Energy Management API

Document WINNF-TS-3001

Version V1.0.0

27 Aug 2018

TERMS, CONDITIONS & NOTICES

Table of Contents

List of Figures

List of Tables

Executive Summary

During March 2017 through June 2018 the volunteer industrial participants of the Wireless Innovation Forum developed this document that defines an Energy Management Application Programming Interface (EM-API) such that radios can manage their energy consumption by interacting with a standardized interface. The same interface may allow Software Defined Systems (SDS), and platforms such as UAVs to interact with subsystem components on the platform such as the platform battery, radio, sensors, displays and applications in the context of the energy consumption and the timeline of a mission profile. The interface is defined in the Interface Definition Language (IDL) of the Object Management Group (OMG).

Contributors

Bruce Fette
Cory Clark
David Hagood
David Murotake
Eric Nicollet
Fabio Casalino
Frank Van Syckle
James Evangelos
Jimmie Marks
John Stine
Joseph Mitola
Kevin Richardson
Latha Kant
Michael Fritz

Energy Management Application Programming Interface

1 Introduction

The Wireless Innovation Forum has defined an Energy Management Application Programming Interface (EM-API) with the objective of making it straight forward to manage energy consumption in Software Defined Systems (SDS), in particular in Software Defined Radios (SDRs) and the platforms with which SDRs may be integrated. Given that SDSs may be integrated with a wide variety of applications and a wide variety of platforms, SDSs may be called upon to adjust their energy consumption in response to platform demands, application demands, communications traffic demands on the system, and spectrum channel conditions. To meet the objective of software portability, it is important for SDSs, network applications, and platforms to be able to adjust energy consumption in response to such environmental operational aspects.

This document proposes standardized mechanisms to interact with SDS software in an industry standard way, with an expectation that following the recommended standard will make it easier for each SDS to make efficient use of available power/energy resource (e.g. battery power, thus enabling long user periods or mission durations between battery recharge or replacement).

The EM-API assumes that an SDS may consist of various components. Such components may include any of the following:

a. an energy source (battery, vehicular power, prime power etc.),
b. a platform (people or vehicles of many different kinds),
c. one or more applications (reasons to exchange voice, video, data) to accomplish some task, and applications may operate embedded inside some device or may reside in a separate unit (like a smart phone),
d. remote controllers (for example a vehicular adapter for a radio, a remote controller for a drone or robot, or a smart display running an application to accomplish something for its user), and/or
e. a software Defined Radio (SDR) or Cognitive Radio (CR)

Furthermore, many such SDSs may interact with each other as a network to accomplish one or more overall objectives, and at the systems of systems level may find adjustment of energy consumption at one location in the network could be desirable to many other system performance metrics. Systems that are battery powered or operating with a constrained power source may wish to prolong their battery life by careful use of energy, particularly if the objective is to be able to continue to operate for a defined period of time, or until a specific objective has been accomplished. Careful energy management can also reduce heat generation and reduce thermal signature.

In order to prolong the operational time window, each component must have a way to interact with the other components in ways that enable taking advantage of opportunities to save energy. For purposes of understanding the EM-APIs, it will be helpful to assume that an energy management optimizer functionality exists within one or more of the system components. Such an energy management optimizer will find it useful to know about how much energy is available, how much energy each function is consuming, and when there are opportunities for energy saving to be able to share that knowledge to the various components. In components where such an energy management optimizer function exists, the EM-API's functions described in this document will enable the sharing of energy saving opportunities within the system and also within the network.

In order to accomplish energy conservation, the components will need to exchange information using a standard representation as defined by the EM-API.

Such information may include:

a. properties of the energy source (watt hours, battery, vehicular or prime power),
b. properties of the energy consuming components (modes, power consumption in those modes, cycle time to change modes), and
c. opportunities to save energy or otherwise manage system performance by shifting energy/power consumption amongst available modes

1.1 What is an API?

An API is an Application Programming Interface.

Application: this is to enable some software application to use some other element of the system.

Programming: it must exist within the context of a program (in the Turing machine sense), not as a non-programming guideline.

Interface: APIs can only really exist at the boundary of two distinct elements of a system. Within an element there may be a function call, but if it does not cross the boundary from one element to another it is not an interface.

1.2 Defining an API means defining:

1) Where the boundary is, that is, between what elements of a system does the boundary exist.
2) Which elements implement the API (provide the function). Which elements use the API.
3) What are the functions in the API. What are the parameters of those functions. What are the preconditions, post conditions, and invariants of those functions. What errors or exceptions can those functions indicate. What are the returned values of those functions.
4) What language mappings exist for those functions..

1.2.1 Overview of EM-APIs

For the appropriate definition of the EM-API functions it is important to define the border between the offered services (through the specific interfaces) and the entities accessing them.

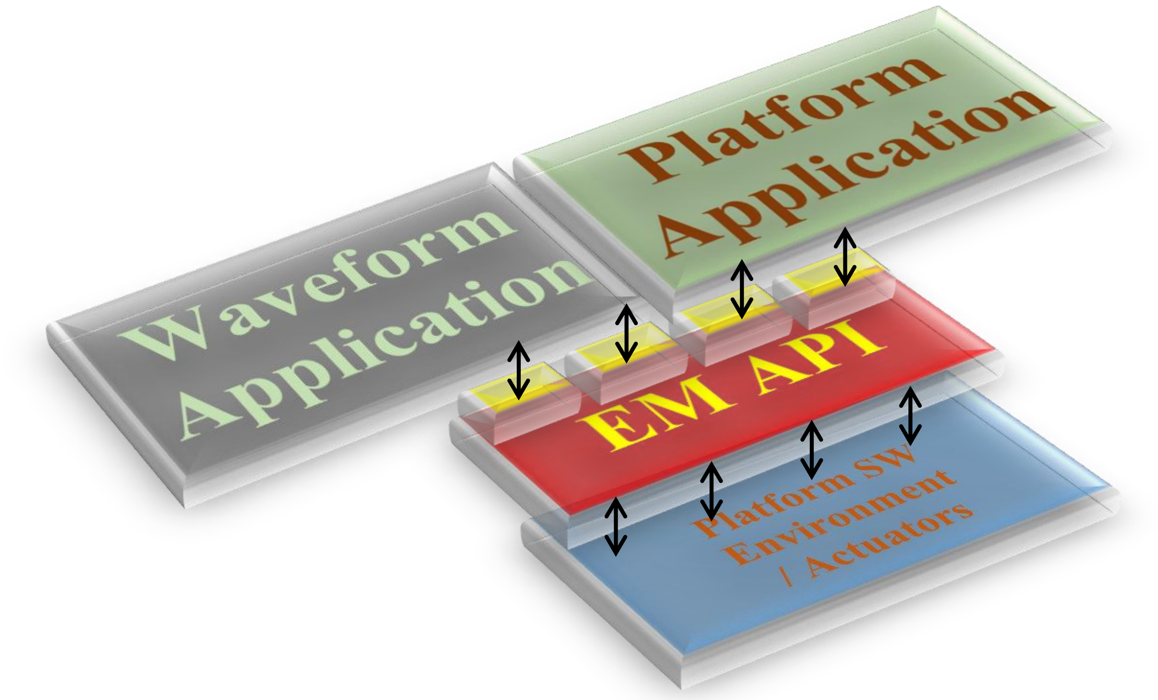

Figure 1 - The EM-API exists between a higher level functions and lower level functions.

The EM-API border location can be identified as being on top of the hosting platform software (SW) environment and below the applications running on it. It provides for a common interface to actuate mechanisms of power savings and management, requested by the higher-level applications and implemented by the specific processes or hardware of the platform underneath.

The user of the EM-API can be a generic application that, due to the shared knowledge of the device type and status, could implement some sort of control intervention through the available API.

For instance, a waveform running on a power constrained platform (e.g. battery powered) could be requested or notified by the API about a specific condition, which requests the radio or radio network to implement some behavior. This behavior may be internal inside the waveform. Alternatively, it may actuate available settings (presets), through the API, to set the platform status/resources in a particular working condition, which still allows it to provide a service even with a possible change of performances. It should be noted that the usage of the API may reflect information exchange in both directions.

Similarly, a platform application could benefit from the API services. For instance, a power profile is managed at a higher level within the platform itself or a platform-embedded service is provided to the user.

1.2.2 Energy Saving Principles

Part of allowing a system to save power or energy is to provide the components of that system with enough information about the over-all goals the user needs to achieve, and the over-all state of the system, such that a component can use that information plus its own self-knowledge to optimize its operation. However, knowing what information a component will need about the state of the system, and designing specifically for that component's needs is antithetical to portability. It tightly couples all the components of a specific system, reducing reuse. A more portable method is to provide a mechanism to make information about the goals and states of the system and the components comprising it, available to any and all components that are interested in the information. Those components should then be allowed to make their own decisions about what component-specific means will best meet those goals. Thus, there should be a means to communicate high-level, non-application specific information about the system state, which can then be applied by specific layers of the system to select specific means to adjust power.

A key part of the information needs to be the overall goals of the system – whether the system should be optimizing for lower power by shedding work, or optimizing for maximum work done at the potential expense of higher energy consumption. That goal selection will have to take into account both items which can be directly introspected by the system (e.g., available power sources, capacities of those power sources), as well as items that the system cannot ascertain on its own and must be informed of (e.g. user preferences). Additionally, in a system composed of many functions or applications, the goals of the overall system may be at odds with the goals of individual subsystems – a life support radio may need to be in a power conserving state overall, but during a rescue activity the actual location beacon function may need to continue to operate at maximum performance to allow the operator to be rescued.

Other key pieces of information are scheduling and deadlines. Combining items of work (information delivery in an SDR) in time can save energy spent in entering and leaving low power states, or allow hardware to be used by more than one item of work at the same time to save energy. If the information about scheduling and deadlines is not available to a component, all the component can do is work on what is assigned when it is assigned. However, if the component is given information about scheduling and deadlines for an item of work, the component may be able to schedule the work more efficiently to save power – coalescing work in time by rescheduling work with loose time constraints, opportunistically scheduling optional work for when it causes the least increase in power, or even discarding optional work when power targets cannot be met.

There are many mechanisms in SDR radio design and operational use that can be recognized and used as energy saving opportunities. In Appendix A, we address a number of energy saving opportunities in ISO Stack layered fashion working from PHY layer, up through the network, application, platform and system level considerations. It should be recognized that the SDR is able to efficiently adapt to conditions and roles in the network. The adaptivity mechanisms are often referred to in Cognitive Radio (CR) principles such as sense and adapt. Often, the adaption mechanisms are also referred to as “knobs” even though these adaption mechanisms are changes internal to the software, not explicitly external knobs to be manipulated by a human. The reader will note that inside the SDR radio, the modem may sense various conditions and perform adaption within the modem. Similarly, the networking function may recognize various conditions and choose to perform energy conserving adaption mechanisms internal to the network. However, if energy management is performed by a mechanism outside the network function it will then do so through an API. Similarly, if energy management is performed outside of the modem, control of the modem function for energy management will be performed through an API. It is rather likely that sensing reports from inside the modem and inside the network may optionally be made available to an energy management function which can integrate the information and can then make good choices of energy management adaption mechanisms.

