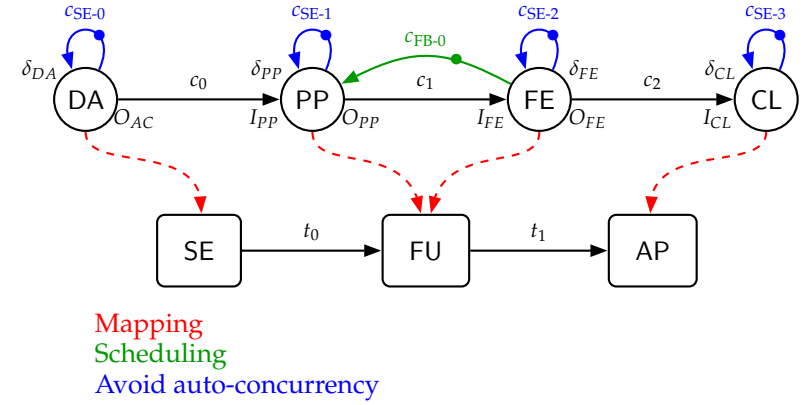
floating point values. By calculating a linear regression over the five measurements, we acquired the dependency of its energy consumption from the output data rate by using the regression slope. We did this for five devices of the custom sensor node and used the mean slope of their regressions.

The same was done for acquiring the energy dependency of the FuserCore. Here we started again with the baseline calibration configuration of sending raw accelerometer data at 100 Hz and implementing four different firmware versions, each introducing artificial computational load, by calculating multiply and accumulate operations in a loop with different loop counters. We measured the computation time of executing the loop and, together with the sampling period of 10 ms with which this code is performed, we calculated the computational load introduced in each configuration. Together with the measured average voltage drops, we calculated a linear regression again over the five devices and used the mean regression slope as the energy model for the FuserCore.

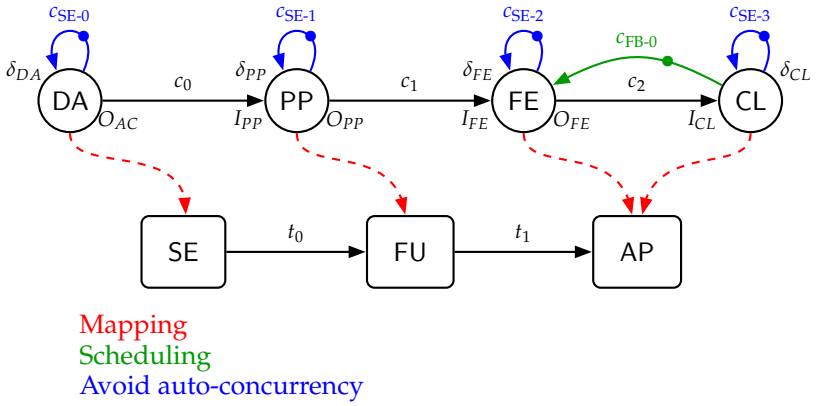
The acquired energy dependency for the BTLE transceiver is 0.01191546 mW/Hz and the energy dependency of the FuserCore is 0.02138247 mW/workload %. The energy consumption for different sensor modalities and sampling frequencies were taken from the BHI160's data sheet [42]. The baseline calibration configuration that sends raw sensor data sampled at 100 Hz without additional processing on the FuserCore is also our reference configuration, from which we estimated all test configurations by our dataflow-based approach in the next section.

