



FP7-ICT-2013- 10 (611146)

CONTREX


Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties

Project Duration

2013-10-01 – 2016-09-30

Type

IP

	WP no.	Deliverable no.	Lead participant
	WP3	D3.3.3	PoliMi
Implementation of the run-time management infrastructure (FINAL)			
Prepared by	Gianluca Palermo, William Fornaciari, Giuseppe Massari, Carlo Brandolese (PoliMi), Massimo Poncino (PoliTo), Paolo Azzoni, Matteo Maiero (EUTH), Sylvian Kaiser (Intel)		
Issued by	POLIMI		
Document Number/Rev.	CONTREX/POLIMI/R/D3.3.3/1.0		
Classification	CONTREX Public		
Submission Date	2016-09-30		
Due Date	2016-09-30		
Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)			

© Copyright 2016 OFFIS e.V., STMicroelectronics srl., GMV Aerospace and Defence SA, Vodafone Automotive SpA, Eurotech SPA, Intecs SPA, iXtronics GmbH, EDALab srl, Docea Power, Politecnico di Milano, Politecnico di Torino, Universidad de Cantabria, Kungliga Tekniska Högskolan, European Electronic Chips & Systems design Initiative, ST-Polito Società consortile a r.l., Intel Corporation SAS.

This document may be copied freely for use in the public domain. Sections of it may be copied provided that acknowledgement is given of this original work. No responsibility is assumed by CONTREX or its members for any application or design, nor for any infringements of patents or rights of others which may result from the use of this document.

History of Changes

ED.	REV.	DATE	PAGES	REASON FOR CHANGES
GP	0.1	2016-09-1	51	Skeleton taken from D332
PA	0.2	2016-09-21	53	Section 4.5 update
GP,CB	0.3	2016-09-23	57	PoliMi contribution update
SK	0.4	2016-09-28	56	Intel contribution
SV, MP	0.5	2016-09-29	62	PoliTo Contribution
GP	1.0	2016-09-30	62	Final review

Contents

1	Introduction	4
2	Overview	5
3	State of the art of Run-Time management strategies	7
3.1	Dynamic Power Management	7
3.2	Thermal Management	7
3.3	Resource Management	9
3.4	Battery Aware management strategies	9
4	Dedicated Run Time Management approach	12
4.1	RTRM for Mixed Critical applications	12
4.1.1	System-Wide RTRM	12
4.1.2	Resources Partitioning Policy	14
4.1.3	Design Time Support	15
4.1.4	Distributed Hierarchical Control	15
4.1.5	Application Program Interface	17
4.1.6	Implementation Overview in Linux-Based Systems	18
	Mixed Criticality Aware Resource Allocation	24
4.2	Battery-Aware Run-Time Management	28
4.2.1	Adopted components for battery and power converters	28
4.2.2	Adopted models for battery and power converters	29
4.2.3	Run-time management	32
4.2.4	Impact on the node (UC2)	33
4.3	Lightweight dynamic power management	37
4.4	Run-Time management for the Cloud	42
4.4.1	Managing accounts and users	42
4.4.2	Managing Kura enabled devices	44
4.4.3	Viewing collected data	46
4.4.4	Using cloud rules	48
4.4.5	The cloud platform REST API	55
5	Conclusions	59
	References	60

1 Introduction

Scope of this deliverable is to describe the various mechanisms envisioned in the CONTREX project to be applied at run-time to manage the execution platform according to extra-functional properties.

This deliverable is to be considered as a final document for reporting all the activities within Task 3.3. The document extends the previous deliverables on the run-time management infrastructure D3.3.1 and D3.3.2 keeping updated all the activity. All the sections have been updated with respect to the previous version of the Deliverable D3.3.2.

In particular this deliverable describes the state of the art concerning run-time resource management both at node level and at system level and outlines the mechanisms and the policies that are under development within CONTREX also showing some results related to the actual implementations of the mechanisms within the Use Cases of the project.

The deliverable is organized into three sections. Section 2 describes the overall context and summarizes the mapping of the activities of Task 3.3 partners onto the specific use cases. Section 3 outlines the state of the art of run-time management activities, while Section 4 contains the details of each of the CONTREX dedicated activities.

2 Overview

The main activities of this task are on the definition of mechanisms and policies to implement run-time management both at node and cloud level.

The run-time management layer can be seen as an intermediate layer between the HW platform and the application layers providing services to the application(s) through the execution platform abstraction layer. The run-time management layer uses the extra-functional models of the platforms (developed in Task 3.1) as the knowledge base to guide its decision-making policies.

The run-time manager can operate autonomously (implementing a typical control loop) and/or based on optimal configuration points defined at compile/design-time through the design space exploration phase.

Figure 1 shows all the activities engaged in Task 3.3 (and described in Section 4 of this document) highlighting the targets of the run-time actions. While obviously the application layer is always impacted by the decisions in terms of extra-functional properties, the arrows identify where are the knobs used by the run-time management layer to dynamically reconfigure the HW/SW infrastructure.

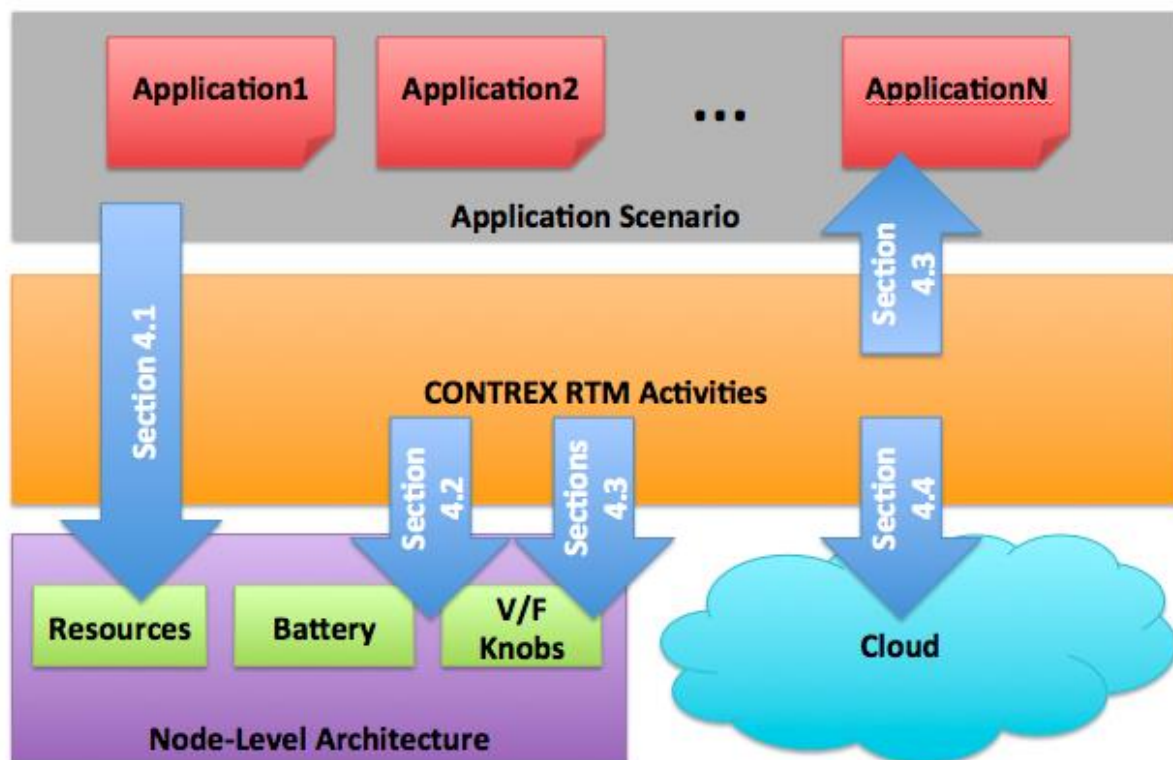


Figure 1 - Overview of the CONTREX RTM Activities

Section 4.1 describes the idea of the framework that will be used for the management of mixed critical application at node level in terms of resources. The modular framework called BarbequeRTRM is enhanced in the context of CONTREX for supporting mixed-criticalities and power/thermal aspects always considering the resources as main knob.

Section 4.2 defines policies for dynamically managing the available energy stored in a storage device according to the workload. This activity encompasses the implementation of the models of the non-functional considered properties, namely power consumed and available in the storage.

Section 4.3 describes a lightweight power manager at node-level that is used to customize the application services depending on the monitoring of the system status. This manager interfaces a monitoring layer developed in Task 3.4 to select predefined power policies determined at design time on the basis of the execution platform model derived in Task 3.2.

Section 4.4 outlines the new software components that extend the pervasive infrastructure to provide a run time management features to the cloud service abstraction. The implementation of these components will be finalized in WP4, in the context of the automotive application scenario.

Overall, the run-time management infrastructure supports the execution platforms proposed in the specific application contexts of the CONTREX use-cases. In particular, except the activity related to the resource management (Section 4.1) that is used in the context of the Use Case 1, all the other activities are planned to be adopted in the context of the Automotive Use Case (Use Case 2).

3 State of the art of Run-Time management strategies

3.1 Dynamic Power Management

Techniques to reduce power consumption in computing systems range from physical layers design up to higher software abstraction levels [Venk05cs] [Pedram01aspdac]. Considering a classification based on the abstraction levels, the main techniques proposed in literature fall into five categories: pure hardware, OS-level, application level and cross-layer adaption approaches.

The *pure hardware* approaches mainly address the processor DVFS. These techniques are based on specific hardware support that measure the current CPU load and configure its frequency according to the inferred system utilization. All these approaches are completely transparent from the user-space, cannot exploit knowledge on future workloads and disregard specific application needs.

The OS-level techniques basically try to improve the memory-less hardware approaches in two main directions: by exploiting OS scheduler knowledge (e.g., [Lorch03mobisys]) and by allowing software system designer to compare different optimization policies [Pettis09tcom]. A number of other works has focused techniques for the power optimization of specific subsystems, e.g., disks, network cards and displays. In others more cooperative approaches, the OS tries to exploit some “application hints” to increase the level of knowledge e about tasks’ requirements [Anand04mobisys].

The ‘application level’ approaches, rather than a partnership between the OS and the applications, try to exports the entire burden of PM to the user level, resembling the philosophy of the Exokernels. Application adaption techniques trade power consumption with quality or data fidelity (e.g., [Tamai04nossdav]). Others techniques propose complete optimization frameworks, such as the Chameleon’s application-level Power Management architecture [Liu10tmc].

The development of holistic approaches, that aggregate data from multiple layers, is nowadays a popular research topic. Indeed, a number of approaches based on ‘cross-layer adaptations’ have already been proposed. Unfortunately, available solutions are frequently designed only for the energy optimization of real-time multimedia tasks with fixed periods and deadlines [Yuan03sosp], require application modifications to feed some meta-information to the optimization layer [Fei08tecs] or are based on complex models to be build off-line and thus not easily portable across different platforms [snowdon09eurososys]. In this class we can find also some Linux kernel framework, such as DPM [Brock03soc].

3.2 Thermal Management

Dynamic thermal management techniques (DTM) [Donald06isca] are approaches that address dynamically the thermal problems either by limiting the power sources (e.g. by using the dynamic voltage and frequency scaling) or by changing the spatial power distribution over the chip (e.g. task migration).

DTM techniques should always consider the computational workload of the systems since the application QoS should be impacted as less as possible [Chantem09isp]. In fact, the trend with DVFS techniques can be to cool down the chip temperature by sacrificing its performance.

An alternative approach to DVFS is to balance the workload among CPUs and thus the power distribution. One of the most used mechanisms for reaching this goal is the task migration. It gives the possibility to reduce the temperature peaks without reducing the CPU frequency by temporally balancing the workload among the cores [Yeo08dac] [Khan09date] [Ge10dac] [Gomaa05asplos]

Some examples are reactive task migration algorithms that migrate the task away from hottest core to the coolest one. Those techniques make a basic assumption that the Thermal Manager has the possibility to know the temperature of the cores on the chip either by means of simplified models or by thermal sensors. The effectiveness of this type of DTM techniques can be severely degraded if the thermal model diverges from the real thermal behaviour, if thermal sensors can lack of accuracy due to their placement location and if long latency occurs between the identification of the critical temperature and the system reaction (actual actuation).

To this end, DTM algorithms and models predicting the potential thermal emergency before reaching the critical situation have been developed [Khan09date] [Ge10dac] [Yeo08dac]. This type of approach takes the system behaviour into consideration, either by knowing the application characteristics or by use a history-based thermal prediction.

Note that even though DPM and DTM can exploit common techniques to modify the power dissipation pattern in space and time in a circuit, they can result in contradictory decisions. For instance, a power management policy may favour a higher processing locality to reduce the energy cost of transferring data between processing units. But such a policy tends to increase the local power density and is thus likely to degrade thermal operating conditions.

An industrial implementation that combines DPM and DTM is the ARM IPA (Intelligent Power Allocation) embedded framework. This solution combine power management and thermal management.

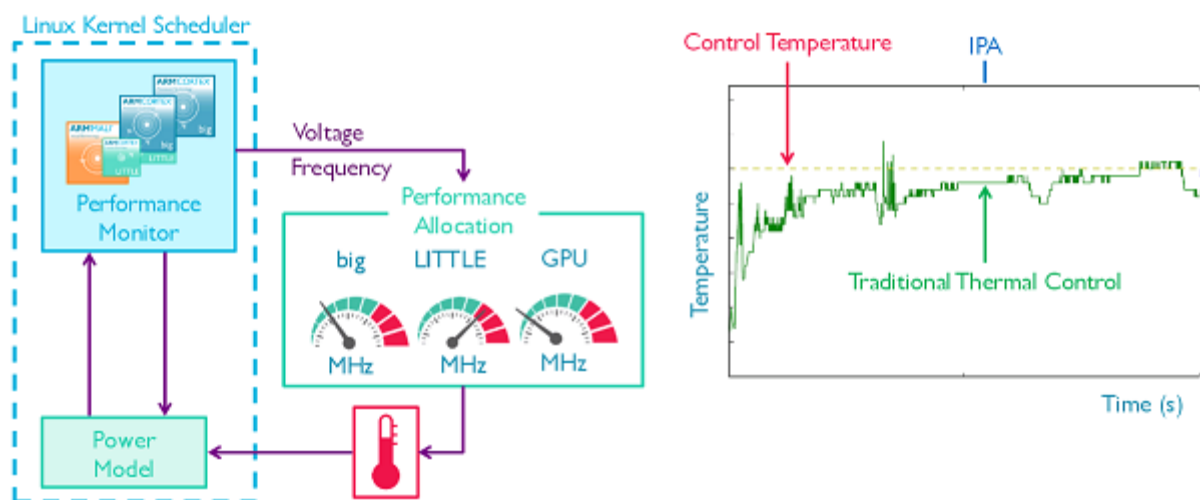


Figure 2 – ARM Intelligent Power Allocation (IPA)

Figure 2 is an illustration prepared by ARM's CTO Mike Muller. The objective of IPA is to help chip makers manage power consumption and heat dissipation in System-on-Chips integrating a big.LITTLE dual-core IP. The system temperature and the power consumed by the three main cores (the big core, the LITTLE core + a Graphical Processing Unit (GPU) core) are monitored at run time. This is exploited to determine the future permitted power allocation with respect to the performance requirements and the tasks positioned in the OS task queue (here the Linux Kernel Scheduler). Therefore, this is an OS level mechanism. The decision on power allocation is used to tune the V/F (Voltage and Frequency) operating points of the IPs.

This example illustrates that monitoring extra-functional properties and employing runtime management techniques to control their effects is now a technique exploited in devices commercialized in high-volume markets.

3.3 Resource Management

Modern multimedia applications for new generation multi-core mobile embedded systems require specific resources and exhibit dynamic QoS requirements. The required support to parallel application execution with both high performance and low energy costs, raised up new classes of problems. Quality of service (QoS) management, which concerns to the definition of mechanisms and policies to enhance user experience in a cost effective manner, represents a first problem. The system design should consider the dynamic nature of user requirements and the support of different optimization strategies, e.g. multimedia boosting and resource saving. This can be done with a run-time fine-tuning on applications' behaviour, for example by dynamically changing the level of parallelism of the applications or by forcing a task migration. A suitable run-time support cannot disregard mixed workload usage scenarios, where both device critical services (e.g., a radio stack) and user- installed applications run aside competing on the usage of the same resources. This requires the ability to specify a priority level for each application and to properly handle it at run-time.

Among the approaches to run-time resource management, the "Pareto optimality" is a widely used concept [Shahriar07jos] [Xu05ts] [Ykman06issoc] [Ykman11ietcdt]. In [Nollet08jsps], [Shojaei10dac] an "utility model" has been described to obtain a resources allocation solution aiming to approach the Pareto optimality. This model constructs a multi-dimensional multi-choice knapsack problem (MMKP) formalizing the concept of profiled applications with QoS constraints and Pareto optimal resource allocation. Many approaches have been proposed for the case of runtime manager in multi-processors architectures [Ykman11ietcdt] [Nollet08jsps] [Shojaei10dac] to manage applications with different operating configurations (resource usage) and QoS constraints. In those cases, the runtime manager selects a system configuration for all the active applications as a MMKP solution. Other works define different models of the solution [Zoni12arcs], such as exploiting the Pareto Algebra [Glein05ACSD][Shojaei10dac].

3.4 Battery Aware management strategies

Traditional power management strategies as the ones described in Section 3.1 assume an ideal power supply, that is, (1) it can satisfy any power demand by the load and (2) the available

battery energy is always available regardless of the load demand. While this simplifying assumption may be considered reasonable for systems connected to a fixed power supply, it is simply wrong in the case of battery-operated systems.

Batteries are in fact nowhere close to be ideal charge storage units, as pointed out in any battery handbook [Linden95]. From a designer's standpoint, three are the main non-idealities of real-life battery cells that need to be considered:

1. **The capacity and the lifetime of a battery depend on the discharge current.** At higher currents, a battery is less efficient in converting its chemically stored energy into available electrical energy. This effect is called the **rated capacity effect**: it is shown in Figure 3, showing battery voltage vs. capacity (which is a measure of available energy) for a sample battery (<http://www.low-powerdesign.com>)

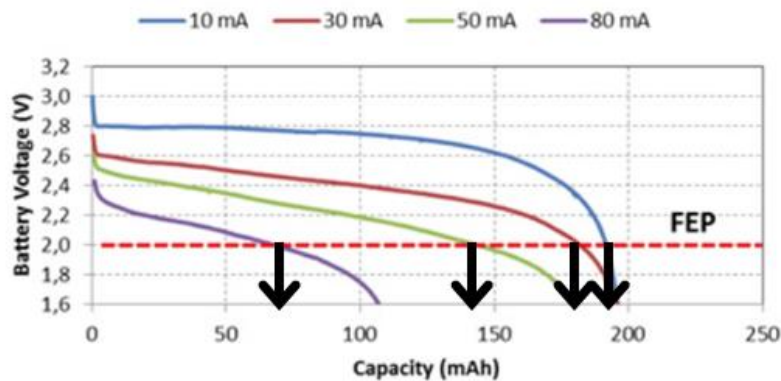


Figure 3. Rated Capacity Effect in a Typical Battery.

We observe that, for increasing load currents, the battery capacity (calculated at the point where battery voltage reaches the cutoff point – the dashed line) progressively decreases. The 180mAh resulting from a load of 10mA become only 70 mAh (about 2.5x less) for a load of 80mA.

2. **The capacity and the lifetime of a battery depends on the discharge current distribution.** In particular, there exists a correlation between battery lifetime and the **variance** of the current load; for a given average current value, a constant load (i.e., no variance) will result in the longest battery lifetime of all load profiles.
3. **Batteries have some (limited) recovery capacity during idle times.** A battery can recover some of its deliverable charge if periods of discharge are interleaved with rest periods, i.e., periods in which no current is drawn (intermittent discharge).

In terms of power management of battery-powered devices these three properties have two important consequences on the strategies for optimizing the energy usage:

- It is better to distribute the load over time, i.e., request a smaller current (power) for a longer time, than to have shorter computations with higher loads; This is *consistent with objective of dynamic speed scaling* in which averaging the execution speeds of different computations over time yields the optimal energy consumption [Govil95,Sinha01].

- Inserting idle times in the battery discharge is beneficial. This is *consistent with the objectives of dynamic power management*, which aims at putting resources into standby states when not used.

Besides this intrinsic non-idealities arising from the electrochemical properties of batteries, there exists another issue related to the *interfacing* of batteries with a load, namely the impact of the voltage converter/regulator.

A battery needs a voltage converter because (1) its output nominal voltage is typically different from the one required by the load, (2) its output voltage is not truly constant due to discharge, and it has to be regulated. Typically, a DC/DC converter (for which a number of possible circuit implementation) is used. The problem is that DC/DC converters are also non ideal, and the conversion process consumes some power: these are the converter losses. It means that the power provided in input to the converter (from the battery) will be transferred to the load only in part (output power). The ratio between output and input power is the converter efficiency. Problem is that converter is not constant but depends on (i) the difference of input and output voltages, and (ii) the output current. While the first is dictated by technology (e.g., lithium-ion batteries have typically 3.7—4.2 V output and a processor core is power at 0.9—1.1 V), the second variable depends on the workload. Typical converters are extremely inefficient at low output current, which incidentally are the values that we are aiming at during low-power design!

