Project acronym:	**PrEstoCloud**
Project full name:	**Proactive Cloud Resources Management at the Edge for efficient Real-Time Big Data Processing**
Grant agreement number:	**732339**

D3.5 Mobile Context Analyser

Deliverable Editor:	**Dimitris Apostolou (ICCS)**
Other contributors:	**Nikos Papageorgiou, Andreas Tsagkaropoulos, Yiannis Verginadis, Gregoris Mentzas (ICCS)**
Deliverable Reviewers:	**Salman Taherizadeh (JSI)**
Deliverable due date:	**31/03/2018**
Submission date:	**31/03/2018**
Distribution level:	**Public**
Version:	**1**

This document is part of a research project funded by the Horizon 2020 Framework Programme of the European Union

Change Log

Version	Date	Amended by	Changes
0.1	15/12/2017	Dimitris Apostolou (ICCS)	Table of Contents
0.2	20/3/2018	Dimitris Apostolou, Nikos Papageorgiou, Andreas Tsagaropoulos, Gregoris Mentzas, Yiannis Verginadis (ICCS)	Draft version ready for internal review
0.3	21/3/2018	Salman Taherizadeh (JSI)	Review
1.0	30/3/2018	Dimitris Apostolou (ICCS)	Final version addressing review comments

Table of Contents

List of Tables

List of Figures

List of Abbreviations

The following table presents the acronyms used in the deliverable.

Abbreviation	*Description*
ACE	Acquisitional Context Engine
AMQP	Advanced Message Queuing Protocol
CPU	Central Processing Unit
CSV	Comma Separated Value
DSL	Domain Specific Languages
HDA	Highly Distributed Applications
JSON	JavaScript Object Notation
MCA	Mobile Context Analyser
MCC	Mobile Cloud Computing
MEC	Mobile Edge Computing
OMG	Object Management Group
OS	Operating System
TOSCA	Topology and Orchestration Specification for Cloud Applications
UAV	Unmanned Areal Vehicle
UML	Unified Modelling Language
VM	Virtual Machine
XMI	XML Metadata Interchange
XML	Extensible Markup Language
LTE	Long-Term Evolution
GPS	Global Positioning System

Executive Summary

This deliverable reports on the work performed under Task 3.5 with respect to the development of a context analysis engine. The engine is part of the Meta management layer of the PrEstoCloud architecture, which mainly provides valuable input to the PrEstoCloud Situation Detection Mechanism (D 5.1) which in turn will support the Control layer perform adaptation of cloud resources Manager. The context analysis engine is able to detect and process the context of devices at the extreme edge of the network and thus greatly influences the recommended adaptations of the processing topology, especially at the edge.

The context analysis engine provides a mechanism to detect the state of the processing resources at the extreme edge and provides to communicating components an abstracted view of this data. Moreover, it implements a Context Model to describe information it receives from all extreme edge devices through the PrEstoCloud Communication Broker. The retrieved information is continuously filtered and normalized in order to reassure that it can be correctly processed. Finally the enhanced context of the devices is published to the Communication broker, while saving intermediate results on scalable data stores.

The Mobile Context Analyser follows the line of thought adopted by modern research in the extensive field of context sensing and processing, and provides a scalable, containerized solution to retrieve basic and derived information for the devices at the edge of the network. It is also extensible and modular, permitting the integration of new components to the existing if a specific type of processing is required for some kinds of data. In order to aid attempt to comprehend and extend the engine, both the conceptual and concrete architectures of the module are provided, mapping sub-components to specific applications. This document also contains a walkthrough of the application of the engine in a device monitoring scenario, illustrating the process of context inferencing.

1. Introduction

1.1 Scope

This work describes the methodology and the architecture followed and implemented, to create a context analysis software component, mainly targeted (but not limited) to devices functioning in the extreme edge of the network, and providing information on the current health status of edge devices as well as their capability to perform some processing tasks. The information published on the PrEstoCloud Broker will be used by other, higher-level components of the Meta-management layer to detect situations requiring adaptation as well as to improve the adaptation recommendation itself.

The need to be able to direct processing tasks not only from the edge to the cloud, but also in the reverse direction, presents the need for a specialized software component to monitor the state of devices at the extreme edge of the network, understand it, and provide decision making components with all data required to propose an efficient adaptation of the processing topology. The Mobile Context Analyzer undertakes this responsibility, and further provides to other components of the PrEstoCloud architecture an abstraction layer over edge devices.

Furthermore, the Mobile Context Analyzer is an active component in the sense that it can learn from the past behaviour of a device type/user or a group of device types/users and therefore can improve the quality of the interpretation it performs on raw data. Additionally, the approaches used to enrich and process raw data are not hard-wired in the core logic of the Analyzer module, but can be further improved and extended should this be desirable.

1.2 Relation to PrEstoCloud Tasks

The Mobile Context Analyzer component has been defined in the description of work of the PrEstoCloud project as part of Task 3.5. It materialises a system able to sense and detect context in the way the latter was defined in Deliverable D2.1 (Scientific and Technological State-of-the-Art analysis) and formulated as a functional requirement in Deliverable D2.2 (High-level requirements analysis for the PrEstoCloud platform). The component also adheres to the foundations set for the entire PrEstoCloud topology in deliverable D2.3 (PrEstoCloud Conceptual Architecture). The Mobile Context Analyser receives input from the Communication broker for real-time data streams and processes it taking into account the specifications of the communication format developed in Deliverable D2.4 (Format and procedures for plugging in real-time data streams).

The Mobile Context Analyser is itself a primary input for the Situation Detection Mechanism (Task T5.1), while it also produces information required for the proper operation of the Resources Adaptation Recommender (Task T5.2). Information gathered and processed by the component is also retrieved by the Autonomic Data-Intensive Application Manager (Task T4.2) and the Application Placement and Scheduling Controller (Task T4.4), which use it in order to direct the processing of application fragments to the edge when this is deemed appropriate.

1.3 Document Structure

The deliverable is structured as follows: Section 2 presents the related work already performed in the general area of Context sensing and processing. Section 3 includes our approach on the development of the PrEstoCloud Context Model. Section 4 presents the methods and the technologies used in our Context analysis approach through an illustrative example. Last, in Section 5 we formulate our conclusions on the approach and the development of the component.

2. Related Works

Interest in mobile context sensing has grown proportionally to the vast increase in the number of computing and IoT devices that are literally attached to the everyday life of individuals. Situations and entities that comprise the environment are collectively termed context (Pejovic & Musolesi, 2015). Context may have numerous aspects: geographical, physical, social, temporal, or organisational, to name a few. Context sensing aims at bridging physical stimuli sensed by the device's sensors, also known as modalities, and high level concepts that describe a context. Various mobile devices are perceptive devices theoretically capable of inferring the surrounding context.

Context sensing has been enabled by two factors. First, mobile computing devices such as smartphones are provisioned with sophisticated sensors, as well as with communication and computation hardware such as GPS, accelerometer, gyroscope, proximity and light sensors, microphones and cameras. Multi-core processors and sizable memory allow mobile devices to locally handle a large amount of data coming from these senses and extract meaningful situation descriptors, while a range of communication interfaces, such as WiFi, Bluetooth, 4G/LTE, and a near-field communication (NFC) interface, allow distributed computation and remote data storage. The second key factor is the increasingly ubiquitous and personal usage of mobile computing devices.

Context inference involves stages such as sensing, classification and prediction (Pejovic & Musolesi, 2015). Such stages are needed to transform raw data to high-level contextual cues. Sensing is the first stage, which aims to be the entry point to physical context-generating devices. A subsequent step is feature extraction which transforms raw data to a form which can be used for context inference. A final step is context modelling which focuses on the creation of models that relate relevant high level contextual information with collected data features.

2.1 Context sensing

Mobile sensors can reveal high level information about the device internal state. A single sensor modality is seldom sufficient for inferring the context in which a device is. In addition, multimodal information can offset the ambiguities that arise when single sensor data are used for inference (Maurer et al., 2006). Multimodal information can uncover relationships previously unknown or difficult to confirm through conventional approaches. For example, app processing can be correlated with user's location or activity (Puiatti et al., 2011).

Mobile sensing is subject to constraints coming from the devices' hardware restrictions. Frequent sensing of different modalities and collaboration of multiple agents are likely to be necessary for accurate context inferencing, emphasizing the need for resource-efficient mobile sensing solutions. Energy-efficient operation, processing, storage and communication constraints are the most common practical mobile sensing challenges. In Table 1 we summarise the state-of-the-art solutions to address these issues. Energy issues are exacerbated by the design of mobile sensors as occasionally used features, rather than constantly sampled sensors. Two popular means of reducing the energy consumption are adaptive sampling, i.e., sampling less often, and, in the case of a device with multiple sensors, powering them on hierarchically, i.e., preferring low-power sensors to more power hungry ones. Adaptive sampling and hierarchical sensing are not the only means of reducing energy usage. The inherent structure of the context inference problem can also be used to improve sensing efficiency. This is the main idea behind the Acquisitional Context Engine (ACE) proposed in (Nath, 2012). Here, Nath develops a speculation-based sensing engine that learns associative rules among contexts, an example of which would be "when a user state is driving, his location is not at home". When a context-sensitive application needs to know if a user is at home or not, it contacts ACE that acts as a middle layer between sensors and the application. ACE initially probes a less energy costly sensor – accelerometer – and only if the sensed data does not imply that a user is driving, it turns the GPS on and infers the actual user's location.

2.2 Context Modelling

(Rivero-Rodriguez, Pileggi & Nykänen, 2016) argue that there exists no universal model for context related data. (Strang & Linnhoff-Popien, 2004) present key-value, markup scheme, graphical, object-oriented, logic-based and ontology-based models for modelling context. Ontologies have been user extensively to model context, providing advantages such as information alignment, and the ability to deal with incomplete or partially understood information. The W3C Semantic Sensor Network (SSN) ontology was developed by reviewing 17 existing sensor ontologies (Lefort, Henson and Taylor 2011, Compton et al., 2012), also aligned with the general DOLCE Ultra Lite upper ontology providing concepts such as Physical Object, Event, etc. Other ontologies acknowledge a more generalized logical context, such as the Service-Oriented Context-Aware Middleware (SOCAM) architecture, which provides efficient infrastructure support for building more complex Context-aware Services in pervasive computing environments (Gu, Hung Keng & Da Qing, 2005). SOCAM also acknowledges the needs of using a two-level information architecture: general contextual information is described using SOCAM ontology, while more application-specific concepts use domain-specific ontologies. In previous work, we developed our own ontology-based context model for real-time processing systems (Verginadis et al., 2015).

The first challenge in context modelling is the identification of those modalities of raw data that are the most descriptive of the context. Interdisciplinary efforts and domain knowledge are crucial in this step. A subsequent step is the choice of the method for representing context data. Distinctive properties extracted from the data are called features. Typical features used in selected context domains are listed in Table 1. Modality and feature selection impact the rest of the context inference; a careful consideration at this stage of the process can help improve classification accuracy or reduce the computational complexity of the learning process. As mobile sensing matures the variety of context types that researchers strive to infer broadens. In addition, the number of sensors available on the smartphone increases steadily. Therefore, identifying and quantifying the strength of a link between a domain and a modality (or a feature) emerges as an important research direction in mobile sensing.

Table 1. Context domains and characteristic features (adopted from Pejovic & Musolesi, 2015)

Domain	Characteristic Features
Audio and speech recognition	Sound spectral entropy, zero crossing rate, low energy frame rate, spectral flux, spectral rolloff, bandwidth, phase deviation (Lane et al., 2010)
	Teager Energy Operator (TEO), pitch range, jitter and standard deviation, spectral centroid, speaking rate, high frequency ratio (Lu et al., 2012)
	Running average of amplitude, sum of absolute differences [Krause et al. 2006]
	Perceptual linear predictive (PCP) coefficients (Rachuri et al., 2011)
	Mean and standard deviation of DFT power (Miluzzo et al., 2008)
Location	Days on which any cell tower was contacted, days on which a specific tower is contacted, contact duration, events during work/home hours (Isaacman et al., 2011)
	Tanimoto Coefficient of WiFi fingerprints (Chon et al., 2012)
	Eigenbehaviors - vectors of time-place pairs (Eagle & Pentland, 2009)
	Hour of day, latitude, longitude, altitude, social ties (De Domenico et al., 2012)

2.3 Context Classification

Many machine learning methods have been used to transfer sensor data into mathematical representations of a mobile device environment. Table 2 lists indicative mobile context domains and the corresponding machine learning methods that have been applied to classify context. Indicative methods used by scholars in Context Classification include Hidden Markov Models, Markov Chains, Bayesian Networks, Nearest Neighbour, Time Series, Threshold based Learning and Gaussian Mixture Models. Scaling up imposes novel challenges with respect to sensing application distribution, data processing and scalable machine learning. We increasingly observe ecosystems of devices, where multiple devices work together towards improved context sensing. For example, the Energy Efficient Mobile Sensing System (EEMSS) proposed by Wang et al. hierarchically orders sensors with respect to their energy consumption, and activates high-resolution power-hungry sensors, only when low-consumption ones sense an interesting event [Wang et al. 2009]. The variety of classification methods and data features can be overwhelming for a mobile sensing application designer and careful analysis of the purpose and the intended use of context classification can help the designer select only relevant features and subsequently the applicable machine learning method.

Table 2. Context domains and relevant machine learning techniques (adopted from Pejovic & Musolesi, 2015)

Domain	Machine learning method
Audio and speech recognition	Hidden Markov Model ((Chon et al., 2012; Choudhury & Pentland, 2003) Threshold based learning (Wang et al., 2009) Gaussian Mixture Model (GMM) (Rachuri et al., 2010; Lu et al., 2012)
Location	Markov chain (Ashbrook & Starner 2003) Non-linear time series (Scellato et al., 2011; De Domenico et al., 2012) Bayesian network (Eagle & Pentland 2006; Eagle et al., 2009) Nearest neighbour (Maurer et al., 2006)

2.4 Context Prediction

Context prediction, especially in lieu of mobile computing devices, has focused on energy-related predictions (Ravi et. all, 2008; Bramble & Swift, 2014; Peltonen et al., 2015 & Wagner et al., 2013) as well as prediction of processing capability (Zhou, et al., 2015; Tillenius et al., 2015) and computation offloading (Xia et al., 2014; Akherfi et al., 2016). Other works exploit multimodal data for context prediction, see e.g., (Eagle & Pentland, 2006; Laurila et al., 2012) which use such data sets served as a proving ground for a number of approaches towards mobility prediction.

Table 3 outlines examples of methods used for the prediction of the movement and location of mobile devices because this has been an area of active research as well as because this application of context prediction is relevant to our work in PrEstoCloud. Historically, the prediction of mobile devices' movement patterns was tied with system optimisations. For small-scale indoor movement predictions, systems can rely on sensors embedded in the buildings. An example of such systems is MavHome: the authors propose a smart home which adjusts indoor light and heating according to predicted movement of house inhabitants (Cook et al., 2003).