7.3. Dataflow Evaluation

We implemented six test configurations of on-board feature extractions or sending raw sensor data on the wireless sensor node with varying sampling frequencies, sliding window length, window overlaps, feature combinations, and mappings. The settings of each configuration and the reference configuration for the energy models can be seen in Table 1. In column *Sensor*, the sensor modality of each configuration is shown, which is either the accelerometer, the gyroscope, or both. The column *SamplingFreq.* describes the sampling frequency of each sensor that is used. If both accelerometer and gyroscope is used, the sampling frequency applies to both. The next column specifies the size of the sliding window in *number of samples* from each sensor modality. The column *Window Overlap* gives the overlap of the sliding windows. Column *Features* shows the implemented feature set on the sliding window calculated for each dimension of all used sensor modalities. The feature set *MVSKPE* is composed of features which have shown good recognition performance in a kitchen task assessment scenario [24]. The feature set is a combination of sample-based processing of Mean, Variance, Skewness, and Kurtosis, and a window-based processing of an FFT where the peak frequency and its magnitude (referred to as Energy in [24]) are used as features. This sums up to six features calculated for each dimension of the sensor data. The feature set *MV* is a set of sample-based processing of the two features Mean and Variance only. The feature $\int$ is an integral over the raw data of each dimension of the sensor signal within the sliding window. The last column, *Mapping*, specifies which mapping is used and refers to Figure 8, showing MAPPING A and MAPPING B, i.e., calculating features on board of the wireless sensor node or sending raw data to the smartphone. Note that in configurations with MAPPING B, the columns *Window Length*, *Window Overlap*, and *Features* contain *ANY*, since in these configurations the feature extraction would be performed on the smartphone and its configuration does not influence the energy consumption of the wireless sensor node.



(**a**) MAPPING A: feature extraction is performed on the sensor's FuserCore.



(**b**) MAPPING B: feature extraction is performed on the smartphone's application processor.

Figure 8. Two different mappings, calculating the feature extraction on (**a**) the FuserCore or (**b**) the smartphone of our hardware setup.

The configurations in Table 1 were chosen to have in each column a variation of parameters that influence the energy consumption of the wireless sensor node. This way, we covered all parameters that can be changed in a particular feature extraction setup to evaluate our approach. In total, this covers three combinations of sensor modalities of accelerometer, gyroscope, or both; sensor sampling frequencies of 50 Hz, 100 Hz, and 200 Hz; sliding window lengths of 50 or 128 samples; overlaps of 0%, 25%, and 75%; three different features sets; and two possible mappings of the feature extraction stage.

Table 1. Sensor settings, sampling frequencies, sliding window parameters, feature sets, and mappings of all configurations.

Cfg.	Sensor	Sampling Freq.	Window Length	Window Overlap	Features	Mapping
Ref.	ACC	100 Hz	*ANY*	*ANY*	*ANY*	B
C1	ACC, GYRO	100 Hz	128	75%	MVSKPE	A
C2	ACC, GYRO	50 Hz	128	75%	MVSKPE	A
C3	ACC	100 Hz	128	75%	MVSKPE	A
C4	GYRO	200 Hz	50	0%	$\int$	A
C5	ACC	100 Hz	128	25%	MV	A
C6	GYRO	200 Hz	*ANY*	*ANY*	*ANY*	B

Each of the configurations has been modeled as a CSDF graph. Figure 8 shows the general graph structures with parameterized input/output rates, actor delays, and two different mappings, namely, calculating the features on the sensor (MAPPING A), or calculating features on the smartphones, i.e., sending raw sensor data from the wireless sensor node (MAPPING B).

The corresponding parameters of each configuration can be found in Table 2. Note that the reference configurations and C6 are used together with MAPPING B, i.e., they are configurations where the wireless sensor node is sending raw samples to the smartphone. Thus, the length of the sliding window, its overlap, and the calculated features are not important for the energy consumption of the wireless sensor node. However, in our graphs we chose configuration C3 with MAPPING B as the reference scenario, and C4 with MAPPING B as C6 to show the impact of shifting the feature extraction of these configurations to the wireless sensor node. The delays δ_{PP} and δ_{CL} are set to zero in the constructed dataflow graphs, as the classification is not part of the evaluation and the preprocessing stage is not changed among different configurations.

Table 2. Graph parameters of all configurations.

Cfg.	δ_{DA} [µs]	$O_{DA}, I_{PP}, O_{PP}, I_{FE}$	$\#cycles_{FE}$	O_{FE}	δ_{FE} [µs]	I_{CL}	Mapping
Ref.	10,000	2	32	$[0,\ldots,0,6]$	$[160,153,\ldots,153,11489]$	6	B
C1	10,000	2	32	$[0,\ldots,0,12]$	$[311,297,\ldots,297,22968]$	12	A
C2	20,000	1	32	$[0,\ldots,0,12]$	$[311,297,\ldots,297,22968]$	12	A
C3	10,000	1	32	$[0,\ldots,0,6]$	$[160,153,\ldots,153,11489]$	6	A
C4	5000	1	50	$[0,\ldots,0,1]$	$[7,\ldots,7,9]$	1	A
C5	10,000	1	96	$[0,\ldots,0,2]$	$[41,33,\ldots,33,62]$	2	A
C6	5000	1	50	$[0,\ldots,0,1]$	$[7,\ldots,7,9]$	1	B

7.4. Results

The throughputs and repetition vectors of the CSDF graphs have been calculated using the SDF3 tools from [34]. The effort functions have been acquired following the principle from Section 6. The throughput of each graph, its repetition vector, the corresponding effort functions, and the predicted energy consumption from these graphs are summarized in Table 3.