Figure 4 shows a typical energy efficiency curve for a sample converter by Linear Technology.

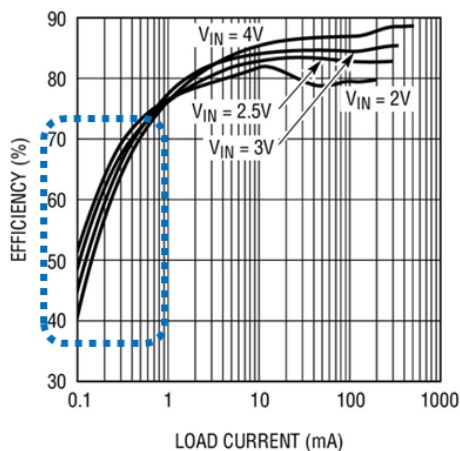


Figure 4. Converter efficiency vs. load current.

It is quite impressive how for currents below 1mA (that could be typical of a low-power standby state) between 30% and 50% of battery energy is lost in the conversion process.

This aspect appears to be consistent with battery property that suggests to average the workload as much as possible. Since converter efficiency is non-linear and increases steeply for low-current, it is best to avoid long intervals of very low current loads, and use an equivalent average current load (which should be as much constant as possible from the battery standpoint).

4 Dedicated Run Time Management approach

4.1 RTRM for Mixed Critical applications

Given the state of the art approaches presented in section 3, the required properties on the run-time resource manager for mixed critical application, to be used within CONTREX, can be defined as follow:

- Management of Homogeneous and Heterogeneous resources: the RTRM should target systems with resources either of the same type (such as Multi-/Many-core CPUs) or different type (e.g., CPU+GPU, multiple memory modules,...);
- Management of application adaptivity and parallelism: the RTRM should target also applications that can require more than one resource and that can be reconfigured at run-time according to different resource assignments;
- Extra-functional properties awareness: the RTRM should include multi-objective scheduling and resource allocation policies also considering possible hints coming from an off-line Design Space Exploration phase. The objective should include extra-functional properties, such as power, or criticality levels and throughput requirements;
- Portability: the RTRM should not target a single platform but should be general enough to be easily portable.

Following the previous requirements, in this section we present the portable and extensible framework for run-time resource management, supporting both homogeneous and heterogeneous platforms in the context of mixed-critical applications.

4.1.1 System-Wide RTRM

The RTRM used for the management of mixed critical application at node level is called BarbequeRTRM and it is implemented as an open source modular framework. It has been developed in the previous project [2parma] and has been enhanced within the current project to manage mixed-critical application and forcing power-awareness in the decisions. Its layered design, with a main software stack, spans from target platform up to managed applications, besides a set of optional components that provide advanced features.

An overall view of the proposed system-wide resource partitioning strategy is depicted in the following figure.

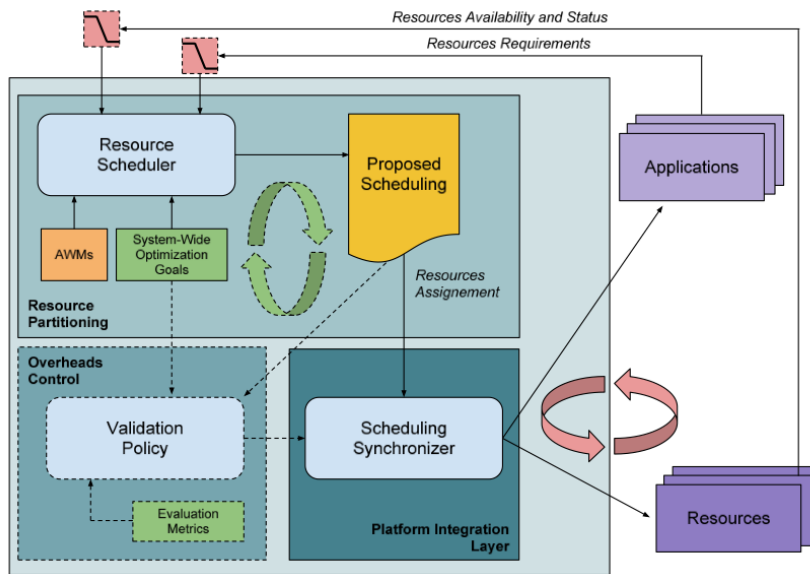


Figure 5 - System Wide Resource Partitioning Scheme

The main goal of this control level is to identify an optimal partitioning of the available computational resources among the demanding applications. It is worth to notice that what we propose is an event based scheduler, where a new scheduling decision is based on events. Events are related either to

- changes on resources availability (or their status) or
- changes on applications requirements in terms of computational resources.

Each time such an event happens, a new run of a “resource scheduler” is triggered to identify a new scheduling.

The proposed resources assignment is based on the scheduler optimization function, which exploits design-time identified Application Working Modes (AWMs – See later), according to a set of system-wide optimization goals. These goals encompass different system optimization aspects, ranging from applications performance to power and thermal profile as well as some overheads and control robustness aspects, thus actually supporting a run-time tuneable and multi-objective optimization strategy.

However, the scheduling identified is not immediately enforced on the system. Indeed, the second step of the proposed strategy is related to “overheads control” and targets a validation of the scheduling decision based on a set of system-wide metrics and a scheduler validation policy. The goal of this policy is two-fold:

- a) to inhibit the actuation of scheduling decisions that are not compliant with constraints on system- wide optimization goals, and
- b) to tune some of the scheduler parameters, in order to improve the quality of the proposed solution on its following runs.

Since a new run of the resource scheduler could be eventually started by the validation policy, the first two steps of the proposed strategy actually represent an inner feedback control loop, for the system-wide control level. Finally, the outer feed- back control loop is closed by a scheduler synchronizer which is in charge to actuate a proposed scheduling decision by

properly configuring both applications and resources. Therefore, reconfigured applications are properly notified about the new assigned AWM, i.e. the set of computational resources reserved to the application, and computational resources are set according to the scheduling decision. The last point is performed through a platform-specific module called Platform Integration Layer (PIL).

The main goal of the inner loop is to reduce the chances to produce instabilities, or higher overheads on the outer loop. To the same purpose respond the two low-pass filters at the input of the resource scheduler. These are targeted to increase control robustness, by avoiding (or eventually delaying) the triggering of new scheduling events following small variations on either application requirements or resources availability.

4.1.2 Resources Partitioning Policy

The resource partitioning, among the demanding applications, must consider 1) the requirements of each application 2) the status of the available resources and 3) a set of optimization goals. It is worth to notice that every application has a predefined set of possible configurations (the AWMs identified at design-time, see later), each one corresponding to a certain amount of required resources. The partitioning policy is in charge to select (at most) one AWM for each active application, where it is assumed that there are a set of platform resources and several active applications, each one with a specific priority and a set of possible configurations, i.e. AWMs;

This configures as a well-studied problem in combinational optimization which is known as multi-choice multi-dimension multiple knapsacks problem (MMMKP). The formulation of the MMMKP problem is completed by the introduction of a profit P to be associated to each possible choice. The profit assigned to such a choice is defined by a multi-objective optimization policy. Such a formulated MMMKP problem is known to be NP-hard [Shojaei10dac], and algorithms for exact solutions are too slow and thus not suitable for an efficient run-time management exploitation. Fortunately, state-of-the-art heuristics have been developed, which allows finding near-optimal solutions and are fast enough for the considered environment. The optimization policy we propose belongs to this class of solutions and it has been inspired by a greedy heuristic (henceforth referenced as “original heuristics”), proposed by Ykman-Couvreur et al. [Ykman06issoc], which has been considered since it has been demonstrated to exhibit reasonable overhead, even in the specific context of resource constrained and real-time embedded system.

To clarify, the resource partitioning among the active applications should take into account a set of optimization goals:

- Maximization of the applications performance or Quality-of-Service (QoS);
- Maximization of the fairness in the resource assignment;
- Minimization of the reconfiguration overheads, whenever a choice leads to a change of resource assignment in terms of amount of resource;
- Minimization of the migration overheads, whenever a choice leads to a change of resource assignment in terms of mapping;

- Maximization of the load balancing, i.e., try to distribute the resource mapping to minimize resource contention and balancing the temperature of the several processing cores.

The results of the resource allocation policy are notified to each application through a specific API, the Run-Time Library (RTLlib). The resource partitioning policy is the key element in case of management of mixed-critical systems since the decision should be taken considering the level of criticality of the applications that are running. Despite it is clear that the proposed RTRM cannot be used in Hard-real time system, the management of mixed-critical application has been implemented within the partitioning policy by modifying the concept of fairness in the resource assignments and QoS. In particular, this issue has been faced either by weighting the resource function considering the priority of the application, or by giving the possibility to describe design-time defined scenarios to be selected at run-time (by adopting a more static approach).

4.1.3 Design Time Support

An effective usage of the computational resources starts from the capability of the applications to run according to different configurations, that means different resource usage levels. We think that design-time activities can provide a good contribution in profiling applications, and thus identify their possible configurations. To this purpose, we currently exploit a Design-Time Exploration (DSE) tool, which performs an optimal quantization of the configuration space of run-time tunable applications, identifying a finite set of configurations. The effectiveness of such an approach to efficiently support run-time resources management has been already demonstrated on many prior works [Ykman06issoc], [Shojaei10dac], [Ykman11ietcdt].

The configuration of a run-time tuneable application is defined by a set of parameters; some of them could affect just on the applications behaviours while others have a direct impact on the class or amount of the required computational resources. For example, if we consider a video decoding application, a tuneable ‘frame-rate’ parameter impacts just on the application behaviours, while the ‘CPU processing time’ is a parameter directly affecting the amount of a required computational resource. This distinction allows classifying the application tuneable parameters into two different granularity levels:

- Operating Point (OP): a collection of application specific parameters, corresponding to an expected QoS for the end user;
- Application Working Mode (AWM): a collection of re- sources requirements, corresponding to an expected QoS for the application.

Indeed, a single AWM could support multiple OPs, i.e. the same kind and amount of computational resources could accommodate multiple values of application specific parameters. Therefore, OPs are bound to application-specific properties, i.e. affecting just the specific application, while AWMs describes system-wide properties, i.e. affecting system resources and thus every other system entity competing on their usage. The AWMs are those of interest for the RTRM.

4.1.4 Distributed Hierarchical Control

The set of AWMs and OPs identified at design-time become valuable feed-forward signals, for the proposed run-time control solution. They lead to an efficient and low-overhead

identification of control actions, as well as to a reduction of the convergence time required to reach a new stable system configuration.

We propose a distributed control scheme, where controllers are distributed throughout the system, acting on a specific subsystem. This design allows: a) to spread the control complexity, b) to design subsystem specific optimal controllers, and c) to scale better with the overall system complexity and number of subsystems.

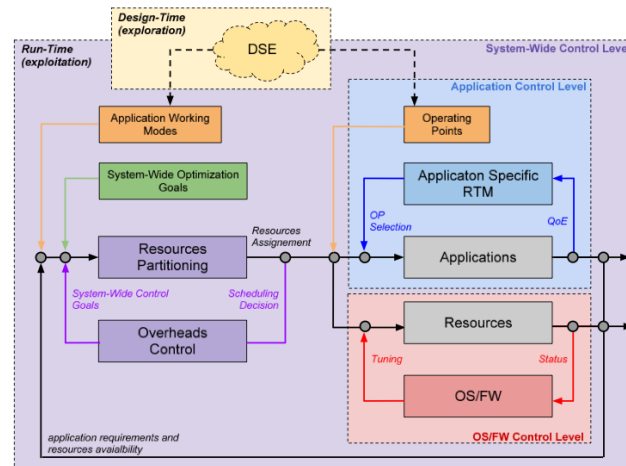


Figure 6 - RTRM Hierarchical Control

The hierarchical approach allows the control at different granularities and abstraction levels. This simplifies the design of the overall control solutions for a complex system. Indeed, higher abstraction levels target system-wide optimizations, which are usually associated to non-negligible overheads and reduced optimization opportunities. Conversely, lower abstraction levels are targeted to fine tunings and optimizations, which usually relate to almost negligible overheads and frequent adjustment opportunities.

We propose three main classes of subsystems, each one corresponding to a different control level, and driven by a specific controller. For each run-time tuneable application it is defined an “application control level” subsystem. Based on the specific characteristics of the target application an Application Specific Run-Time Manager (AS-RTM), is in charge to define a suitable policy to select an OP based on the amount of resources being assigned to the application (AWM), and the actual quality of experience obtained at run-time. Such an approach allows exploiting the detailed knowledge on: applications internals, performances evaluation and user perceived quality. Hence, this control level is in charge to: a) evaluate actual run-time application behaviours and b) tune its specific parameters or eventually to request more resources to the higher control level.

At the same hierarchical level of the previous one, we relay an “OS/FW control level”, which is platform-specific. Here are located mechanisms like DVFS and thermal control capabilities, which can be exploited to react to risky conditions, e.g. hot spots and thermal alarms. Finally, the System-Wide Run-Time Resource Manager (SW-RTRM) targets a set of optimization goals that are achieved by a proper assignment of the available resources to the demanding applications.

4.1.5 Application Program Interface

In this distributed and hierarchical view, the applications play an active role on the self-adaptiveness of the system. All these activities are supported by a Run-Time Library (RTLib), which provides a rich set of features related not only to properly manage the interaction with the framework, but also to support the application specific run-time management activities.

To simplify the coding (and the integration of existing) stream processing application, the RTLib provides the Abstract Execution Model (AEM). Basically, it is defined via a callback based API, meaning that the developer just needs to implement the application specific logic into the body of few methods, as represented in figure. The implementation must define proper actions related to a) the processing of a bunch of data (onRun), e.g. decoding a frame, b) reconfiguring into a different working mode (onConfigure) or c) suspending the workload processing (onSuspend), e.g. when the required computational resources are currently not available.

As previously discussed, the AS-RTM has in charge to monitor the application specific QoS in order either to fine-tune its parameters or to switch to a different AWM, once an agreement has been met with the SW-RTRM. The RTLib provides a collection of utilities to monitor metrics of interest (e.g. timings and usages on different resources, such as memory or bandwidth). Given such metrics, an Operating Point Manager (OP-Manager) can easily order and filter the set of operating points, based on an optimization goal. By selecting an OP, the AS-RTM tunes some application specific parameters, without affecting the resource requirements. However, whenever the target QoS is not satisfactory, the application can ask for a change of AWM to the RTRM. This API includes a collection of methods which could be exploited during the so called “QoS Evaluation and Run-Time Tuning” phase of the AEM.

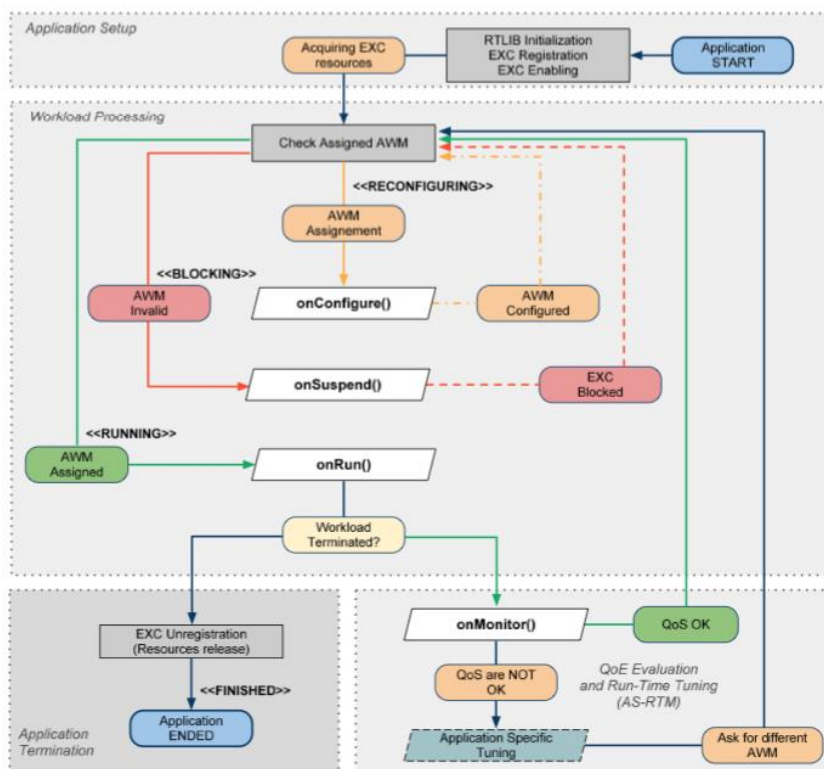


Figure 7 - Application Execution Model

The execution model, shown in the figure, is exposed to the application developer through a C++ class, called EXC. The definition of the class is included in the RTLib, introduced in the previous section.

The basic idea is that a control thread (not visible to the developer) invokes a specific member function of the EXC class (white parallelepipeds), according to what happens from the RTRM side. Therefore, considering a C++ application, the developer is in charge of defining a class derived from the base class EXC, and implements the member functions in a state-machine fashion. In other words, each member function represents what the application does according to a specific execution state:

- onSetup(): the function will includes only initialization code;
- onConfigure(): the application is notified about the the amount of resources, assigned. Whatever is resource-dependent must be configured, like for example the number of active threads and the values of some algorithm-specific parameters;
- onRun(): useful computation is performed here, e.g. processing of a frame image, scanning of a text block, processing of a frequency spectrum block, etc...
- onMonitor(): if the application needs to monitor the current performance/QoS level, the code in charge of do that can be placed here. The RTLib provides also a function through which the application can notify the RTRM about the “distance” between the expected performance goal, and the goal estimated at run-time.
- onRelease(): cleanup operations.

The control thread iterates until the application does not signal an exit condition. In most cases, this condition is the end of the input data or an exit command coming from the user.

4.1.6 Implementation Overview in Linux-Based Systems

The figure below provides an overview of the architecture of the Run-Time Resource Management (RTRM) solution proposed.

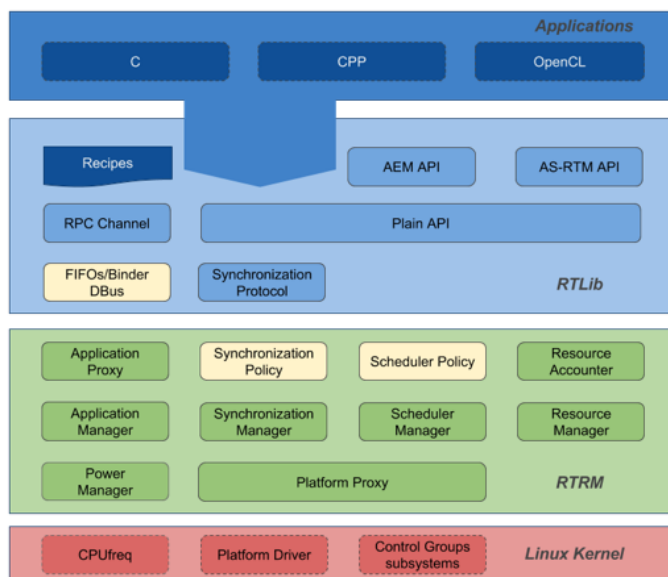


Figure 8 - Overview of the layered implementation in Linux-based systems

The topmost layer shows the programming languages actually supported for the integration of the run-time manageable applications, i.e. C, C++ and OpenCL.

The second layer is based on the Run-Time Library (RTLib) provided by the RTRM framework for the management of the applications. The library completely masks the communication infrastructure between the RTRM and the application. Moreover, it defines an execution model (discussed in the next section) and a set of functions for the interaction with the RTRM.

The third layer is the RTRM itself. As it is possible to see from the picture, it features a modular design, with the core modules (e.g., Application Manager, Scheduler Manager, Resource Accounter, etc...) and plugin modules (e.g., Scheduler Policy, Synchronization Policy, Communication channel implementation). This means that the framework offers several possibilities to customize its behaviour according to the system workload or the hardware platform.

Finally, the fourth layer is represented by the OS interfaces. Despite the interface with this layer called Platform Integration Layer (PIL) is general and not limited to a specific implementation, to the sake of portability and due to CONTREX platform requirements, the RTRM will be based on Linux. Additionally the Linux OS should include the frameworks required to enforce the resource assignment control, see next section.

4.1.6.1 Resource allocation enforcement and OS interfaces

The execution of the RTRM framework services on one of the CONTREX platform requires the presence of a Linux OS installed on the system. Being the portability one of the most important design requirements, the framework has been implemented to be executed as a daemon in user-space. However, to enforce a control over the allocation of hardware resources, a pair of needs must be satisfied: a) the execution of the daemon with root privileges; b) the availability of low-level interfaces towards the hardware resources. While the former does not represent a hitch, the latter imposes that the operating systems would export such interfaces. Depending from the hardware configuration of the system, and the resources that aim to manage, it can be possible to exploit already existing interfaces and mechanisms, or to implement ad-hoc platform supports.