In addition, predicted location can be considered on a level higher than geographical coordinates. The NextPlace project aims to predict not only user's future location, but also the time of arrival and the interval of time spent at that location (Scellato et al., 2011). The authors base the prediction on a non-linear

time series analysis. SmartDC merges significant location prediction with energy-efficient sensing, and proposes an adaptive duty cycling scheme to provide contextual information about mobility of users (Chon et al., 2013).

Table 3. Modelling methods for mobility prediction (adopted from Pejovic & Musolesi, 2015)

Method	Example
Markovian	Markov process (MP) (Ashbrook & Starner 2003; Song et al. 2004)
Nonlinear time series analysis	NTSA [Scellato et al. 2011], (De Domenico et al., 2012)
Bayesian	Dynamic Bayesian Network (Eagle & Pentland 2006; Eagle et al., 2009) (McInerney et al., 2013; Etter et al., 2013)
Other/Hybrid	MP with NTSA (Chon et al., 2013) Information-theoretic uncertainty minimization (Bhattacharya & Das, 2001; Cook et al., 2003) Statistical regularity-based model (McNamara et al., 2008), Temporal, spatial probabilistic model (Cho et al., 2011), Frequent meaningful pattern extraction (Sadilek & Krumm, 2012) M5 trees and linear regression (Noulas et al. 2012)

In another noteworthy approach coming from Big Data world, the Apache Metron[1] provides a security analytics framework built with the Hadoop Community evolving from the Cisco OpenSOC[2] Project. It is a cyber security application framework that provides organizations the ability to detect cyber anomalies and enable organizations to rapidly respond to identified anomalies. Although the core features and scope of Metron is not directly linked to PrEstoCloud, its real time processing capabilities include contextual analysis of the raw streaming data in order to infer context information related to threat intelligence, geolocation, and DNS information to telemetry being collected. The immediate application of this information to incoming telemetry provides the context and situational awareness, as well as the "who" and "where" information that is critical for investigation.

[1] http://metron.apache.org

[2] http://opensoc.github.io/

3. Mobile Context Model & Approach

We can find many complementary definitions of context in the literature. The survey paper of Li et al. (2015) summarizes some of the most known definitions of context and context awareness. From the perspective of the PrEstoCloud Mobile Context Analyzer (MCA), context is defined by specializing in our domain one of the most cited definitions given in (Dey, 2001). The term context here refers to any information (such as CPU, memory or utilization, network type and traffic, battery state, software and hardware configuration, etc.) that can help to infer conclusions about current and future state of processing topologies in which participate multiple mobile or extreme edge devices (such as phones, tablets or drones).

The specification of the PrEstoCloud context model is needed in order to store, process and distribute mobile context. Moreover, the context model will be the stepping stone for facilitating event-based context detection and inference functionality, in order to better understand situations in dynamic cloud and fog computing platform. Specifically, context will be used to better respond to situations that demand for new computing resources at the edge or/and lead to a number of service adaptations. In order to achieve the goal of extracting contextual information, analysing them and then deriving higher level context, we follow an event-based context modelling approach.

In this section, we present the PrEstoCloud Context Model (Figure 1), expressed in UML 2.0 class diagram. We chose UML class diagrams because of their high expressivity, the fact that they are well understood by developers and analysts and because of the plethora of UML complaint tools. The model is based on the W4H model (Truong et al., 2009) that describes the five main elements associated within a context; the five elements are arranged into a quintuple (When, What, Where, Who, How). Our model also takes into account the Contextual Metamodel proposed by (Santos, 2008), which depicts the key entities and relations between entities that represent contextual information.

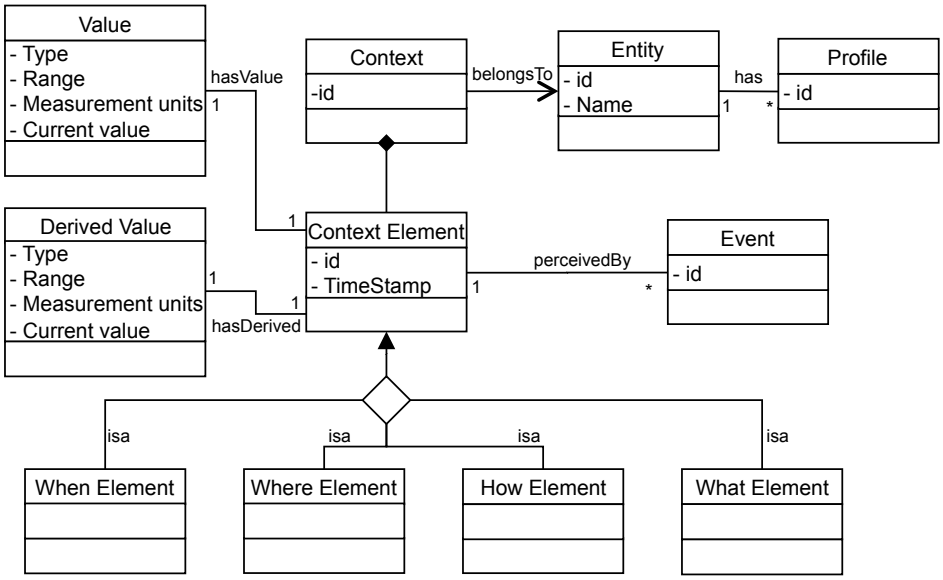

Figure 1. PrEstoCloud context model

The Context Model expresses the temporal (i.e. When), spatial (i.e. Where), declarative (i.e. Who, What) and explanatory (i.e. How) dimensions of context having as central point of focus the notion of Entity. Entities refer to either physical or virtual entities with specific profiles and preferences that characterise them (e.g. a mobile device). We use the context class to refer to a number of context elements that are related to the five dimensions of context. Each Context element can have a value that can be asserted and/or a derived value that is inferred from any kind of reasoning or computing process or function. All context related information should be captured by the Mobile Context Analyser as objects which can store either a single scalar value or multiple values such as vectors, sets, lists etc.

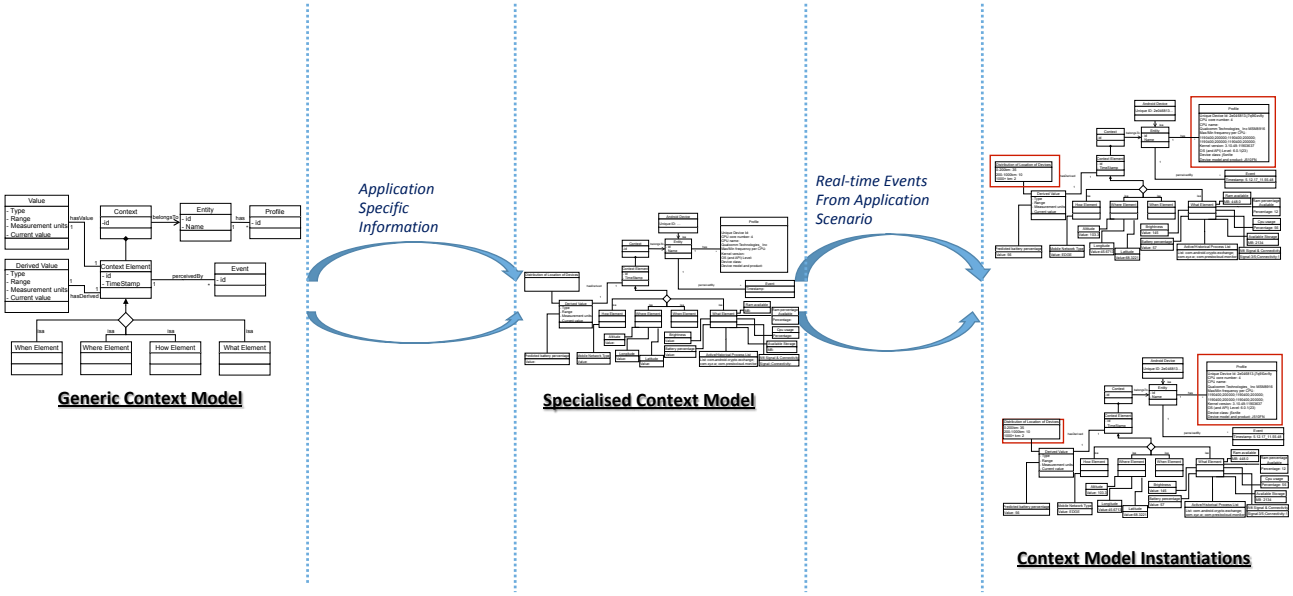

Figure 2. Context model lifecycle

As any of the available context models (Strang & Linnhoff-Popien, 2004), the context model of PrEstoCloud needs to become domain- or application-specific in order to be useful. Hence, we follow the process of generic model usage, model specialization and instantiation (Figure 2). Later in this deliverable, we illustrate how context model instantiation is realized for a specific application scenario.

4. Context Analysis Methods and Technologies

4.1 Requirements

As mentioned in section 3, in our work the term context refers to any information (such as CPU, memory or utilization, network type and traffic, battery state, software and hardware configuration, etc.) that can help to infer conclusions about current and future state of processing topologies in which participate multiple mobile or extreme edge devices (such as phones, tablets or drones). These conclusions may be deduced from a single device (for example a device cannot handle a specific task because it does not have sufficient battery) or by combining information from multiple devices either deterministically (i.e. there are more than 4 devices with 8 cores and low CPU utilization within a range of 100m of a WiFi hotspot) or probabilistically (i.e. from historical information it is inferred that a device that has brightness level 90% and CPU utilization 60% will consume 5% of its battery in the next 30 minutes).

In order to be able to analyse context and answer to a broad range of queries as the above mentioned we must design, implement and operate a software component with specific characteristics. We have identified the following high-level functional and non-functional requirements for the Mobile Context Analyzer (Table 4).

Table 4. MCA high-level requirements

No	Requirement Description	Requirement Type
1	Collect, store and process contextual information from different types of edge devices.	Functional
2	Perceive raw contextual information in the form of events.	Functional
3	Read and store events that contain contextual information with payload which can be in multiple formats (such as CSV or JSON) upon appropriate configuration without developing new parsing software components for each type.	Functional
4	Produce inferred context by combining and analysing raw information, deterministically or not.	Functional
5	Publish enriched contextual information as events.	Functional
6	Answer to contextual queries with a RESTful API in JSON format.	Functional
7	Validate the quality of input and output information.	Functional
8	Assess new information efficiently incrementally in order to provide inferred context with lower resource consumption.	Non-Functional
9	Operate in an efficient and scalable way.	Non-Functional
10	Process and deliver contextual information with low latency.	Non-Functional
11	Be flexible and easily integrated with external software and the other components of PrEstoCloud.	Non-Functional

4.2 Context Analysis Approach

MCA should be able to operate in an environment where different kinds of extreme edge devices, cloud services, operating systems and hardware architectures coexist. Under these circumstances it is expected that contextual information from such entities will arrive to MCA in different formats depending on the capabilities and limitations of them. There are already many open source tools that are designed for the collection, filtering and transformation of logs from multiple sources like Logstash

(https://www.elastic.co/products/logstash), Fluentd (https://www.fluentd.org/), Syslog-ng https://syslog-ng.com/open-source-log-management and Rsyslog (http://www.rsyslog.com/). These tools can be used during the initial stages of context processing pipeline in order to pre-process the input data. Each tool has its own strengths and weaknesses, see https://sematext.com/blog/logstash-alternatives/ and (Andreassen et al., 2015; Vega et al., 2017; Vaarandi et al., 2013), but most of them are considered adequate for this task. After taking in consideration their capabilities, we have concluded that in the first version of MCA we will use Logstash for the context pre-processing functionalities of MCA that include the transformation of input data from CSV to JSON format, the conversion of different date-time strings to a common format, the sending and receiving of streaming events based on the Advanced Message Queuing Protocol (AMQP) protocol and the writing of context data to Elasticsearch[3]. (Note that for the transmission of raw context data from edge devices we will rely on MQTT.) Logstash provides all these functionalities out-of-the-box without the need to install plugins or to write extra software. As we will show later MCA should be integrated with the PrEstoCloud Broker which will be implemented on top of RabbitMQ[4] and Elasticsearch. RabbitMQ supports among others the lightweight MQTT and more advanced AMQP protocols for publishing and receiving data in the form of events.

With MCA we aim not only to collect, transform and distribute context as it is provided by each source but we have the intention to provide higher value context analysis services. With these services the user will be able to receive beyond the raw context data, inferred values of context features. These inferences may be produced either by quantitative statistical methods or by more advanced machine learning algorithms. In both cases the supported inferences will be data-driven. Elasticsearch (https://www.elastic.co/products/elasticsearch) is an open-source scalable analytics engine with a RESTful API and many embedded aggregation functions. It can store and index data in JSON format with good performance (Abubakar et al., 2014; Teodoro et al., 2018) and scalability. MCA will exploit the data aggregation framework of Elasticsearch in order to provide higher-level (aggregated) context to other PrEstoCloud components.

Beyond aggregated context, MCA aims to provide real-time context inference functionalities based on models that are constructed with machine learning methods. Streaming data which contain raw context from extreme-edge devices will be enriched with predicted features in real-time. For example, for each mobile phone MCA will predict the remaining battery percentage in the next minutes. In order to implement the context prediction features of MCA, we will adopt a microservice-based architecture implemented on top of Docker (https://www.docker.com) containers. In this way, as we will demonstrate scalable processing topologies, based on loosely-coupled micro-services that may operate in different execution environments and may be written even in different programming languages.

In this version of MCA, in order to implement the high level context inferencing, we used the machine learning capabilities of a variation of the NEAT (NeuroEvolution of Augmenting Topologies) algorithm (Stanley et al. 2002). Neuro-evolution, has the ability to construct artificial neural networks by using data-driven genetic algorithms without requiring from the designer to possess extensive experience on neural networks or to describe the rules that hold on the specific domain. Recent studies (Such et al. 2017) claim that neural networks trained with genetic algorithms can achieve very good results in reinforcement learning.

NEAT is a genetic algorithm that aims to generate and evolve Artificial Neural Networks (ANN). The Instinct algorithm (https://towardsdatascience.com/neuro-evolution-on-steroids-82bd14ddc2f6) is a variation of the NEAT algorithm which, according to the author, overcomes a set of problems of NEAT in order to better support complex datasets. It adds new capabilities to NEAT such as using different activation functions and memory cells during the mutation process.

[3] https://www.elastic.co/

[4] https://www.rabbitmq.com/

The Neataptic library (https://wagenaartje.github.io/neataptic/docs/) is an implementation of the Instinct algorithm in Javascript. Neataptic can combine up to fifteen types of activation functions such as Sigmoid, Relu, Step and Gaussian in networks with multiple layers. Another important feature of Neataptic for the context inferencing in MCA is that it can be used to incrementally train artificial neural networks with new training datasets by starting from the output of a previous dataset or to retrain them with the same dataset in order to reduce the error (by running additional iterations of the algorithm). In this way machine learning models learned from an initial dataset can be used to build faster models for other datasets that are produced by similar devices.

4.3 Methods and Algorithms to Infer High Level Context

The conceptual architecture of MCA is depicted is Figure 3. It is organized in three layers. The *Online Processing layer* contains the tasks that are executed in real-time as the events that contain raw contextual information arrive from the Broker. *The Batch Processing layer* contains the tasks that are executed on demand or periodically with input from historical data. The *Storage layer* depicts the long term data stores of the MCA. The tasks that are depicted with dashed outlines will be implemented in the second version of MCA (due M30).