When it is understood how much time is available for an energy conservation mode, for example how long before the radio will again transmit or receive, the energy management function can also perform a tradeoff of timing and energy to change state in and out of lower energy states.

We anticipate that energy conserving mechanisms can be grouped into various state presets, so that energy management can consist of recommending a preset and a duration. Such preset mechanisms can be used to internalize vendor unique intellectual property, and simplify energy management.

It is important to recognize that some of the principles discussed may not be applicable in various designs for various reasons due to system design considerations, however, the intent of this document is to offer a rich portfolio of energy saving management functionality, which may be discretionally adopted by developers.

1.2.3 Managing Energy Sources

Energy resource management is essential for power and energy constrained devices, which must typically last for an entire mission duration.

Assuming a single energy source is available to power the device, it has to be properly managed in order to optimize its life time (e.g. battery) while providing efficiency and needed performance.

For this reason achieving the most effective energy management would mean to reach the best balance between power/energy usage and functionality to be performed.

1.2.4 Software Developer Perspective

From the software developer perspective, the EM-API provides a means to access information provided by the platform and to set proper configuration parameters or presets in order for the platform to perform in the most optimized and efficient way for saving power.

Multiple profiles of developers are identified: platform software and waveform software, application software and energy source software.

Based on platform type characteristics, the platform SW developers implement the mechanisms (e.g. in form of drivers or low level applications) to associate each specific EM-API interface to a proper function, which could impact the operating environment or a hardware (HW) resource.

The waveform developer implements the proper code to gather information from the system and set the available configuration to actuate a specific power management profile.

The configuration could be manual (commanded) or automatic, meaning that a specific configuration is performed by the platform upon one or more individual settings carried out through the API interface.

1.2.5 Hardware developer perspective

From the hardware developer point of view, specific solutions have to be designed and implemented in the platform processing elements and actuators in order to activate the specific setting through the EM-API. Such solutions are in accordance with the software interfaces actuated by the platform software and by the overall platform requirements and design.

1.2.6 Reporting Supported Capabilities

Some radio designs may have few or no EM-API features, while other radios may support some, many, or all of the EM-API features recommended in this document. When radios and other components are integrated as a system, the system components may interact with each other to determine what features are supported in each subsystem. SDR radios that support energy management APIs may provide a means to respond to queries in order to establish the most effective means of managing energy in the composite system, network, or platform, and subsequently to make use of available energy management features.

1.2.7 Saving Energy at the System and Platform Level

Looking at the system architect's perspective, a system-of-systems can benefit from energy management across a variety of different platform types including airborne, maritime, ground mobile, man portable and fixed site nodes/platforms. Ideally, platforms within a system should be able to query and discover the energy management characteristics of other platforms within the network using a common framework of a common EM-API. This ability is especially critical for nodes/platforms that have system management responsibility for major portions of the system-of-systems. Within any single system-of-systems, the system manager node should have as a minimum, the capability to respond to EM-API discovery queries from trusted external networks and network member nodes/platforms,

Some smaller platforms, such as man-portable systems and unmanned systems (drones and robots) can benefit from energy management. At a system and network level, the system manager can spare size, weight and power (SWAP) constrained platforms from extensive use of energy consuming functions when doing so does not impact system performance (see Figure 2).

To accomplish system-wide energy management, the interfaces with system- and network-level interfaces should be harmonized across all platforms.

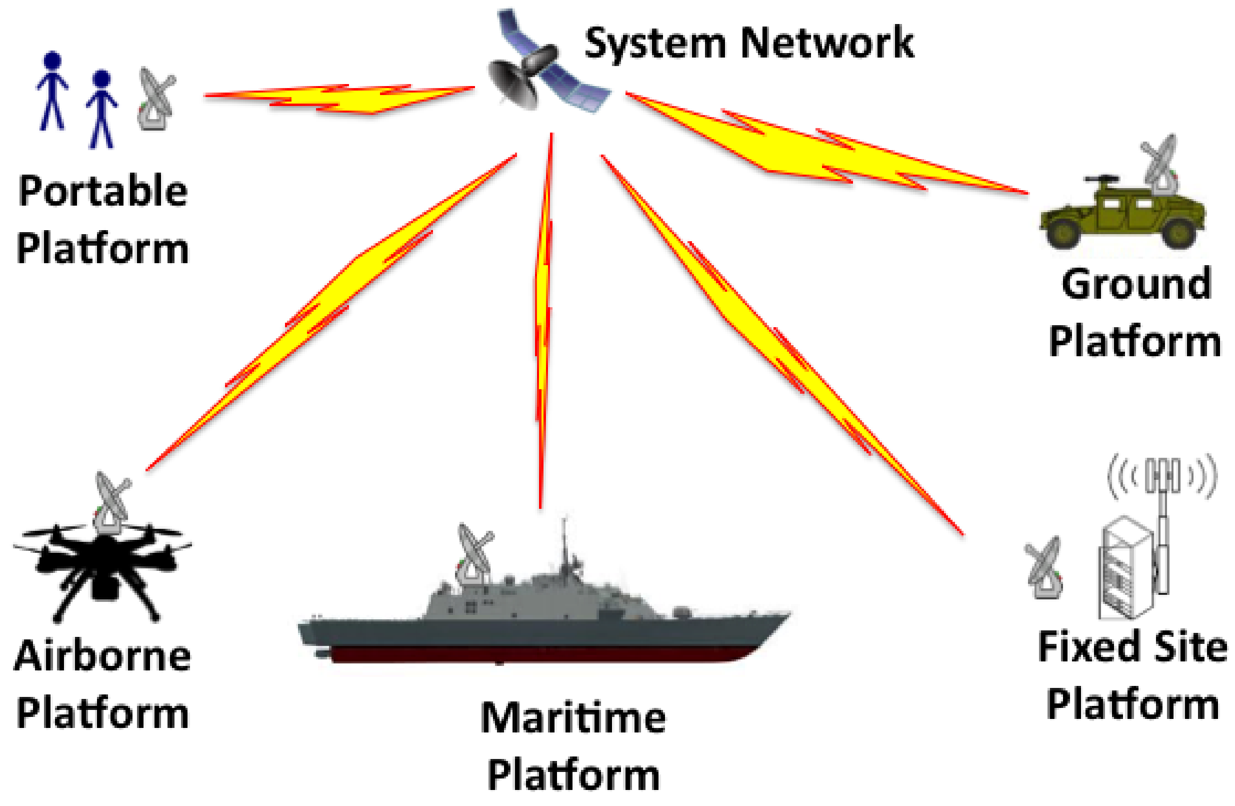

Figure 2 - A system-of-systems benefits from system-wide energy management.

The platform ideally should have the capability to query its components, such as a data link transceiver, navigation, security, battery management or flight control subsystem, using a common EM-API, and to respond to queries and reports from component subsystems. (see Figure 3). Such queries may occur every time the system is booted, or, depending on platform and system manager policies, may store recent discovery responses. The means to query this overall information is done within the greater context of a CR protocol, cooperative sensing and synchronization between platforms in both heterogeneous and homogenous system-of-systems. The EM-API work group plans to identify a common EM-API that can provide a common method within this greater context, in a manner consistent with the Forum’s bylaws and mission statement.

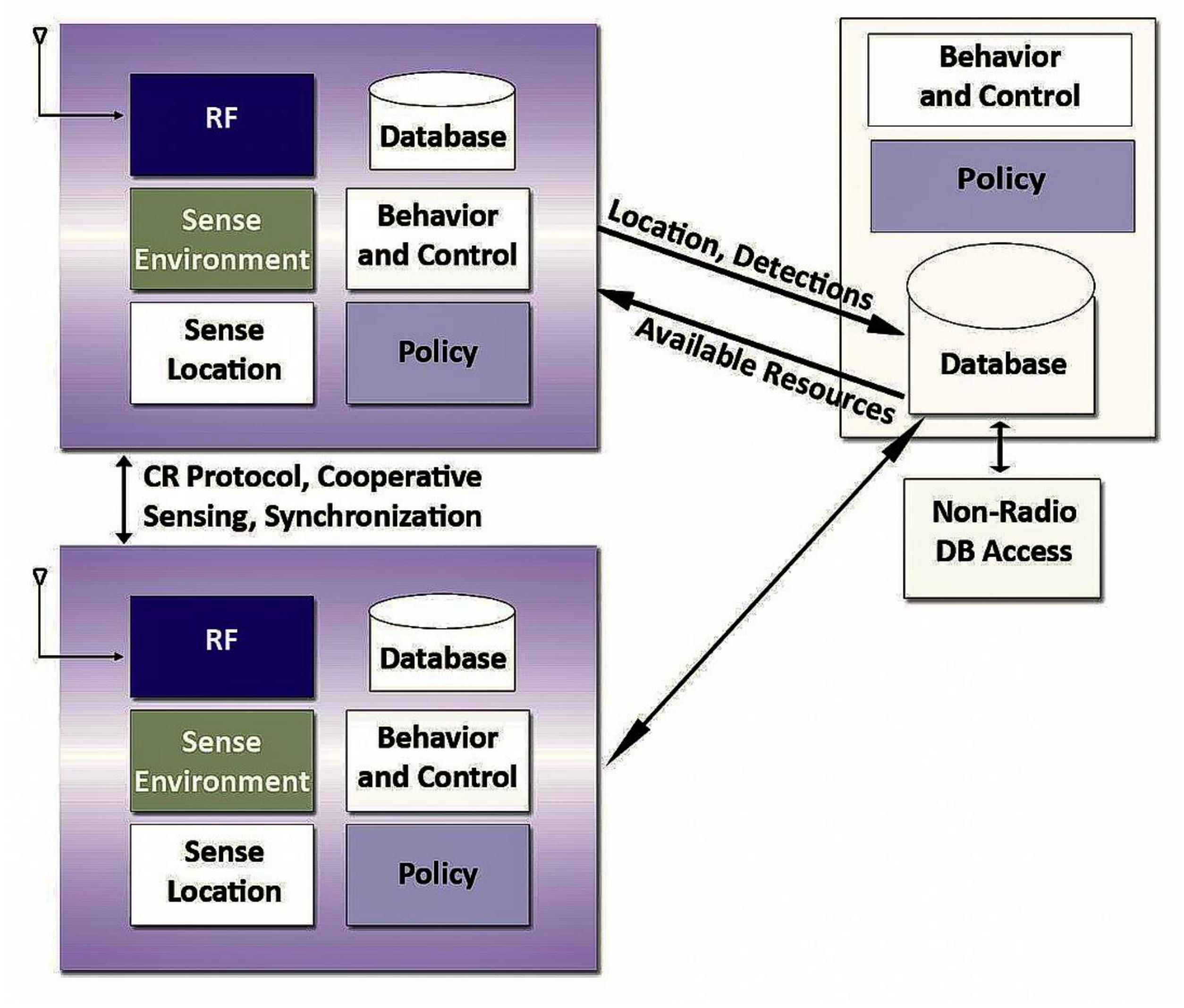

Figure 3- A platform/node architecture supports a common EM-API within the greater context of a CR protocol, cooperative sensing and synchronization between platforms in both heterogeneous and homogenous system-of-systems. (Figure: Excerpt from Slide 16, WInnF 2016 Corporate Presentation, 26 November 2016)

EM-API protocols can be incorporated as part of other inter-platform protocols, such as CR protocols, cooperative sensing and synchronization, between platforms employing WInnForum APIs. For communication with external entities, a “wrapper” embedding the EM-API protocols can be developed (see Figure 4). Specific use cases of this diagram, with examples from airborne, maritime, etc. domains, will be considered in the following sections, including energy management parameters which might be standardized.