In the case of a Linux OS, usually we can rely on an interesting set of frameworks, thanks to which it becomes possible to constrain the usages of the systems resources by the running applications. Linux Control Groups (CG) is a noticeable example of that. It includes a set of “subsystems”, each aiming at controlling a specific hardware resource, i.e. the CPU, the memory, the network bandwidth, and so on. The framework provides an extremely simple user-space interface, based on a virtual file-system. The user can “mount” a so-called Control Group hierarchy, by specifying the subsystems to include. This generates a file-system hierarchy, featuring a set of files, representing a subsystem attribute that can be read, written or both. Such files provide access to properties of the specific hardware resource that can be controlled. Therefore, the user can exploit a Control Group hierarchy to specify a set of tasks that are subjected to the constraints on the hardware resources, asserted through the values of the subsystems attributes. Accordingly, it is possible to isolate the execution of specific tasks on a reserved set of computing resources. This kind of capabilities allows addressing the typical mixed-criticality scenario requirements, on hardware platforms based on multi-core processors. Indeed, the contention generated by the execution of multiple tasks on the same

set of computing resources can be mitigated by suitably partitioning the resource set, and isolating the execution of tasks according to their criticality.

Specifically, the RTRM Platform Integration Layer and Resource Accounter modules build an abstraction level on top of such interfaces and mechanisms. Actually, the resource allocation and scheduling policies can act on top of this abstraction, by considering the resource control capabilities offered by the subsystems cpu, cpuset and memory, i.e. 1) CPU bandwidth quota reservation; 2) CPU core set assignment; 3) Amount of main memory assignment.

4.1.6.2 Power and thermal management

Actually to enforce power/thermal management, the RTRM provides a pair of ways:

- a) Dynamic tuning of the resource assignment: According to the criticality level of the application, the resource allocation policy can suitably shrink the amount of resources assigned, e.g. CPU cores, allowing the unused resource to go into an idle state, and thus saving dynamic power consumption.
- b) Exploitation of the Power Manager module: Similar to the Platform Proxy, this module builds an abstraction layer on top of the Linux framework providing the support for checking and controlling the operating frequencies and the temperature of the CPU resources. If the system exports the possibility to dynamically change such operating frequencies, it is possible to implement a custom power/thermal management policy to address the problem the specific mixed-criticality scenario.

Additionally, we can enhance the RTRM to support better support for mobile battery-supplied devices, by introducing the possibility of adding energy-aware policies that would perform resource allocation taking into account battery-related characteristics and the remaining capacity of the battery.

To test the effects of the RTRM control action, a set of tests have been performed on a embedded development board (Pandaboard ES), featuring a dual-core ARM Cortex A9 CPU as the one target of the Use Case where the RTRM is going to be used. More in detail, we executed a multi-threaded benchmark from the PARSEC 2.1 suite (blackscholes), allocating different amounts of CPU bandwidth quota, and observing the behaviour of the system in terms of CPU temperature, Linux thermal throttling and overall system power consumption.

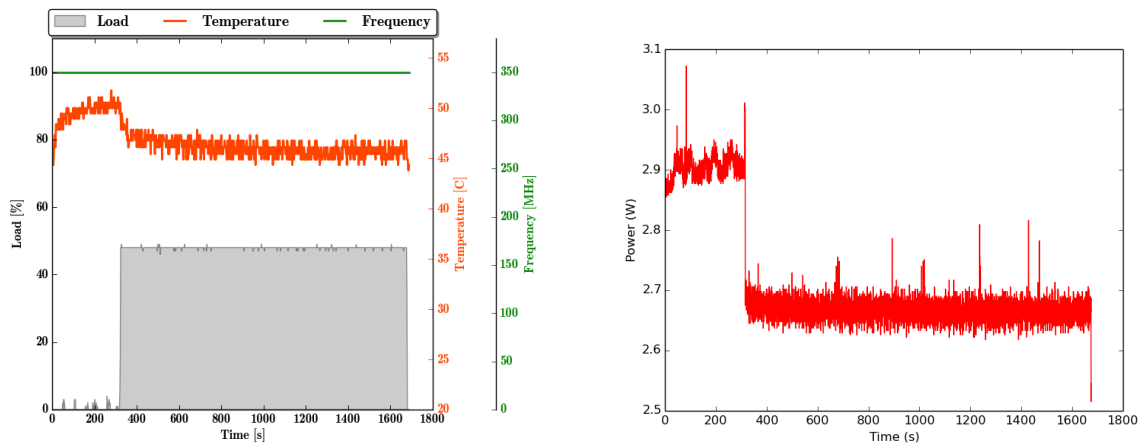


Figure 9 - Thermal response of a ARM Cortex A9 dual-core CPU @ 350 MHz, and system power consumption (Pandaboard ES), with the RTRM enforcing the allocation of 50% CPU bandwidth of a single core.

In the test cases limiting the CPU usage, we can clearly distinguish two different execution stages. During the first stage, the application performs some initialization steps. Afterwards, the execution goes on with the RTRM enforcing its resource allocation control.

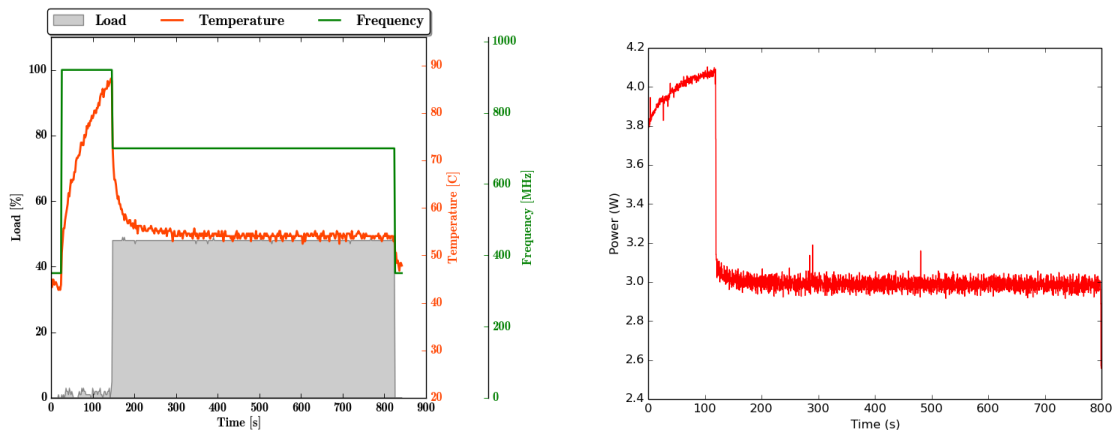


Figure 10: Thermal response of a ARM Cortex A9 dual-core CPU @ 920 MHz, and system power consumption (Pandaboard ES), with the RTRM enforcing the allocation of 50% CPU bandwidth of a single core.

In the test case shown in Figure 8, the CPU is set at the minimum frequency value supported (350 MHz), with the CPU bandwidth allocation limited to 50%, i.e., half a core. During the execution of the benchmark, after the setup stage, the CPU temperature settles in the range 45-47°C and the average power consumption around 2.68 W.

Raising the frequency up to 920 MHz (Figure 9), during the benchmark setup stage, the CPU temperature overpass the 85°C, triggering the scaling of the frequency to 700 MHz. The enforcing of the CPU bandwidth quota by the RTRM bounds the load to 50%, inhibiting the possibility of re-scaling the frequency up. As a consequence, the temperature reported is fixed in the range 52-54°C and system power consumption around 3 W.

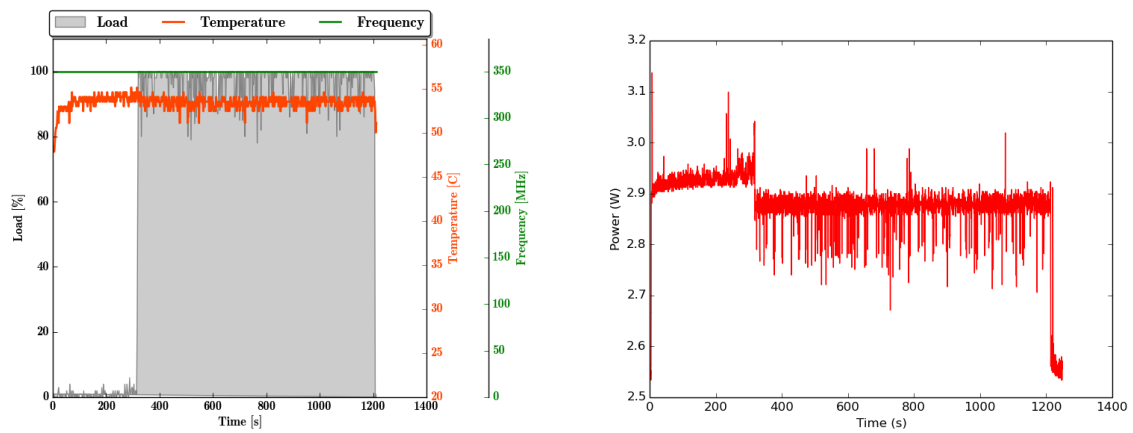


Figure 11: Thermal response of a ARM Cortex A9 dual-core CPU @ 350 MHz, and system power consumption (*Pandaboard ES*), with the RTRM enforcing the allocation of a single core CPU core.

In the case single core case, the benchmark is isolated in only one of the two CPU cores, while the second core is allocated to system processes that usually do not require the full core utilization for long periods. What is it possible to see in Figure 11, is that at 350 MHz the full load of the core leads to a temperature quite stable in the 52-54°C range and a system power consumption of about 2.9 W. These values makes this scenario comparable with the previous, from the power-thermal point of view, and demonstrate how we can have run-time workload conditions for whom CPU bandwidth quota control can be exploited as a mechanism alternative to frequency scaling, to cool the chip or reduce power consumption.

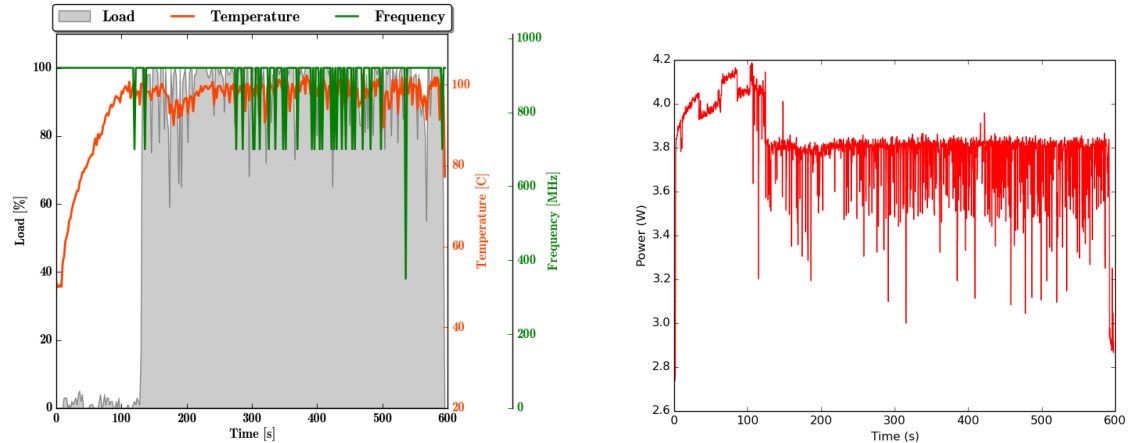


Figure 12: Thermal response of a ARM Cortex A9 dual-core CPU @ 920 MHz, and system power consumption (*Pandaboard ES*), with the RTRM enforcing the allocation of a single core CPU core.

In Figure 11 we faced the case of single core at full speed (920 MHz). The load generated by the benchmark, along with the CPU temperature reaching the 100°C, lead the Linux thermal management policy to perform several frequency scaling, down to 700 MHz and in some cases 350 MHz. Accordingly the system power consumption widely fluctuates between 3 W and 4.2 W.

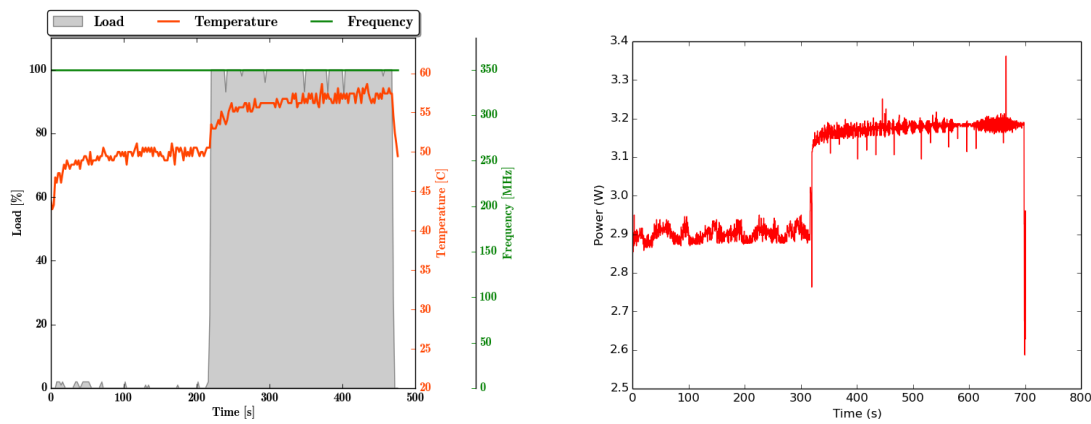


Figure 13: Thermal response of a ARM Cortex A9 dual-core CPU @ 350 MHz, and system power consumption (*Pandaboard ES*), without RTRM enforcing, i.e., 200% CPU bandwidth quota allocation.

Finally, the case of benchmark running on both the CPU cores is of particular interest (Figure 12). Setting the frequency to the minimum value supported (350 MHz), the performance level delivered are quite constant, since the Linux OS does not trigger any frequency scaling. In fact, the CPU temperature is quite stable in the 55-60°C range. The average system power consumption is around 2.85 W, with some peaks slightly above the 3 W.

Conversely, the observations reported in case of CPU frequency set to 920 MHz (Figure 13) show how the full utilization of both the CPU cores raises the chip temperature above the 110°C, triggering continuous frequency scaling actions (to 700 and 250 MHz). This behaviour leads to both performance and power consumption variability. As we can see in the figure, the system power consumption drops to 3.3 W and jumps up to 4.8 W, according to the frequency scaling.

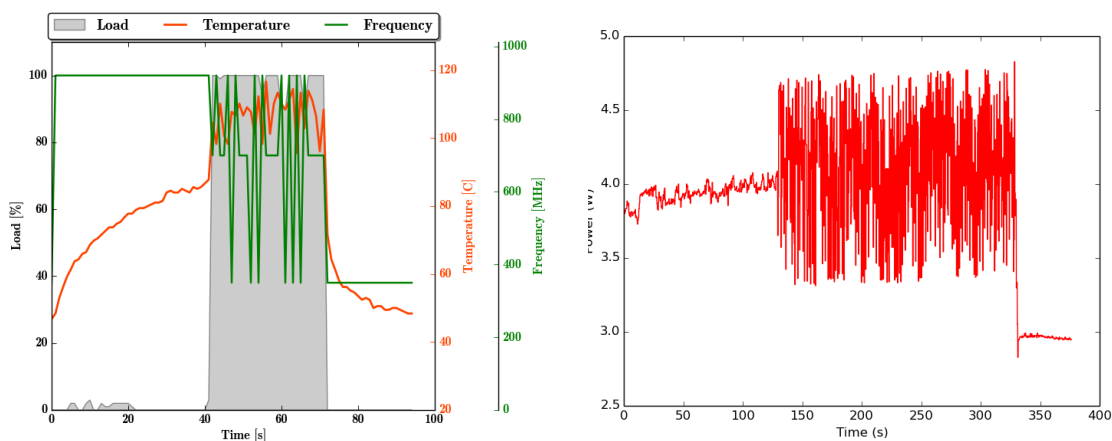


Figure 14: Thermal response of a ARM Cortex A9 dual-core CPU @ 920 MHz, and system power consumption (*Pandaboard ES*), without RTRM enforcing, i.e., 200% CPU bandwidth quota allocation.

A further system behaviour experienced, that is worth to report, is the shutdown operated by the Linux OS as a reaction to the CPU temperature overpassing the 120°C. The experiment in Figure 13 has been indeed repeated, shortening the execution time of the benchmark, by stopping it before the shutdown control was invoked again.

To conclude, the experiments performed so far show how systems based on ARM Cortex A9 dual-core processors can exhibit critical behaviours from the thermal point of view, with the temperature that can raise up to 120°C and trigger a system shutdown. Therefore, chip vendor should provide chip-packaging solutions with proper heat dissipation capabilities.

Furthermore, the system power consumption can span from 1.7 to 5 W, meaning that we can identify suitable power budgets for specific application scenarios, and by exploiting DVFS or bounding the CPU resource allocation, we can enforce the power budget during the workload execution.

Overall, the observations highlight the needs of a Run-Time Resource Manager to safely adapt to run-time variable application scenarios, under thermal and power constraints, also systems based on these high-end embedded CPUs.

Mixed Criticality Aware Resource Allocation

In this section, we provide a concrete picture about how the Run-Time Resource Manager would handle the allocation of computing resources, given a mixed-criticality workload.

The rationale behind the implementation of the resource allocation policy is based on the following points:

- Critical applications must be scheduled in any case;
- The policy must guarantee the minimal performance requirements of critical applications;
- The policy must take into account also non-functional requirements, like the CPU temperature.

We performed some tests on the same development board, introduced in the previous section, to illustrate three different mixed workload scenarios. Moreover, the board does not include any cooling support, neither a heat sink nor a fan, leaving to the policy the complete responsibility of the CPU thermal management task.

Scenario A: One critical (multi-threaded) application

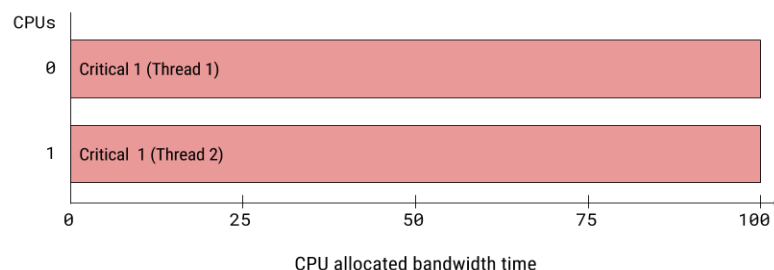


Figure 15 -: Resource allocation output with a single multi-threaded critical application. The two CPU cores are allocated to a single critical application (featuring 2 threads).

This is a trivial choice, since no other applications are competing for the CPU assignment. However, we have seen in the previous experiments how a CPU-intensive workload would easily raise up the CPU temperature in a few seconds, leading the underlying operating system to shut-down the board to prevent damages.

Consequently, in order to avoid the triggering of this emergency action and thus guarantee the system activity, the policy can perform two actions: 1) to shrink the amount of CPU allocated (as shown previously); 2) to do frequency scaling. In these experiments, we show the exploitation of the second option, with the intent of introducing a further capability of the Run-Time Resource Manager.

In this scenario, we ran *fluidanimate* (2 threads configuration) from the PARSEC benchmark suite, as a critical application. We executed just five cycles, which are enough to show the behaviour of the RTRM and its policy.

In the figure below, we considered two cases:

(i) In the first case the CPU frequency is bound to 350 MHz. In this case the peak temperature experienced is safely at 70°C and the minimal requirements of the critical application is satisfied;

(ii) In a second test, the policy was free to set the frequency up to 700 MHz with constraints on temperature and on the minimum QoS for the critical apps. In the first period the frequency has been set to 700MHz, after 10s the temperature reached 100°C triggering the scaling of the frequency down to 350 MHz. Compared to the first test, in this case we gained performance, since the execution time has passed from 27s to 14s, thus not only respecting the constraints but also maximizing the QoS of the Critical App when needed.

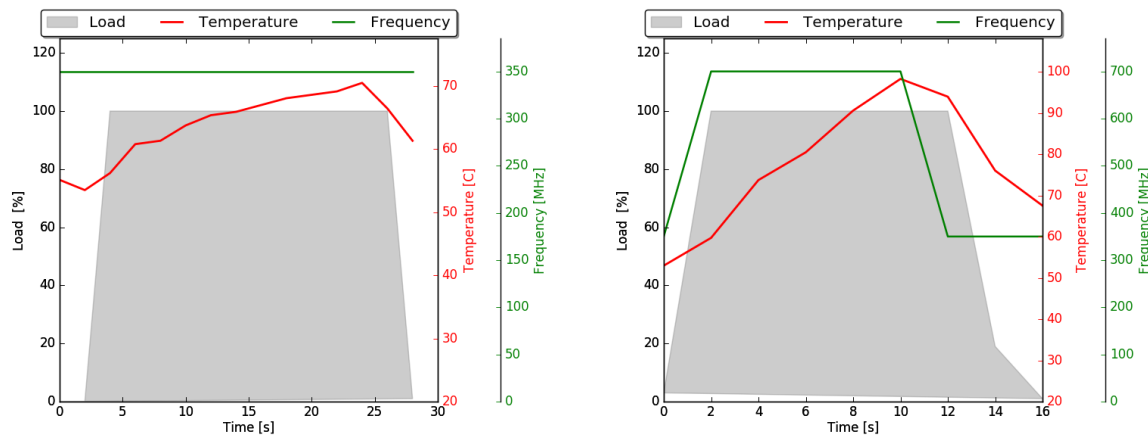


Figure 16 - Load, temperature and frequency values of the CPU (ARM Cortex A9 dual-core) during the testing of Scenario A.