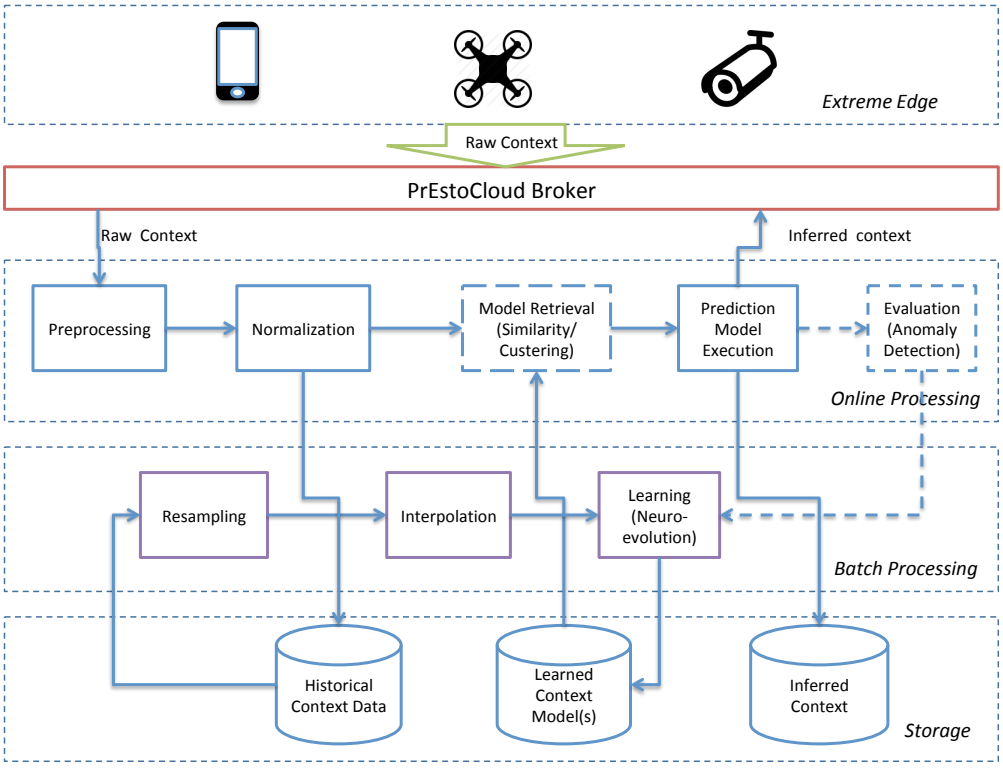

Figure 3. Mobile Context Analyzer conceptual architecture

Raw context from edge devices like mobile phones, drones, IP cameras is published in the form of events to the PrEstoCloud Broker. During the pre-processing stage, the MCA subscribes to the Broker and receives the published context events. The main role of the pre-processing stage is to convert the input events to a format that is suitable for the next stages. During the pre-processing stage raw events are converted to the appropriate format (for example CSV data may be converted to JSON), the values of the different fields are validated, date/time strings are converted from one format to another. After the pre-processing, the input data fields that are related to features of the prediction model must be normalized. These features can be derived from device sensor values such as the current memory utilization, the remaining battery capacity and the current screen brightness setting or device specifications such as the number of cpu cores, the total RAM, or the flash storage capacity. This task is performed during the Normalization stage. After the normalization, the events are stored to the Historical Context Data store and in parallel are forwarded to the Model Retrieval component. The model retrieval component chooses the right models according to the

characteristics of the device that sends the event (device type, model, software and hardware configuration). Then the appropriate prediction model is executed. An example of a model that will be presented in this document is one that is designed in order to predict the battery percentage of a mobile phone in the next minutes based on the current state of it.

The parameters of the prediction models are learned from historical data inside the Batch Processing layer. Initial data are retrieved from the Historical Context Data store. Then these data are processed by the Resampling and Interpolation components where the numeric values are aligned in regular time intervals. The output of this processing is injected to the (machine) Learning component. This component tries to construct a model that is able to predict the desired features with the minimum error possible. The machine learning method that will be used in MCA is described in the corresponding paragraph of this document (paragraph 4.4.3). It is based on neural networks that are designed and evolved automatically by a genetic algorithm. The output of the Learning component is a prediction model that can be used in real-time for the context prediction of the desired feature (e.g. future battery percentage). It is possible to construct different models for different types of devices. These models are stored in the Learned Context Model store.

4.4 Technologies & Implementation

In this section we describe the implementation details of the MCA while we discuss all the relevant technological decisions. To illustrate the use of various technologies and implementation details, we will follow an example scenario.

4.4.1 Data Acquisition

A primary asset for the Mobile Context Analyzer is the Context-Logging agent, developed for exemplifying the data acquisition with respect to one type of edge devices (i.e. Android devices). Over the last few years the Android-powered devices and especially smartphones have become broadly popular with the number of such devices surpassing the 2 billion (https://twitter.com/Google/status/864890655906070529). The data which can be gathered in a typical Android device ranges from sensor data (such as GPS, temperature, accelerometer, barometer) to processing state metrics (CPU,RAM consumption, running processes) and data concerning the operational environment of the device (WiFi state and signal strength, mobile network type etc.). The Android agent we developed collects and submits context data to the PrEstoCloud Broker, which in turn makes it available for further analysis (for example by the Mobile Context Analyzer processing pipeline) to any PrEstoCloud component. All communication is realised using MQTT minimizing the power consumption overhead arising from the usage of heavier protocols such as http (Joshi et al., 2017), and secured with Transport Layer Security (TLS).

A necessary precondition for the mobile application to be installed and function properly, is the use of Android version of 4.4 or newer, and the availability of internet connectivity (WiFi or cellular). Following its installation, the application sends an announcement message, and thereafter regularly sends monitoring data to the PrEstoCloud Broker. The two message types (for the announcement and monitoring of the device) used by the application, are both described in CSV format. The announcement event is used only once. The native characteristics of the mobile phone reported to the Broker are presented in Table 5:

Table 5. Transmitted native characteristics of mobiles

Field ID	Field Name	Description
1	Information Protocol version	The version of the information protocol used as an integer, concatenated with the character 'a' (to signify an announcement event).
2	Unique Device Id	The unique id of the device, derived from the characteristics of the operating system run by the phone, is anonymised and transmitted.

3	CPU core number	The number of cores reported by the Android runtime.
4	CPU name	The name of the CPU of the device.
5	Maximum /Minimum frequency per CPU	The maximum and minimum frequency for each CPU of the device, in Hz.
6	Kernel version	The Linux kernel version used by the device.
7	OS (and API) Level	The Android OS installed on the device, and the corresponding Android API which can be used for programming on it.
8	Device class	An Id describing the family of mobile devices that this device belongs to.
9	Device model and product	A string containing the product series of the mobile devices that this device belongs to, and the model.

The information contained in each transmission is shown in Table 6:

Table 6. Mobile phone tranmitted events

Field ID	Field Name	Field Update	Description
1	Information protocol version	None (constant variable)	The version of the communication format represented as an integer.
2	Unique device id	None (constant variable)	The unique id of the device, derived from the characteristics of the operating system run by the phone, concatenated with a random string.
3	Current frequency for all CPU's	Every 5 seconds	The frequency for every CPU of the mobile device in Hz.
4	Screen brightness level	Every 30 seconds	An integer value from 0 to 255, representing the brightness of the screen of the device, from darkest to brightest.
5	Available RAM	Every 10 seconds	A real number denoting the amount of MBs free in the main memory of the device.
6	Available RAM percentage	Every 10 seconds	An integer percentage value denoting the percentage of main memory of the device, which is free.
7	Available storage	Every 10 seconds	An integer number denoting the amount of MBs in the filesystem available for applications to use.
8	Latitude	On location change	A float value in degrees, denoting the latitude of the device.
9	Longitude	On location change	A float value in degrees, denoting the longitude of the device.
10	Altitude	On location change	A float value denoting the altitude of the device
11	Wifi signal and connectivity status	Every 10 seconds	An integer number from 0 to 5, denoting the strength of the of the WiFi signal (from none to excellent),

			accompanied by a value '1' when WiFi is on or '0' when WiFi is off.
12	Active processes list or historical processes list	Every 10 seconds	A list of strings containing the methods which are currently executed[5] or have been executed in the past 24 hours[6].
13	Mobile network type	Every 60 seconds	The mobile network type (e.g. LTE, EDGE, etc.) used by the device.
14	Battery percentage, voltage, temperature and charging status	On battery status change (Upon ACTION_BATTERY_CHANGED intent reception)	A list of boolean, integer and real number values, containing the remaining battery percentage, the voltage of the battery, the temperature of the battery, and the charging status of the device ('0' when not charging and '1' when charging)
15	CPU consumption	Every 5 seconds	An integer percentage value containing the average load from samples collected over the last 3 minutes.
16	Timestamp	Upon creation of the record	The time that the record was created, in the format of day.month.abbreviatedYear_hour.minute.second

The mobile application consists of a number of monitoring modules (classes), which refresh their data either when an event occurs (for example the GPS location update) or after a set time interval. The instantiation of the context model for the case of the mobile phone is shown in Figure 4.

[5] Applies to Android versions prior to Android Nougat

[6] Applies to Android Nougat and succeeding Android versions

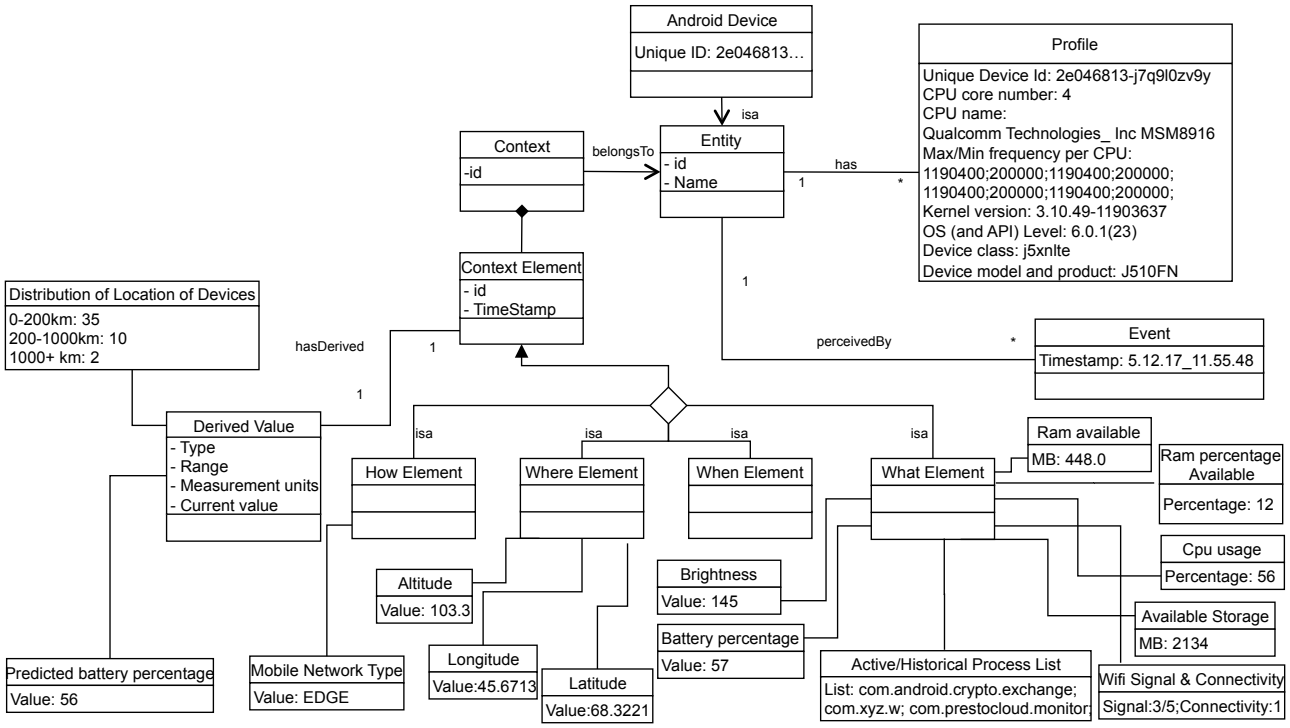

Figure 4. Context model instantiation

The architectural layout of the application is shown in Figure 5. The Main Activity retrieves information from data modules which is then published to the PrEstoCloud broker and displayed on the screen of the user.

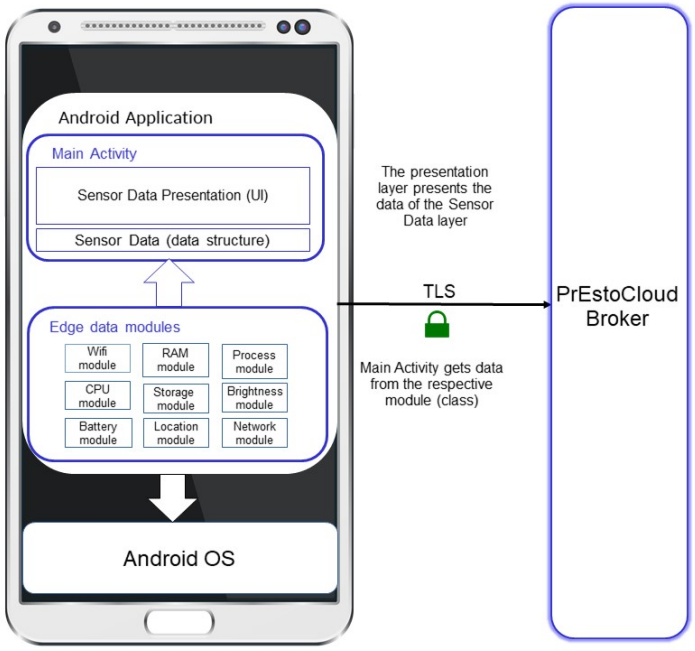

Figure 5. The internal architecture of the Android monitoring agent.

The application was deployed on five different Android devices, which published all data on a message Broker using the MQTT protocol. The monitoring data collected was representative of a variety of use-cases – from simple discharging with low user interaction to purposeful heavy usage. The application was configured with the required TLS certificates in order to communicate with the broker, which in turn logged all incoming data for further analysis. During a time period of over two months, between December 1st and

February 15th, more than 250 Megabytes of usable data have been captured, featuring several thousands of state records. The device types used are shown in **Table 7**:

Table 7. Specification of device types

Device Brand	Device Model
Samsung	J510FN
LG	H525n
Samsung	GT-I9301i
Samsung	GT-I9505
Samsung	A-520F

In the following sections we present the processing methods of the context data collected through the Android agent. Our objective is to demonstrate the utilization of the transmitted information, in order to predict the battery consumption over a specific time interval. This prediction is key to determine a proper recommendation on which processing tasks should be sent for execution on Android devices. Specifically, forthcoming WP5 components such as the PrEstoCloud Situation Detector and the Adaptation Recommender will exploit such inferred context in order to propose the execution of lighter tasks on devices which are low on battery, and the execution of more demanding tasks on devices which are adequately charged.