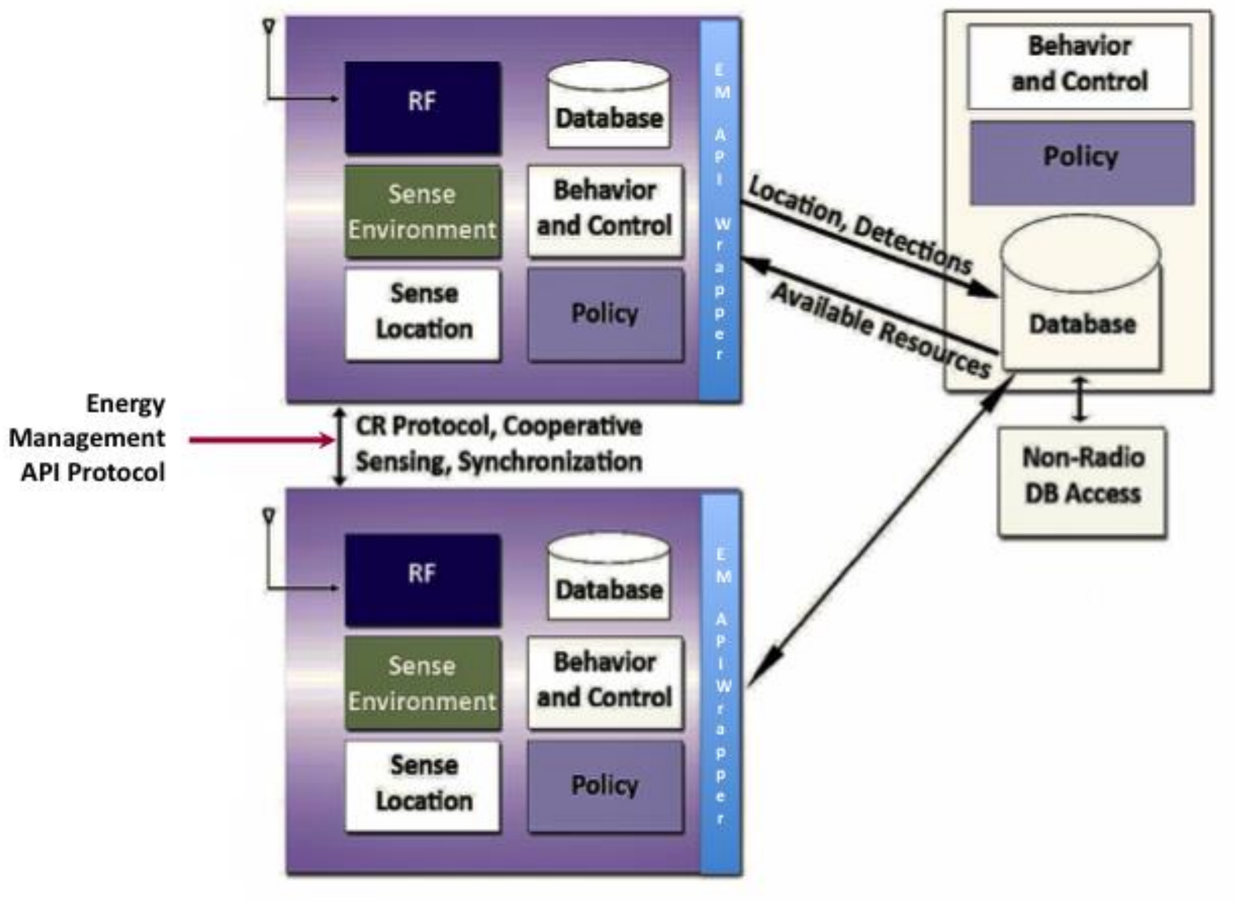

Figure 4 - EM-API protocols can be incorporated as part of other inter-platform protocols, such as CR protocols, cooperative sensing and synchronization, between platforms employing WInnForum APIs. For communication with external entities, a "wrapper" embedding the EM-API protocols (shown in light blue) can be developed.

1.3 Referenced Documents

Earlier documents developed within the Wireless Innovation Forum may be valuable for the SDR development community. Of particular note, the Transceiver API (WINNF-TS-0008 Version 2.0.0) is believed to be particularly helpful.

2 Use Cases and Considerations

2.1 Use Cases

In order to create a collection of EM-APIs, it is important to understand how the EM-APIs may need to be applied. The following section provides a number of energy management use cases that EM-APIs should support.

2.1.1 Example EM API Use cases

1) User wants the system to perform a task for which there is insufficient power/energy.
2) User wants the system to perform a task which will require a potentially damaging amount of energy/power; e.g., discharging a battery beyond the safe limit, or operating a generator when there is insufficient oil pressure.
3) An entity in the system wishes to indicate intended energy/power requirements for a future time: for example, the system is unable to meet the need, or system is able to meet the need by rescheduling or shutting down other entities.
4) Conversely to the previous item: the system wishes to indicate future energy/power availability to allow entities in the system to adapt their behavior.
5) User wishes to achieve a goal that could be satisfied by one of N possible entities in the system, with different entities having different energy sources (and thus different energy availability), and the system will have to select which entity to use.
6) An entity in the system needs to be given priority over other entities in the system (e.g., life critical systems vs. routine systems). Activities are prioritized between entities claiming priority, and preventing the “Pointy-Haired Boss approach” of "Make ALL THE THINGS highest priority!"
7) User wishes to achieve a goal that can be met by two (or more) methods: one with lower power but higher energy requirements (e.g. continuous transmission at +10dBm), or one with lower energy but higher power (e.g. pulsed transmission at +20dBm with a 1% duty cycle). The system must select which approach based upon power and energy availability and limitations.

2.1.2 Other Work in Energy Management APIs

This document focuses on EM-APIs, which amongst other things, involve managing energy such that a radio or wireless application is involved. However, as background, the Internet Engineering Task Force (IETF) has also addressed energy management for large scale terrestrial systems, and these publications offer some interesting perspectives. The following Request For Comments (RFC)s provide interesting insights: RFC 6988 and RFC 7326. The following sections extract principles potentially relevant to development of EM-APIs for systems that integrate with wireless functions.

2.1.2.1 RFC 6988

A few highlights of RFC 6988 relevant to energy management in wireless subsystems that are part of larger systems are highly synopsized below:

1) Power states of a system often consist of at least these conditions: off, sleep, full power.
2) There are tradeoffs between power consumption and service level performance.
3) There is local energy management control, but there may also be network wide energy management controls which may provide different optimizations. It may be desirable to combine these capabilities.
4) Energy management addresses total energy consumption. Power management addresses the rate at which energy is consumed. (It is feasible to manage power to control instantaneous grade of service level controls; it is feasible to manage energy to be able to provide a grade of service over an extended period of time or over specific instances of time.)
5) Mission critical systems may be managed differently than other components.
6) Awareness of energy source properties and their power limitations is important to energy management. This may consist of understanding battery properties and states and properties of any battery charging and/or discharging components. Similarly, awareness of energy consumers, and their power consumption properties is important to energy management. This may be in the form of awareness of power consumption states.

2.1.2.2 RFC 7326

A few highlights of RFC 7326 relevant to energy management in wireless subsystems that are part of larger systems are synopsized below:

1) An energy management system may consist of policies and procedures.
2) Target devices or components may be controlled directly or indirectly.
3) There may be more than one energy source and/or more than one energy consumer in a system.
4) Sensing the states of energy sources and consumers is useful to effective energy management.
5) Example Class structures are shown

2.2 Use of Presets

1) The use of any preset (for any system, not just the Transceiver) can have an associated power and energy implication. This can then allow the user of that entity to make choices based upon the energy implications of different presets. For example, a waveform may know a priori that presets 1 and 2 can achieve the same goal, but preset 2 may indicate it has a lower power consumption than preset 1. The waveform can then use that information to respond to power management instructions from other entities; selecting the lower power preset when needed, using a priori knowledge of how those presets change the performance of the entity.
2) The idea that a "preset" for an entity may have many sub-presets of different power implications, and that the entity may choose within that preset as needed to meet different commanded power needs. Those changes would have to be communicated to users of the entity, as they likely will change performance aspects of the entity.

2.3 Interactions between Energy Management Elements

2.3.1 Flow Between System and Subsystem Elements

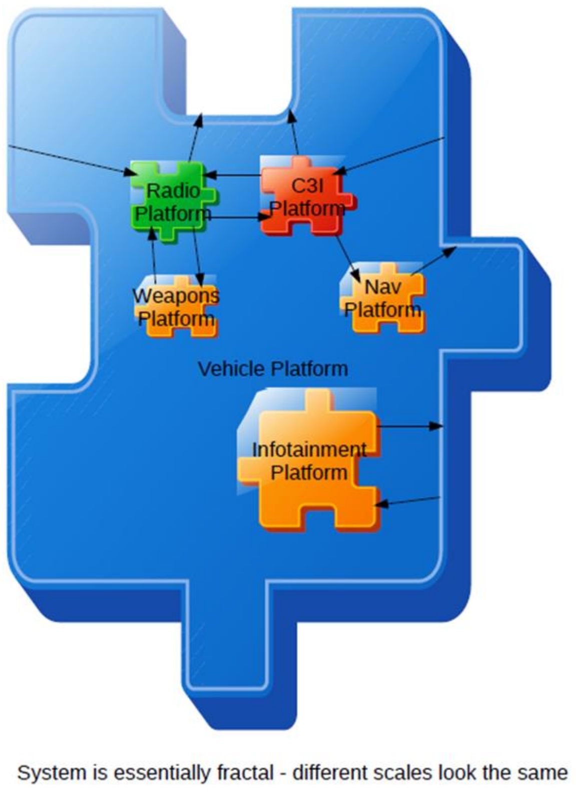

Figure 5 - Energy Management at the platform level and amongst subsystems inside of a radio may behave similarly.

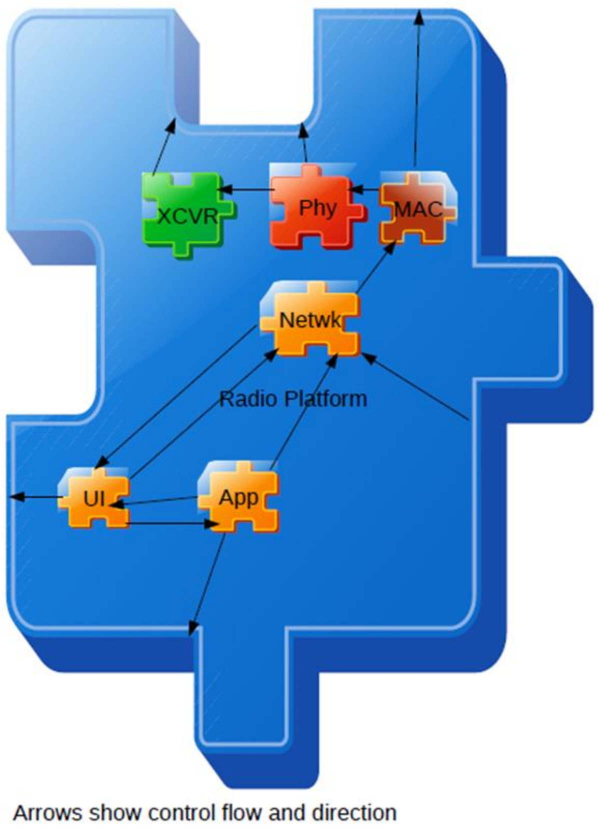

Figure 6 - Energy management inside of a radio may have processes similar to those at the platform level.