Scenario B: One critical application and two non-critical applications

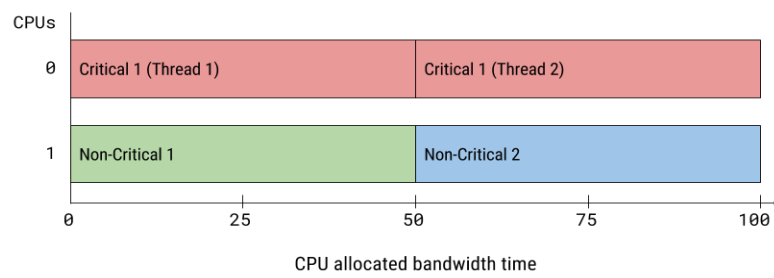


Figure 17 - Resource allocation output with the isolation of the multi-threaded critical application and the two non-critical applications in separated CPU cores.

In this second scenario, the critical application is isolated in one CPU core, while the second core is assigned to a pair of non-critical applications.

In this case, the policy assigns the exclusive usage of one CPU core to the critical application. We assume this has been defined at design-time as the minimum resource requirement for the application execution. This is a mixed-criticality scenario, showing how the RTRM can isolate the execution of critical application from non-critical ones.

For the test, we used again *fluidanimate* as a critical application, while the non-critical workload is made by a pair of “toy” applications requiring with much less CPU.

In the first test, similar to the previous case, the policy is to scale the frequency only when it is needed. Initially the CPU frequency has been set to 700 MHz and scaled it down to 350 MHz once the temperature reached 100°C. When the scaling to 350 MHz occurs, the non-critical apps are stopped to respect the constraint of the min QoS (the two critical thread are now in two different CPUs). When the critical task ends, the non-critical tasks restart running. In the second test, we forced the policy to start setting the frequency to 920 MHz, but after 5s of execution it has been necessary to scale down the frequency to 700 MHz and quickly to 350 MHz. The result has been a slight decrease of performance compared to the first test.

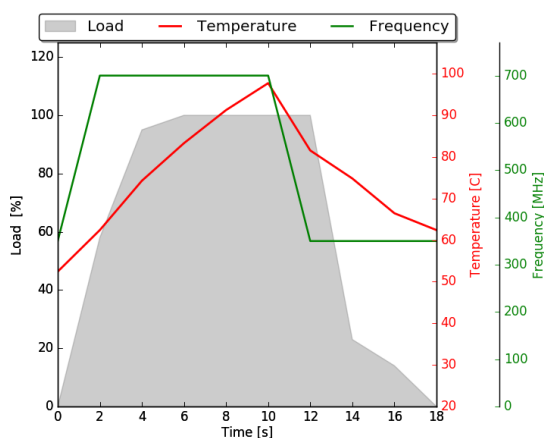


Figure 18 - Load, temperature and frequency values of the CPU (ARM Cortex A9 dual-core) during the testing of Scenario B.

Scenario C: Two critical applications

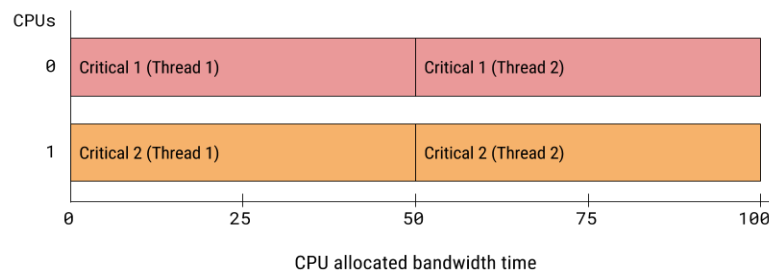


Figure 19 - Resource allocation output with the isolation of two multi-threaded critical applications in two separated CPU cores.

The third and last scenario, the critical applications are two. In this case, since the policy must guarantee the resource requirements to two applications, there is no space for non-critical applications. A CPU core is assigned in exclusive mode to each application. Again, as we have seen in Scenario A, being our critical applications CPU-intensive we need to scale down the frequency in order to keep the system alive.

In the first test, keeping the cores at 350 MHz, the temperature is under control (78°C max). While in the second test, the attempt of forcing the policy to set the frequency at 700 MHz lasted a few seconds. The CPU temperature reached 100°C in 6s, triggering the scaling to 350 MHz. The frequency has been scaled up again only at the end of the execution of both the applications. This case is similar to the Scenario A, however this time having two critical applications running together the advantages in having the possibility to scale up and down the frequency when the temperature is under the threshold does not give any advantage.

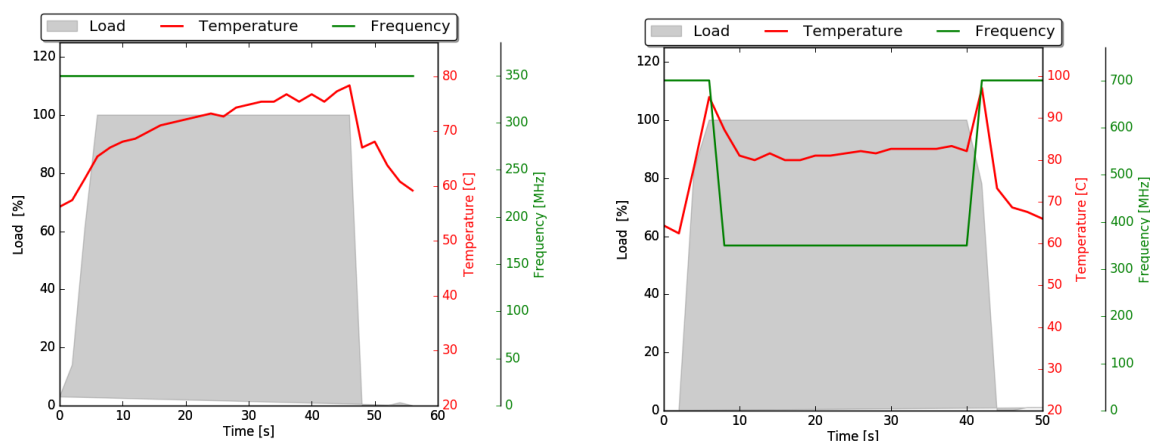


Figure 20 - Load, temperature and frequency values of the CPU (ARM Cortex A9 dual-core) during the testing of Scenario C.

4.2 Battery-Aware Run-Time Management

Dynamic management techniques must be applied also to the power dimension of a system, since the available energy is limited. In an automotive scenario, this is true at least when the car is turned off. This is the case, e.g., for sensing and monitoring equipment natively installed on the car. However, the same system-on-chip may be sold also after-sales, and installed as a distributed battery-powered device. This would enlarge the market for the service providers, as a wider range of vehicles may be equipped with the desired sensing and monitoring equipment. The price is that the battery provides limited power, thus heavily affecting the lifetime of the system: power availability determines whether the functionality can be performed and delivered to the environment/user or not.

The adoption of battery-powered devices introduces new challenges, since energy is not available at any time and at any operating conditions. For this reason, the run-time manager must be extended with knowledge of the power dynamics of the system. The goal is both to be aware of the operating conditions of the device (i.e., how much power is available and how effectively it is used) and to adjust the device operating mode to the surrounding conditions (i.e., to adjust power consumption depending on power availability). As a result, the final device will not only be optimized in terms of energy consumption and responsiveness, but it will also adjust its operating mode to different “scenarios” by suitable trade-offs between contrasting criticalities.

This deliverable introduces run-time battery management, applied to the node used in the CONTREX Automotive Use Case (UC2). To highlight the relevance of the power dimension, we focus on the battery-powered scenario, depicted in Figure 21.

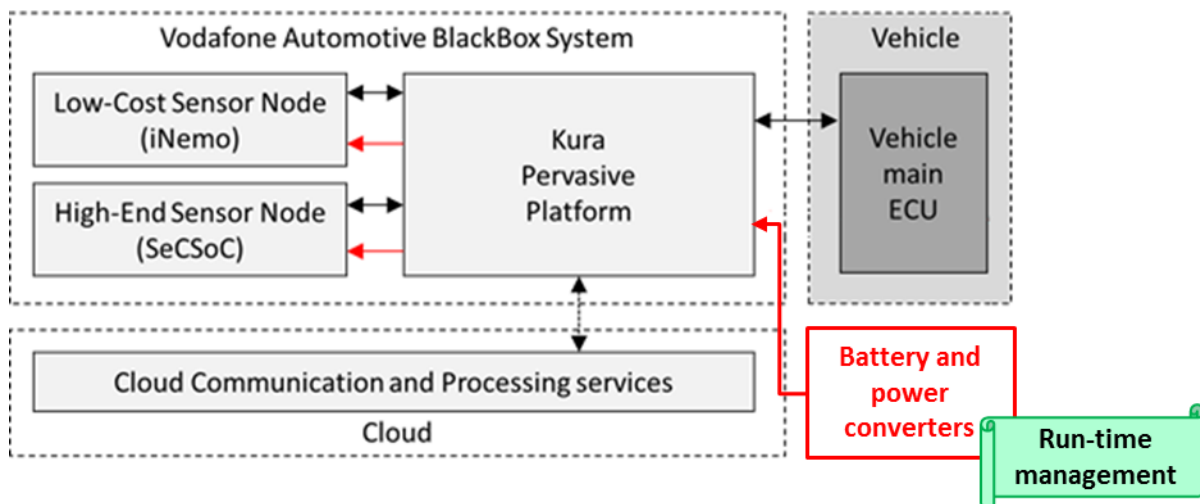


Figure 21. Run-time management applied to the battery-powered version of the node used in the CONTREX Automotive Use Case (UC2).

The reminder of this section presents the activities of POLITO in this scenario, including the construction of the necessary models for power monitoring, their integration in the run-time framework, and the effects on the node used in the CONTREX Automotive Use Case (UC2).

4.2.1 Adopted components for battery and power converters

The battery-powered node does not contain solely the battery. As presented in D3.1.3, each system includes (at least) a power bus, that allows for the energy to combine and propagate

within the system. Furthermore, each component is connected to the power bus through a converter module, necessary to maintain compatibility of voltage levels. The resulting system is depicted in Figure 22.

The UC2 node component is a wrapper for the functional node, producing values of voltage and current demand over time. The node is powered by a Panasonic CG18650CG lithium-ion battery [Pana18650]. The battery converter is a Texas Instruments TPS54122-Q1 DC-DC converter [Texas54122], while the converter used for the functional block is a Linear Technology LTC3407-3 converter [Linear3407].

Of all power dynamics, only two signals are exported to the run-time manager: the battery voltage ($V(battery)$) and the battery state-of charge ($SOC(battery)$). They constitute indeed the necessary information to apply battery-aware management policies, to tune the device operating mode and power consumption to the evolving power availability. Note that also power converters are necessary to accurately model power dissipation. However, power dissipation affects the battery state of charge, and thus is intrinsically taken into account, even if it is not made explicit to the run-time manager.

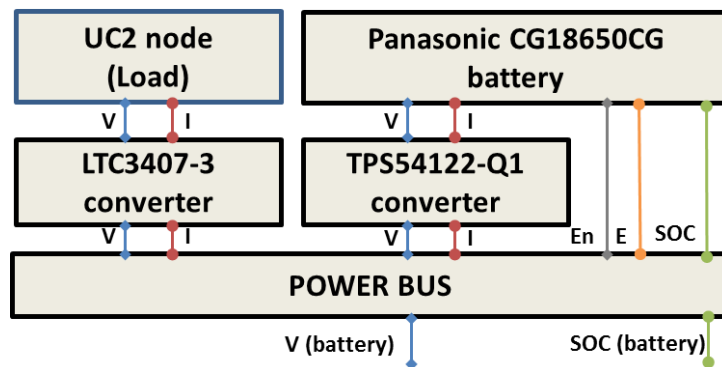


Figure 22. Power components for the node used in the CONTREX Automotive Use Case (UC2).

4.2.2 Adopted models for battery and power converters

The run-time management of the power dimension requires the construction of models for the power components. These models may replace sensors that are not available on the node (e.g., of the battery state of charge, or of the dissipated power over time), and they provide an effective and accurate representation of the power flows.

Battery model

Run-time management requires an accurate modelling of the battery dynamics, that requires to take into account not only average power consumption, but also the distribution of power demand and its impact on battery voltage.

For this reason, the battery is modelled with a level 2 model (D3.1.3). This *electrical circuit equivalent* model mimics the battery behaviour [Petricca2014islpd]. An overview of the model and of its construction process is provided in Figure 23. C_M denotes the nominal capacity, I_B the current drawn from the battery, V_{OC} the open-circuit voltage of the battery, R its internal resistance. Both the resistance and the open-circuit voltage depends on the state of charge (SOC) of the battery, represented electrically as the voltage V_{soc} .

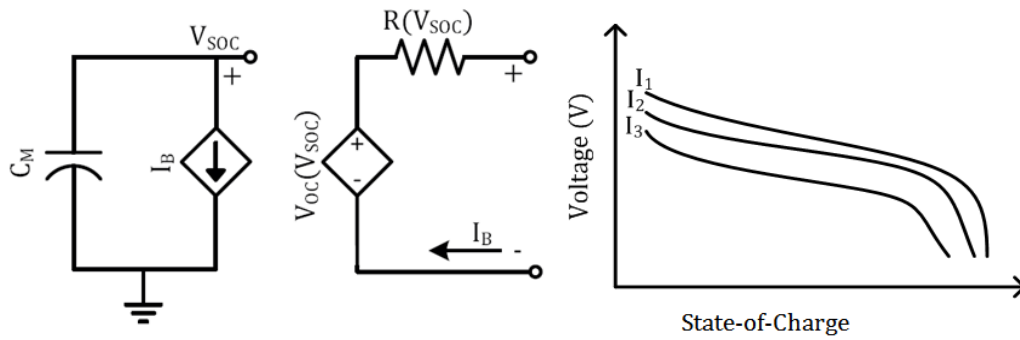


Figure 23. Chosen circuit template (left), and datasheet curves used for model population (right).

Specifications

Nominal Voltage		3.6 V
Standard Capacity ^{*1}		2250mAh
Dimensions ^{*2}	Diameter	18.6 + 0/-0.7mm
	Height	65.2 + 0/-1.0mm
	Weight	Approx. 45g

Discharge Characteristics

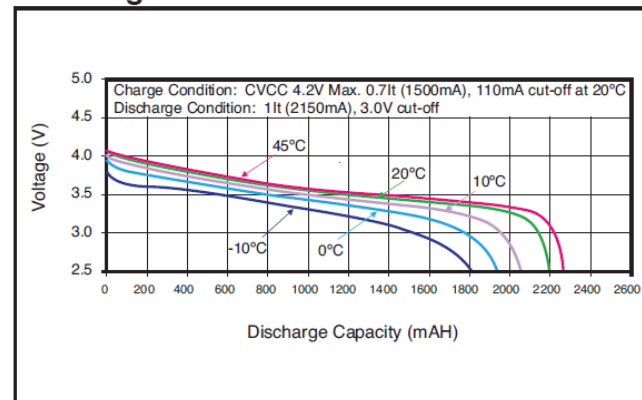


Figure 24. Characteristics of the Panasonic CG18650CG lithium-ion battery from the datasheet.

Population of the model requires availability of voltage discharge curves vs. or SOC (or capacity), possibly for different values of discharge current, as shown in Figure 23.

Given the battery datasheet (Figure 24), the circuit elements are populated as follows. Capacity C_M is simply obtained by converting the nominal battery capacity C_{NOM} (2250mAh, provided with the data-sheet) into A·s, i.e., as $C_M = 3600 \cdot C_{NOM}$. The voltage-controlled voltage generator V_B is implemented as a function $V_B^{C_1}(V_{SOC})$, derived automatically by means of a curve fitting process applied to the (voltage, SOC) points in the datasheet curve corresponding to C-rate C_1 (bottom of Figure 24). Finally, another curve fitting derives a second function $V_B^{C_2}(V_{SOC})$. This allows to derive $V_{OC}(V_{SOC})$ and $R(V_{SOC})$ by solving the equations associated with the electrical circuit.

Figure 25 recaps the resulting model built for the Panasonic CG18650CG lithium-ion battery. This model can be easily encapsulated in the run-time monitoring sub-system (Figure 22), to expose information about battery voltage and state of charge over time. This information is crucial to determine run-time management policies and to guide the functional evolution with run-time power management.

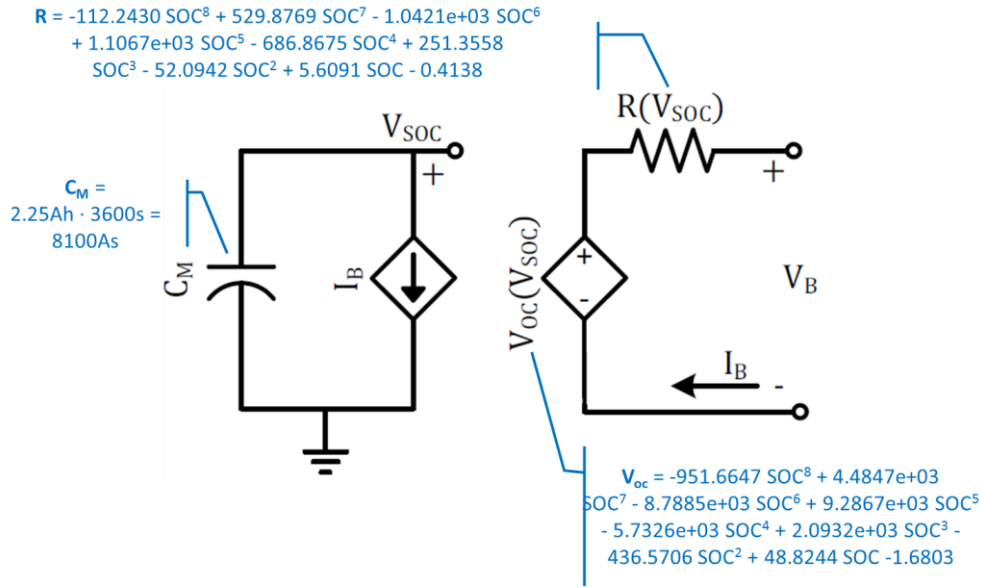


Figure 25. Circuit-equivalent model for the Panasonic CG18650CG lithium-ion battery.

Converter models

Appropriate voltage conversion is necessary, since it is not possible to directly interface components that are at different voltage levels. This is crucial for the node used in the CONTREX Automotive Use Case (UC2): the functional components operate at 1.8V, while the power bus has reference voltage of 3.0V. Furthermore, note that battery voltage strictly depends on the residual state-of-charge. For this reason, it is subject to wide variations over time, ranging from its nominal voltage (3.6V) to almost zero.

Any DC/DC converter simply adapts input power to match output power by mean of appropriate circuitry. This process can be characterized by the efficiency of the conversion η :

$$\eta = \frac{P_{out}}{P_{in}} = \frac{V_{out} I_{out}}{V_{in} I_{in}}$$

where the difference between P_{out} and P_{in} represents the losses of the converter.

Since the DC–DC converter is an electronic device, in principle a circuit-level model consisting of the interconnection of the discrete components would guarantee the highest accuracy. However, this would require a specific model for any specific type of converter (e.g., switching vs. linear) and would slow down the overall simulation, thus heavily affecting the responsiveness of the run-time manager. Thus, adopting a higher level model allows a better compromise between accuracy and performance.

As a result, conversion efficiency is modelled as an equation, depending, in order of relevance, on output current I_{out} , difference between input and output voltage $\Delta V = V_{in} - V_{out}$, and absolute values of V_{in} and V_{out} . Note that approximating converters with their nominal efficiency is too far from the real behaviour of the device. Thus, we rather adopted a functional model, that reproduces conversion efficiency w.r.t. the operating conditions.

Figure 26 recaps the most relevant datasheet information extracted for the Texas Instruments TPS54122-Q1 DC-DC converter. The model is derived from the efficiency versus current plot via curve fitting, by adopting a mixed exponential and polynomial equation. The model

approximates well the converter behaviour, and still it avoids the overhead implied by circuit-equivalent models.