4.4.2 Online Processing

The implementation details of the online processing part of MCA is depicted in Figure 6. MCA consists of a RabbitMQ broker, two at least Logstash instances, a set of set of context inferencing workers implemented as NodeJS services, an Elasticsearch cluster and a Kibana instance. All the MCA components are implemented as Docker containers. Docker containers can be deployed and executed by an administrator very easily with tools such as Docker Compose. The events that are exchanged either between the other PrEstoCloud components and the MCA or between the internal components that comprise the MCA are handled by a RabbitMQ broker. Initially events containing raw context data from extreme edge devices (such as mobile phones, IP cameras, or drones) are published to RabbitMQ with the MQTT protocol. Then the rest of the MCA components communicate by exchanging events with RabbitMQ.

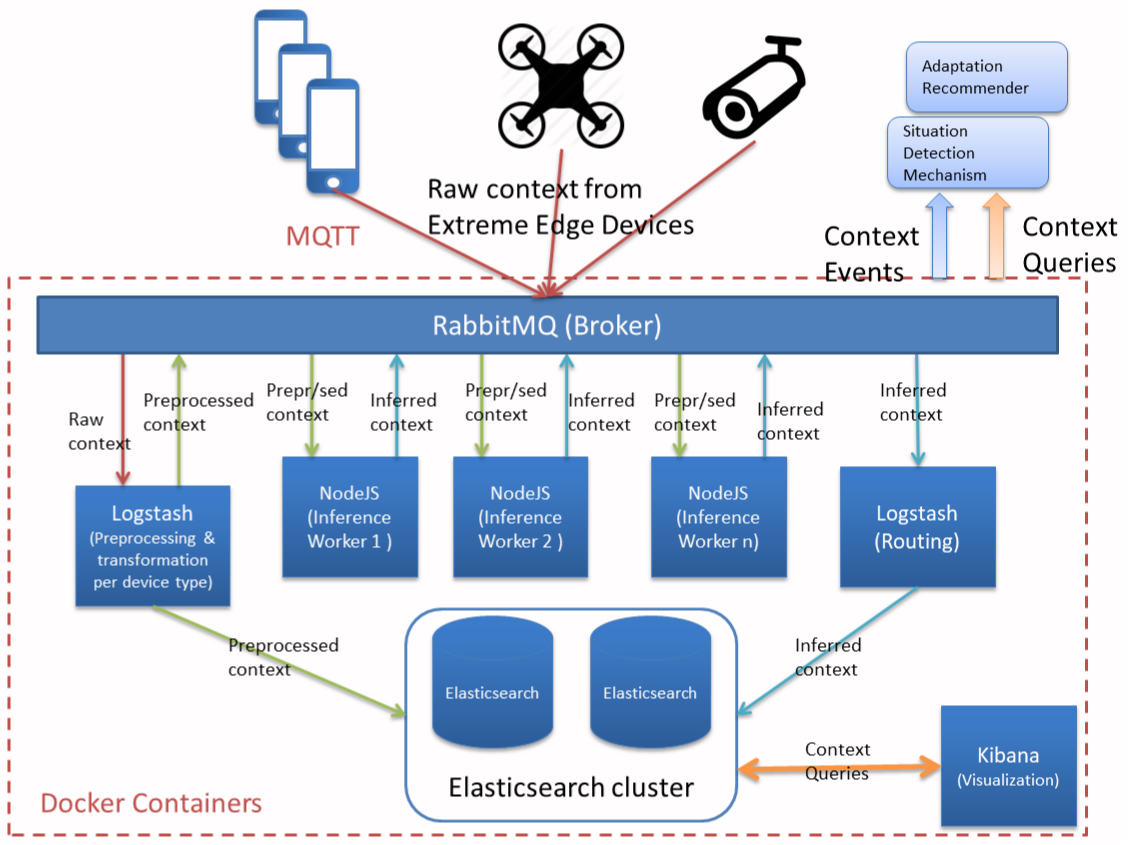

Figure 6. MCA online processing infrastructure

The internal components of MCA are communicating based on the AMQP protocol. Different messaging patterns are used in different stages of the MCA.

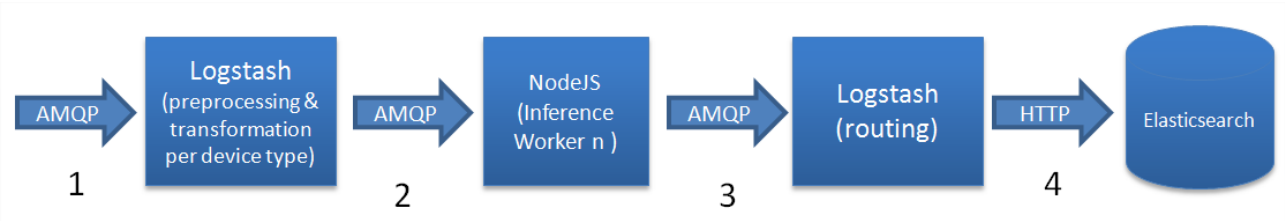

Figure 7. MCA messaging patterns

Data from edge-devices are transmitted by the Broker (RabbitMQ) over the AMQP protocol to the Logstash instance that is configured for to implement the pre-processing of the data (figure 6, point 1). Here we have used the publish/subscribe messaging pattern according to which the Broker delivers each incoming message to all subscribers.

The Logstash (pre-processing instance) then transmits messages to one or more inference workers by publishing them to the Broker (figure 6, point 2). Here we use a work queue with round-robin dispatching. This means that the Broker will deliver each message to exactly one worker. The workload will be distributed among workers.

In the final stage the inference workers publish the results to the Broker (figure 6, point 3). Then a separate Logstash instance is used to receive messages from the Broker and send them to Elasticsearch (over HTTP protocol) (figure 6, point 4. In point 3 we use a simple RabbitMQ queue. With this configuration the Broker (RabbitMQ) will store messages from the producers (inference workers) and deliver them to the consumer (routing Logstash instance).

The MCA microservices are implemented using docker containers. MCA can be executed with docker-compose. The structure of the docker-compose file is the following :

MCA Microservices / Docker-Compose file

- **Services**
  - **Elastic search cluster**
    - **Elasticsearch1, Elasticsearch2**
  - **Kibana**
  - **RabbitMQ**
  - **Logstash (pre-processing)**
  - **Logstash (routing)**
  - **Inference Workers**
- **Volumes**
  - **Esdata1, Esdata2**

Inference worker scaling with docker-compose

Multiple inference worker microservices (docker containers) can run simultaneously. The Broker (RabbitMQ) automatically load balances the workload (incoming pre-processed events) between all the started (online) inference services. By issuing an appropriate docker-compose command we can execute the MCA with multiple instances of the Docker-Compose service "inference_svc" as described in **Figure 8**.

```
inference_svc:
    image: "node:8"
    user: "node"
    working_dir: /home/node/app
    environment:
      - NODE_ENV=production
    volumes:
      - ./:/home/node/app
    expose:
      - "8081"
    command: "npm start"
```

Figure 8. Docker compose inference service definition

This docker-compose service is a Node.js service that implements the context inference functionalities of MCA. The definition in **Figure 8** tells docker-compose which OS image to use (image: "node:8") and defines the required execution parameters such as exposed ports (8081), volume mappings (. to /home/node/app) and start commands (npm start) in order to be integrated with the other MCA components.

By issuing the following command we can execute the MCA with two instances of the Docker-Compose service "inference_svc".

```
$ sudo docker-compose up --scale inference_svc=2
```

Figure 9 depicts the result of the above command. Input workload (events from mobile devices) are distributed in a round-robin manner between the two inference worker instances.

```
$ sudo docker-compose up --scale inference_svc=2
```

```
inference_svc_1  |  [x] from routing key : mobile.preproc.out
inference_svc_1  |      DEVICE_UUID:2e046813 , TS=2018-01-18T16:32:11.000Z , PROTVER:2
inference_svc_1  |      BAT_LVL_PCT:91
inference_svc_1  |  [x] Inferred context :     [{"model_id":1,"bat_state_30m":81.9}]
inference_svc_2  |  [x] from routing key : mobile.preproc.out
inference_svc_2  |      DEVICE_UUID:2e046813 , TS=2018-01-18T16:32:21.000Z , PROTVER:2
inference_svc_2  |      BAT_LVL_PCT:91
inference_svc_2  |  [x] Inferred context :     [{"model_id":1,"bat_state_30m":81.9}]
inference_svc_1  |  [x] from routing key : mobile.preproc.out
inference_svc_1  |      DEVICE_UUID:2e046813 , TS=2018-01-18T16:32:31.000Z , PROTVER:2
inference_svc_1  |      BAT_LVL_PCT:91
inference_svc_1  |  [x] Inferred context :     [{"model_id":1,"bat_state_30m":81.9}]
inference_svc_2  |  [x] from routing key : mobile.preproc.out
inference_svc_2  |      DEVICE_UUID:dacc3411 , TS=2018-03-05T13:17:19.115Z , PROTVER:2
inference_svc_2  |      BAT_LVL_PCT:99
inference_svc_2  |  [x] Inferred context :     [{"model_id":1,"bat_state_30m":89.10000000000001}]
inference_svc_1  |  [x] from routing key : mobile.preproc.out
inference_svc_1  |      DEVICE_UUID:dacc3411 , TS=2018-03-05T13:17:19.615Z , PROTVER:2
```

Figure 9. Online processing running with 2 inference workers

We can scale up the MCA by telling docker-compose to execute more instances of the inference_svc. For example with the following command we can start four inference worker instances:

```
$ sudo docker-compose up --scale inference_svc=4
```

Figure 9 depicts the result of the above command. Input workload (events from mobile devices) are now distributed in a round-robin manner between the four inference worker instances.

```
inference_svc_2  |  [x] from routing key : mobile.preproc.out
inference_svc_2  |      DEVICE_UUID:2e046813 , TS=2018-01-18T11:31:11.000Z , PROTVER:2
inference_svc_2  |      BAT_LVL_PCT:93
inference_svc_2  |  [x] Inferred context :     [{"model_id":1,"bat_state_30m":83.7}]
inference_svc_1  |  [x] from routing key : mobile.preproc.out
inference_svc_1  |      DEVICE_UUID:2e046813 , TS=2018-01-18T11:46:36.000Z , PROTVER:2
inference_svc_1  |      BAT_LVL_PCT:93
inference_svc_1  |  [x] Inferred context :     [{"model_id":1,"bat_state_30m":83.7}]
inference_svc_3  |  [x] from routing key : mobile.preproc.out
inference_svc_3  |      DEVICE_UUID:2e046813 , TS=2018-01-18T12:25:34.000Z , PROTVER:2
inference_svc_3  |      BAT_LVL_PCT:93
inference_svc_3  |  [x] Inferred context :     [{"model_id":1,"bat_state_30m":83.7}]
inference_svc_4  |  [x] from routing key : mobile.preproc.out
inference_svc_4  |      DEVICE_UUID:2e046813 , TS=2018-01-18T12:27:00.000Z , PROTVER:2
inference_svc_4  |      BAT_LVL_PCT:93
inference_svc_4  |  [x] Inferred context :     [{"model_id":1,"bat_state_30m":83.7}]
inference_svc_2  |  [x] from routing key : mobile.preproc.out
inference_svc_2  |      DEVICE_UUID:2e046813 , TS=2018-01-18T12:45:07.000Z , PROTVER:2
inference_svc_2  |      BAT_LVL_PCT:92
inference_svc_2  |  [x] Inferred context :     [{"model_id":1,"bat_state_30m":82.8}]
inference_svc_1  |  [x] from routing key : mobile.preproc.out
inference_svc_1  |      DEVICE_UUID:2e046813 , TS=2018-01-18T13:26:51.000Z , PROTVER:2
inference_svc_1  |      BAT_LVL_PCT:92
inference_svc_1  |  [x] Inferred context :     [{"model_id":1,"bat_state_30m":82.8}]
```

Figure 10. Online processing running with 4 inference workers

Logstash Pre-processing functionalities

Logstash is used in the pre-processing stage. By defining appropriate configuration files (logstash pipelines) logstash can perform event format transformation and enrichment. The following figures depict how Logstash can be used for the pre-processing of events that are published by the Android mobile devices with the application that we designed and implemented.

By configuring a filter we can tell logstash to convert events in CSV format to JSON (Figure 10). The parameter "columns" assigns names to fields. The parameter "convert" can be used in order to validate an

input value according to the declared datatype and give logstash directions in order to encode it appropriately in a JSON object.

```
csv {
  separator => ","
  columns =>
["bright_lvl","ram_pct","store_mb","lat","lon","alt","wifi_state","plist","net_type","bat_state","cpu_load_pct","ts"]
  convert => {
                "lat" => "float"
                "lon" => "float"
                "bright_lvl" => "integer"
                … more mappings follow …
  }
}
```

Figure 11. Read a comma-separated event and map it to json field names

In Figure 11 we can see how we can parse a datetime string with Logstash and convert it to a timestamp.

```
date {
        match => [ "ts", "dd.MM.yy_HH.mm.ss"]
}
```

Figure 12. Parse a date/time string and convert it to an event timestamp

In Figure 12 we show how a list separated by ";" character can be converted to a JSON array with Logstash,

```
split => {
                "plist" => ";"
}
```

Figure 13. Split a list separated by ";"

In Figure 13 we show how Logstash can be configured to normalize sensor values in order to enter them in a machine learning algorithm. In the specific example we normalize the values of mobile phone brightness levels and map them from the range (0,255) to the range (0,1.0) by multiplying them with 0.0039215686274.

```
ruby {
        code => 'event.set("bright_lvl_nr", event.get("bright_lvl").to_i * 0.0039215686274 )'
}
```

Figure 14. Normalize a value (map a value from 0 to 255 to the range: 0.0 - 1.0)

Elasticsearch context aggregation queries

Elasticsearch can be used in order to produce aggregated context. It supports four categories of aggregations (Metrics, Bucket, Pipeline and Matrix). Each category contains many types of aggregations. Different aggregations can be combined. In (figure 14) we can see an example of a geographical aggregation that can be used for context inference. With this query we can see how many mobile phones operate in specific distance ranges from a location (for example the location of an edge router).

QUERY:

```
POST /mobile*/_search?size=0
{
    "aggs" : {
        "devices_around_athens" : {
            "geo_distance" : {
                "field" : "mobile_geoip.location",
                "origin" : "37.983810, 23.727539",
                "unit" : "km",
                "ranges" : [
                    { "to" : 200 },
                    { "from" : 200, "to" : 1000 },
                    { "from" : 1000 }
                ]
            }
        }
    }
}
```

RESULTS :

```
"aggregations": {
    "devices_around_athens": {
      "buckets": [
        {
          "key": "*-200.0",
          "from": 0,
          "to": 200,
          "doc_count": 52802
        },
        {
          "key": "200.0-1000.0",
          "from": 200,
          "to": 1000,
          "doc_count": 930
        },
        {
          "key": "1000.0-*",
          "from": 1000,
          "doc_count": 257962
```

Figure 15. Count mobile phones around a location and get results per distance range (0 to 200km, 200km to 1000km, over 1000km)

The example context aggregation query depicted in (Figure 15) can be used to find the hour of the day that the average free ram of mobile devices is the maximum or the minimum.