2.3.2 Signaling Exchanges Between Energy Management Function and other Energy Components.

Above, the concept of energy understanding throughout the system, energy presets, and the concept of an Energy Management Function that interacts with the energy sources and energy consumers were introduced. In this section, we provide examples of message exchange sequences to illustrate how the Energy Management Function might exchange messages to accomplish energy management functionality. Below, are six illustrations of message exchanges between system elements or subsystem elements to illustrate the basic principles of energy management exchanges.

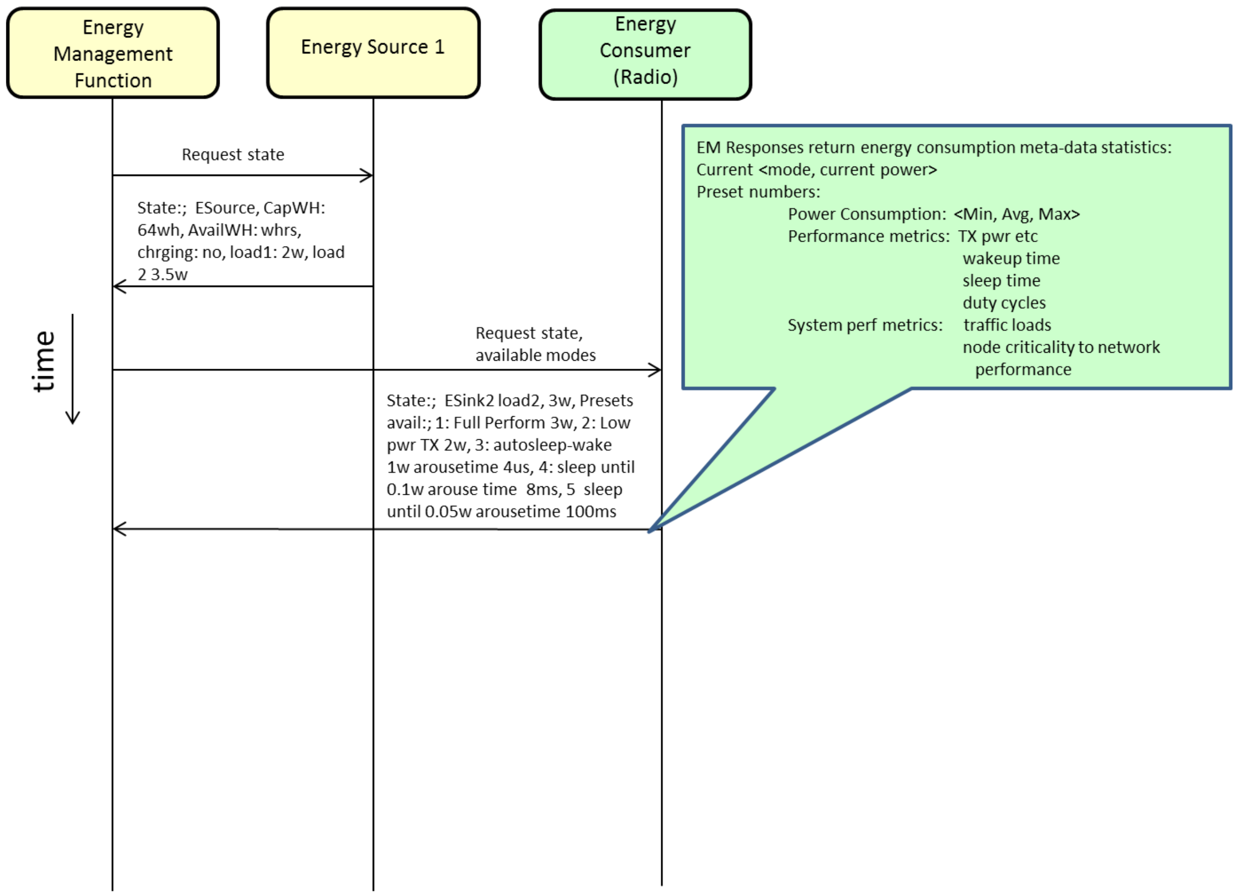
Energy Management Function
Energy Source 1
Energy Consumer (Radio)
Request state
State:; ESource, CapWH: 64wh, AvailWH: whrs, chrging: no, load1: 2w, load 2 3.5w
time
Request state, available modes
State:; ESink2 load2, 3w, Presets avail:; 1: Full Perform 3w, 2: Low pwr TX 2w, 3: autosleep-wake 1w arousetime 4us, 4: sleep until 0.1w arouse time 8ms, 5 sleep until 0.05w arousetime 100ms
EM Responses return energy consumption meta-data statistics:
Current <mode, current power>
Preset numbers:
Power Consumption: <Min, Avg, Max>
Performance metrics: TX pwr etc
wakeup time
sleep time
duty cycles
System perf metrics: traffic loads
node criticality to network performance

Figure 7 - Interactions between the energy management function, energy sources and energy consumers.

In this example, the Energy Management Function requests state, and statistics about the energy sources and energy consumers. The energy source reports its capabilities in watt hours, available watt hours (current charge), whether or not it is charging, what its current load is, and any other important information. The energy consumer (potentially a radio) reports its current state, presets that are available, energy statistics of the presets as well as the corresponding impact to performance. In this example, the energy consumer reports 4 presets and corresponding impact to performance. Since the lower energy states are created through use of sleep modes, and managing transmit power, the corresponding impact is provided regarding wakeup time cycles to shift between presets and thus respond to network traffic.

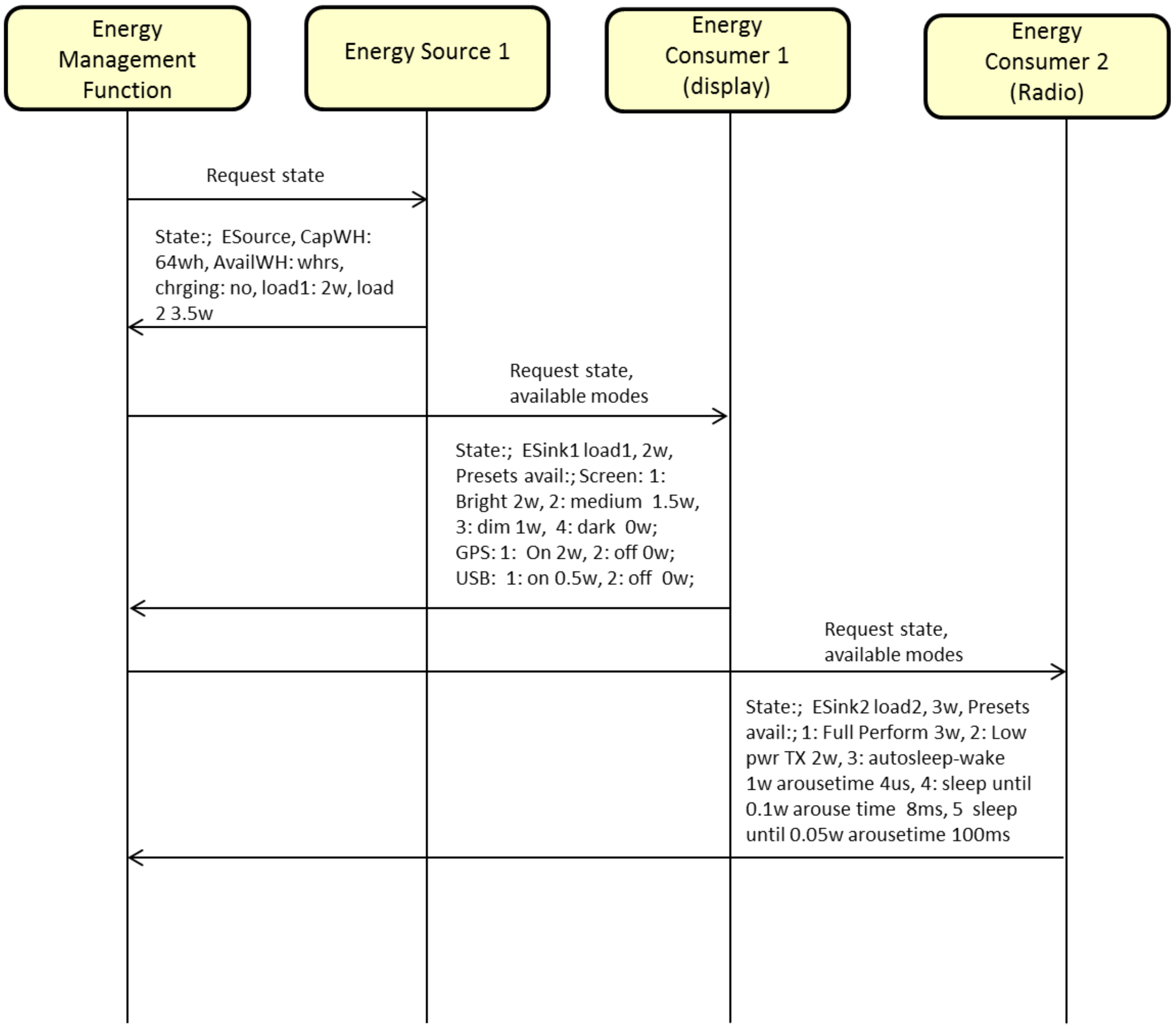
Energy Management Function
Energy Source 1
Energy Consumer 1 (display)
Energy Consumer 2 (Radio)
Request state
State:; ESource, CapWH: 64wh, AvailWH: whrs, chrging: no, load1: 2w, load 2 3.5w
Request state, available modes
State:; ESink1 load1, 2w, Presets avail:; Screen: 1: Bright 2w, 2: medium 1.5w, 3: dim 1w, 4: dark 0w; GPS: 1: On 2w, 2: off 0w; USB: 1: on 0.5w, 2: off 0w;
Request state, available modes
State:; ESink2 load2, 3w, Presets avail:; 1: Full Perform 3w, 2: Low pwr TX 2w, 3: autosleep-wake 1w arousetime 4us, 4: sleep until 0.1w arouse time 8ms, 5 sleep until 0.05w arousetime 100ms

Figure 8 - The energy management function interacts with both the energy source and with two energy loads.

In this signaling thread the energy management function interacts with the source and two energy consumers. In this case, the energy consumers are a display and a radio. Additional information regarding these modes, and their significance (bright – dark, etc.) is provided for understanding by the reader, but need not be understood by the energy management function. Primarily, the energy management function wishes to know the power consumption levels of the various presets and their criticality to various phases of the mission. This kind of representation will enable portability of energy management function designs. In a similar manner, there can also be more than one energy source, with different properties.