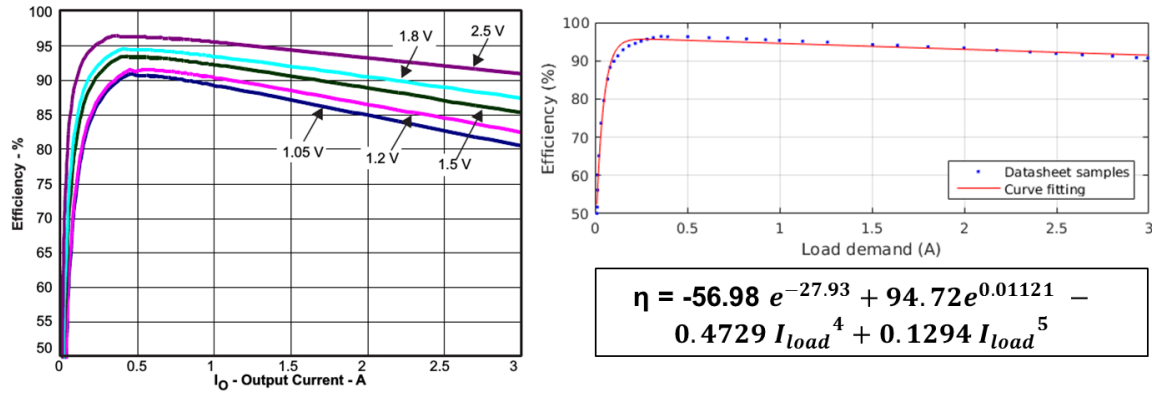


Figure 26. Datasheet information (left) and adopted model (right) for the Texas Instruments TPS54122-Q1 DC-DC converter.

Similarly, Figure 27 recaps the same information for the Linear Technology LTC3407-3 converter. The model is derived from the efficiency versus current plot by fitting the sampled points to a polynomial. The resulting model is a reasonable trade-off between computational complexity and accuracy.

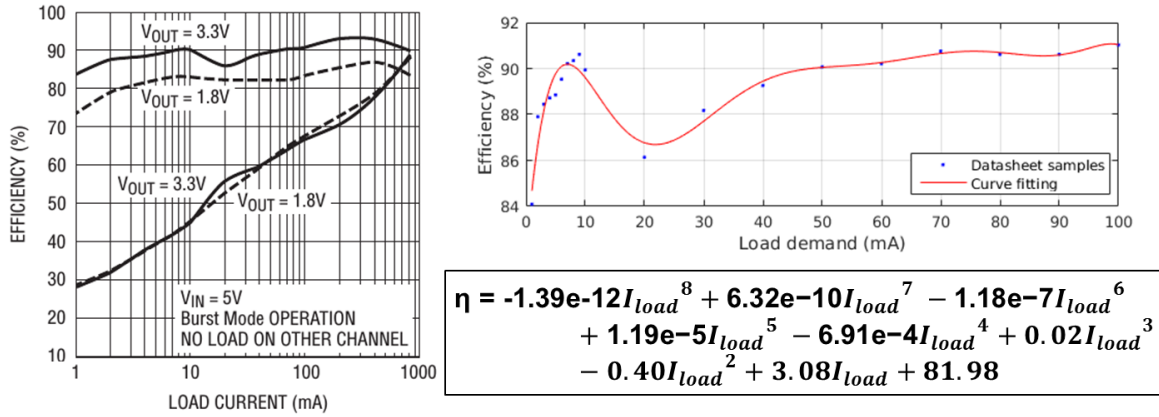


Figure 27. Datasheet information (left) and adopted model (right) for the Linear Technology LTC3407-3 converter.

4.2.3 Run-time management

The possible alternatives for integration in the run-time manager are offline simulation and online simulation. In case of *offline simulation*, the power models are executed stand alone, and they are fed with traces originated by functional simulation. This integration approach would not be beneficial w.r.t. the runtime management infrastructure, as the power estimation would be conducted a posteriori w.r.t. system simulation and management, and this would not allow the application of power-aware policies.

The preferable integration strategy is thus be *online integration*. In this scenario, the battery model is executed simultaneously w.r.t. the functional model and the runtime manager, thus allowing runtime monitoring and to increment the management infrastructure with power-aware policies. Examples of policies may include shutting down non critical services and components whenever the battery has limited residual charge.

To achieve this kind of integration, the runtime management infrastructure must be wrapped by a SystemC module (the SYSTEMC WRAPPER in Figure 28), that exports to the battery model the power samples generated at runtime. The interface of such a SystemC module features:

- An output port for power consumption over time (P); this information easily allows to derive current values, given the constant voltage (1.8V);
- An input port containing the values of the battery state of charge over time (SOC, in percentage);
- An input port E, for the battery residual capacity.

The latter two ports are inputs for the runtime manager, and they are used to apply power-aware policies. This strategy allows to augment the run-time manager with policies that tune the execution of the functional sub-system depending on the power sub-system. An example of these strategies is provided in the next section.

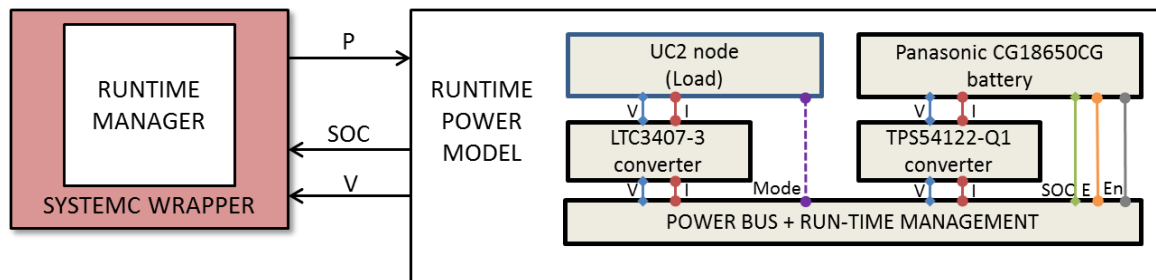


Figure 28. Integration of the battery model in a runtime management infrastructure.

4.2.4 Impact on the node (UC2)

This section provides an example of run-time battery-aware management strategy, applied to the node developed for the CONTREX Automotive Use Case.

Given the distributed nature of the node, battery capacity tends to expire quickly in response to high demand periods. On the other hand, the functionality of the node implies that its lifetime must be as long as possible, to ensure both passenger safety (i.e., crashes must be detected) and product effectiveness (i.e., lifetime must be reasonable).

To this extent, this deliverable compares the result of two different configuration scenarios:

- **Scenario 1:** without battery-aware management. The node executes full functionality at the maximum quality of service level, with no knowledge of the available residual power;
- **Scenario 2:** the level of quality of service is tuned to the battery residual state of charge, thus finding run-time trade-offs between performance and energy.

The scenarios take into account four different execution modes of the node, described in D3.1.3, that are recapped in Figure 29:

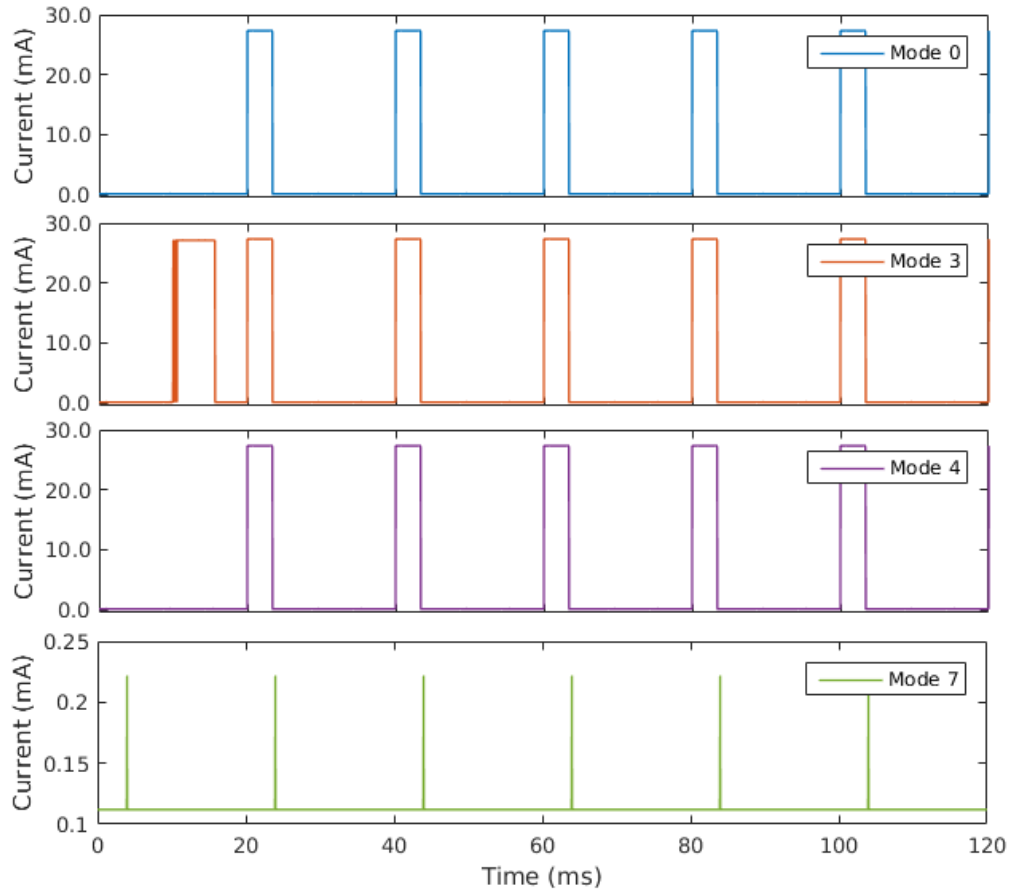


Figure 29. Considered execution modes for the CONTREX UC2 node: full-active at key-on (mode 0), full-active with polling to detect dynamic conditions (including crashes detection, mode 3), reduced system activity to save energy (mode 4), and low-energy activity (with disabled self-calibration, mode 7).

- **Mode 0:** key-on mode, full active device;
- **Mode 3:** full-active device with self-calibration, includes polling to gather dynamic system and environment conditions;
- **Mode 4:** reduced system activity to save energy;
- **Mode 7:** Battery-saving mode; low energy version of the activity, self-calibration and data processing are disabled.

Scenarios and discussion of results

The goal of this section is to mimic the operation of the node with and without the battery-aware management policy. This will highlight that power-awareness is necessary in the run-time manager to apply more effective policies and to prolong node lifetime.

The first scenario reproduces the behaviour of the node *without battery-aware management*. In this scenario, the designer is not aware of the power available in the system, and the main concern is node performance. To this extent, the choice falls on the most complete operating mode, i.e., mode 3. In this mode, the node executes full functionality, and it supports all services, including self-calibration, environment monitoring, and data processing. It is thus very effective at monitoring crashes and in its interaction with the remainder of the distributed

system. The drawback is a high power consumption: the high demand peaks are more frequent and last longer than in other execution modes.

The second scenario applies a *battery-aware management policy*. The models for battery and power converters allow to estimate the power flows in the system at any time. Thus, information about power availability is forwarded to the run-time manager in the form of the battery state of charge and voltage over time. Given this information, the run-time manager can apply a management policy that gracefully decreases the quality of service, depending on the available power:

- With a fully charge battery, the node operates according to mode 3;
- As soon as the battery state of charge reaches the 90%, the node switches to mode 4 to save power;
- When the state of charge is as low as 20%, the node switches to mode 7, to prolong battery lifetime and to increase the effectiveness of the node, that allows monitoring the vehicle for longer.

Power, and in particular batteries, have very different time dynamics w.r.t. the functionality. Even if the power demand of the node changes every μs , in response to task activation and management, the battery can not react to such sudden changes. To reproduce this behaviour, it is possible to *average the power demand at a coarser grain*, still preserving the accuracy of battery modelling. This is a very nice feature, as it reduces the overhead of the battery model, that can be computed less often than the functionality.

To this extent, power demand from the node is averaged over periods of time. A reasonable period, given battery dynamics and functional data, is in the order of ms. This deliverable compares the behaviour of the run-time manager with two grains:

- **1ms period:** the power samples generated from the functionality (one every μs) are accumulated and averaged, so that one sample is provided to the battery model every 1ms. The value provided to the battery is thus the average over 1,000 values produced by the functionality;
- **10ms period:** the power samples generated from the functionality (one every μs) are accumulated and averaged, so that one sample is provided to the battery model every 10ms. The value provided to the battery is thus the average over 10,000 values produced by the functionality.

The length of the period affects both the accuracy and the timing behaviour of the run-time manager.

The following table reports the characteristics of the different configurations:

- **Configuration 1:** scenario 1 (no battery-aware management), one sample every 1ms;
- **Configuration 2:** scenario 1 (no battery-aware management), one sample every 10ms;
- **Configuration 3:** scenario 2 (battery-aware management), one sample every 1ms;
- **Configuration 4:** scenario 2 (battery-aware management), one sample every 10ms.

	Scenario	Time period	Execution mode application	Lifetime	Time for models computation
#1	1	1ms	Mode 3@all time	1,317,639.204s	2,777.154s
#2	1	10ms	Mode 3@all time	1,518,841.960s	387.885s
#3	2	1ms	Mode 3 @ 0s Mode 4 @ 216,681.363s Mode 7 @ 1,117,828.465s	4,741,260.479s	12,557.427s
#4	2	10ms	Mode 3 @ 0s Mode 4 @ 215,769.140s Mode 7 @ 1,176,439.460s	3,896,648.710s	995.134s

The table highlights the *effectiveness of the battery-aware run-time management*. The adoption of power models, and the construction of the battery-aware policy (scenario 2) prolongs the lifetime of the node by a factor 3.55x (compare lifetime of configuration #1 and #3). Indeed, the battery-aware policy tunes the power consumption to the state of charge of the battery, and it allows to perform node functionality even when the battery state of charge is as low as 20%.

Figure 30 depicts the evolution of the state of charge over time for configuration #1 and #3, where the boxes highlight when configuration #3 changes execution mode. The figure highlights the effectiveness of the run-time manager, that prolongs lifetime from 16 days 15 hours to 54 days 21 hours.

The table highlights also the *impact of the sampling period* used for power modelling. The impact is dual. On one hand, using a coarser grain sample implies an approximation. Firstly, the power demand from the battery is less accurate, as demand peaks are averaged over the period. The battery dynamics is sensitive not only to average power, but also to the distribution of power demand and to the presence of peaks. The averaging thus affects battery behaviour.

Secondly, the power system is simulated at larger steps, thus missing crossing events. This implies that the estimation of the battery state of charge is approximated, and that the changes in the execution mode happen at slightly different times (avg. error 2.832%). The consequence is that also system lifetime is approximated (avg. error 16.66%).

On the other hand, accepting a certain degree of approximation allows to reduce the *impact in terms of time* for the run-time manager. Indeed, the simulation of power models achieves a 7.16x speedup for scenario 1, and an even higher speedup of 12.62x for scenario 2. This speedup of one order of magnitude is due to the coarser grain, that enlarges the timestep of one order of magnitude (from 1ms to 10ms).

Figure 31 recaps the results of this contribution. The analysis conducted in this deliverable allows to state that battery-aware run-time management allows to substantially prolong the lifetime of distributed devices, as proven by the 3.55x lifetime for the considered case study (configuration #3 as opposed to configuration #1). Furthermore, the run-time manager may

adopt different trade-offs of accuracy versus performance, to either have a more accurate estimation of residual battery power or a lighter implementation of the battery-aware extension of the run-time manager. This increases the effectiveness of the proposed run-time management strategy.

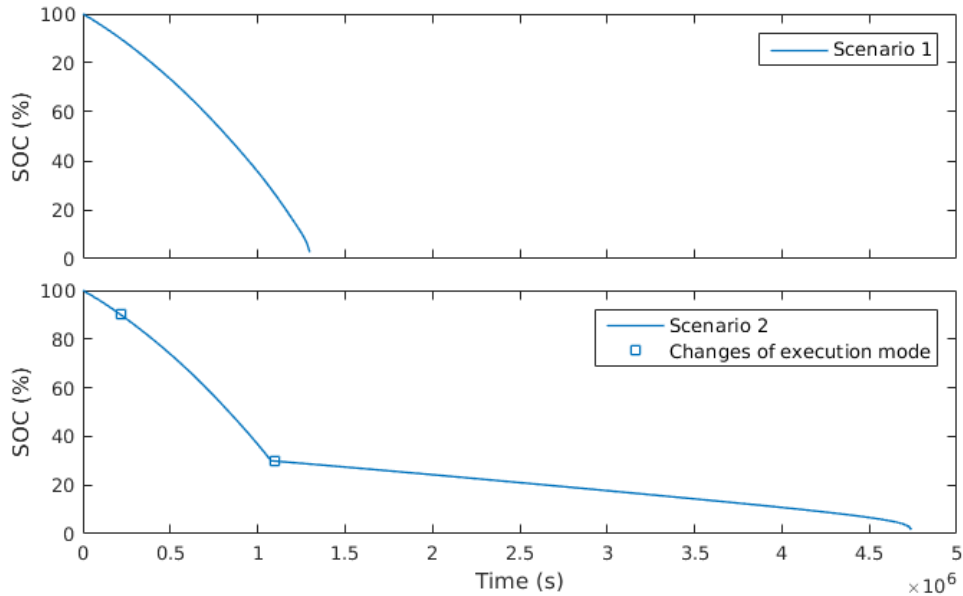


Figure 30. Evolution of the battery state of charge without (configuration #1) and with battery-aware run-time management (configuration #3).

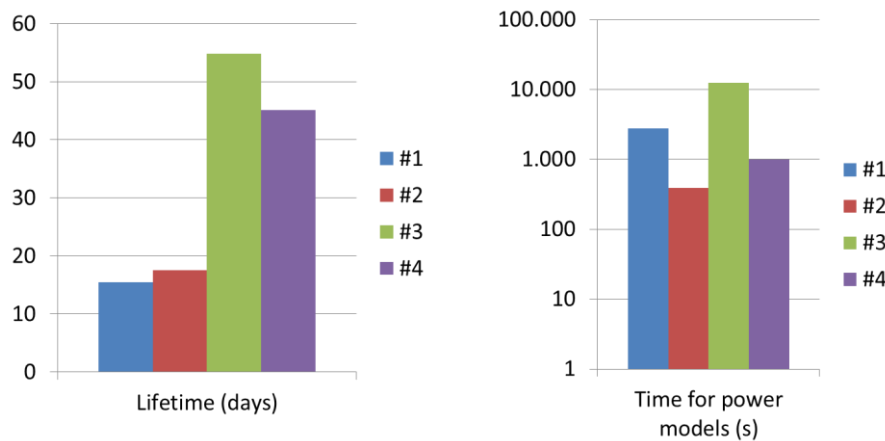


Figure 31. Performance of battery-aware run-time monitoring in the four configurations: scenario1+1ms (#1), scenario1+10ms (#2), scenario2+1ms (#3), scenario2+10ms (#4).

4.3 Lightweight dynamic power management

This section describes the lightweight implementation of the dynamic power manager designed and customized for the requirements coming from the node used in the CONTREX Automotive Use Case (UC2).

Dynamic power management at node level has been implemented by a non-functional manager (called BBQLite) and the related infrastructure. The key components and the necessary abstractions that of BBQLite are the following:

- A module for autonomous management of the power consumption whose behaviour is defined by policies determined at design-time as a result of node-level simulations. The possibility to change the policies at run-time is not envisaged.
- A control layer exposing an API to the application to explicitly perform power management operations. This layer might not be strictly necessary for the application of the specific use-case. Nevertheless, its availability might allow to fine-tune the behaviour of the application in specific critical conditions.
- An interface to the extra-functional properties monitoring layer and/or to the module holding the functional status (i.e. operating mode of the application). Specifically for the automotive use-case, the system status depends also from its extra-functional status (e.g. the level of charge of the battery) and thus it is expected that most of the information required by BBQLite will be accessed indirectly through the functional-status module.

Figure 32 shows the overall architecture of the BBQLite infrastructure, where in blue can be identified the application including all the requested functionalities, in green there are the blocks providing extra-functional information (see Activities in T3.4) and the system status, while in Orange there are the BBQLite components divided in the API abstraction, the core implementation and the design-time profiled policies (in gray).

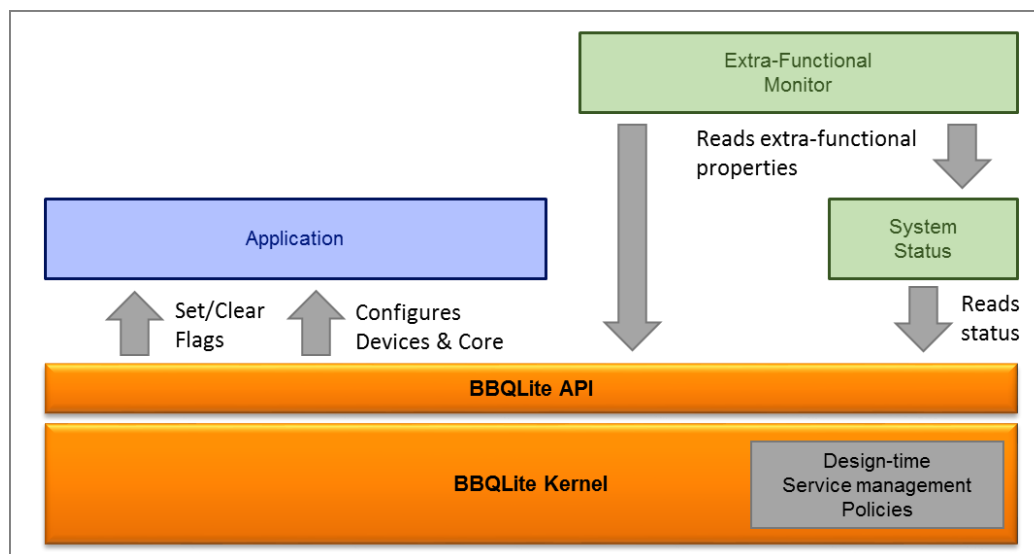


Figure 32 – Architecture of the BBQLite service manager

It is worth noting that the data memory and code size requirements of the automotive use-case are very tight and this implies preferring an optimized code-generation approach rather than a generic service manager dynamically configurable. Additionally the knobs available are the activations/deactivations of application functionalities (services) and power state control of attached devices.