QUERY:

```
POST /mobile*/_search?size=0
{
    "size": 0,
    "aggs" : {
        "ram_hour" : {
            "date_histogram" : {
                "field" : "@timestamp",
                "interval" : "hour"
            },
            "aggs": {
                "fram": {
                    "avg": {
                        "field": "ram"
                    }
                }
            }
        },
        "stats_hr_ram": {
            "stats_bucket": {
                "buckets_path": "ram_hour>fram"
            }
        }
    }
}
```

RESULTS :

```
"aggregations": {
    "ram_hour": {
      "buckets": [
        {
          "key_as_string": "2018-03-
16T00:00:00.000Z",
          "key": 1521158400000,
          "doc_count": 2108,
          "fram": {
            "value": 27787.723748773307
          }
        },
…more results here…
      ]
    },
    "stats_hr_ram": {
      "count": 15,
      "min": 20198.5146539548,
      "max": 56876.050105857445,
      "avg": 37195.654750736125,
      "sum": 557934.8212610419
    }
```

Figure 16. Find the average ram of all devices per hour and get statistics about it (min, max, average, sum)

Kibana

Kibana (https://www.elastic.co/products/kibana) is a visualization tool for Elasticsearch. With it an analyst can examine events from edge devices and build custom context aggregation queries and visualizations.

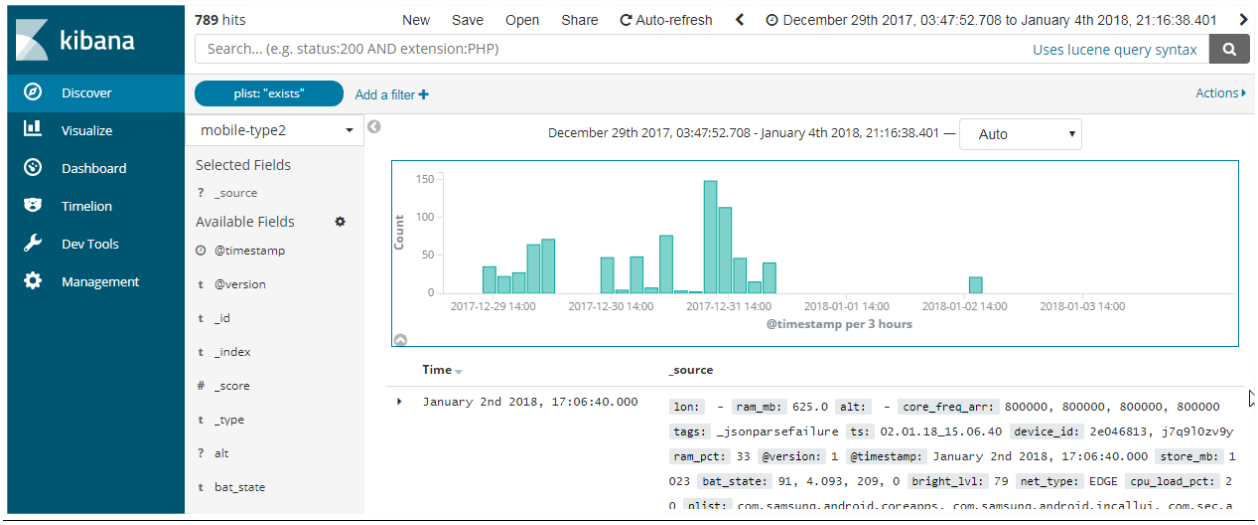

Figure 17. Discovery for mobile phone events during a specific time period (29/12/'17 to 4/1/'18)

With Kibana the analyst can build many types of visualizations. In (Figure 17) we can see a visualization of the number of mobile devices per location with a heatmap diagram.

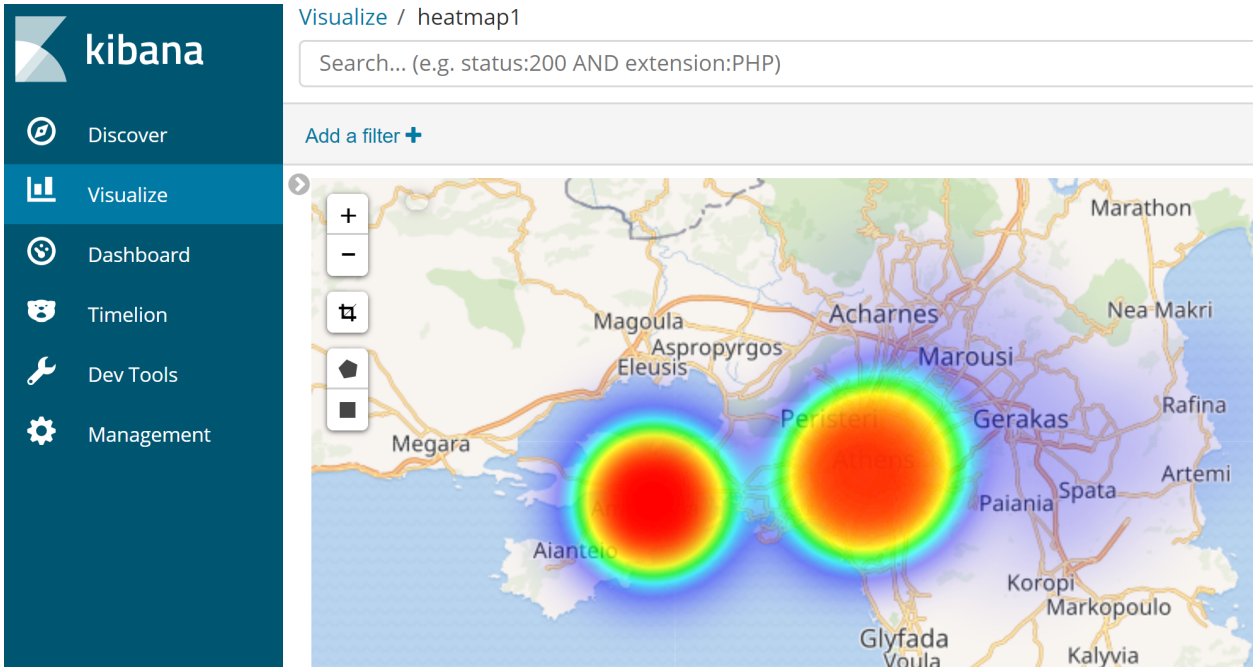

Figure 18. Heatmap Kibana visualization depicts the number of mobile devices at specific locations

4.4.3 Batch Processing

MCA context inference in batch mode refers to the supervised learning of neural networks that predict context parameters based on historical context data. This functionality is designed in order to satisfy the need for non-deterministic context inference as documented in Requirement 4 (Produce inferred context by combining and analysing raw information, deterministically or not). Data-driven machine-learning methods can be used in order to build models that are capable to predict context parameters (such as the future battery capacity of a mobile device that has specific hardware characteristics and state). As we will show in this chapter, such methods can be applied to devices never seen before by MCA (as long as they

have similar characteristics with devices already analyzed). The output of these models are predicted context parameter values which we expect to have an error bellow a threshold.

Input dataset

Building on the illustrative example mentioned above, we will present the way that the Batch Processing layer of MCA operates. Specifically, we show how the machine learning model is created in order to be able to predict the battery of a specific mobile device based on four features: Memory utilization (ram), Battery capacity (bat), Screen brightness (bright) and the hour of the day.

The field real_next_bat is pre-computed from historical data and corresponds to the battery capacity 30 minutes after the time of the measurement.

The raw data from the mobile device are resampled and interpolated with the Pandas Python library (McKinney, 2018) in order to generate four uniform time series with time-steps that correspond to one minute.

The dataset depicted in (Figure 18) contains data from one mobile phone and covers a time period from 1-Dec-2017 to 19-Jan-2018.

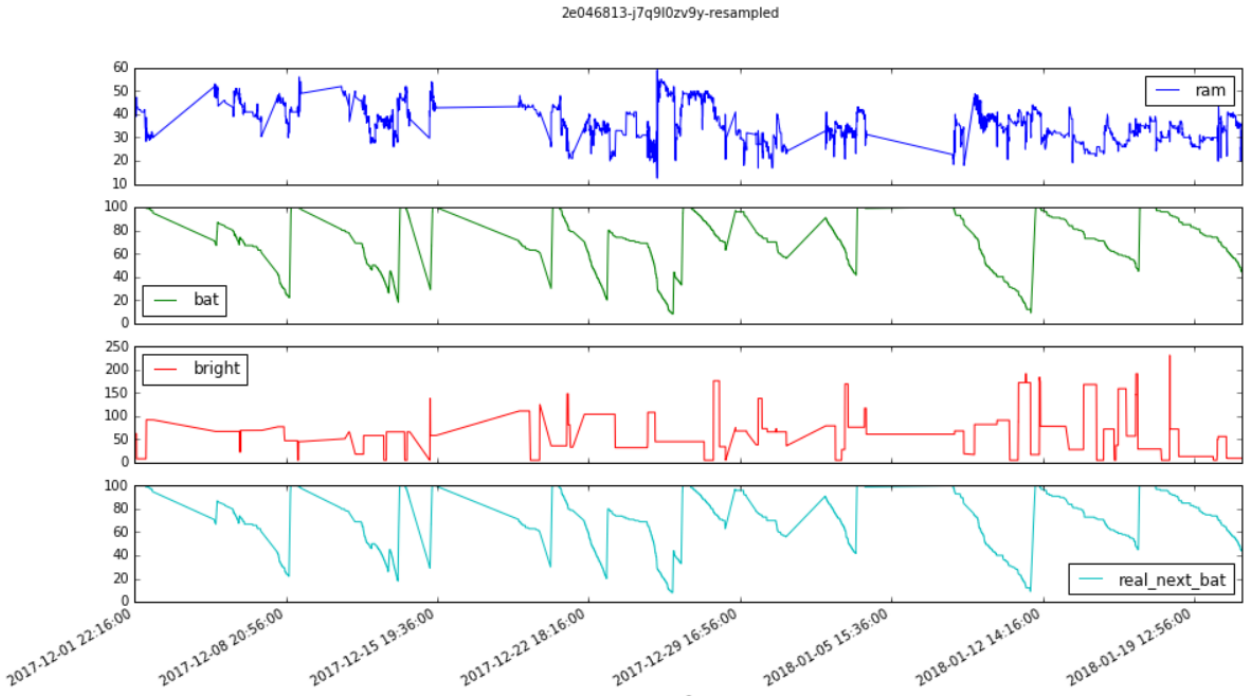

Figure 19. Input dataset

Training Dataset (subset 16 to 21 – Jan – 2018)

With the historical data from this mobile phone we will attempt to build with different methods machine learning models (neural networks) capable to predict the future battery capacity of a mobile phone. As training dataset for supervised learning we used a subset of the above dataset covering the time period from 16-Jan-2018 to 21-Jan-2018.

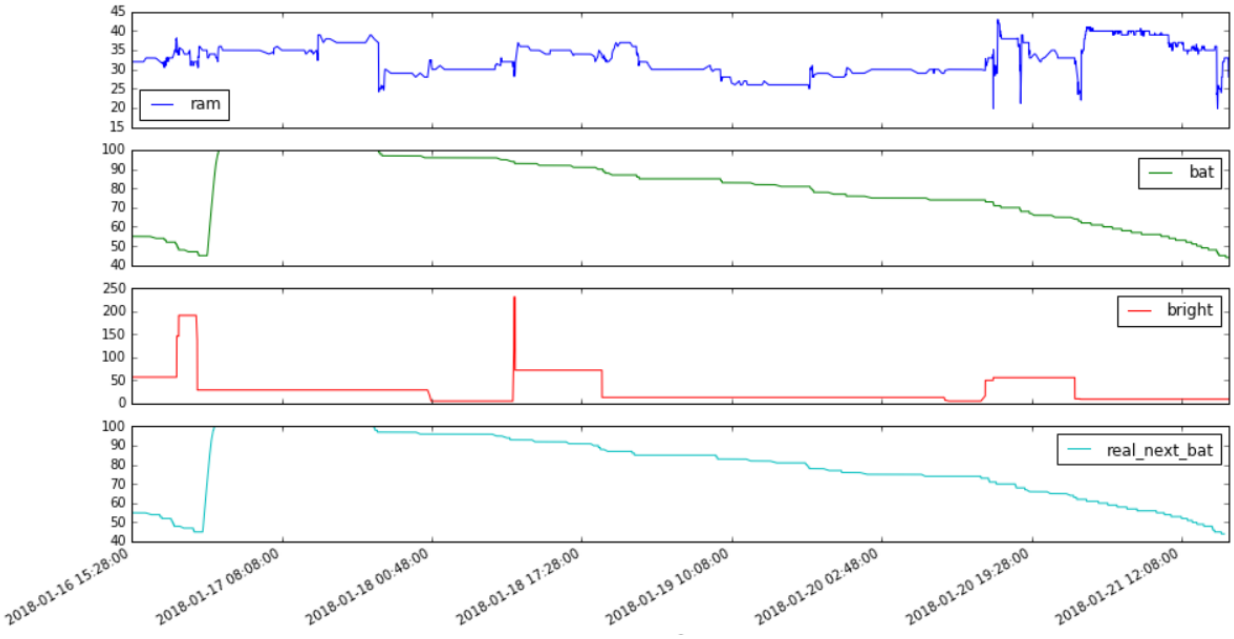

Figure 20.Training dataset

All the developed neural networks have four inputs and one output. The mapping that we will use is the following:

- Input 0 : Normalized hour of day (hour/24.0)
- Input 1 : Normalized percent of ram (ram/100.0)
- Input 2 : Normalized screen brightness level (bright/255.0)
- Input 3 : Normalized percentage of battery (bat/100.0)
- Output 0 : Normalized percentage of battery after 30 minutes (real_next_bat/100.0)

Perceptron trials

With the help of the Neataptic library (https://wagenaartje.github.io/neataptic/docs/) we have built and trained many types of multi-layer Perceptron (Rosenblatt, 1958) neural networks by using the error backpropagation algorithm (Hecht-Nielsen, 1992), see Table 8.

Table 8. Description of tested Perceptron Neural Networks

Neural Network Type	Number of input layer neurons	Neurons per hidden layer	Number of output layer neurons	Cost function used
Perceptron(4, 10, 10, 10, 10, 1)	4	10, 10, 10, 10	1	Mean Squared Error (MSE)
Perceptron(4, 40, 4, 40, 4, 1)	4	40, 4, 40, 4	1	Mean Squared Error (MSE)
Perceptron(4, 40, 20, 1)	4	40, 20	1	Mean Squared Error (MSE)
Perceptron(4, 4, 1)	4	4	1	Mean Squared Error (MSE), Mean Absolute Error (MAE)
Perceptron(4, 1, 1)	4	1	1	Mean Squared Error (MSE)
Perceptron(4, 16, 16, 16, 4, 1)	4	16, 16, 16, 4	1	Mean Squared Error (MSE)
Perceptron(4, 4, 4, 4, 4, 1)	4	4, 4, 4, 4	1	Mean Squared Error (MSE)
Perceptron(4, 40, 40, 40 , 40, 1)	4	40, 40, 40, 40	1	Mean Squared Error (MSE)

The best results were achieved by the Perceptron(4,4,1) neural network with Mean Squared Error cost function.