The actual measured energy consumption of all configurations on all five devices and their mean are shown in Table 4. The prediction errors, i.e., the differences in the predicted values of energy consumption by the model and the measured consumption scaled with the measured values of all five devices and their mean, are summarized in Table 5. The highest prediction error of 7.5% was achieved for configuration C3. This is on the one hand caused by an outlier with Sensor 3, but also a slightly higher prediction error for this configuration compared to the others. In general, the predicted energy consumption with the model-based approach is close enough to the measured values to substantiate early design decisions. Shifting the feature extraction of the reference configuration towards the sensor (C3) instead of sending raw data (reference configuration), we can predict a consumption of 3.0842 mV instead of 4.0513 mV. The measured consumption is 3.3164 mV (3.192 mV excluding S3), which deviates by 7.5% (3.4% if S3 is excluded) from the predicted value. However, the energy saving from C1 to C3 is predicted to be 23.9% and measured as 18.1% (21.2% excluding S3), which is accurate enough to substantiate at design time the decision of shifting the feature extraction to the sensor to save energy. Similarly, shifting the feature extraction from C6 to the sensor (C4) instead of sending raw data, the model-based approach predicts an energy saving of 30.3%, while the measured energy saving is about 30.2%. Other design decisions, like using an additional sensor, can be evaluated similarly. From configuration C1 to C3, the gyroscope was excluded from the setup. The predicted energy consumption saving of this change is 48.7%. The measured difference is about 45.3% (47.4% if interpreting the value of S3 as an outlier).

Table 3. Graph-extracted model properties and estimated energy consumption of all configurations.

Cfg.	$TH[\mu s^{-1}]$	$rep.vec.^T$	$ef_{freq}(\mathsf{SE})$	$ef_{load}(\mathsf{FU})$	$ef_{trans}(t_1)$	Energy Cons. [mV]
Ref.	3.125×10^{-6}	$[32, 32, 32, 32]^T$	100	0	100	4.0513
C1	3.125×10^{-6}	$[32, 32, 32, 32]^T$	100	0.1006	37.5	6.0117
C2	1.5625×10^{-6}	$[32, 32, 32, 32]^T$	50	0.0503	18.75	5.7872
C3	3.125×10^{-6}	$[32, 32, 32, 32]^T$	100	0.0504	18.75	3.0842
C4	4×10^{-6}	$[50, 50, 50, 50]^T$	200	0.0015	4	5.3854
C5	1.04167×10^{-6}	$[96, 96, 96, 96]^T$	100	0.0033	2.08334	2.8846
C6	4×10^{-6}	$[50, 50, 50, 50]^T$	200	0	200	7.7208

Table 4. Measured energy consumption for five sensor devices of the same type.

Cfg.	S1 [mV]	S2 [mV]	S3 [mV]	S4 [mV]	S5 [mV]	Mean [mV]
C1	6.0318	6.0570	6.1503	5.9913	6.0987	6.0658
C2	5.7789	5.6256	5.7873	5.6421	5.7441	5.7156
C3	3.1545	3.2118	3.8139	3.2058	3.1962	3.3164
C4	5.2497	5.2848	5.4513	5.2356	5.3532	5.3149
C5	2.8485	2.9021	2.9264	2.8981	2.8765	2.8903
C6	7.4898	7.6254	7.8780	7.4613	7.6260	7.6161

Table 5. Errors of the predicted energy consumption compared to the measurements.