The following list summarizes the logical APIs of the BBQLite manager.

BBQL_GetStatus()

This function returns the current functional status of the application by querying the application specific module holding this information. The implementation of this method is application-dependent and its code is automatically generated based on a simple high-level model specifying types, status encodings and function names for accessing the required information.

`BBQL_GetProperty(property_id)`

This function returns the value of the extra-functional property specified by a suitable property identifier. Properties are stored into objects belonging to the monitoring infrastructure and access to such properties is granted in a very general form through a name-object map.

`BBQL_SetCoreMode(core_mode, core_frequency)`

This function is hardware-dependent and allows changing the operating frequency and operating mode of the microcontroller core. It can be used both by the kernel of the service manager and directly by the application, depending of the specific needs that will emerge from node-level analysis. A typical usage of this function consists in invoking it from the idle task to switch the core to a low-power mode and from the tasks and the interrupts service routines to bring the core back to its normal mode of operation. It is worth noting that the behaviour of this function – if needed – can be made dependent to application-level flags managed, in turns, by the kernel of the service manager.

`BBQL_SetDeviceStatus(device_id, device_mode)`

This function collects the code for managing the hardware devices similarly to what the function `BBQL_SetCoreMode()` does for the microprocessor core.

`BBQL_SetFlag(flag_id, flag_value)`

Specific functionality performed by the tasks and the interrupt service routines can be enabled, disabled or configured according to the policies defined by the service manager. This function is invoked by the service manager itself to “implement” the policies defined at design-time.

`BBQL_GetFlag(flag_id)`

This function is the counterpart of the previous function and is used by interrupt service routines and tasks to change their execution paths (i.e. executing or skipping some processing).

`BBQL_Manage()`

This function is the kernel of the service manager. It uses the information stored in a data structure representing the service management policies and modifies the behaviour of the application using the abovementioned functions. This function shall be either called periodically or upon specific system events.

As mentioned in D3.2.3 (and previous D3.2.2 deliverable), the integration of BBQLite in the application is rather simple as it only requires to add a few macros in selected code points. Such macros expand to function calls to the actual BBQLite API. In particular a call shall be added in the idle operating system task to bring the microcontroller into the right power

mode and a call shall be added to each interrupt service routine (UART, SPI, I2C) to bring the microcontroller back to its normal operating mode (e.g. to wake-up from sleep mode).

The integration and configuration of BBQLite in the final and complete application on the iNemo platform has been completed and tested. The following figures show some execution traces obtained with a logic analyzer and average current measurements in three different power modes.

The first figure below shows the execution times of the different jobs performed by the analysis task during normal mode of operation. It worth noting that these execution times do not depend on the operating mode selected by BBQLite, nor on the data being processed. Changing the operating mode, in fact, has the effect of activating or deactivating such jobs. These figures are the same that have been estimated using uVision Keil cycle-accurate simulation and that have been used as input to the non-functional simulator N2Sim.

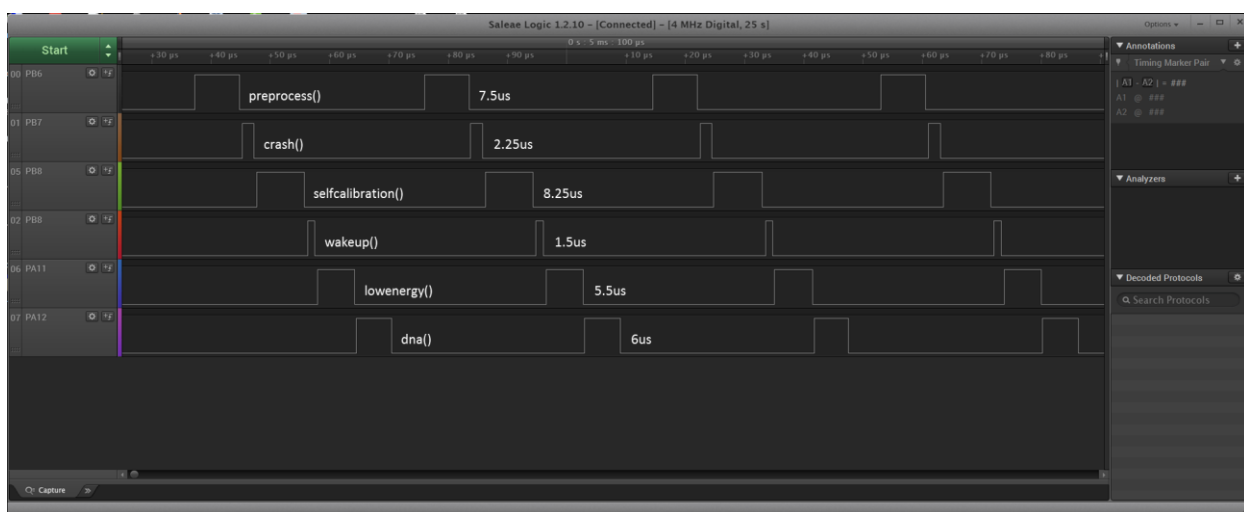


Figure 33 – Acceleration analysis jobs timing

The second trace shows the two tasks of data acquisition (below) and data analysis (above), again during normal operation. As it can be seen, the acquisition period is approximately 15ms: this results from the fact that the sampling frequency of the accelerometer is 1344Hz and an internal hardware FIFO is used and data is read when the FIFO level reaches 20 samples.

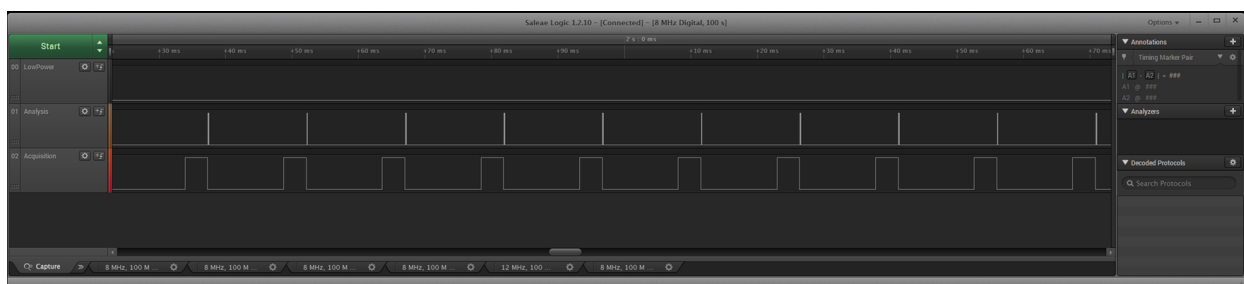


Figure 34 – Acquisition and analysis tasks periods

When 20 samples are read from the device, they are filtered as a single chunk, decimated by a factor of 5, inserted into a software queue and passed to the analysis task. This task pops elements from the software queue until not empty and process each sample, passing it through

the different jobs that are active, depending on the operating mode. A detail on the duration of the analysis task shows that after the acquisition task, $20/5=4$ executions of the jobs are performed.

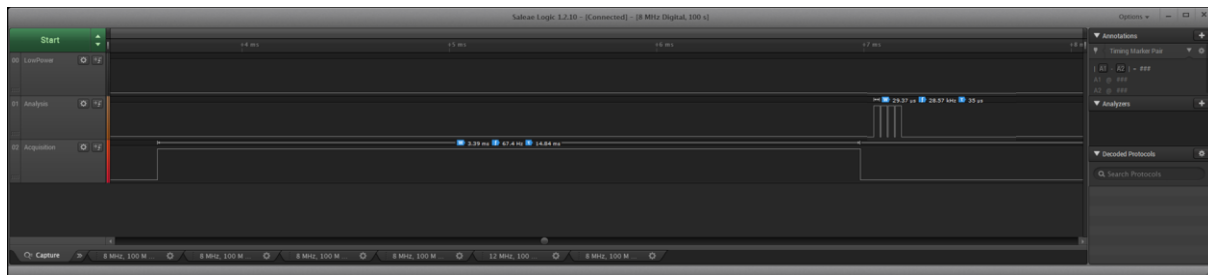


Figure 35 – Acquisition and analysis tasks timing

When the low-power mode is entered upon request from BBQLite, the sampling frequency is reduced and some of the analysis tasks are disabled. The frequency reduction is clearly shown by the figure below, where the topmost trace indicates whether the low-power mode is active or not, while the two other traces are, again, the acquisition and analysis tasks.

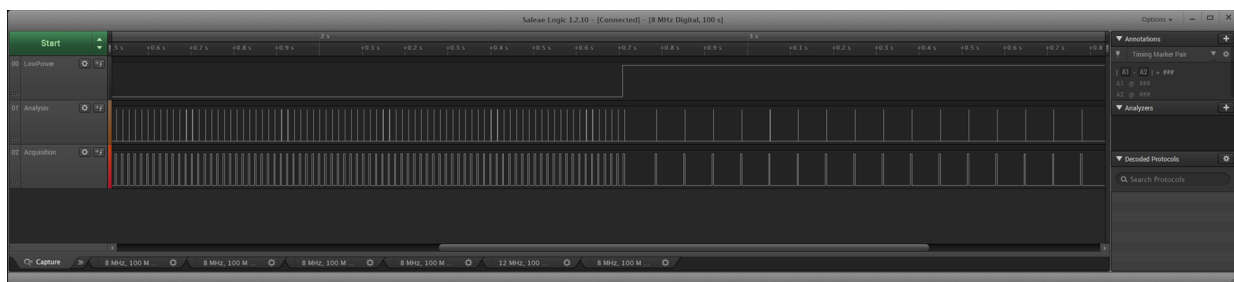


Figure 36 – Acquisition and analysis tasks timing: switch to low-power mode

Measurement of the average power consumption (actually, average current absorbed by the system) have also been performed leading to the results summarized in the table below and shown in Figure 37, where the constant contribution due to the EVB and connected measurements/communication subsystems has been isolated and the power consumption reduction with respect to the original system (i.e. full functionality but no power management) have been calculated.

Operating mode	Measure (EVB+iNemo)	iNemo Only	Reduction
Without BBQLite	28.70 mA	14.85 mA	N/A
With BBQLite in normal mode	17.06 mA	3.21 mA	78 %
With BBQLite in low-power mode	14.38 mA	0.53 mA	96 %

Figure 37 – Average power consumption of the iNemo SoB

Further traces on the usage of the BBQLite power manager have been included in D3.4.3 for highlighting the characteristics of the monitoring infrastructure to extract EFP both for internal and external usage.

4.4 Run-Time management for the Cloud

The cloud platform run time management is responsible for the management of cloud account and users, remote devices control, data collection, and use case specific application. The run time management is based on:

- a web based console and
- a REST API.

The web console is devoted to the management of every day cloud-related activities, while the REST APIs provide support to administrators and developers in the implementation/execution of use case specific applications (see Figure 38).

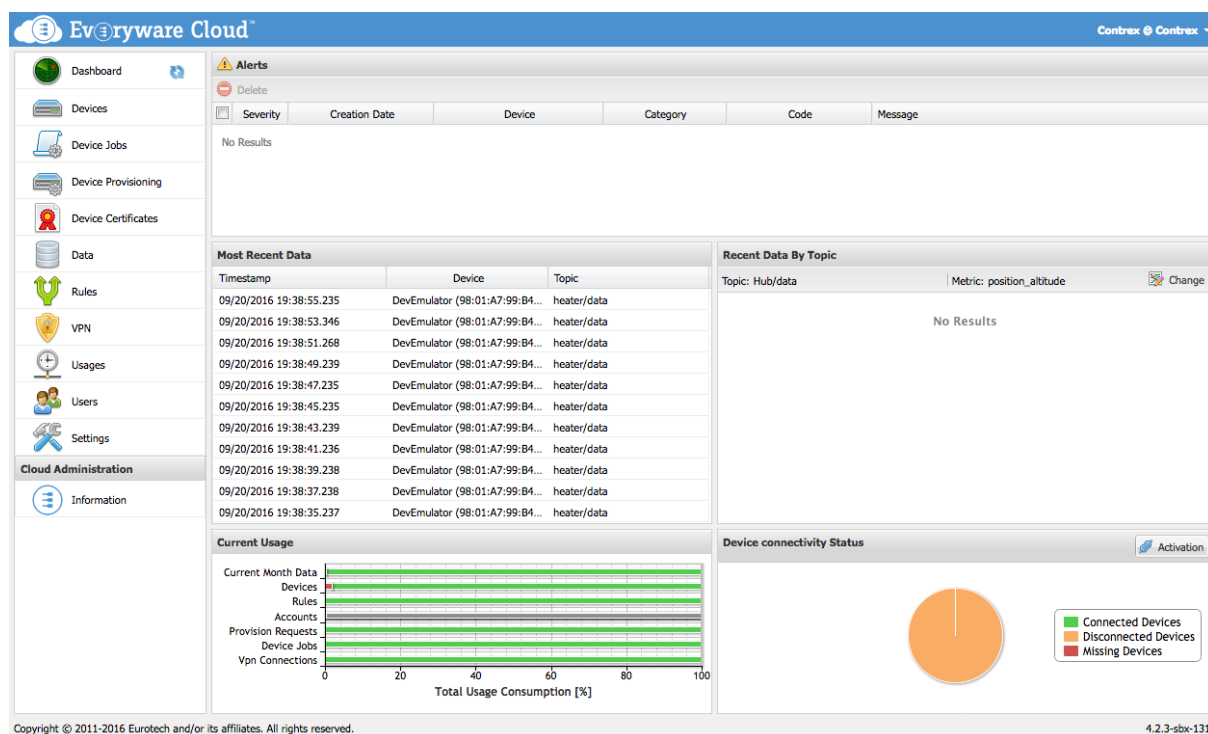


Figure 38 – Screenshot of the cloud console dashboard.

4.4.1 Managing accounts and users

The web-based console allows the management of cloud accounts and users. The cloud platform adopts a centralized Role-Based Access Control (RBAC) security model, where each account may have multiple users and where a different set of permissions may be granted to each user. When a user connects to the cloud using his/her credentials, he/she will have access to the limited set of functionalities granted by the assigned permissions. The same principle applies to users accessing the cloud platform through the cloud console, through the cloud broker or through the cloud REST APIs.

Each account must have at least one user with administrator role (which implicitly grants all the permissions). Other users may have limited access to the cloud determining whether they can:

- view or manage the account or users;

- view or manage data or rules;
- connect to the MQTT broker (see D3.4.1).

The following table defines the permissions for users. The administrator has all the following permissions assigned automatically.

Permission	Allows user to
account:view	View the account detail
account:manage	Manage the account
broker:connect	Connect to the cloud broker
data:view	View data on this account
data:manage	Publish data on this account
device:view	View the devices on this account
device:manage	Manage the devices on this account
rule:view	View account rules
rule:manage	Add or modify rules
user:view	View users
user:manage	Modify, add, or delete users

The following figures illustrate the cloud web console that provide accounts and users' information and related functionalities.

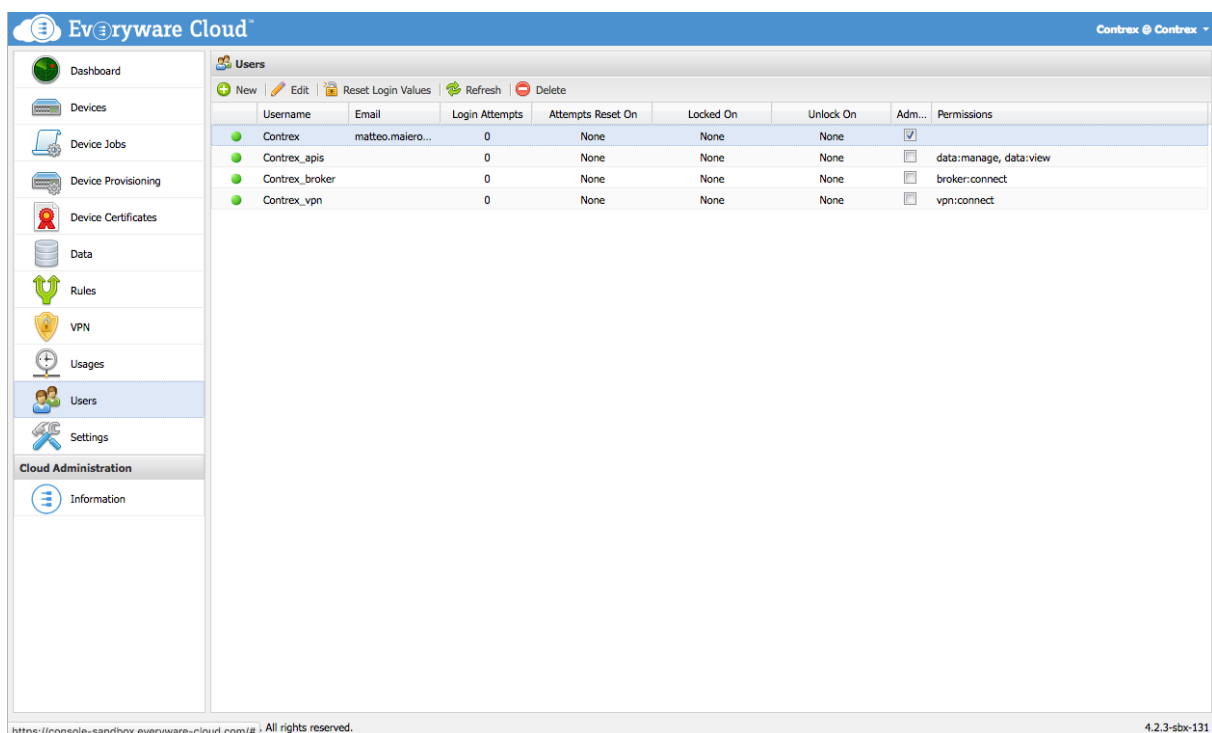


Figure 39 – Screenshot of the account and user summary.

The screenshot shows a dialog box titled "Update User: Contrex" with a close button (X) in the top right corner. It has two tabs: "User Information" (selected) and "Two factor authentication".

User Information Tab:

- Information Section:**
 - * Username: Contrex
 - * Password: [masked with dots]
 - * Confirm Password: [masked with dots]
 - Note: If the credentials are used by devices, update them accordingly in the device configuration.
 - Display Name: [empty field]
 - Email: matteo.maiero@eurotech.com
 - Phone Number: [empty field]
- Status Section:**
 - Status: Enabled (dropdown menu)
 - Last Login On: Today 3:29:14 PM
 - Failed Login Attempts: 0
 - Attempts Reset On: None
 - Locked On: None
 - Unlock On: None

At the bottom are "Submit" and "Cancel" buttons.

Figure 40 – Screenshot of the account and user settings of the cloud console.

4.4.2 Managing Kura enabled devices

The cloud console provides a specific page that reports the status information about the devices that are connected to the current cloud account.

Every device that connects to a specific account in the cloud platform has to follow a connection procedure. Upon establishing a connection to the cloud, the device publishes a Birth Certificate message that contains information about the hardware and software configuration. Furthermore, the device will notify the cloud platform when it disconnects gracefully by publishing a Death Certificate message. Through the characteristics of the MQTT protocol, should a device lose its connection to the Broker, the cloud platform is informed with a Last-Will Testament message. These three life-cycle messages allow the cloud to track the current status of each device in a specific account.

The devices panel in the console allows account administrators to monitor the status and review the full life-cycle history of their devices. This is possible because the Kura framework allows devices to be remotely configured and controlled while they are connected to the MQTT broker. The following figure illustrates a mock-up of the devices page of the cloud console.