Perceptron (4.4.1) training results

If we try to train with backpropagation a simple perceptron neural network with four inputs, 1 hidden layer with four nodes and one output layer we get results as those that we show in figure 20. The green line corresponds to the future (after 30 minutes) battery capacity that was predicted by the trained neural network. The blue line corresponds to the actual future battery capacity. We can clearly see that while this model follows some of the patterns of the battery capacity time series it is not very close to the reality.

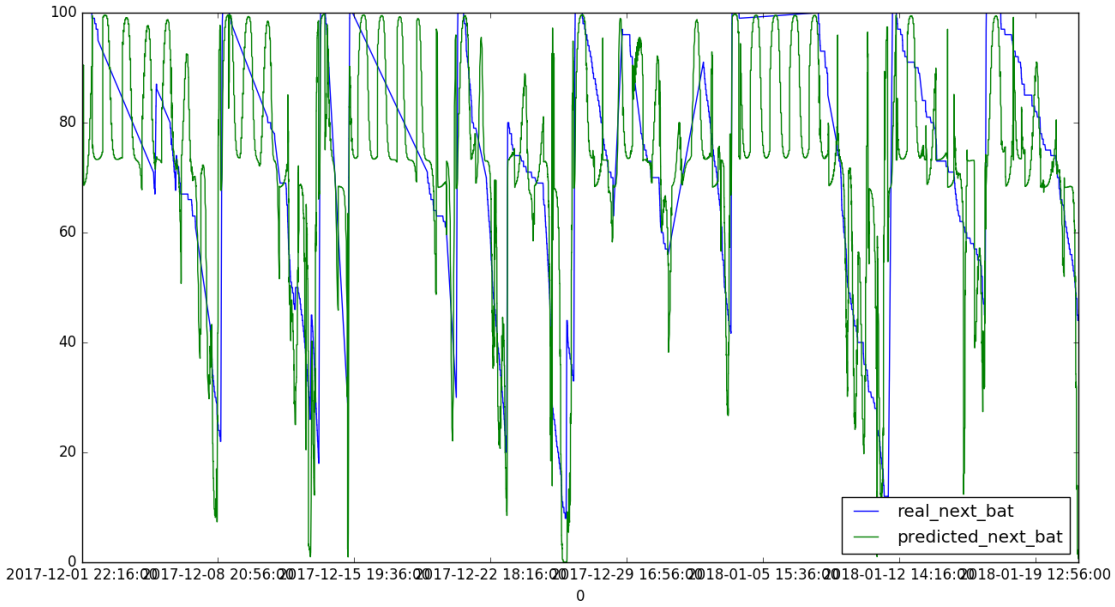

Figure 21. Perceptron (4,4,1) training results

After 1000 iterations the training algorithm converged to a neural network with error 0.0001292101064029624 (Figure 21).

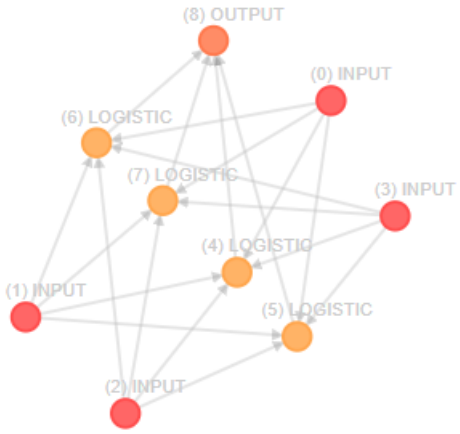

Figure 22. Perceptron (4,4,1) neural network

LSTM trials

LSTM (Long Short-Term Memory) [23] neural networks are better at predicting patterns in time series data with unknown time lags. Several LSTM units connected form a recurrent neural network (RNN). With the help of the Neataptic library we have built and trained many types of multi-layer LSTM neural networks by using the error backpropagation algorithm (Hecht-Nielsen, 1992), see Table 9.

Table 9. Description of tested LSTM Neural Networks

Neural Network Type	Number of input layer	Memory cells per memory block	Number of output layer	Cost function used

	neurons	assembly	neurons	
LSTM(4, 8, 8, 8, 8, 1)	4	8, 8, 8, 8	1	MSE
LSTM(4, 4, 1)	4	4	1	MSE
LSTM(4, 16, 1)	4	16	1	MSE, MAE
LSTM(4, 1, 1)	4	1	1	MSE, MAE
LSTM(4, 4, 16, 4, 1)	4	4, 16, 4	1	MSE

The best results were achieved by the LSTM (4,4,1) neural network.

LSTM (4,4,1) training results

A trained LSTM network with four inputs, one memory block with 4 memory assemblies and an output node while it is much more complex than the perceptron network does not achieve better results than the latter.

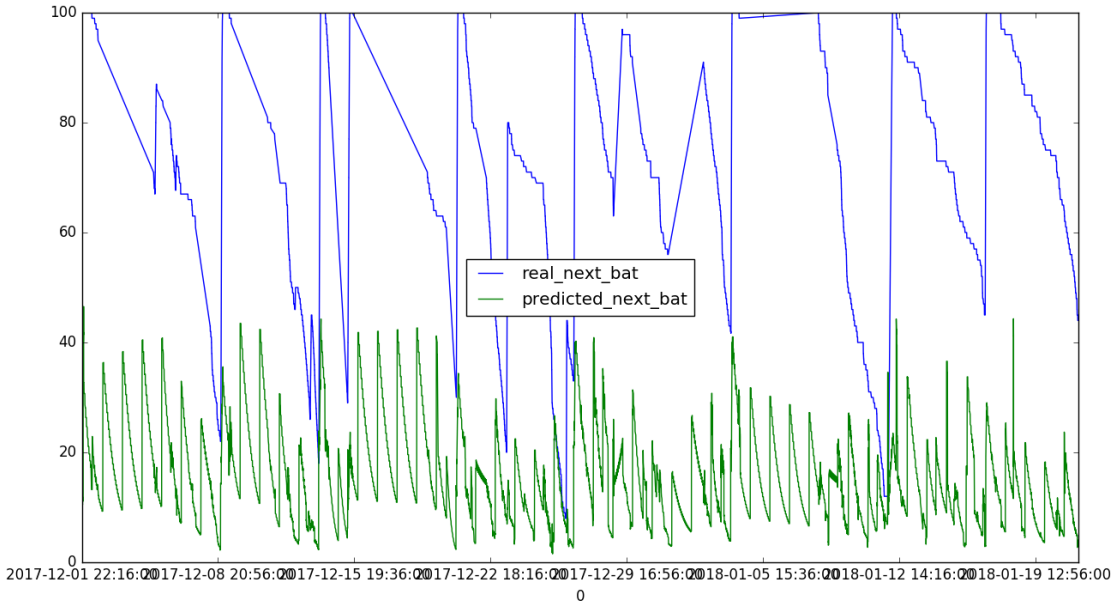

Figure 23. LSTM (4,4,1) training results

After 180 iterations the training algorithm converged to a neural network with error .00012184095632515447 (Figure 23).

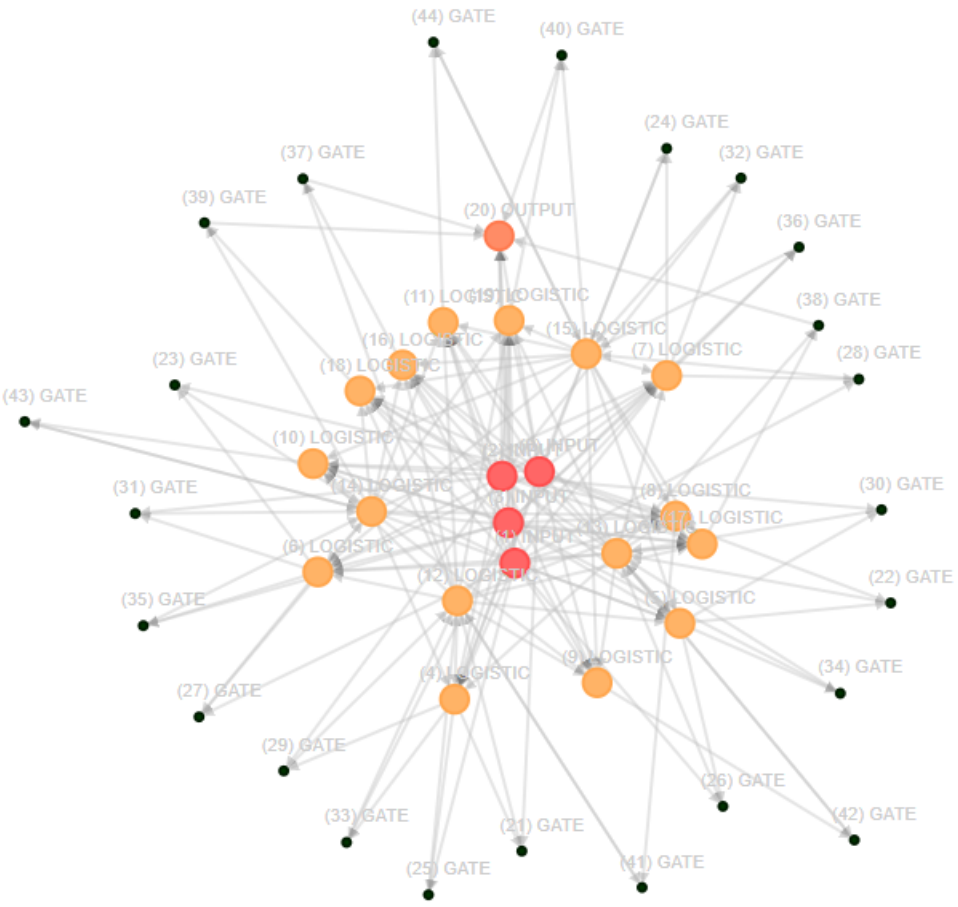

Figure 24. LSTM (4,4,1) neural network

NARX trials

NARX (Nonlinear AutoRegressive with eXogenous inputs) (Billings, 2013) neural networks have been also applied for time series prediction. We have trained the following types of NARX neural networks by using the error backpropagation algorithm (Table 10).

Table 10. Description of tested NARX Neural Networks

Neural Network Type	Number of input layer neurons	Number of nodes per hidden layer	Number of output layer neurons	Number of previous inputs that the NN remembers	Number of previous outputs that the NN remembers	Cost function used
NARX(4, 5, 1, 1, 5)	4	5	1	**1**	**5**	**MSE**
NARX(4, 5, 1, 3, 3)	4	**5**	1	**3**	**3**	**MSE**
NARX(4, [10,20,10], 1, 30, 30)	4	**10, 20, 10**	1	**30**	**30**	**MSE, MAE**
NARX(4, 10, 1, 10, 5)	4	**10**	1	**10**	**5**	**MSE**
NARX(4, 100, 1, 10, 5)	4	**100**	1	**10**	**5**	**MSE**

The best results were achieved by the NARX(4, 5, 1, 1, 5) neural network.

NARX 4,5,1,1,5 training results

While the following NARX neural network predicts better than the LSTM it is not as good as the Perceptron.

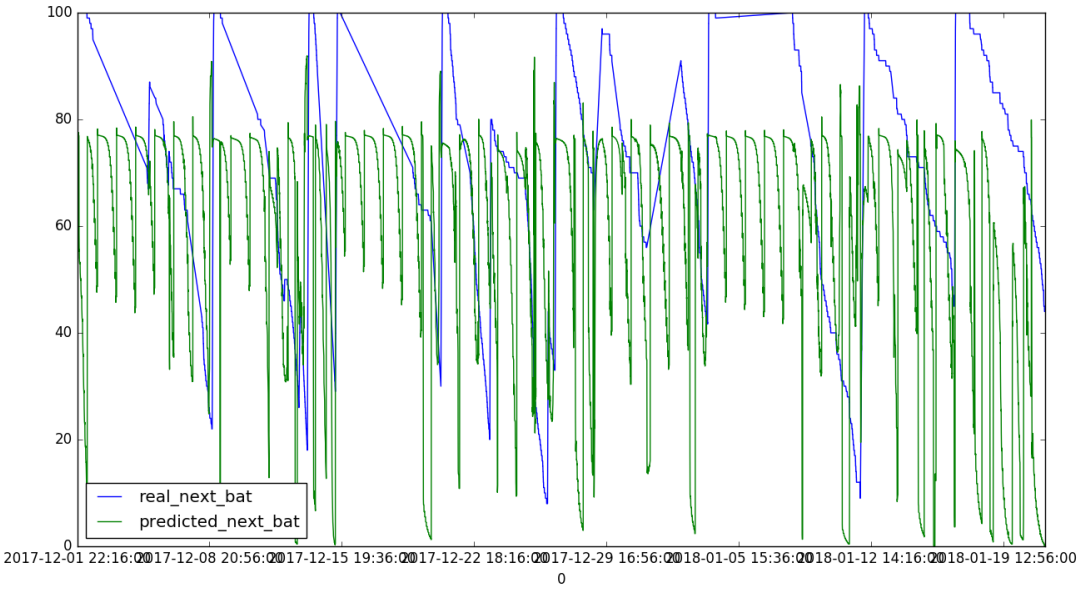

Figure 25. NARX (4,5,1,1,5) training results

After 139 iterations which lasted 3797 seconds the training algorithm converged to a neural network with error 0.00009992158278938505 (Figure 25).

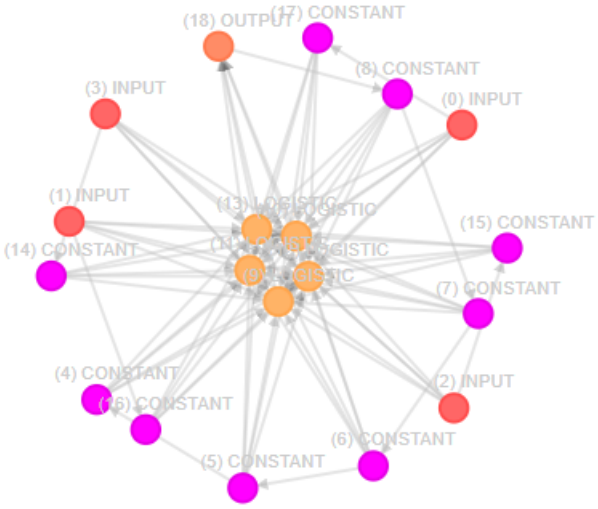

Figure 26. NARX 4,5,1,1,5 neural network

Conclusion from manually building neural networks

By using software libraries we can continue the process described above and build quickly many kinds of neural networks by adding more nodes and layers. But many times, as in the use case that we describe here, this trial and error process does not always provide better results. There is a need for an automated process that evaluates different types of neural networks until it finds the one that fits best to a given (context-inferencing) problem. There are already methods such as genetic algorithms (Auger & Doerr, 2011) that are good at finding optimum solutions in large search spaces.