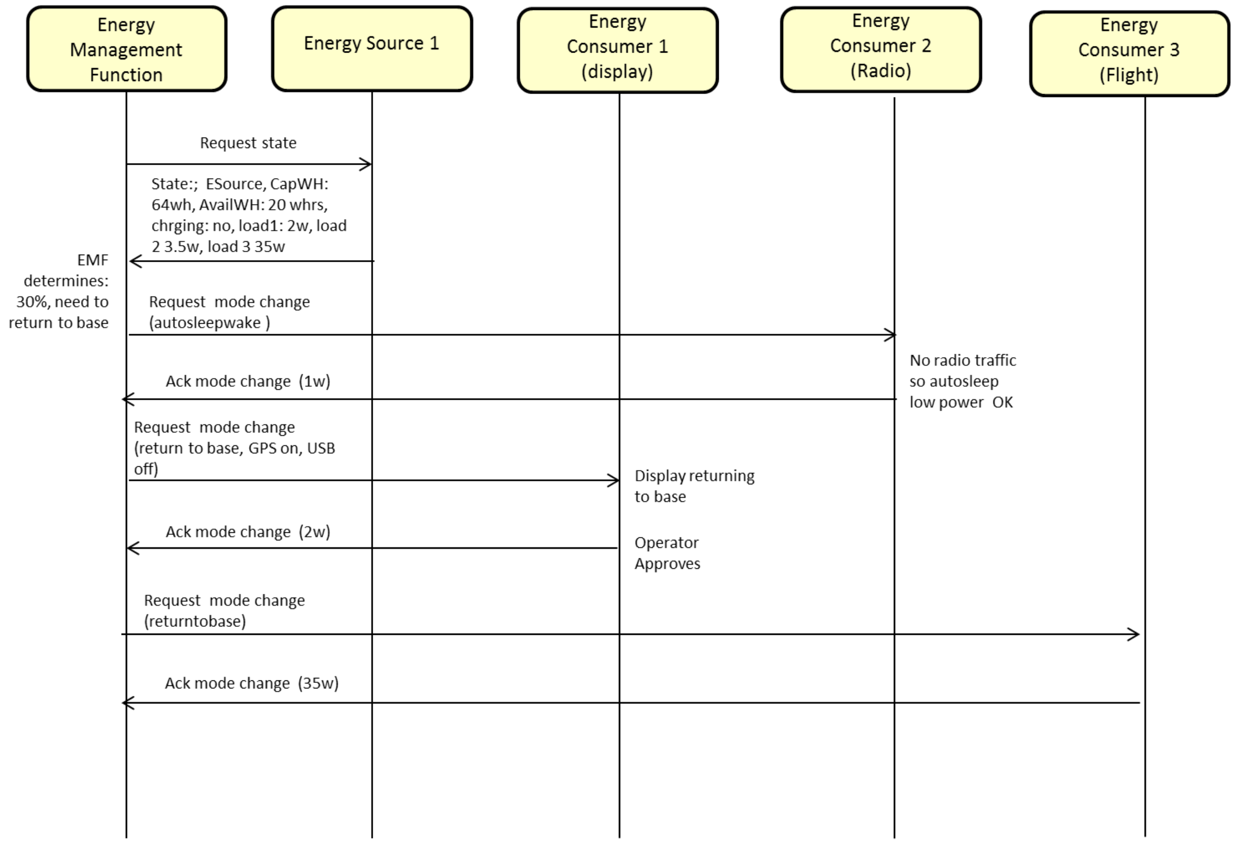
Energy Management Function
Energy Source 1
Energy Consumer 1 (display)
Energy Consumer 2 (Radio)
Energy Consumer 3 (Flight)
Request state
State:; ESource, CapWH: 64wh, AvailWH: 20 whrs, chrging: no, load1: 2w, load 2 3.5w, load 3 35w
EMF determines: 30%, need to return to base
Request mode change (autosleepwake)
No radio traffic so autosleep low power OK
Ack mode change (1w)
Request mode change (return to base, GPS on, USB off)
Display returning to base
Ack mode change (2w)
Operator Approves
Request mode change (returntobase)
Ack mode change (35w)

Figure 9 - An Energy management function may also need to support many energy consuming functions.

In Figure 9, the energy management function detects that the energy source has depleted to the 30 percent level. It's programmed function is then to request reduced energy consumption of display and radio, and corresponding reduced performance, and to instruct the flight system to return to base at this point.

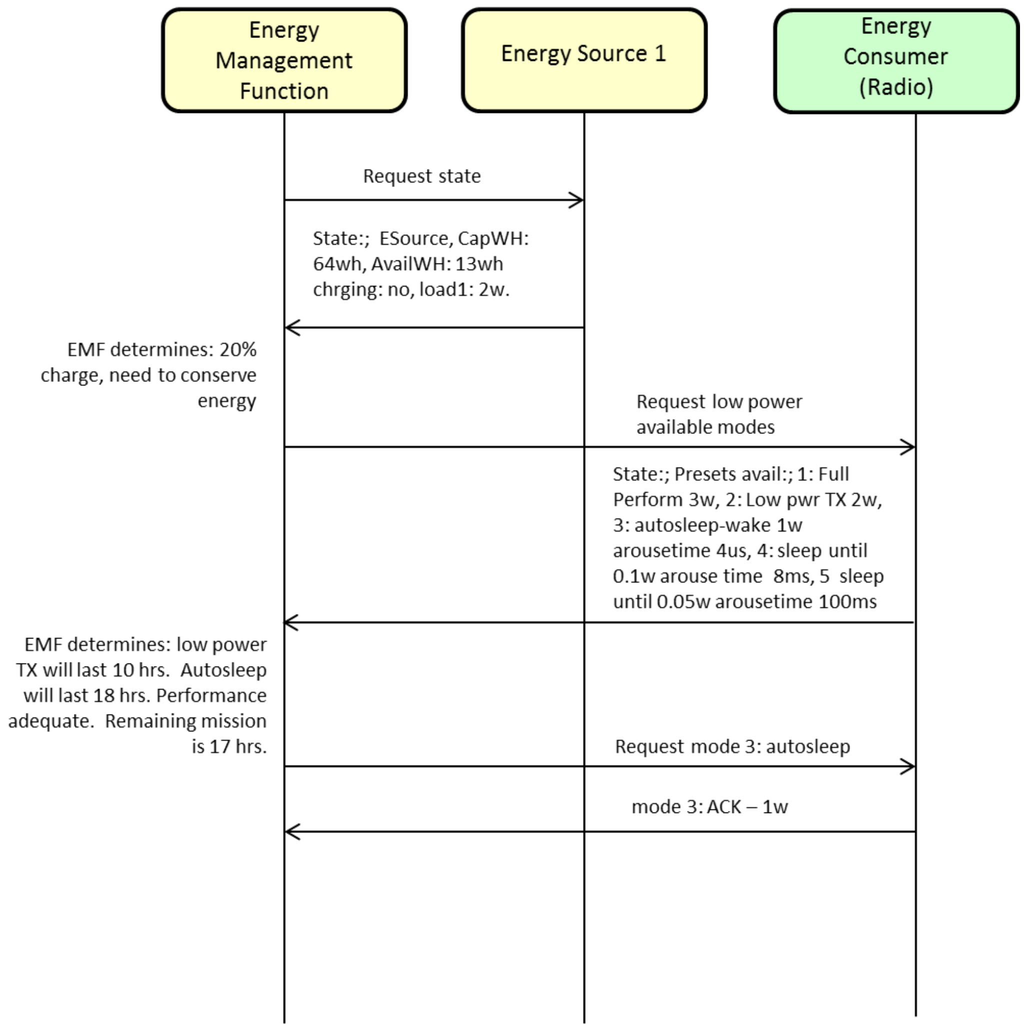
Energy Management Function
Energy Source 1
Energy Consumer (Radio)
Request state
State:; ESource, CapWH: 64wh, AvailWH: 13wh chrging: no, load1: 2w.
EMF determines: 20% charge, need to conserve energy
Request low power available modes
State:; Presets avail:; 1: Full Perform 3w, 2: Low pwr TX 2w, 3: autosleep-wake 1w arousetime 4us, 4: sleep until 0.1w arouse time 8ms, 5 sleep until 0.05w arousetime 100ms
EMF determines: low power TX will last 10 hrs. Autosleep will last 18 hrs. Performance adequate. Remaining mission is 17 hrs.
Request mode 3: autosleep
mode 3: ACK – 1w

Figure 10 - The energy management function determines that it must further reduce energy management in order to complete its mission profile. It requests a lower energy preset after determining that it will not substantially impact the final stage of the mission.

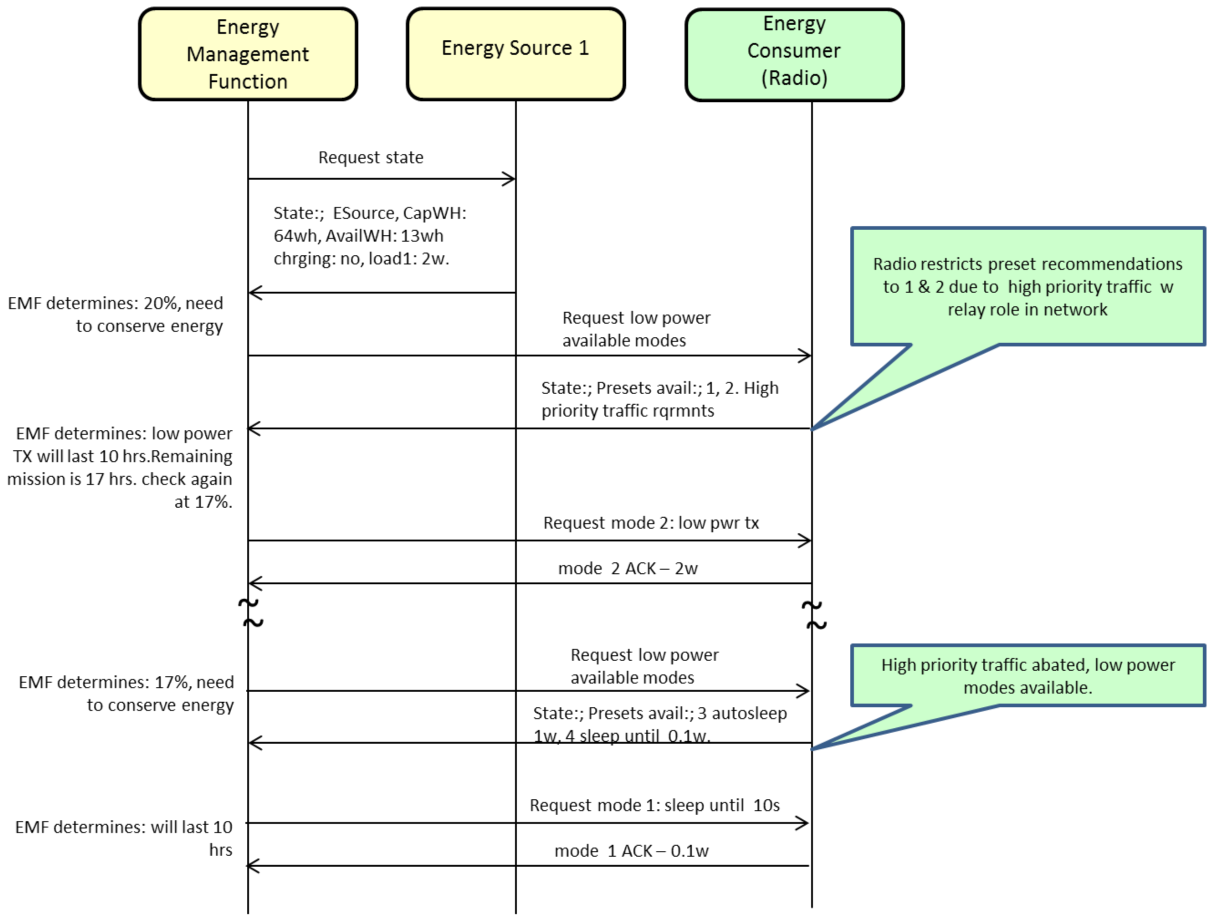
Energy Management Function
Energy Source 1
Energy Consumer (Radio)
Request state
State:; ESource, CapWH: 64wh, AvailWH: 13wh chrging: no, load1: 2w.
EMF determines: 20%, need to conserve energy
Request low power available modes
Radio restricts preset recommendations to 1 & 2 due to high priority traffic w relay role in network
State:; Presets avail:; 1, 2. High priority traffic rqrmnts
EMF determines: low power TX will last 10 hrs.Remaining mission is 17 hrs. check again at 17%.
Request mode 2: low pwr tx
mode 2 ACK – 2w
Request low power available modes
High priority traffic abated, low power modes available.
EMF determines: 17%, need to conserve energy
State:; Presets avail:; 3 autosleep 1w, 4 sleep until 0.1w.
Request mode 1: sleep until 10s
EMF determines: will last 10 hrs
mode 1 ACK – 0.1w