Cfg.	Err. S1	Err. S2	Err. S3	Err. S4	Err. S5	Mean Error
C1	0.0033	0.0075	0.0231	0.0034	0.0145	0.0090
C2	0.0014	0.0279	0.0000	0.0251	0.0074	0.0124
C3	0.0228	0.0414	0.2366	0.0394	0.0363	0.0753
C4	0.0252	0.0187	0.0122	0.0278	0.0060	0.0131
C5	0.0125	0.0061	0.0145	0.0047	0.0028	0.0020
C6	0.0299	0.0124	0.0204	0.0336	0.0123	0.0136

8. Discussion

In the previous section, we showed that we can predict the energy consumption of a wireless sensor node in different configurations and mappings of an ARC with a considerable estimation accuracy. Considering a mean prediction error of all configurations of 2.4% (1.6% excluding the outlier of Sensor 3 in configuration C3), a design decision which causes a change of energy consumption of less than 3% is close to the range of prediction error, which might on the other hand not need an accurate prediction, as the potential energy saving is marginal for such cases anyway.

Thus, we evaluate the model-based energy prediction approach as a useful methodology at design time to substantiate design decisions impacting the energy consumption of wireless sensor nodes when designing online activity recognition systems. In the following, we discuss the benefits and limitations of the proposed design method.

8.1. Benefits

The model-based approach allows the comparison of different configurations and mappings of an ARC. These differences can also include energy efficiency strategies, such as sensor selection and feature selection approaches, often found in the literature [9–11,45]. They can be separately modeled and compared to their non-optimized counterparts, showing their impact on the energy consumption. The major benefit of the model-based approach is the estimation at design time, which avoids the actual time-consuming and error-prone implementation of different configurations in attempts to find the most energy-efficient design, which is refined later on. As an additional step

compared to an ad hoc implementation, our proposed approach requires a model calibration. This requires either proper analysis methods, a lot of experience of the system designer (manual calibration), or the partial implementation of sub-functionalities, perhaps available through previous product versions. Nevertheless, the calibration typically only results in additional effort if the final system implementation is obvious in early design steps. Considering today's systems complexity, which is expected to further increase in the future, the spent effort for modeling and calibration will be much less compared to implementing and checking all possible configurations of the system.

8.2. Limitations

The proposed modeling approach with CSDF is limited to the cyclo-static behavior of activity recognition. As an example, a context-aware activity recognition system that changes its configuration (e.g., dynamic sensor selection [9,10] or dynamic feature selection [11,45]) based on the situation cannot be fully captured by CSDF. The data-dependent context changes would require dataflow variants allowing the modeling of dynamic, data-dependent behaviors. Possible dataflow variants could be scenario-aware dataflow (SADF) [4] or enable-invoke dataflow (EIDF) [46]. In contrast to these, our proposed CSDF-based method could only model each configuration present in a context-aware activity recognition system individually and analyze them separately.

9. Conclusions and Future Work

In the article at hand, a formal modeling approach for activity recognition systems is introduced, which is based on dataflow graphs. Furthermore, dataflow analysis techniques are extended to enable estimations of the energy consumption of wearable sensor nodes at early design stages of the corresponding activity recognition system. The proposed modeling approach is shown at different levels of abstraction by applying it to the segmentation and feature extraction stages of typical activity recognition systems. The accuracy of the proposed energy estimation technique is evaluated on a set of configurations, composed of different sensor sampling frequencies, sensor modalities, sliding window parameters like size and overlap, sample-based and window-based feature sets, and mappings of either calculating features on board of wireless sensor nodes, or sending raw data over wireless channels. All configurations have also been implemented in a setup consisting of a custom low-power wireless sensor node and a smartphone. In our experiments, the energy consumption of the wireless sensor node estimated by our model-based approach is compared to the measured energy consumption values. We achieved a system-level average accuracy above 97%, which is accurate enough to substantiate decisions about the configuration of activity recognition systems at early design stages.

In our future work, we want to extend the proposed approach to activity recognition systems with dynamic changes in configuration, such as for context-awareness. Therefore, dataflow variants allowing the modeling of dynamic behavior will be subject to our research in the future.

Author Contributions: F.G. conceived and designed the concepts of the modeling approach and the dataflow graph analysis extension; F.G., A.H., T.K. and C.H. conceived and designed and F.G. performed the experimental evaluation; F.G. and C.H. wrote the paper. All authors critically reviewed the manuscript and approved the final version.

Funding: This work is supported by the German Federal Ministry of Education and Research (BMBF), grant number 03ZZ0519D.