Incremental Neuroevolution

By utilizing neuro-evolution we can automatically generate and evaluate neural networks based on historical data. Neuro-evolution starts from a random neural-network and then with a genetic algorithm tries to find an optimum solution (a neural network that minimizes a cost function) with a process that involves creation of offsprings and mutations (Sher,2012). With the help of the Neataptic libary we can incrementally learn neural networks with neuro-evolution. We can stop the algorithm when it achieves a specific error level or after it completes a specific number of iterations. In the subsequent execution of the algorithm the neuroevolution process can be initiated from a previous learned neural network (instead of starting from a random network). The ability to incrementally learn (better) neural networks is very useful because you can find an initial solution and then improve it by utilizing more computation resources (processing time) or more data (bigger in volume or more specific to a device or a time period).

In the following paragraphs we will demonstrate this incremental process. Figure 26 encapsulates the prediction results of a model (neural network) built with neuro-evolution after 25 iterations of the process. It depicts the predicted and the real values for the time period that was excluded from the training dataset. The results are already better than those that were produced by manually constructed NNs.

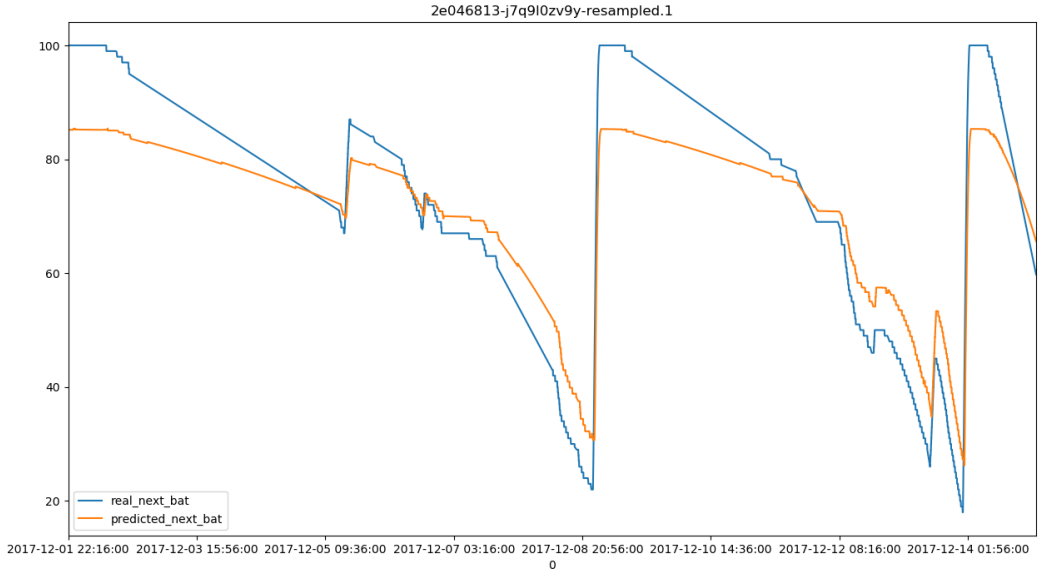

Figure 27. Results after 25 neuro-evolution iterations

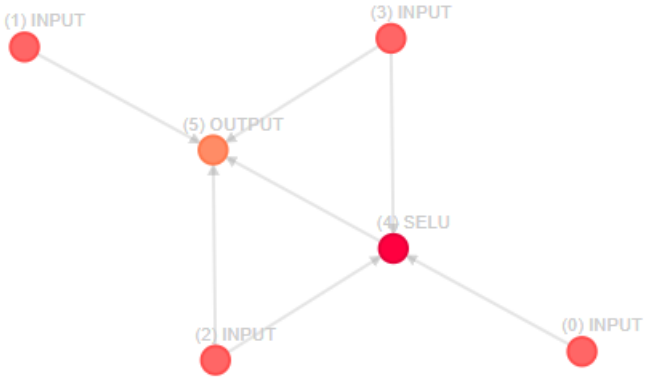

Figure 28. Neural network selected by the genetic algorithm after 25 iterations

If we allow the neuro-evolution algorithm to perform additional iterations (providing more computation time) we see that it converges to even better results. Figures 28,29,31 depict the results after 50,250 and 500 iterations. We have found that after 250 iterations there is very little difference and the neuroevolution algorithm from this point starts to converge very slowly (for the specific dataset).

Figure 29. Results after 50 neuro-evolution iterations

Figure 30. Results after 250 neuro-evolution iterations

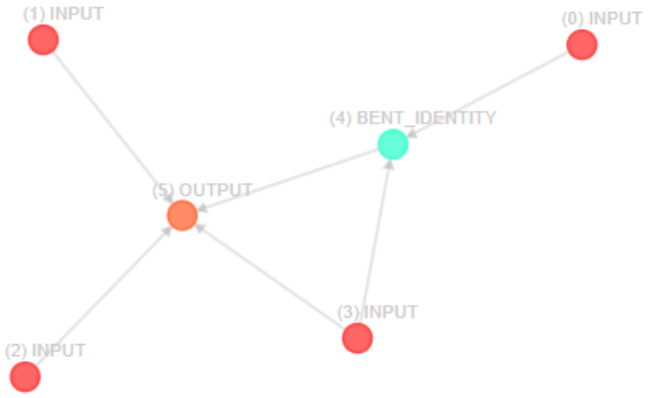

Figure 31. Neural network selected by the genetic algorithm after 250 iterations

Figure 32. Results after 500 neuro-evolution iterations

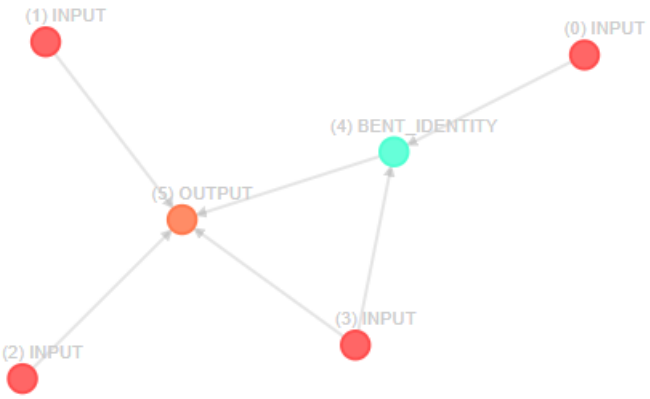

Figure 33. Neural network selected by the genetic algorithm after 500 iterations

Prediction of different phones with model built by neuro-evolution

In the following figures (33, 34, 35, 36, 37) we present how capable is the neural network learned by historical data from one mobile phone (the one presented previously) to predict the battery percentage in the next 30 minutes of five different mobile phones.

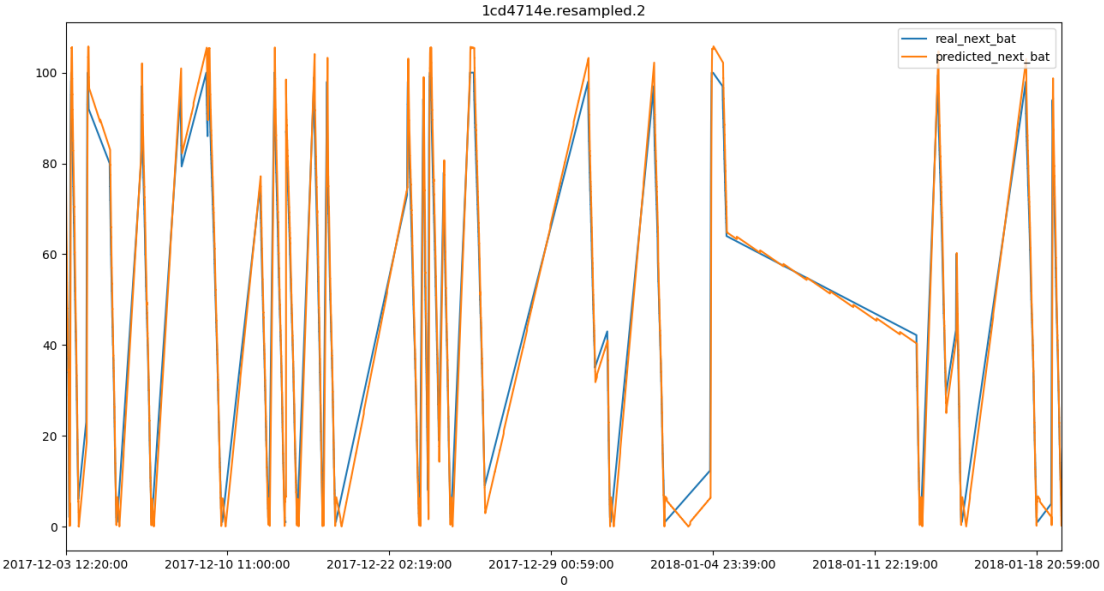

Figure 34. Battery of mobile "1cd471e" predicted by a model trained with data from "2e046813"

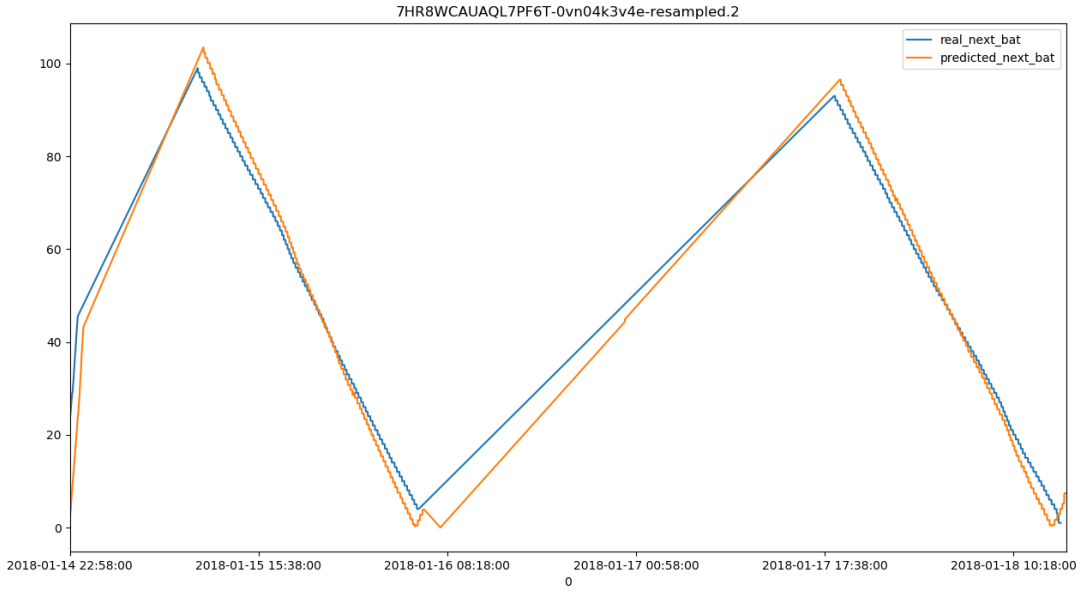

Figure 35. Battery of mobile "7HR8WC.." predicted by a model trained with data from "2e046813"

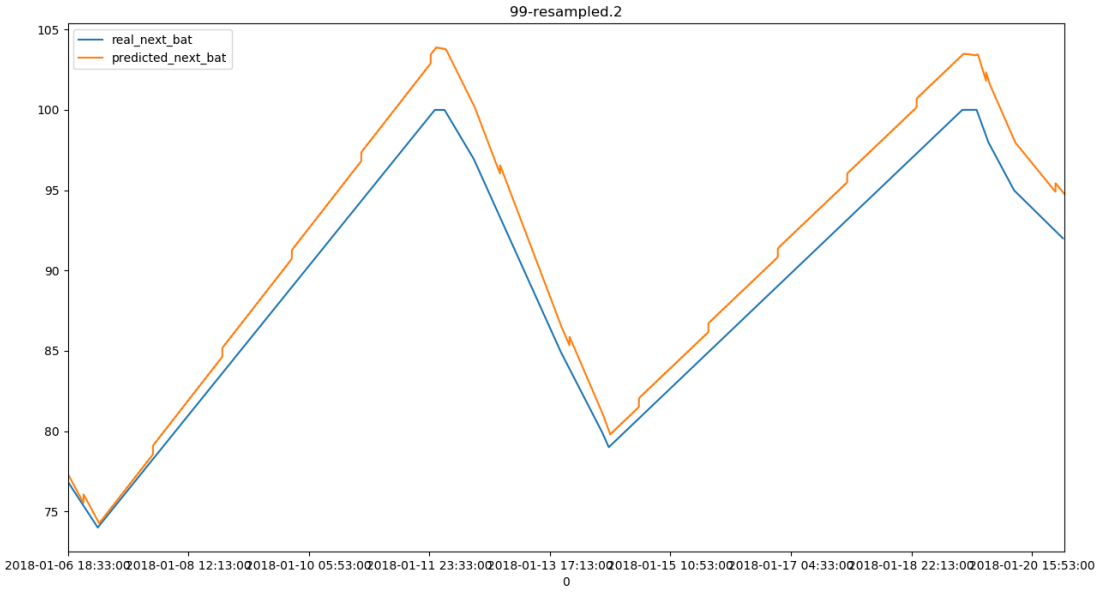

Figure 36. Battery of mobile "99…" predicted by a model trained with data from "2e046813"

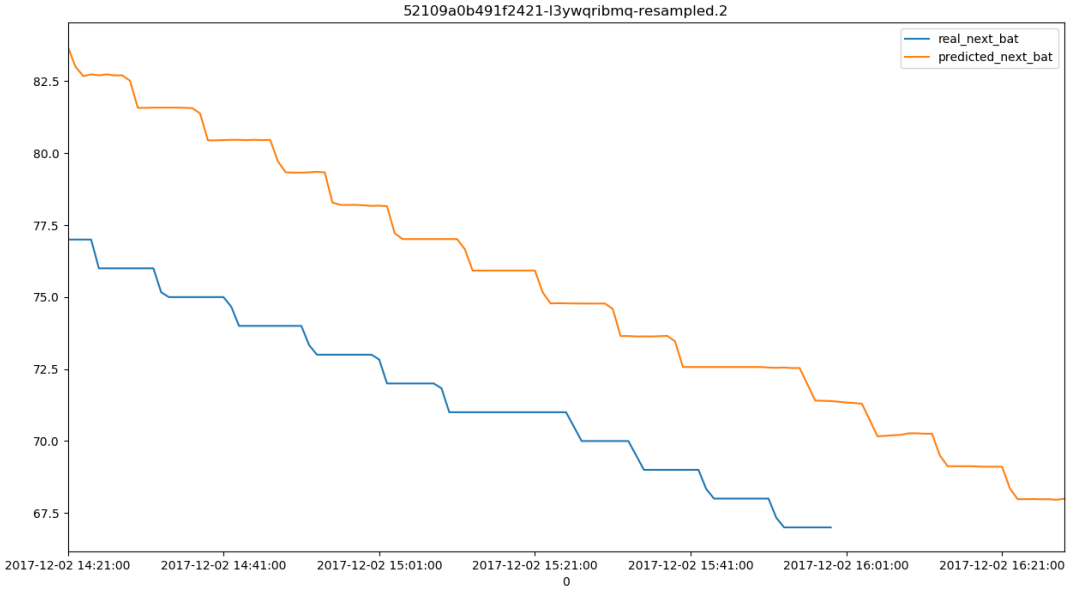

Figure 37. Battery of mobile "52109a0.." predicted by a model trained with data from "2e046813"