Figure 11 - The energy management function attempts to lower energy consumption in order to complete the mission, however, the energy consumer is briefly unable to comply with the request. The energy manager asks again after a brief interval.

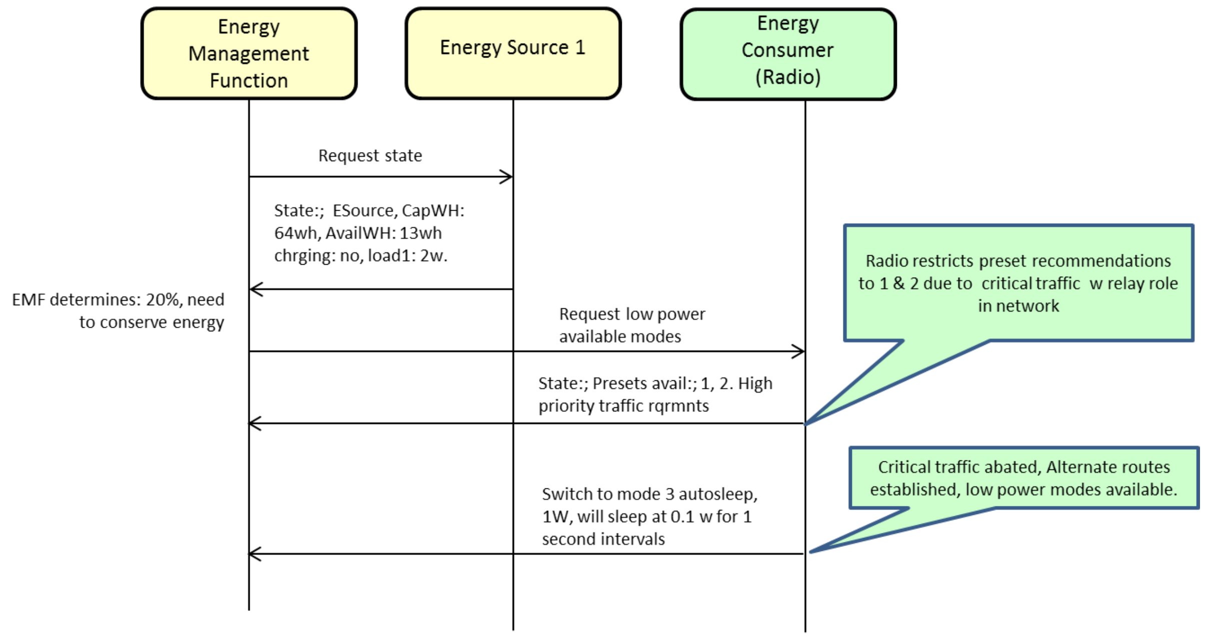
Energy Management Function
Energy Source 1
Energy Consumer (Radio)
Request state
State:; ESource, CapWH: 64wh, AvailWH: 13wh chrging: no, load1: 2w.
EMF determines: 20%, need to conserve energy
Request low power available modes
State:; Presets avail:; 1, 2. High priority traffic rqrmnts
Radio restricts preset recommendations to 1 & 2 due to critical traffic w relay role in network
Critical traffic abated, Alternate routes established, low power modes available.
Switch to mode 3 autosleep, 1W, will sleep at 0.1 w for 1 second intervals

Figure 12 - Similarly to Figure 12, the energy manager makes requests to lower energy consumption. However, the energy consumer answers that only selected presets are available due to a critical mission function (critical mission traffic in the radio network). After successfully shedding its critical activity (relay traffic transfer handed to an alternative relay node), it responds to the energy management function that it has now switched to a lower energy preset, without being asked again.

3 Introduction to the Energy Management API Interface Description Language

There are a few entities in the system that this introduction will discuss. Software Communications Architecture (SCA) terminology may be occasionally utilized, but that should not imply that this EM-API should only applied to SCA based SDR designs. Rather, it is hoped that the EM-API enables energy management in a very wide variety of software defined radios.

To begin considering the context of the EM-API, consider the SDR radio as an integrated component of a platform. The platform is an over-all framework into which all the components of the system have been inserted. The platform has both a physical (hardware) existence and a software existence.[1]

The energy managing supervisor of the platform interacts with components, which may have been supplied by different organizations than the supplier of the platform. These include Application(s) and the Resource(s) that make up the Application, and Device(s) that manage hardware.

There needs to be only a single energy managing functionality in the system - there's ONE Energy Management Supervisor in a system (note: if the system is actually a system-of-systems, the master system has only one Energy Management supervisor, and the sub-systems have their own internal Energy Management Supervisor at that level that only addresses energy management within that specific sub-system). Being a singleton, the Energy Management Supervisor should be a service.

The energy management functionality of the Energy Management Supervisor is difficult to acquire because the energy source, the radio, the platform and the network functionality may have each been developed by separate organizations. For this reason, it is essential to energy management portability that each function contain the corresponding knowledge inside each EM-API driver. The role of the implementer of the energy management function scopes the level of knowledge integrated into the Energy Management service. If it is supplied by the platform vendor, the EM cannot have detailed knowledge of all the devices, or of the mission. If it is supplied by the integrator, it can know about devices, but cannot have detailed knowledge about the mission. If supplied by the end user, it can know the mission, but the EM cannot have detailed knowledge about the devices. Because of the implied hierarchy of energy management supervisors the system is able to provide the performance in response to the energy management supervisor. The platform energy management supervisor sets the overall energy management goal for the platform and mission and the lower levels of supervisors respond to and implement the energy management corresponding to that goal.

3.1 Making Energy Management Broadly Applicable to Multiple Systems

The EM-API process defines only the API itself; determination of the API's use is left to industry.

[1] In SCA implementations, "the platform" would be the Domain Manager and would provide energy management service(s) on the platform.

Conceivably the Energy Management Supervisor can be loaded with extra information by the Applications[2]. What is loaded needs to provide enough information to allow the energy management system program to make deterministic decisions.

Given the above, the EM-API is very careful about what information is passed to the EM Supervisor. For example, micromanaging power sources should not be based upon battery voltage - that interface should be handled by a device driver for the energy source hardware, written with detailed knowledge of how the hardware works, rather than trying to hand the management of the hardware off to an EM Supervisor that is written with no knowledge of the hardware.

The approach to creation of this EM-API IDL is based on the premise that the more specialized the information, and the less that is abstracted, the more likely it is that no one has enough information to perform the design. If the EM-API is too detailed it becomes important to understand the platform hardware, the device hardware, the application designs, and the mission, and thus becomes less reusable.

3.2 The API as Described

Every energy source SHALL have an EM device that represents it. That device is responsible for any low-level management of that source (there is a device driver):

1) it provides two-way communication with the hardware
2) it knows how to make hardware adjustments as required
3) it abstracts hardware management details and provides information about itself to the energy management supervisor
4) it implements the PowerSource interface

In cases where a Platform can have many energy sources, there shall be many Devices, one per source. The energy management function can receive a list of Energy Sources[3]: There are two ways to handle systems that have power sources that are only sometimes available:

1. When the power source is available, a Device is registered with the Platform, and when the power source is not available the Device is removed[4].
2. There is always a Device for that power source; when the power source is not actually connected the Device reports SourceInfo::SourceStates=NotConnected, SourceInfo::maximum=0; SourceInfo::surge = 0; SourceInfo::capacity=0, SourceInfo::runTime = 0.

[2] Making it a Loadable Device in SCA terms rather than a Service

[3] If using an SCA system this type of functionality can be easily handled by APIs associated with SCA.

[4] In SCA this requires more dynamic registration, which is available in SCA 4.1 but difficult in SCA 2.2.2.

To be clear: The concept of "component" as deliberately defined within the EnergyManagement namespace to differentiate it from any other concept of Component, e.g. the SCA concept of "Component".

For an SCA implementation of these APIs, it is anticipated they would be ports on the SCA resource - that is to say, an SCA "Component" (device, resource, service, etc.) would have a port named "EnergyMgmt" that implemented the EnergyManangment::Component interface. Likewise, whatever is serving as the Energy Management controller (be that a platform resource like a Service, a platform Device, or a platform Application Assembly) would have a port named EnergyMgmtCtrl, implementing the energy management controller APIs (the event notification services). That SCA level component might also implement the EnergyMgmt port and controls as well, to allow a system-of-systems access to control the whole platform as an energy management component.

Notes on PowerStates

The reason for having power states is that some of them represent situations where the component cannot "do its job" - called the dozing and sleeping states. These states are useful when a managing entity knows that a component isn't needed right now, but will be at some point in the future. Imagine a TDMA system - the PHY layer may know that the receiver isn't needed right now, but will be needed in 100usec. If it is known that the receiver can wake from dozing in 20usec, but takes 1ms to wake from sleeping, the receiver can be commanded to doze, and in 70usec command it to full operation, but the receiver cannot be commanded to sleep, as it will miss the time slot.

There is no requirement that a component implement a preset in any of those states - just that presets exist. The semantics of those states are defined precisely so that code can rely upon them to behave in a meaningful way - thus the idea that "dozing" means anything to a device is not valid; rather dozing means "unable to function but able to resume that function more quickly than if sleeping".

There are two main API domains: one implemented by a component and used by other components to manage energy, and one implemented by an energy manager function and used by components to report energy states. There will also be some data structures. The entity making the decisions is a singleton. However, each entity providing or consuming energy must enact those decisions (one boss, many workers).