Acknowledgments: We want to thank Bosch Sensortec GmbH for providing the required sensor hardware and their support for programming the experimental environment (BMBF grant number 03ZZ0519G). We acknowledge financial support by Deutsche Forschungsgemeinschaft and Universität Rostock/Universitätsmedizin Rostock within the funding program Open Access Publishing.

Conflicts of Interest: The authors declare no conflict of interest. The funding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

CPS	Cyber-Physical Systems
DVFS	Dynamic Voltage Frequency Scaling
MPSoC	Multi-Processor System-on-Chip
ARC	Activity Recognition Chain
SDF	Synchronous Data Flow
CSDF	Cyclo-Static Data Flow
PP	Preprocessing
FFT	Fast Fourier Transform
APU	Application Processing Unit
BTLE	Bluetooth Low Energy
SoC	System-on-Chip
SADF	Scenario-Aware Dataflow
EIDF	Enable-Invoke Dataflow

References

1. Ghamarian, A.H.; Geilen, M.; Stuijk, S.; Basten, T.; Theelen, B.D.; Mousavi, M.R.; Moonen, A.; Bekooij, M. Throughput analysis of synchronous data flow graphs. In Proceedings of the Sixth International Conference on Application of Concurrency to System Design, Turku, Finland, 28–30 June 2006; pp. 25–36.
2. Stuijk, S.; Geilen, M.; Basten, T. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *IEEE Trans. Comput.* **2008**, *57*, 1331–1345. [CrossRef]
3. Geilen, M.; Stuijk, S. Worst-case performance analysis of synchronous dataflow scenarios. In Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Arizona, AZ, USA, 24–29 October 2010; pp. 125–134.
4. Theelen, B.D.; Geilen, M.C.; Basten, T.; Voeten, J.P.; Gheorghita, S.V.; Stuijk, S. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In Proceedings of the Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design, Washington, WA, USA, 27–30 July 2006; pp. 185–194.
5. Siyoum, F.; Geilen, M.; Corporaal, H. End-to-end latency analysis of dataflow scenarios mapped onto shared heterogeneous resources. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2016**, *35*, 535–548. [CrossRef]
6. Siyoum, F.; Geilen, M.; Corporaal, H. Symbolic analysis of dataflow applications mapped onto shared heterogeneous resources. In Proceedings of the 51st ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 1–5 June 2014; pp. 1–6.
7. Grützmacher, F.; Beichler, B.; Haubelt, C.; Theelen, B. Dataflow-based modeling and performance analysis for online gesture recognition. In Proceedings of the 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data), Vienna, Austria, 11 April 2016; pp. 1–8
8. Das, A.; Kumar, A.; Veeravalli, B. Energy-aware task mapping and scheduling for reliable embedded computing systems. *ACM Trans. Embed. Comput. Syst.* **2014**, *13*, 72. [CrossRef]
9. Gordon, D.; Czerny, J.; Miyaki, T.; Beigl, M. Energy-Efficient Activity Recognition Using Prediction. In Proceedings of the 16th International Symposium on Wearable Computers, Newcastle, UK, 18–22 June 2012; pp. 29–36.
10. Wang, Y.; Lin, J.; Annavaram, M.; Jacobson, Q.A.; Hong, J.; Krishnamachari, B.; Sadeh, N. A framework of energy efficient mobile sensing for automatic user state recognition. In Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services, Kraków, Poland, 22–25 June 2009; pp. 179–192.
11. Yan, Z.; Subbaraju, V.; Chakraborty, D.; Misra, A.; Aberer, K. Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. In Proceedings of the 16th International Symposium on Wearable Computers, Newcastle, UK, 18–22 June 2012; pp. 17–24.
12. Liang, Y.; Zhou, X.; Yu, Z.; Guo, B. Energy-efficient motion related activity recognition on mobile devices for pervasive healthcare. *Mob. Netw. Appl.* **2014**, *19*, 303–317. [CrossRef]