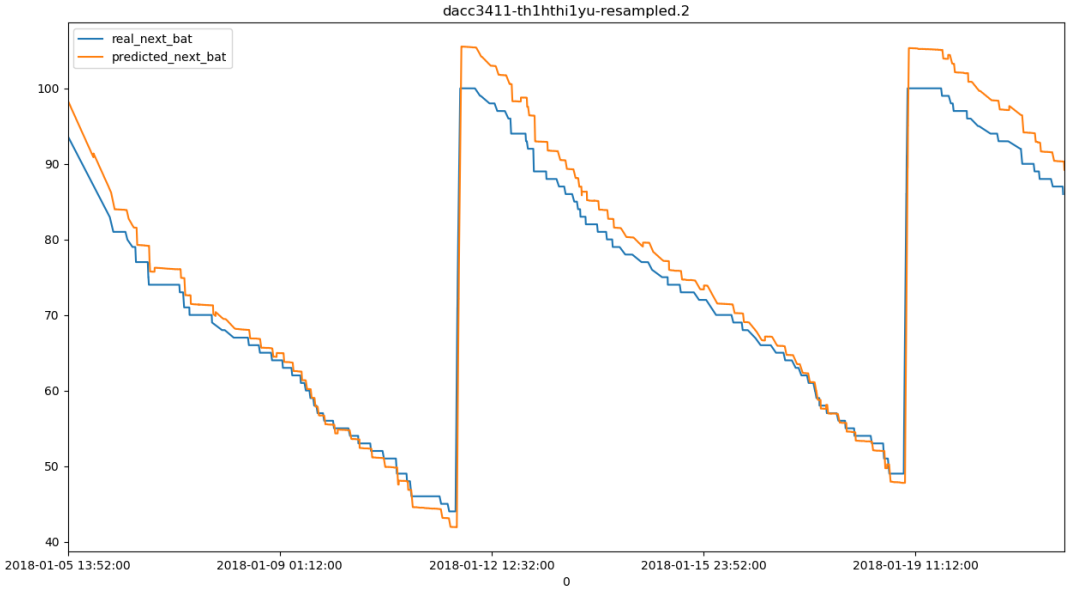

Figure 38. Battery of mobile "dacc3411" predicted by a model trained with data from "2e046813"

5. Conclusions

This deliverable presented the first version of the Mobile Context Analyzer component, which allows the detection and enhancement of context gathered from devices at the extreme edge of the network. We described our approach which factors in the current State-of-the-Art and builds upon reputable programming frameworks and solutions. The Mobile Context Analyser makes use of and implements a Context Model, which can accurately represent the Context gathered and processed.

We designed the component so as it is modular and can be easily deployed and used. An advantage of this approach is that it is possible to improve, extend and reuse it to acquire and monitor the context of a wide range of computing infrastructure types. We also demonstrated a real-world scenario indicative of the usage of our component, and its capabilities.

Our next objective will be the completion of the Situation Detection Mechanism (part of Task T5.1 "Situation Awareness at the Extreme Edge of the Network"), which will produce and handle information from the whole processing topology and provide a complete view of it. This will enable us afterwards to focus on the adaptation and reconfiguration mechanisms of the processing topology so that its performance requirements can be satisfied.

6. References

Abubakar, Y., Adeyi, T. S., & Auta, I. G. (2014). Performance evaluation of NoSQL systems using YCSB in a resource austere environment. *Performance Evaluation*, *7*(8), 23-27.

Akherfi, K., Gerndt, M., & Harroud, H. (2016). Mobile cloud computing for computation offloading: Issues and challenges. *Applied Computing and Informatics*.

Andreassen, O., De Dios Fuente, A., & Charrondière, C. (2015). Monitoring Mixed-Language Applications with Elastic Search, Logstash and Kibana (ELK).

Ashbrook, D., & Starner, T. (2003). Using GPS to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous computing*, *7*(5), 275-286.

Auger, A., & Doerr, B. (2011). *Theory of randomized search heuristics: Foundations and recent developments* (Vol. 1). World Scientific.

Bhattacharya, A., & Das, S. K. (2002). LeZi-update: An information-theoretic framework for personal mobility tracking in PCS networks. *Wireless Networks*, *8*(2-3), 121-135.

Billings, S. A. (2013). Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains. John Wiley & Sons.

Bramble, B., & Swift, M. (2014). Predicting Power Usage of Android Applications.
Tillenius, M., Larsson, E., Badia, R. M., & Martorell, X. (2015). Resource-aware task scheduling. *ACM Transactions on Embedded Computing Systems (TECS)*, *14*(1), 5.

Cho, E., Myers, S. A., & Leskovec, J. (2011, August). Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 1082-1090). ACM.

Chon, Y., Lane, N. D., Li, F., Cha, H., & Zhao, F. (2012, September). Automatically characterizing places with opportunistic crowdsensing using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (pp. 481-490). ACM.

Chon, Y., Talipov, E., Shin, H., & Cha, H. (2014). SmartDC: Mobility prediction-based adaptive duty cycling for everyday location monitoring. *IEEE Transactions on Mobile Computing*, *13*(3), 512-525.

Choudhury, T., & Pentland, A. (2003, October). Sensing and modeling human networks using the sociometer. In *null* (p. 216). IEEE.

Compton, M., Barnaghi, P., Bermudez, L., GarcíA-Castro, R., Corcho, O., Cox, S., ... & Huang, V. (2012). The SSN ontology of the W3C semantic sensor network incubator group. *Web semantics: science, services and agents on the World Wide Web*, *17*, 25-32.

Cook, D. J., Youngblood, M., Heierman, E. O., Gopalratnam, K., Rao, S., Litvin, A., & Khawaja, F. (2003, March). MavHome: An agent-based smart home. In *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on* (pp. 521-524). IEEE.

De Domenico, M., Lima, A., & Musolesi, M. (2013). Interdependence and predictability of human mobility and social interactions. *Pervasive and Mobile Computing*, *9*(6), 798-807.

Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing*, *5*(1), 4-7.

Eagle, N., Clauset, A., & Quinn, J. A. (2009). Location Segmentation, Inference and Prediction for Anticipatory Computing. In *AAAI Spring Symposium: Technosocial Predictive Analytics* (pp. 20-25).

Eagle, N., & Pentland, A. S. (2009). Eigenbehaviors: Identifying structure in routine. *Behavioral Ecology and Sociobiology*, *63*(7), 1057-1066.

Etter, V., Kafsi, M., Kazemi, E., Grossglauser, M., & Thiran, P. (2013). Where to go from here? Mobility prediction from instantaneous information. *Pervasive and Mobile Computing*, *9*(6), 784-797.

Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.

Gu, T., Pung, H. K., & Zhang, D. Q. (2005). A service-oriented middleware for building context-aware services. *Journal of Network and computer applications*, *28*(1), 1-18.

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception* (pp. 65-93).

Isaacman, S., Becker, R., Cáceres, R., Kobourov, S., Martonosi, M., Rowland, J., & Varshavsky, A. (2011, June). Identifying important places in people's lives from cellular network data. In *International Conference on Pervasive Computing* (pp. 133-151). Springer, Berlin, Heidelberg.

Joshi, J., Rajapriya, V., Rahul, S. R., Kumar, P., Polepally, S., Samineni, R., & Tej, D. K. (2017, January). Performance enhancement and IoT based monitoring for smart home. In *Information Networking (ICOIN), 2017 International Conference on* (pp. 468-473). IEEE.

Krause, A., Smailagic, A., & Siewiorek, D. P. (2006). Context-aware mobile computing: Learning context-dependent personal preferences from a wearable sensor array. *IEEE Transactions on Mobile Computing*, *5*(2), 113-127.

Lane, N. D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., & Campbell, A. T. (2010). A survey of mobile phone sensing. *IEEE Communications magazine*, *48*(9).

Laurila, J. K., Gatica-Perez, D., Aad, I., Bornet, O., Do, T. M. T., Dousse, O.,Eberle J. & Miettinen, M. (2012). The mobile data challenge: Big data for mobile computing research. In Pervasive Computing (No. EPFL-CONF-192489).

Lefort, L., Henson, C., Taylor, K., Barnaghi, P., Compton, M., Corcho, O., ... & Neuhaus, H. (2011). Semantic sensor network xg final report.

Li, X., Eckert, M., Martinez, J. F., & Rubio, G. (2015). Context aware middleware architectures: survey and challenges. *Sensors*, *15*(8), 20570-20607.

Lu, H., Frauendorfer, D., Rabbi, M., Mast, M. S., Chittaranjan, G. T., Campbell, A. T., ... & Choudhury, T. (2012, September). Stresssense: Detecting stress in unconstrained acoustic environments using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (pp. 351-360). ACM.

Maurer, U., Rowe, A., Smailagic, A., & Siewiorek, D. P. (2006, April). eWatch: a wearable sensor and notification platform. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on* (pp. 4-pp). IEEE.

Mcinerney, J., Stein, S., Rogers, A., & Jennings, N. R. (2013). Breaking the habit: Measuring and predicting departures from routine in individual human mobility. *Pervasive and Mobile Computing*, *9*(6), 808-822.

McKinney, W. (2018). "Python Data Analysis Library—pandas: Python Data Analysis Library." Online at: http://pandas.pydata.org

McNamara, L., Mascolo, C., & Capra, L. (2008, September). Media sharing based on colocation prediction in urban transport. In *Proceedings of the 14th ACM international conference on Mobile computing and networking* (pp. 58-69). ACM.

Miluzzo, E., Lane, N. D., Fodor, K., Peterson, R., Lu, H., Musolesi, M., ... & Campbell, A. T. (2008, November). Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (pp. 337-350). ACM.

Nath, S. (2012, June). ACE: exploiting correlation for energy-efficient and continuous context sensing. In *Proceedings of the 10th international conference on Mobile systems, applications, and services* (pp. 29-42). ACM.

Noulas, A., Scellato, S., Lathia, N., & Mascolo, C. (2012, December). Mining user mobility features for next place prediction in location-based services. In *Data mining (ICDM), 2012 IEEE 12th international conference on* (pp. 1038-1043). IEEE.

Pejovic, V., & Musolesi, M. (2015). Anticipatory mobile computing: A survey of the state of the art and research challenges. *ACM Computing Surveys (CSUR)*, *47*(3), 47.

Peltonen, E., Lagerspetz, E., Nurmi, P., & Tarkoma, S. (2016). Constella: Crowdsourced system setting recommendations for mobile devices. *Pervasive and Mobile Computing*, *26*, 71-90.

Puiatti, A., Mudda, S., Giordano, S., & Mayora, O. (2011, August). Smartphone-centred wearable sensors network for monitoring patients with bipolar disorder. In *Engineering in Medicine and Biology Society, EMBC, 2011 annual international conference of the IEEE* (pp. 3644-3647). IEEE.

Rachuri, K. K., Mascolo, C., Musolesi, M., & Rentfrow, P. J. (2011, September). Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proceedings of the 17th annual international conference on Mobile computing and networking* (pp. 73-84). ACM.

Rachuri, K. K., Musolesi, M., Mascolo, C., Rentfrow, P. J., Longworth, C., & Aucinas, A. (2010, September). EmotionSense: a mobile phones based adaptive platform for experimental social psychology research. In *Proceedings of the 12th ACM international conference on Ubiquitous computing* (pp. 281-290). ACM.

Ravi N., Scott J., Han L., and Iftode L. (2008). Context-aware Battery Management for Mobile Phones. In Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM '08). IEEE Computer Society, Washington, DC, USA, 224-233. DOI: https://doi.org/10.1109/PERCOM.2008.108

Rivero-Rodriguez, A., Pileggi, P., & Nykänen, O. A. (2016). Mobile context-aware systems: technologies, resources and applications. International Journal of Interactive Mobile Technologies (iJIM), 10(2), 25-32.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

Sadilek, A., & Krumm, J. (2012, July). Far Out: Predicting Long-Term Human Mobility. In *AAAI*.

SANTOS, Vaninha Vieira dos. (2008) "CEManTIKA: a Domain-independent framework for designing context sensitive systems." PhD Thesis.

Scellato, S., Musolesi, M., Mascolo, C., Latora, V., & Campbell, A. T. (2011, June). Nextplace: a spatio-temporal prediction framework for pervasive systems. In *International Conference on Pervasive Computing* (pp. 152-169). Springer, Berlin, Heidelberg.

Sher, G. I. (2012). *Handbook of neuroevolution through Erlang*. Springer Science & Business Media.

Song, L., Kotz, D., Jain, R., & He, X. (2004, March). Evaluating location predictors with extensive Wi-Fi mobility data. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies* (Vol. 2, pp. 1414-1424). IEEE.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, *10*(2), 99-127.

Strang, T. and Linnhoff-Popien, C. (2004) A Context Modeling Survey, First International Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp.

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2017). Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *arXiv preprint arXiv:1712.06567*.

Teodoro, D., Sundvall, E., Junior, M. J., Ruch, P., & Freire, S. M. (2018). ORBDA: An openEHR benchmark dataset for performance assessment of electronic health record servers. *PloS one*, *13*(1), e0190028.

Vaarandi, R., & Niziński, P. (2013, July). Comparative analysis of open-source log management solutions for security monitoring and network forensics. In *Proceedings of the 2013 European Conference on Information Warfare and Security* (pp. 278-287).

Vega, C., Roquero, P., Leira, R., Gonzalez, I., & Aracil, J. (2017). Loginson: a transform and load system for very large-scale log analysis in large IT infrastructures. *The Journal of Supercomputing*, *73*(9), 3879-3900.

Verginadis, Y., Patiniotakis, I., Papageorgiou, N., Apostolou, D. , Mentzas, G. , Stojanovic, N. (2015). Context Management in Event Marketplaces. The Semantic Web: ESWC 2012 Satellite Events: ESWC 2012 Satellite Events, Heraklion, Crete, Greece, May 27-31, 2012. Revised Selected Papers, 7540, 313.

Wagner, D. T., Rice, A., & Beresford, A. R. (2013, December). Device analyzer: Understanding smartphone usage. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services* (pp. 195-208). Springer, Cham.

Wang, Y., Lin, J., Annavaram, M., Jacobson, Q. A., Hong, J., Krishnamachari, B., & Sadeh, N. (2009, June). 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* (pp. 179-192). ACM.

Xia, F., Ding, F., Li, J., Kong, X., Yang, L. T., & Ma, J. (2014). Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Information Systems Frontiers*, *16*(1), 95-111.

Zhou, B., Dastjerdi, A. V., Calheiros, R. N., Srirama, S. N., & Buyya, R. (2015, June). A context sensitive offloading scheme for mobile cloud computing service. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on* (pp. 869-876). IEEE.