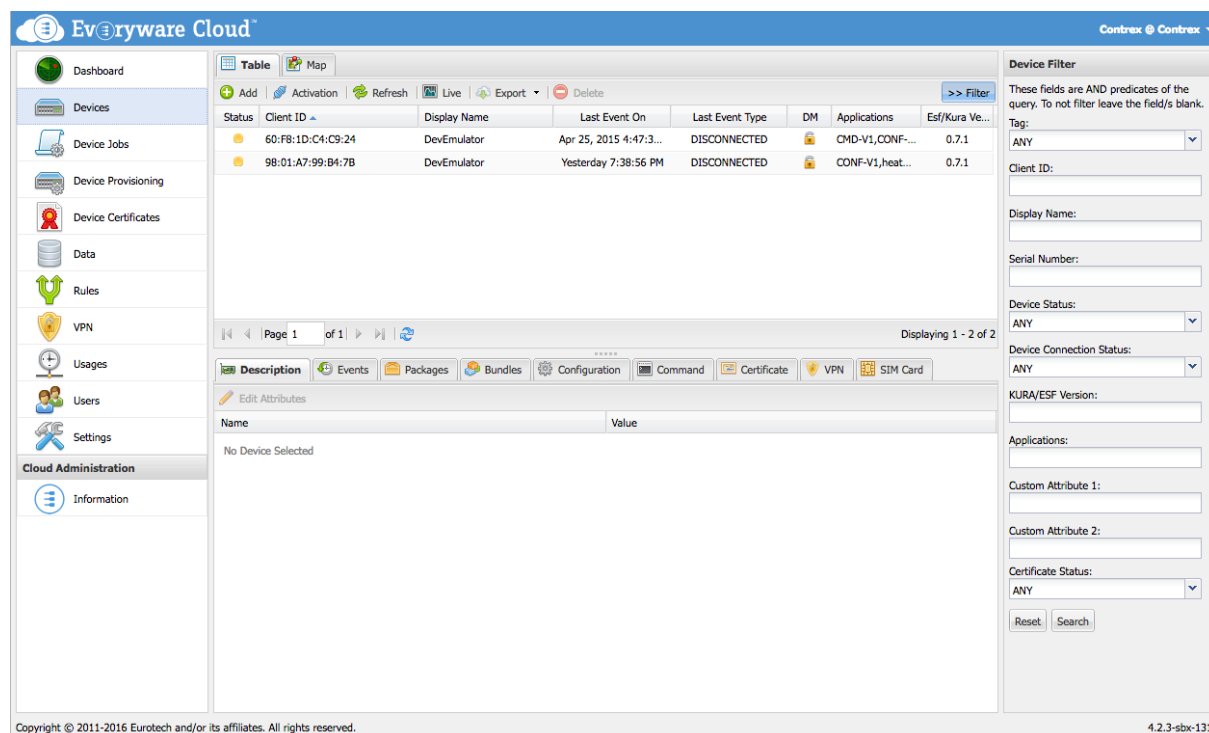


Figure 41 – Screenshot of the devices page of the cloud console.

The Table tab lists all the devices that have previously been connected to the specific cloud account. This table shows the status of the device in real time, exporting device information and deleting a device. Deleting a device only deletes the profile and history information, while it does not delete the data published by the device itself.

For each device the table reports: the status, the client identifier, the name and model of the device, the last report date and the uptime.

The status provides a visual indication of the device status in relationship to its connection to the cloud broker: green indicates that the device is connected properly, yellow that the device performed an orderly disconnect and red that it has lost the connection to the broker.

The client identifier is a unique identifier within a cloud account that represents a single gateway device. The Client ID maps to the Client Identifier defined in the CONNECT message of the MQTT specifications. For a gateway, the MAC address of its primary network interface is generally used as the Client ID. The cloud platform client libraries and the Kura framework follow the MAC address convention. Client ID identifies a physical device and its MQTT connection. The hierarchical nature of the MQTT topic namespace can be leveraged to describe a topology of gateway devices, applications running on the gateways, and/or sensors connected to the gateways.

Finally, the device page of the console provides the profile of the device, the historical data, the packages and bundles installed/running on the device and allows to manual issue commands with a terminal session.

4.4.2.1 Device management security improvements

The device management has been improved with new security-related features. The new functionalities have been introduced mainly to provide security and reliability mechanisms

that guarantee that the control messages, received by a device from the Cloud Platform, have not been altered by an attacker.

The security mechanisms are based on the verification of signatures that are exchanged between the remote devices and the cloud platform. The Cloud platform creates a signature of the message and adds the signature to the message itself, when it is sent. The device running Kura extracts the signature from the message, recalculates the signature from the message content and compares the two signature. The control message is accepted and processed only if the extracted signature and the calculated one match.

To increase the security level, the signatures is encrypt both at cloud and device sides.

The following figure illustrates the certificates tab of the cloud console.

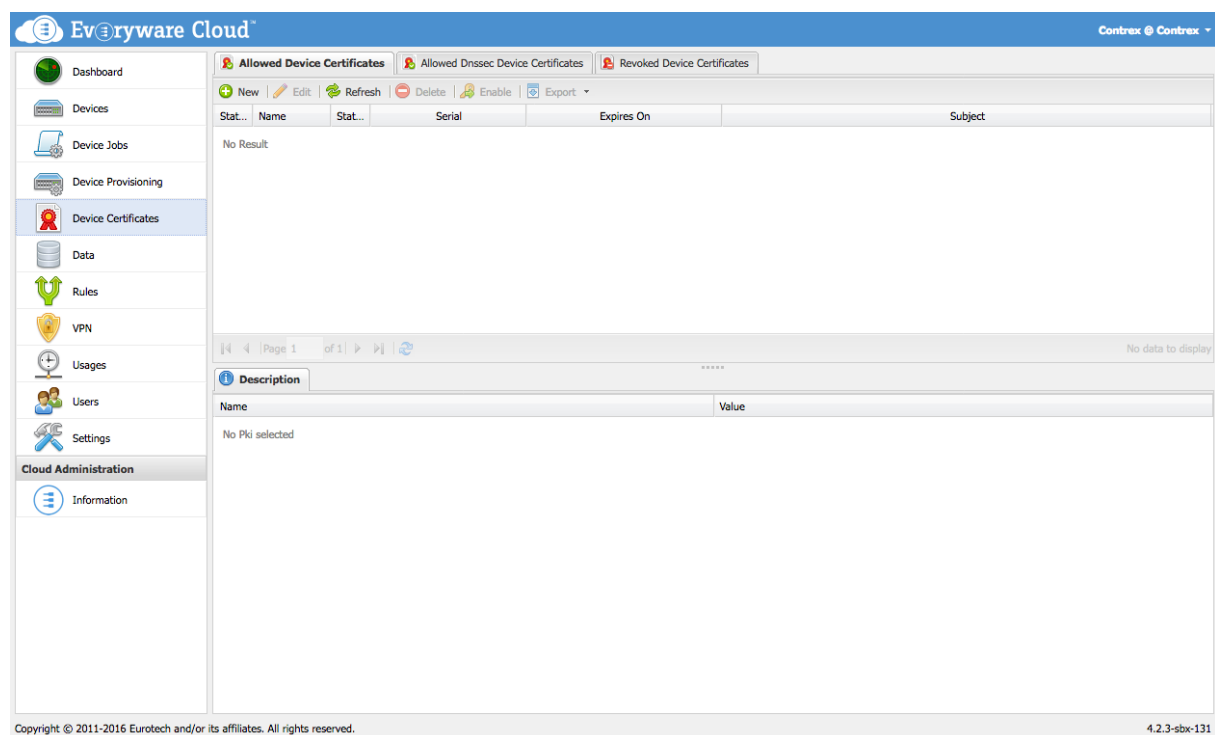


Figure 42 – Screenshot of the certificates tab of the cloud console.

4.4.3 Viewing collected data

The cloud console allows the user to display, query and inspect the data collected from the remote devices. The “Data by Topic” and “Data by Asset” panels allow to perform these operations. As data is published to the cloud, the platform automatically creates data registries to track the following information:

- the asset from which the data originated,
- the topic to which the data was addressed,
- the metric names that are contained into each message and, therefore, the names of the metric which received any values for a given asset or a given topic.

It is important to emphasize that topics are hierarchical: data published to a topic called “state/city” is implicitly addressed to “city” and “state”. This is possible because the cloud platform performs topic-level aggregations automatically for topics up to 5-levels deep.

Through the data registries, the cloud platform provides simplified data browsing and data discovery. The data queries in the console operate on the data published using metrics. For more information and examples about assets, topics, data, and metrics, please refer to the deliverables D3.2.1 and D3.4.1.

Using the platform data registries, the cloud console allows browsing and querying of connected devices data in two ways:

- “Data by Asset” – This method is appropriate when the query on the device data follows the physical view, knowing the device and asset that originally produced the data.
- “Data by Topic” – This method is appropriate when it is not important to know the device that published the data but rather the topic to which that data was addressed. Typically, this is the case when you defined a topic namespace for your application and you leveraged the cloud platform to aggregate your data at each topic level.

The following figure illustrates a mock-up of the data page of the console, where a query by asset has been performed.

The screenshot displays the Evryware Cloud console interface. On the left is a sidebar with navigation icons and labels: Dashboard, Devices, Device Jobs, Device Provisioning, Device Certificates, Data (highlighted), Rules, VPN, Usages, Users, Settings, and Cloud Administration. The main content area is titled 'By Asset' and shows a table of available topics and metrics. Below this, there is a 'Query' button and a 'Results Table' tab. The 'Results Table' displays a list of data points with columns for Timestamp, Asset, Topic, and various metrics (tempera..., errorCode). The footer of the console shows 'Copyright © 2011-2016 Eurotech and/or its affiliates. All rights reserved.' and '4.2.3-sbx-131'.

Timestamp	Asset	Topic	tempera...	tempera...	tempera...	errorCode
09/20/2016 19:38:55.235	98-01:A7-99:B4:...	heater/data	19.5	5	30	0
09/20/2016 19:38:53.346	98-01:A7-99:B4:...	heater/data	19.25	5	30	0
09/20/2016 19:38:51.268	98-01:A7-99:B4:...	heater/data	20.25	5	30	0
09/20/2016 19:38:49.239	98-01:A7-99:B4:...	heater/data	20	5	30	0
09/20/2016 19:38:47.235	98-01:A7-99:B4:...	heater/data	19.75	5	30	-937045...
09/20/2016 19:38:45.235	98-01:A7-99:B4:...	heater/data	19.5	5	30	0
09/20/2016 19:38:43.239	98-01:A7-99:B4:...	heater/data	19.25	5	30	0
09/20/2016 19:38:41.236	98-01:A7-99:B4:...	heater/data	20.25	5	30	0
09/20/2016 19:38:39.238	98-01:A7-99:B4:...	heater/data	20	5	30	0
09/20/2016 19:38:37.238	98-01:A7-99:B4:...	heater/data	19.75	5	30	0
09/20/2016 19:38:35.237	98-01:A7-99:B4:...	heater/data	19.5	5	30	0
09/20/2016 19:38:33.243	98-01:A7-99:B4:...	heater/data	19.25	5	30	0
09/20/2016 19:38:31.235	98-01:A7-99:B4:...	heater/data	20.25	5	30	0
09/20/2016 19:38:29.238	98-01:A7-99:B4:...	heater/data	20	5	30	0

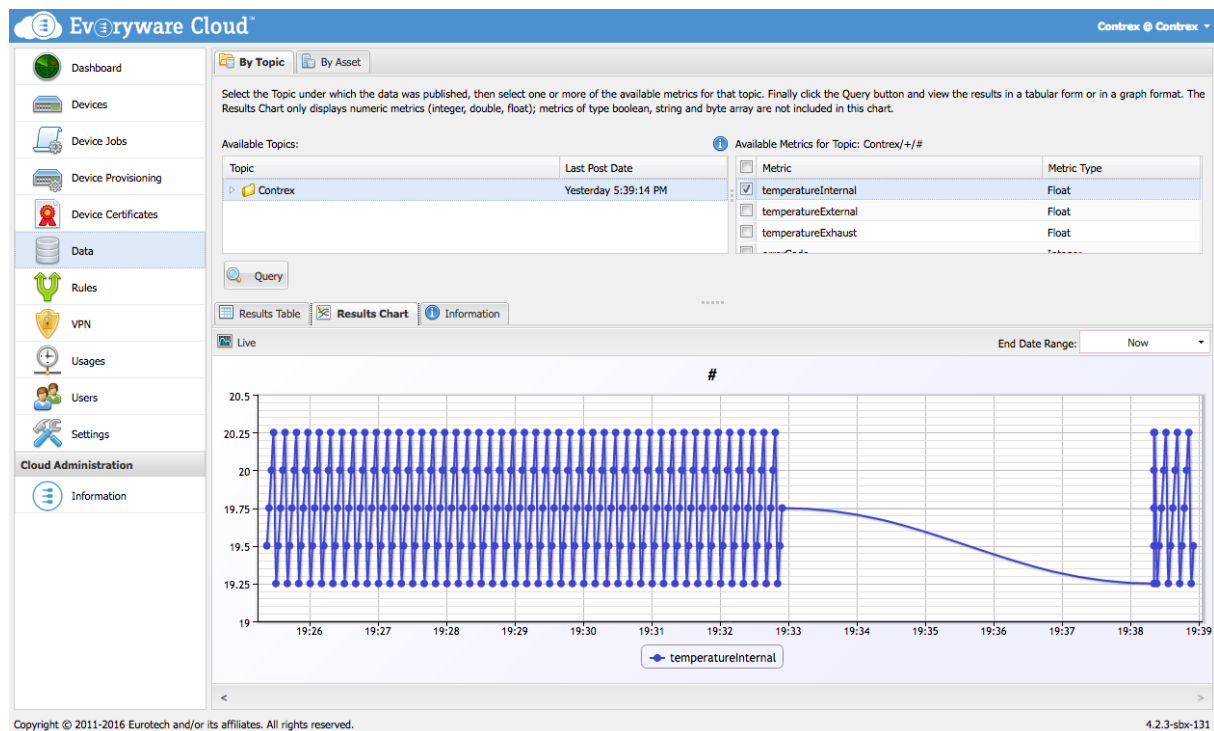


Figure 43 – Screenshots of the data query page of the cloud console.

The “Data by Asset” panel queries the data by asset: by default, an asset represents the device gateway (and its MQTT connection) from which the data was originated. To view the data, select an asset in the Available Assets list, click the checkboxes in the Available Metrics list for the metrics you want to display, and then perform the query.

The “Data by Topic” panel demonstrates the powerful flexibility of cloud’s platform schema-less data storage. In the query process, the limitation is only determined by the schema that devices use when publishing. To display collected data, select the topic or topic branch from the Available Topics list, select the data metrics to display from the Available Metrics list, and then click Query. The Console will fill the table with collected data.

4.4.4 Using cloud rules

The rules panel in the console provides real-time business intelligence driven by data. The rules panel allows the user to perform the following tasks:

- create a new rule,
- edit an existing,
- refresh the list of rules,
- delete a rule.

The following figure describes a mock-up of the rules page of the cloud console.

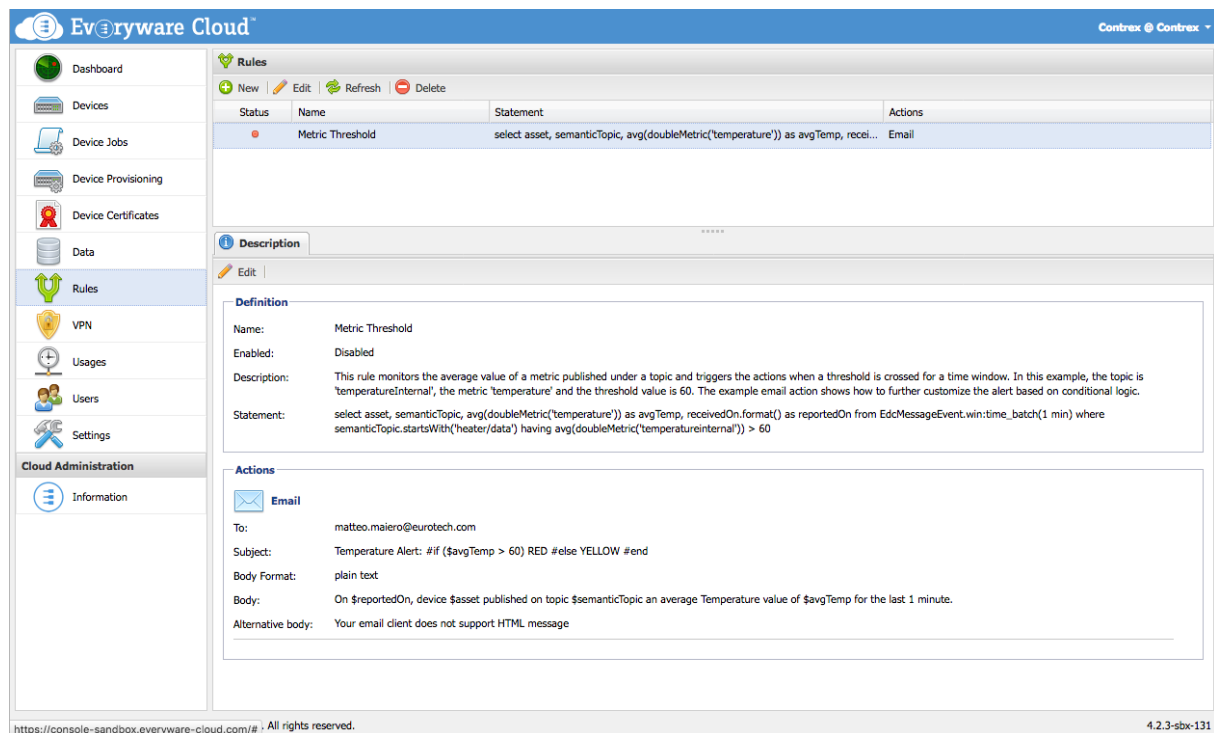


Figure 44 – Screenshot of the rules page of the cloud console.

For each rule, the interface provides the status, the name, the statement and the related actions. The status can be, respectively, green or red if the rule is enabled or disabled. The name is the name that the user associated to the rule. The statement explains what the rule should perform and is written using the Esper event processing language (<http://esper.codehaus.org/>). Finally, the actions are the tasks performed by the rule whenever it matches the conditions described in the statement. The rule can trigger the following actions:

- send an email,
- send a SMS message,
- send a real-time message via Twitter,
- publish an event back to the cloud (via MQTT),
- perform a REST call.

4.4.4.1 New rules actions

The improvement activities on the cloud platform required the update of the supported services that can be used to send messages when the associated statement is triggered. In particular, a new set of services have been implemented and the new related rule actions are:

- IFTT: is an abbreviation of "If This Then That" and stands for a web-based service that allows users to create chains of simple conditional statements, called "recipes";
- Alert: allows to create an alert that will be displayed in the cloud platform dashboard.

4.4.4.2 Rules example

The next example illustrates the Rules syntax in the context of the UC2 application.

The screenshot shows a 'New Rule' window with a 'Definition' tab. The 'Status' is set to 'Enabled'. The 'Name' is 'Power Consumption rule'. The 'Description' is 'Send e-mail whenever the power level exceeds the given threshold in a defined period of time'. The 'Statement' field contains the following Esper query: `select *, intMetric('power') as power from EdcMessageEvent.win:time(10 minutes).std:firstunique(intMetric('power')) where semanticTopic.startsWith('contrex/power/values') and intMetric('power') > 10`. A link to 'Esper Documentation' is provided for more information on the Statement syntax. The window has 'Submit' and 'Cancel' buttons at the bottom.

Figure 45 – Rule definition

The rule creation window has two main sections that are displayed as a “Definition” and an “Actions” tabs.

Referring to the “Definition” tab, it allows to define the rule in terms of a statement that will specify how the rule will be triggered. The available fields are:

- Status: that specifies if the current rule is enabled or disabled;
- Name: the rule name;
- Description; the rule description that allows to give a more user-friendly description of what the rule does.

The Statement field allows to specify the effective rule using Esper syntax language that require SQL-like statements to derive and aggregate data from an event stream.

The previous figure depicts how a rule in the domain of UC2 use-case has been defined: in the UC2 context, a high power consumption can be the symptom of an electric malfunctioning of one or more components with, potentially, very negative effects on the service quality. For this reason, a corresponding rule that can be defined in the cloud platform is based on the power values acquired from the vehicle and sent to the cloud. It’s plausible that the rule could be triggered when the measured power consumption exceeds the value of 10 W in a defined period of 10 minutes. If these conditions are met, an e-mail has to be sent from the Cloud platform to warn about an uncommon and potentially dangerous event.

To compose the previously described rule, few elements must be analysed in more detail: the “std:firstunique()” view of the EdcMessageEvent triggers when a unique value of ‘power’ is received, and the “win:time(10 minutes)” view limits this uniqueness to a time window of 10 minutes.

The “Action” tab allows to specify the type of action to take, when the rule is triggered.

The following image illustrates a possible definition of the email message that has to be sent when the rule condition is matched.

The screenshot shows the 'New Rule' dialog box with the 'Actions' tab selected. The 'Definition' tab is also visible. The 'Add' button is highlighted. The 'Email' action is selected, and its details are shown in the right pane:

- To:** recipient@emailaddress.com
- Subject:** Warning! Power consumption too high!
- Body Format:** plain text
- Body:** On date \$sentOn, asset \$asset, located at latitude \$position.latitude, longitude \$position.longitude, belonging to account \$account, published on topic \$semanticTopic the power consumption exceeded the threshold value of 10 W. Position: \$position
- Alternative body:** Your email client does not support HTML message

At the bottom of the dialog, there are 'Submit' and 'Cancel' buttons.

Figure 46 – Action definition

The available fields for the action definition are:

- **To:** the recipient(s) mail address;
- **Subject:** the subject of the email;
- **Body Format:** the type of body message to create. Possible choices are:
 - Plain text;
 - Html;
- **Body:** the effective message that will be sent. The body can contain plain text but also dynamically generated messages using \$property_name placeholders that, when the message is sent, will be replaced by the query results;
- **Alternative body:** allows to specify error messages or backup links that are displayed if the email message sent contains HTML code and if the email program is not enabled to display HTML messages.

The following images provides an overview of the needed fields for the other types of action available: IFTTT, Alert, MQTT Event, REST Call, SMS and Twitter.