13. Anguita, D.; Ghio, A.; Oneto, L.; Parra, X.; Reyes-Ortiz, J.L. Energy Efficient Smartphone-Based Activity Recognition Using Fixed-Point Arithmetic. *J. Univ. Comput. Sci.* **2013**, *19*, 1295–1314.
14. Van Laerhoven, K.; Aronsen, A.K. Memorizing what you did last week: Towards detailed actigraphy with a wearable sensor. In Proceedings of the 27th International Conference on Distributed Computing Systems Workshops, Toronto, ON, Canada, 22–29 June 2007.
15. Laerhoven, K.V.; Gellersen, H.W.; Malliaris, Y.G. Long term activity monitoring with a wearable sensor node. In Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks, Cambridge, MA, USA, 3–5 April 2006.
16. Lorincz, K.; Chen, B.R.; Challen, G.W.; Chowdhury, A.R.; Patel, S.; Bonato, P.; Welsh, M. Mercury: A wearable sensor network platform for high-fidelity motion analysis. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, California, CA, USA, 4–6 November 2009; pp. 183–196.
17. Grützmacher, F.; Wolff, J.P.; Hein, A.; Lepidis, P.; Dorsch, R.; Kirste, T.; Haubelt, C. Towards energy efficient sensor nodes for online activity recognition. In Proceedings of the IECON 2017-43rd Annual Conference of the Industrial Electronics Society, Beijing, China, 29 October–1 September 2017; pp. 8291–8296.
18. Elsts, A.; McConville, R.; Fafoutis, X.; Twomey, N.; Piechocki, R.; Santos-Rodriguez, R.; Craddock, I. On-Board Feature Extraction from Acceleration Data for Activity Recognition. In Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks, Madrid, Spain, 14–16 February 2018; pp. 14–16.
19. Fafoutis, X.; Marchegiani, L.; Elsts, A.; Pope, J.; Piechocki, R.; Craddock, I. Extending the battery lifetime of wearable sensors with embedded machine learning. In Proceedings of the 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), Singapore, 5–8 February 2018; pp. 269–274.
20. Rault, T.; Bouabdallah, A.; Challal, Y.; Marin, F. A survey of energy-efficient context recognition systems using wearable sensors for healthcare applications. *Pervasive Mob. Comput.* **2017**, *37*, 23–44. [CrossRef]
21. Grützmacher, F.; Hein, A.; Beichler, B.; Lepidis, P.; Dorsch, R.; Kirste, T.; Haubelt, C. Energy Efficient On-Sensor Processing for Online Activity Recognition. In Proceedings of the 8th International Joint Conference on Pervasive and Embedded Computing and Communication Systems, Porto, Portugal, 29–30 July 2018; pp. 85–92.
22. Bulling, A.; Blanke, U.; Schiele, B. A Tutorial on Human Activity Recognition Using Body-worn Inertial Sensors. *ACM Comput. Surv.* **2014**, *46*, 33. [CrossRef]
23. Huynh, T.; Schiele, B. Analyzing Features for Activity Recognition. In Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-aware Services: Usages and Technologies, Grenoble, France, 12–14 October 2005; pp. 159–163.
24. Krüger, F.; Nyolt, M.; Yordanova, K.; Hein, A.; Kirste, T. Computational state space models for activity and intention recognition. A feasibility study. *PLoS ONE* **2014**, *9*, e109381. [CrossRef] [PubMed]
25. Bao, L.; Intille, S.S. Activity recognition from user-annotated acceleration data. In Proceedings of the Pervasive Computing: Second International Conference, Vienna, Austria, 18–13 April 2004; pp. 1–17.
26. Atallah, L.; Lo, B.; King, R.; Yang, G.Z. Sensor positioning for activity recognition using wearable accelerometers. *IEEE Trans. Biomed. Circuits Syst.* **2011**, *5*, 320–329. [CrossRef] [PubMed]
27. Capela, N.A.; Lemaire, E.D.; Baddour, N. Feature selection for wearable smartphone-based human activity recognition with able bodied, elderly, and stroke patients. *PLoS ONE* **2015**, *10*, e0124414. [CrossRef] [PubMed]
28. Damaševičius, R.; Vasiljevas, M.; Šalkevičius, J.; Woźniak, M. Human activity recognition in aal environments using random projections. *Comput. Math. Methods Med.* **2016**, *2016*, 4073584 [CrossRef] [PubMed]
29. Ravi, N.; Dandekar, N.; Mysore, P.; Littman, M.L. Activity Recognition from Accelerometer Data. In Proceedings of the 17th Conference on Innovative Applications of Artificial intelligence, Pittsburgh, PA, USA, 9–13 July 2005; pp. 1541–1546.
30. Chavarriaga, R.; Sagha, H.; Calatroni, A.; Digumarti, S.T.; Tröster, G.; Millán, J.d.R.; Roggen, D. The Opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recognit. Lett.* **2013**, *34*, 2033–2042. [CrossRef]
31. Lee, E.; Messerschmitt, D. Synchronous data flow. *Proc. IEEE* **1987**, *75*, 1235–1245. [CrossRef]