The work group on EM-API found a disagreement regarding SourceInfo::type. The two opinions regarding SourceInfo::type are reflected immediately below in paragraphs labeled 1) and 2).
1) “A type enumeration can include a list of product part numbers that would allow proprietary implementations of power management. This degree of flexibility in Group IV facsimile substantially expanded the FAX market, but at the cost of resulting in many proprietary implementations. An enumerated types list could be generated by API users, with the master list of nonproprietary considerations maintained on line, e.g. by the Wireless Innovation Forum. This implementation can allow more highly optimized power management, at the risk of non-standard implementations. API users may decide to augment this API specification with such a type enumeration.”
2) “While the SourceInfo structure contains a field for the type of energy source, code should not test against SourceInfo::type (and even more-so SourceInfo::name), but rather test against the various feature fields in the structure. Testing against the type (or name) is brittle: code using this approach will not be

able to handle a new energy source type, and will have to fall back to either default behavior, or testing against the feature fields (as recommended). If the feature fields do not adequately describe a concept that energy management needs, then *revise the specification* to add fields for those concepts. It was suggested during the drafting of this specification that by testing against type or name, it becomes possible to allow for vendor extensions outside the specification to be used - but the whole point of creating this specification is to allow interoperability and portability, which is prevented by using vendor extensions."

Likewise, there is a list of Components that consume power. Again, in SCA we can iterate over all Devices and Resources, and attempt to _narrow() to the EnergyManagment::Component interface. In a non-SCA system we will need another way to do this. We can then use the EnergyManagment::Component interface to influence how the component uses power. Now, in an SCA world, one should expect the EM would only access Devices and Applications, not the individual Resources in an Application - let the Application Assembly Controller manage its Resources.

3.2.1 CORBA IDL representation of the Energy Management API follows:[5]

```
/* The Energy Management API IDL is broken up into several portions.  Standard Typedefs, Standard Energy Consumer Power
states (enums), 2 struct sequences, one for Preset, and another for PresetTransitionInfo, enums for SouceTypes, and
SourceStates, and stucts for SouceInfo.  The main interfaces are Component, ComponetEvent, PowerSouce, and PlatformPower.

The Energy Management API is designed so that the Application layer may have a generic interface to the hardware, attempting
to assist the hardware in making decisions on power usage and consumption.  Information is also obtained on an event
notification basis to the Application layer informing that layer of Power usage and consumption details. The API does not
supersede the Vendor's inherent energy management efficiencies made at the hardware level, rather the intent is to enhance or
complement such features if they are available.
*/

{

typedef float        Watts_T; // Describes a power in watts
typedef float        Joules_T;
typedef float        Seconds_T; // time value
typedef unsigned short PresetID;

// Energy consumer power states.
enum PowerStates
{
 Off,          // Unable to perform operations, no power consumption. May take some time to come back to operation
 MinimalPower,   // Unable to perform operations. Minimal power consumption. May take some time to come back
 LowPower,      // Unable to perform operations. Low power consumption. Will quickly resume operations
 Standby,       // Unable to perform operation. Can resume operations "very quickly"
 LowPerformance, // Able to perform operations, but potentially not at an optimal level for some values of "optimal"
 HighPerformance // Able to perform all operations at best performance.
};
```

[5] CORBA http://www.omg.org/corba/omg_IDL.htm

```
// Consumer preset
struct Preset
{
    PresetID    id;    // Preset # - used by the APIs to specify which Preset
    string      name;  // Human meaningful name for preset, largely ignored by the APIs. SHOULD be unique.
    Watts_T     rms;   // RMS power consumption over "a long time"
    Watts_T     peak;  // maximum power consumption for short periods of time. peak/rms should yield the crest factor.
    PowerStates state; // Over-all concept of what we are doing.
    Any         SpecialParams;
};
typedef sequence<Preset> Presets;

// Description of the cost of transitioning from one preset to another.
struct PresetTransitionInfo
{
   PresetID  from;   // the state being left
   PresetID  to;     // the state being entered
   Seconds_T time;   // Time the transition will take to occur
   Joules_T  energy; // Energy cost for the transition
};
typedef sequence<PresetTransitionInfo> TransitionsInfo;

// Energy source types.
enum SourceTypes
{
   Primary, /**
            * Primary energy sources that have a finite energy and cannot be replenished.
            * Includes alkaline batteries, RTGs,
            **/
   Storage, /**
            * Secondary energy source that can store power from other sources and/or provide power.
            * Includes rechargeable batteries, capacitors, flywheels
            **/
   External, /**
             * Sources nominally external to the "system", such as grid power,
             * power from a vehicle not considered part of the system, etc.
             **/
   Generator  /**
              * Sources that consume a fuel to make power, that are under some degree of control
              * from the system, such as being able to start/stop, read state, etc.
              * Fuel cells, generators,
              **/
};
// Energy source power states. This enumeration also subsumes the role of a source type, as some states only
// apply to certain source types. Source types listed in () at beginning of comment.
enum SourceStates
{
   NotConnected // (all types) the actual source of power isn't present.
   Offline,     // (all types) Not able to supply power, but theoretically could change states if asked
   Grid,        // (External) Power grid sources.
   Running,     // (Generator) - operating and able to supply power
   Charging,    // (Storage) - Replenishing stored energy. May supply power as well
   Discharging, // (Storage) - Consuming stored power
   Floating,    // (Storage) - neither replenishing nor consuming stored energy, but providing power
   SoftFault,   // (all) There's something wrong, but can still supply power...
   HardFault    // (all) Unable to supply power.
};

// Energy source description
struct SourceInfo
{
   const Joules_T  UnlimitedEnergy = -1;
   const Watts_T   UnknownPower    = -1;
   const Watts_T   UnlimitedPower  = -2;
   const Seconds_T Forever         = -1;



   string       name;      // Human readable name
   SourceStates state;
   SourceTypes  type;
   Watts_T      maximum;   // Maximum sustained power available.
   Watts_T      surge;     // Maximum power available for "short periods" of time...
   Seconds_T    surgeTime; // Maximum duration of a "surge". May be 0 for sources with no surge capability.
   Joules_T     capacity;  // available energy left. Things like wall power should have UnlimitedEnergy;
   Watts_T      load;      // amount of power consumed by loads, steady state. If unable to measure, report UnknownPower
   Watts_T      peakLoad;  // amount of surge power being consumed. If unable to measure surge, report UnknownPower
   Seconds_T    runTime;   // Amount of run time at current load. May be Forever for things like grid power.
};

// the interface a component that consumes energy will implement
interface Component
{
   exception UnknownPreset // PresetID not found in the preset list.
   {
   };
   exception SettingsConflict // Selected preset not compatible with current component settings
   {
      string why; // Human readable description of what the problem is.
   }

   Presets        GetPresets();                  // get *all* presets, including ones that are not currently allowed due to component
needs.
   Presets        GetALlowedPresets();           // Get presets compatible with current component operation.
   PresetID        GetCurrentID();               // Get what preset is currently in force.
```

```
  Preset        GetCurrentPreset();                // This is just a convenience function, to avoid GetPresets() then GetCurrentID()
and indexing.
  void          SetCurrentID(in PresetID id)        // Set the desired preset
    raises (UnknownPreset,SettingsConflict);
  TransitionsInfo getTransitions();                 // Report our transitions
  void          SetPowerTarget(in PowerStates state); // Set the goal, let the component pick the preset.
};
typedef sequence<Component> ComponentList;

/**
 * The interface that an energy consumer MAY use to report events.
 * Would be implemented by any component in the system that wants to be advised
 * of events
 **/
interface ComponentEvent
{
  oneway void StateChange(in Component component, in PresetID preset);
};

// The interface an energy source will implement
interface PowerSource
{
  SourceInfo getInfo();
  void      Enable(); // Some sources, like generators, may need to be enabled, a.k.a. started
  void      Disable();
};
typedef sequence<PowerSource> PowerSourceList;
/**
 * The interface an energy source MAY use to report events.
 * Would be implemented by any component in the system that wants to be advised
 * of events
 **/
interface SourceEvent
{
  oneway void StateChange(in PowerSource source,in SourceInfo info);
};

/**
 * API to be provided by the platform for use by energy management component or other components.
 * This may be a service within the system.This could also be accomplished in an SCA system by
 * getting a list of Devices from the Domain Manager and iterating over them, attemping to _narrow()
 * to the PowerSource interface.
 */
interface PlatformPower
{
  ComponentList   GetConsumers();
  PowerSourceList GetSources();
};

};
```

Appendix A
Examples of energy saving opportunities in typical SDR radio architectures

The objective of Appendix A is to provide examples of methods that common radio architectures may consider as means to manage power at relevant layers. As such, concepts in this chapter are examples but are not required for compliance to the standard EM-API because some radio designs may differ, or have different implementation requirements. This chapter is simply an example of various means to save energy in common radio designs.

3.3 Examples of Energy Saving Controls In Typical Radio Designs

Within an SDR radio there are very many design tradeoffs. Often these tradeoffs are conscious choices made between complexity and energy consumption and the probability of requiring the level of performance indicated by the complexity. However, in software the ability to select various levels of complexity and energy consumption to adapt to spectrum or traffic conditions may easily be accommodated within the software design. In this chapter, various means to adapt energy consumption of an example of a current radio architecture are identified as examples of functions which could either be aggregated into presets or otherwise controlled through energy management APIs from an external EM-API.

It is recognized that not all of the listed features will be relevant to every SDR Radio architecture or system implementation. As such these functions are less portable across various radio architectures, and are only provided as example functions for radio designers to consider rather than required components of the standard API.

3.4 Transmitter Energy Management Opportunities

Transmitter functionality includes management of many RF transmit functions including: Antenna, Power Amplifier, and Filtering. Each of these areas have many further features which may be controlled with regard to mode and the mode’s corresponding power/energy consumption.

Example Transmitter Attributes of interest to the system include:

1. power consumption at a given frequency and power level,
2. power consumption in sleep and idle modes, and
3. turnaround time from sleep to active

Typical Transmitter (Transceiver) interfaces already have mechanisms to control the transmitter output power, frequency, and Transceiver state (sleep,idle,on). Once the system has queried for the above Transmitter properties, it can then, for instance, decide whether changing the RF transmit power level of this node results in the desired savings and simply use the existing interfaces to accomplish the energy savings. Transmitter information can also be passed upward and aggregated over the network so that larger decisions can be made about whether a particular node would be better suited to go to higher RF power level to take advantage of efficiency and allow other less efficient nodes to idle or sleep. The system may also find that some nodes are more efficient at certain frequencies and may decide to use alternate frequencies in order to conserve energy. The turnaround time from sleep to active can be used by the Waveform application to find opportunities for the Transmitter to enter a lower power state.

3.4.1 Smart Antenna Control

Antenna control may involve a smart antenna. A smart antenna may be capable of adjusting its frequency response, impedance match, polarization and beam patterns. Mechanisms for accomplishing management of antenna functions may involve energy consumption. Such functions may either continuously consume power or may intermittently consume power upon a control parameter state change, after which they cease to consume additional energy.

3.4.1.1 Antenna Set Frequency

The following controls address Antenna Frequency response:

- Antenna Center Frequency (center frequency, bandwidth)
- Sleep Antenna ()

3.4.2 Power Amplifier Controls

It is well understood that there are many modes of modulation, and that there are many ways of configuring a power amplifier (PA) to be as efficient as possible when transmitting, and to achieve the lowest possible energy state when not transmitting. In order to achieve high energy efficiency, it may be necessary to provide various forms of bias adaption to the PA, adaption of the filtering circuits, adaption of distortion management techniques, management of transmit output power, management of the power supplies feeding the power amplifier(s), as well as controlling center frequencies and bandwidths. Depending on circuit design properties, these circuits may continuously consume energy, or may consume energy to accomplish control state changes.