The screenshot shows the 'New Rule' dialog box with the 'Custom Rule' template selected. The 'Definition' tab is active, and the 'IFTTT' action is selected. The configuration fields are as follows:

- * SMTP server: [Text Field]
- * SMTP port: [Text Field]
- * IFTTT Account Email: [Text Field]
- * IFTTT Email Password: [Text Field]
- * Tag: [Text Field]
- * Subject: [Text Field]
- * Body: [Text Area]

Buttons at the bottom: Submit, Cancel.

Figure 47 – IFTTT action

The IFTTT actions are more oriented to the interoperability with other applications and to the notification of events to the final user.

To define a IFTTT action, the following parameters must be specified:

- SMTP server: the SMPT server of the IFTTT service;
- SMPT port: the port of the IFTTT SMPT server;
- Account Email: the email specified in the IFTTT account;
- Email Password: the password of the email specified in the IFTTT account;
- Tag: specifies a hashtag# that must be put before the subject that will launch the trigger IFTTT, for example rulename#edcalert.
- Subject: the email subject;
- Body: the message that will be processed by the IFTTT rule.

The screenshot shows the 'New Rule' dialog box with the 'Custom Rule' template selected. The 'Definition' tab is active, and the 'Alert' action is selected. The configuration fields are as follows:

- * Severity: [Dropdown Menu] (CRITICAL)
- Category: [Text Field]
- Code: [Text Field]
- Message: [Text Field]

Buttons at the bottom: Submit, Cancel.

Figure 48 – Alert action

The Alerts, when triggered, appears as messages that are visualized in the landing page when the operator accesses the cloud console.

To define an Alert the following parameters must be specified:

- Severity: specifies the severity level and can be CRITICAL, WARNING or INFO;
- Category: specifies the category of the alert, for example Performance, Security, Other etc. .
- Code: a code defined by the user.
- Message: the message to be visualised.

The screenshot shows the 'New Rule' dialog box with the 'Custom Rule' template selected. The 'Definition' tab is active, and the 'MQTT' action is selected. The configuration fields are as follows:

- * Topic: \$account/RulesAssistant/\$semanticTopic/alert
- Custom message metrics:

Name	Type	Value
sentOn	String	\$sentOn
asset	String	\$asset
account	String	\$account
semanticTopic	String	\$semanticTopic

Buttons at the bottom: Submit, Cancel.

Figure 49 – MQTT action

The MQTT action allows to create alert events that can be stored on the cloud as a standard data. This option provides the possibility to process the events generated by the rules with external application (for example an external web based control panel).

To define a MQTT action the following parameters must be specified:

- Topic: MQTT full topic name;
- Metrics: all the properties that are selected in the Statement are available as a token to populate the metrics in the event.

The screenshot shows the 'New Rule' dialog box with the 'Custom Rule' template selected. The 'Definition' tab is active, and the 'REST' action is selected. The configuration fields are as follows:

- * Method: POST
- * Url: (empty field)
- Username: (empty field)
- Password: (empty field)
- Post body content type: application/xml
- Post body content: rule triggering message
- Custom message metrics:

Name	Type	Value
sentOn	String	\$sentOn
asset	String	\$asset
account	String	\$account
semanticTopic	String	\$semanticTopic

Buttons at the bottom: Submit, Cancel.

Figure 50 – REST action

This action allows to make a call to the Cloud APIs or to external APIs provided by third parties (for examples APIs that implements other external services).

To define a REST action, the following parameters must be specified:

- Method: specifies the HTTP method to be used to perform the REST call (e.g., GET, POST, PUT, DELETE).
- Url: specifies the URL of the endpoint to be invoked.
- Username: specifies the username to be used, if the endpoint is protected by HTTP Basic Authentication (optional).
- Password: specifies the password to be used, if the endpoint is protected by HTTP Basic Authentication (optional).
- Metrics: all the properties that are selected in the Statement are available as a token to populate the metrics in the call.

The screenshot shows a 'New Rule' dialog box with a 'Custom Rule' template selected. The 'Actions' tab is active, displaying a list of actions. The 'Twilio SMS' action is selected, indicated by a red dot. The fields for this action are:

- * Twilio Phone Number: [text input]
- * Twilio Account ID: [text input]
- * Twilio Auth Token: [text input]
- * Numbers: [text input]
- Message: [text area]

 At the bottom of the dialog are 'Submit' and 'Cancel' buttons.

Figure 51 – SMS action

SMS-based actions rely on a paid service, and an account at Twilio must be setup before use this actions (<http://www.twilio.com>).

To define a SMS-based action, the following parameters must be specified:

- Phone Number: this is a Twilio phone number;
- Account ID: Twilio account identifier;
- Auth Token: Twilio authorization token;
- Numbers: specifies the recipient's phone number(s) including area code but with no other punctuation. Multiple numbers can be entered, comma-separated.
- Message: all the properties that are selected in the Statement are available as a token to populate the message.

The screenshot shows a 'New Rule' dialog box with a 'Custom Rule' template selected. It has two tabs: 'Definition' and 'Actions'. The 'Actions' tab is active, showing a list of actions with a 'Twitter' action selected. The 'Twitter' action is represented by a blue bird icon. To the right of the icon are five input fields labeled: '* Consumer Key:', '* Consumer Secret:', '* Access Token:', '* Access Token Secret:', and '* Message:'. The 'Message' field is a larger text area. At the bottom of the dialog are 'Submit' and 'Cancel' buttons.

Figure 52 – Twitter action

Finally, to define a twitter action the parameters must be specified:

- Consumer Key: consumer key received after you create your Twitter application;
- Consumer Secret: consumer secret received after you create your application;
- Access Token: access token for your Twitter account and current application;
- Access Token Secret: access token secret for your Twitter account and current application;
- Message: all properties that are selected in the Statement are available as a token to populate the message. The maximum length is 140 characters.

4.4.5 The cloud platform REST API

The cloud platform exposes a comprehensive set of Web Service APIs for application integration purposes. The cloud platform API conforms to the standard REpresentational State Transfer (REST) protocol. REST has emerged over the past few years as a predominant Web service design model. REST-style architectures consist of clients and servers: clients initiate requests to servers, while servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource.

The cloud platform RESTful APIs expose the standard action types (create, read, update, delete) over the platform objects. They are capable of retrieving a resource representation in XML or JSON format.

It is possible to use the REST HTTP Accept Header to specify the representation requested using the "application/xml" and "application/json" Media Types. As an alternative to the Accept header, it is possible to specify the requested data representation by appending an ".xml" or ".json" suffix to the path part of the request URLs. When the URL string includes characters that are reserved to the URL syntax or are outside the ASCII character set the URL encoding is necessary. In these cases, URL-encoding replaces the offending characters with a percent sign (%) followed by the two hexadecimal digits identifying the original character. For instance, the character "/" is replaced with "%2F", or "#" with "%23".

This API supports a Representational State Transfer (REST) model for accessing a set of resources through a fixed set of operations. Depending on the role of the resource, the methods GET, POST, PUT and DELETE are provided. The cloud resources required for the monitoring infrastructure are accessible through the following RESTful model:

- **Accounts:** allows to create/delete a new cloud account or list the accounts available for the current user.
- **Alerts:** create/delete an alert or list all alerts under the account of the currently connected user.
- **Assets:** manages the list of all the assets that published some data for the account of the currently connected user. For each returned asset, the Asset ID and timestamp of its last received message will be returned.
- **Devices:** returns the list of all the Devices that are connected under the account of the currently connected user. For each returned Device, its latest Device profile will be returned. With the various methods it is possible to manage the devices: create a new device, get information on a specific device, update a device profile, delete a device, issue a command on a specific device, etc..
- **Messages:** manage the messages published under the account of the currently connected user. The methods of this resource allow to list all received messages, delete messages and publish new messages. The methods provide the possibility to specify a rich set of query in order to select messages by date, by ID, by Metrics, etc.. When a new message is posted, it will be published to the broker associated to the current account. In this way, devices and applications subscribed to the topic specified in the Message will receive a copy of the published message.
- **Metrics:** returns the list of all metrics that were published under a specified topic. For each returned Metric, its name, timestamp, type and last value will be returned.
- **Rules:** allows to manage the rules created in the current account to manage and process data acquired from the field. Rules can be created, deleted or simply listed.
- **Streams:** this resource allows subscribing to a given topic and receiving messages published under that topic. Upon receiving a message on the subscribed topic, the message will be formatted in XML or JSON format as requested and sent as response body to the suspended request.
- **Topics:** returns the list of all the Topics that received some data for the account of the currently connected user. For each returned Topic, the topic string and timestamp of its last received message will be returned.
- **Users:** allows to manage the users associated with the current account. The methods allow listing the users, creating a new user, updating a user profile and deleting an existing user.
- **Vpn:** this resource allows to manage vpn connections when required by the specific installation of the device to cloud monitoring infrastructure.

The REST resources expose a data model that is supported by a set of client-side libraries that allow the development of application in different operating systems and development

environments. The libraries currently available provide support for the following programming languages: C, .NET, Java, Java JSON, Objective C, PHP and Ruby. A WADL document describing the REST API is available.

In the data model, the data can be represented with difference media (i.e. "MIME") types, depending on the endpoint that consumes and/or produces the data. The data can be described by XML Schema, which definitively describes the XML representation of the data, but is also useful for describing the other formats of the data, such as JSON. In the XML Schema, data can be grouped by namespace, with a schema document describing the elements and types of the namespace. Generally speaking, types define the structure of the data and elements are instances of a type. For example, elements are usually produced by (or consumed by) a REST endpoint, and the structure of each element is described by its type. The data model specifies for each data a namespace and, for each namespace, it specifies the available Data Elements and the related Data Type.

A simple example of use of the REST APIs consists in displaying cloud resources information with a web browser. After the device to cloud infrastructure has been installed, configured and running, it is possible to display the list of devices of the current account navigating with the browser to the following address:

<https://api-sandbox.everyware-cloud.com/v2/devices.xml>

The following figure illustrates the XML file displayed in the browser.

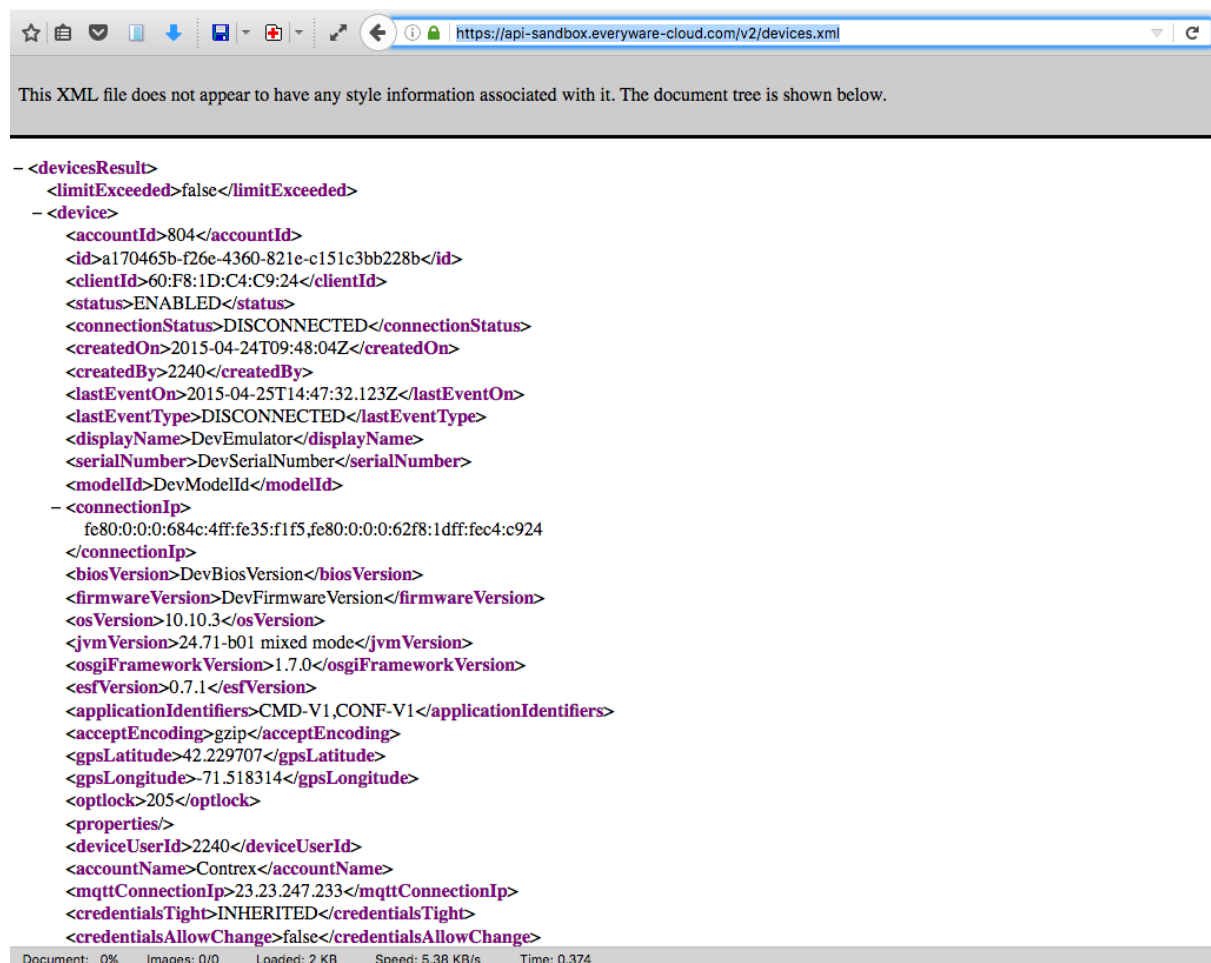


Figure 53 – List of devices connected to the cloud.

4.4.5.1 REST APIs Documentation

The previous sections provide a description of the final version of the REST APIs designed and developed to interact with the runtime management of the cloud platform. The last version includes also the updates introduced in deliverable D3.3.2.

The full documentation of the API is available in html format at the following address:

<https://api-sandbox.everyware-cloud.com/docs/index.html>

The documentation includes a description of the REST resources and of the associated data model.

Examples of source code are available on github at the following address:

<https://github.com/eurotech/edc-examples>

5 Conclusions

This deliverable described the results of the activities on run-time management developed in Task 3.3. This document not only defines the several run-time management activities but also defines the knobs that are subject of the run-time actions. The content of Section 4 is to be a refinement of the previous D3.3.2 deliverable by presenting results of their adoptions in the context of the CONTREX Use Cases. This deliverable is closes the activities on run-time management developed in Task 3.3.

References

- [Venk05cs] Venkatachalam, V., Franz, M.: Power reduction techniques for microprocessor systems. *ACM Comput. Surv.* 37(3), 195–237 (2005)
- [Pedram01aspdac] Pedram, M.: Power optimization and management in embedded systems. In: *ASP-DAC 2001: Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, pp. 239–244. ACM, New York (2001)
- [Lorch03mobisys] Lorch, J.R., Smith, A.J.: Operating system modifications for task-based speed and voltage. In: *MobiSys 2003: Proceedings of the 1st international conference on Mobile systems, applications and services*, pp. 215–229. ACM, New York (2003)
- [Pettis09tcom] Pettis, N., Lu, Y.H.: A homogeneous architecture for power policy integration in operating systems. *IEEE Transactions on Computers* 58(7), 945–955 (2009)
- [Anand04mobisys] Anand, M., Nightingale, E.B., Flinn, J.: Ghosts in the machine: interfaces for better power management. In: *MobiSys 2004: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pp. 23–35. ACM, New York (2004)
- [Tamai04nossdav] Tamai, M., Sun, T., Yasumoto, K., Shibata, N., Ito, M.: Energy-aware video streaming with QoS control for portable computing devices. In: *NOSSDAV 2004: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pp. 68–73. ACM, New York (2004)
- [Liu10tmc] Liu, X., Shenoy, P., Corner, M.D.: Chameleon: Application-level power management. *IEEE Transactions on Mobile Computing* 7(8), 995–1010 (2008)
- [Yuan03sosp] Yuan, W., Nahrstedt, K.: Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In: *SOSP 2003: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 149–163. ACM, New York (2003)
- [Fei08tecs] Fei, Y., Zhong, L., Jha, N.K.: An energy-aware framework for dynamic software management in mobile computing systems. *ACM Trans. Embed. Comput. Syst.* 7(3), 1–31 (2008)
- [snowdon09eurosos] Snowdon, D.C., Sueur, E.L., Petters, S.M., Heiser, G.: Koala: a platform for os-level power management. In: *EuroSys 2009: Proceedings of the 4th ACM European conference on Computer systems*, pp. 289–302. ACM, New York (2009)
- [Brock03soc] Brock, B., Rajamani, K.: Dynamic power management for embedded systems. In: *Proceedings on IEEE International SoC Conference*, September 2003, pp. 416–419 (2003)
- [Donald06isca] Donald, J., Martonosi, M., "Techniques for Multicore Thermal Management: Classification and New Exploration," *Computer Architecture, 2006. ISCA '06. 33rd International Symposium on*, vol., no., pp.78,88
- [Khan09date] O. Khan and S. Kundu, "Hardware/software co-design architecture for thermal management of chip multiprocessors," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09*. pp.952 –957.

- [Chantem09ispled] T. Chantem, X. S. Hu, and R. P. Dick, "Online work maximization under a peak temperature constraint," in Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design, ISLPED '09. pp. 105–110.
- [Yeo08dac] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in Design Automation Conference, 2008. DAC. pp. 734 –739.
- [Ge10dac] Y. Ge, P. Malani, and Q. Qiu, "Distributed task migration for thermal management in many-core systems," in Design Automation Conference (DAC), 2010. pp. 579 –584.
- [Gomaa05asplos] Mohamed Gomaa, Michael D. Powell, and T. N. Vijaykumar. Heat-and-run: leveraging SMT and CMP to manage power density through the operating system. In Proceedings of the 11th international conference on Architectural support for programming languages and operating systems (ASPLOS XI). 2004
- [Ykman11ietcdt] Chantal Ykman-Couvreur, Prabhat Avasare, Giovanni Mariani, Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria. "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration", IET Computers and Digital Techniques. Vol. 5. Issue 2, 2011, pp. 123-135.
- [Zoni12arcs] D. Zoni, P. Bellasi, W. Fornaciari. "A Low-Overhead Heuristic for Mixed Workload Resource Partitioning in Cluster-Based Architectures". International Conference on Architecture of Computing Systems (ARCS'12), Berlin, Germany, 02/2012.
- [Geilen05acs] dGeilen, M., Basten, T., Theelen, B., Otten, R.: An Algebra of Pareto Points. In: IEEE ACSD 2005, pp. 88–97 (2005)
- [Govil95] K. Govil, E. Chan, H. Wasserwan, "Comparing algorithm for dynamic speed-setting of a low-power CPU," MobiCom '95:1st International Conference on Mobile Computing and Networking, pp. 13–25, November 1995.
- [Linden95] D. Linden, "Handbook of batteries," 2nd. ed., McGraw Hill, Hightstown, N. J., 1995
- [Nollet08jsps] Nollet, V., Verkest, D., Corporaal, H.: A Safari Through the MPSoC Run-Time Management Jungle. Journal of Signal Processing Systems (2008)
- [Shahriar07jos] Shahriar, A.Z.M., Akbar, M.M., Rahman, M.S., Newton, M.A.H.: A multiprocessor based heuristic for multi-dimensional multiple-choice knapsack problem. The Journal of Supercomputing 43(3), 257–280 (2007)
- [Shojaei10dac] Shojaei, H., Ghamarian, A.H., Basten, T., Geilen, M., Stuijk, S., Hoes, R.: A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management. In: 46th ACM/IEEE DAC 2009, pp. 917–922 (2009)
- [Sinha01] A. Sinha, A. P. Chandrakasan, "Dynamic Voltage Scheduling Using Adaptive Filtering of Workload Traces," VLSID'01: 14th International Conference on VLSI Design, pp. 221–226, January 2001.
- [Xu05ts] Xu, P., Michailidis, G., Devetsikiotis, M.: Profit-Oriented Resource Allocation Using Online Scheduling in Flexible Heterogeneous Networks. Telecommunication Systems (2005)

[Ykman06issoc] Ykman-Couvreur, C., Nollet, V., Catthoor, F., Corporaal, H.: Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management. In: International Symposium on System-on-Chip, pp. 1–4. IEEE (November 2006)

[2parma] PARallel PARadigms and Run-time MAnagement techniques for Many-core Architectures Project ID: FP7-ICT-2009-4-248716 <http://www.2parma.eu>

[Pana18650] Panasonic, CG18650CG lithium-ion battery, <http://www.meircell.co.il/files/Panasonic%20CGR18650CG.pdf>.

[Texas54122] Texas Instruments, TPS54122-Q1 DC-DC converter, <http://www.ti.com/lit/gpn/TPS54122-Q1>.

[Linear3407] Linear Technology, LTC3407-3 converter, <http://www.linear.com/docs/16678>.

[Petricca2014islpd] M. Petricca, D. Shin, et al. An automated framework for generating variable-accuracy battery models from datasheet information. ACM/IEEE ISLPED'14, pages 365–370, 2013.