32. Damavandpeyma, M.; Stuijk, S.; Basten, T.; Geilen, M.; Corporaal, H. Modeling static-order schedules in synchronous dataflow graphs. In Proceedings of the Conference on Design, Automation and Test in Europe, Dresden, Germany, 12–16 March 2012; pp. 775–780.
33. Bilsen, G.; Engels, M.; Lauwereins, R.; Peperstraete, J. Cycle-static dataflow. *IEEE Trans. Signal Process.* **1996**, *44*, 397–408. [CrossRef]
34. Stuijk, S.; Geilen, M.; Basten, T. Sdf^ 3: Sdf for free. In Proceedings of the Sixth International Conference on Application of Concurrency to System Design, Turku, Finland, 28–30 June 2006; pp. 276–278.
35. Welford, B. Note on a method for calculating corrected sums of squares and products. *Technometrics* **1962**, *4*, 419–420. [CrossRef]
36. Pebay, P.P. Formulas for Robust, One-Pass Parallel Computation of Covariances and Arbitrary-order Statistical Moments. Available online: https://digital.library.unt.edu/ark:/67531/metadc837537/ (accessed on 27 September 2018).
37. Damavandpeyma, M.; Stuijk, S.; Basten, T.; Geilen, M.; Corporaal, H. Schedule-extended synchronous dataflow graphs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2013**, *32*, 1495–1508. [CrossRef]
38. Lele, A.; Moreira, O.; Cuijpers, P.J. A new data flow analysis model for TDM. In Proceedings of the Tenth ACM International Conference on Embedded Software, Tampere, Finland, 7–12 October 2012; pp. 237–246.
39. Bodin, B.; Munier-Kordon, A.; de Dinechin, B.D. Periodic schedules for cyclo-static dataflow. In Proceedings of the 11th IEEE Symposium on Embedded Systems for Real-time Multimedia, Montreal, QC, Canada, 3–4 October 2013; pp. 105–114.
40. Zebelein, C.; Haubelt, C.; Falk, J.; Teich, J. Model-Based Representation of Schedules for Dataflow Graphs. Available online: https://pdfs.semanticscholar.org/6b77/819dcaa5fd88b453335bfe05528ab3ae7184.pdf (accessed on 27 September 2018).
41. Zebelein, C.; Haubelt, C.; Falk, J.; Schwarzer, T.; Teich, J. Representing mapping and scheduling decisions within dataflow graphs. In Proceedings of the 2013 Forum on specification and Design Languages (FDL), Paris, France, 24–26 September 2013; pp. 1–8.
42. Bosch Sensortec. BHI160 / BHI160B - Ultra Low-Power Sensor Hub Incl. Integrated Imu. Available online: https://www.mouser.com/ds/2/783/BST-BHI160-DS000-01-Datasheet-967967.pdf (accessed on 27 September 2018).
43. Geilen, M.; Falk, J.; Haubelt, C.; Basten, T.; Theelen, B.; Stuijk, S. Performance analysis of weakly-consistent scenario-aware dataflow graphs. *J. Signal Process. Syst.* **2017**, *87*, 157–175. [CrossRef]
44. Russell, J.T.; Jacome, M.F. Software power estimation and optimization for high performance, 32-bit embedded processors. In Proceedings of the International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273), Austin, TX, USA, 5–7 October 1998; pp. 328–333.
45. Zappi, P.; Lombriser, C.; Stiefmeier, T.; Farella, E.; Roggen, D.; Benini, L.; Tröster, G. Activity recognition from on-body sensors: accuracy-power trade-off by dynamic sensor selection. In Proceedings of the 5th European Conference on Wireless Sensor Networks, Bologna, Italy, 30 January–1 February 2008; pp. 17–33.
46. Plishker, W.; Sane, N.; Kiemb, M.; Anand, K.; Bhattacharyya, S.S. Functional DIF for rapid prototyping. In Proceedings of the 19th IEEE/IFIP International Symposium on Rapid System Prototyping, Monterey, CA, USA, 2–5 June 2008; pp. 17–23.