3.4.2.1 Bias / PA circuit Mode Configuration

Features can be provided to enable PA mode controls that support selection of the most energy efficient mode available to support the currently selected modulation type:

- Set PA Bias Mode (mode)
- Set PA Distortion Management (mode)
- Set Filters (center frequency, configuration)
- Set D/A (sample rate)
- Set PA Power (on/off, rise/fall rate, envelope tracking modes)
- Set TX (on/off, tx pwr)
- Sleep PA()

PA Bias modes can be used to configure the bias to the PA to various operating modes that provide various levels of efficiency and which relate to the linearity required for a given modulation. Older designs may support various bias levels for class A, B, or C operation.

Advanced designs may require controls for modes D,E,F,G, or S if switching amongst these modes is supported in hardware and if waveform choices need a variety of modes.
PA Distortion management modes provide for configuring predistortion, or feedback based distortion controls. Examples include Volterra Predistortion, or I/Q feedback distortion. Circuits associated with these distortion management modes are likely to need to have clocks gated to off when not needed in order to save energy.

Many SDR radios are agile in frequency. Energy may be conserved if corresponding filters are tuned to the current frequency, rather than set to a single frequency for an entire band. This is particularly important to the energy efficiency of the RF power amplifier. Furthermore the power amplifier filter topology can be configured to various modes to gain the maximum energy added efficiency (such as selecting for modes E, F, G, S) consistent with the modulation type.

Adjusting the sample rate for the D/A converters can be used to make the corresponding signal processing functionality run at a clock rate that corresponds to maximum efficiency required to support the intended bandwidth.

PA power can be used to manage the power supplies that feed the transmit power amplifier. As these are likely rather complex it is desirable to turn all of the power supplies off when there will be a reasonable period of time without transmission. The same controls can be used to control Transmit (TX) power rise & fall times to shape the transmission envelope, and can be used to control envelope tracking supplies on or off if that feature is available. It is expected that envelope tracking circuits can consume significant energy, so the ability to disable them when not needed is significant to energy saving.

Ability to cause the PA to sleep can be is used to turn off as many energy consuming circuits as possible when there is likely to be a time period with no transmissions long enough that energy can be saved.

3.4.3 Filter Controls

Filter properties are also fundamental to the energy efficiency of the power amplifier. Filters are used in many ways: as pass band filters, as band reject filters, and as spectrum shaping filters to the modulation. Some filters may be implemented digitally, and some as tuned inductors and capacitors (LC) or strip line circuits. All of these are able to be managed or even configured for maximum energy efficiency. As an example, circuits in the output power end of the power amplifier (collector or drain) may be configured to achieve mode E, F, G, or S to properly support the most efficient PA configuration for the current modulation type. Digital filters that shape the spectrum as supplied to the input of the power amplifier often may be implemented as Finite Impulse Response (FIR) filters. Depending on many factors, the numbers of taps may vary, and so making use of the FIR filter with the minimum taps required to meet the required spectral mask can be an energy efficiency management. Even more importantly, if the clocks for the FIR filter computations can be stopped when the node is not transmitting, that can result in high energy efficiency as regards the spectrum shaping function. And certainly setting the center frequency of the filters to the intended center frequency of the signal will also deliver maximum power added efficiency.

3.4.3.1 Domains to support transmit filter configuration and tuning

The following domains are available to support transmit filter configuration and tuning:

- Set TX PA Filter Configuration (mode)
- Set TX PA Center Frequency (Fc, BW, Configuration)
- Set TX Spectrum Shaping Filter (Ntaps, coefficients)
- Set TX Mode (on/off)
- Sleep TX ()

3.5 Receiver Energy Management Interfaces

As with the transmitter, there are many circuit functions where energy can be efficiently managed in the receiver function. Which functions are applicable will be strongly dependent on the receiver design, including whether it is direct digitization, single conversion, or double conversion, and properties of Local Oscillators, analog and digital and filters used in the receiver.

3.5.1 Receiver Energy Management Configuration

Similar to the Transmitter, the system should have opportunity to manage receiver energy consumption.

Example Receiver Attributes of interest to the system:

- Power consumption in sleep and idle modes.
- Turnaround time from sleep to active.

Again, similar to the Transmitter, the system can query the Receiver attributes and use the returned information to make decisions about when to sleep the Receiver functions and for how long.

A receiver may have a band select tuning function. If so, it will be desirable to be able to tune the RF center frequency and bandwidth. The desired tuning may need to consider out of band interference, and the digital signal processing complexity to suppress interference if interference currently exists. Setting the filter edge tuning may reduce the number of digital filter taps needed to suppress spurs, and out of band interference as an interaction with receive sensitivity.

It may also be relevant to adjust the sensitivity of the receiver by adjusting the bias currents of Low Noise Amplifier (LNA), Mixer, and Intermediate Frequency (IF) circuits depending on sensitivity and signal strength.

It may also be relevant to be able to adjust the number of bits from the A/D converter also depending on sensitivity and signal strength.

As with the transmitter, spectrum shaping filters may be adjusted to the minimum number of taps to achieve the required IF bandwidth and receive modulation shape factor, and clocks may be halted when the receive process is not required. The same concept can be applied to the equalizer or rake filter.

Local Oscillators may not be gated on and off, but drive to the mixer may potentially be gated when not receiving. Similarly, LNA and IF amplifiers may be gated off when unneeded.
The A/D sample rate may be set to a sample rate that minimizes the computational complexity of the filtering and demodulation processes 4.1.1
The following energy management opportunities may be available in common receiver designs:

- Set RX LNA Filter (frequency)
- Set Bias currents (LNA, mixer, IF)
- Set A/D (Number of bits, samplerate)
- Set FIR Filter (Filter#, Ntaps, Coefficients)
- Sleep RX ()

3.6 Processor Energy Management

The extent to which an SDR Radio architecture has a General Purpose Processor (GPP) and how it is used if present, directly affects which mechanisms can have the greatest effect on energy consumption of the processing functions. Analysis of common SDR architectures indicates that processors consume a relatively large percentage of the energy consumed in the radio. Processors commonly consist of a mixture of general purpose processors (GPPs), digital signal processors (DSPs), and custom processors (consisting of either Field Programmable Gate Array or other special purpose chips). Which of these processor types are utilized, and what they are utilized for has a very profound effect on energy.

Equally importantly, how they interact with network level controls to recognize and make use of periods when the processors can save energy is an extremely important consideration. Since processors usually include Dynamic Voltage Scaling (DVS), Dynamic Frequency Scaling (DFS), as well as a variety of sleep modes that adjust sleep power consumption and wakeup recovery time, these DVS, DFS, and sleep modes must be used to full advantage by the network to find and make use of the best mechanism to reduce power when it is possible, specifically when the applications and network traffic allow the radio an opportunity to save energy in some fashion.

Field programmable Gate Arrays (FPGAs) do not typically come with clock management schemes for each signal process that they may be asked to implement by the product designer. This is left to the FPGA designer. For this reason it is incumbent on the SDR designer to build in clock management for each function that consumes meaningful amounts of energy implemented in the FPGA. And while the entire FPGA may have many different functions that it performs in parallel, some FPGAs do support a limited degree of DVS and DFS.

It is essential that the energy management control mechanisms provide a high degree of DVS, DFS, and clock gating in order to save energy in every opportunity that occurs when the network believes there is an opportunity to reduce energy consumption and can afford the processors the opportunity to take advantage of it.

It must be recognized that the radio will pay a cost of some time and some energy to change mode to a reduced energy state. As such, logic must assess whether the energy saved and the transition time required to change modes is a net savings. If the reduced energy time window is long enough the answer will be positive, but the system must also choose amongst multiple reduced energy consumption states, based largely on how long is needed to save state, restore state, and to return to full performance and then be ready to handle transmit and receive traffic. Energy saving APIs for the processors within a given SDR architecture can be organized by processor type or by radio functionality while hiding which processor type performs the implementation. However, in order to keep management of energy implementation independent, it is recommended that processing functions focus on identifying the opportunity to sleep the receive or transmit functions and the duration of the opportunity, and leave the implementation to choose the best match to the energy saving opportunity.

3.6.1 Energy Saving for processing engines

Functions can provide awareness to processing engines of opportunity to reduce energy consumption of the processing associated with transmit functionality and with receive functionality. Duration of the opportunity enables the underlying drivers to choose appropriate sleep mechanisms:

- SET TX Processors (active/sleep, sleep duration)
- SET RX Processors (active/sleep, sleep duration)

3.7 EM Interactions with the network

The network level of the ISO stack is able to be aware of network level activity. It is able to know expected activity level not only of one node, but also the neighbor community of nodes, so that it can be aware if it may be called upon to interact with neighboring radios. It can also be aware of the state of network alert beyond immediate neighbors, and the current activity cycle of when a node will be called upon to wake up and receive, transmit, or relay. Therefore, the network may also be aware of local and network level applications, their importance, and their effect on radio level activity and alertness required to support that activity.

This section is devoted to discussing examples of energy saving opportunities that provide the ability for EM algorithm designers and software architects to interact with the network and harness cross-layer information in order to control energy expenditure at the PHY layer. In particular, we discuss APIs that provide access to knobs at the MAC and Networking/IP layer to adapt networking-related functionalities and control the amount of RF power consumed by the nodes in the network. As mentioned in Chapter 1, knobs at the aforementioned layers can be set based on mission level information (e.g., CONOPs) and network level information (e.g., battery levels of intermediate nodes in a network) to control the amount RF transmissions at the PHY layer.

Control Knob	Description	Network (OSI) Layer	Notes
Neighbor_Discovery_Timer <t>	Proactive Routing Protocol Neighbor Discovery Frequency	Networking (IP) Layer	• Used with proactive link-state routing to control sleep-wake duty cycles at routing layer (and in turn frequency of RF transmissions at PHY layer) • N/A (don't care) in case of reactive routing protocol
Topology_Message_Timer <t>	Proactive Routing Protocol Topology Information Dissemination Frequency	Networking (IP) Layer	• Used with proactive link-state routing to control sleep-wake duty cycles at routing layer (and in turn frequency of RF transmissions at PHY layer) • N/A (don't care) in case of reactive routing protocol
Route_Flush&Recompute_Timer<t>	Reactive Routing Protocol Stale Route Flush and Re-compute Frequency	Networking (IP) Layer	• Used with reactive/on-demand routing to control sleep-wake duty cycles at routing layer (and in turn, frequency of RF transmissions at the PHY layer) • N/A (don't care) in case of proactive link-state routing protocol
Channel_Sensing&Backoff<t>	Contention Type MAC Channel Sensing Frequency for allocation to application s	Media Access Control Layer	• Used with a CSMA-type MAC to control sleep-wake duty cycles (and in turn RF transmissions) • N/A (don't care) with TDMA type MAC
Num_IDLE_Slots<k>	Non-Contention Type MAC Channel allocation to applications	Media Access Control Layer	• Used with TDMA-type MAC to control sleep-wake duty cycles (and in turn RF transmissions) • N/A (don't care) with CSMA-type MAC
Traffic_Type_Multicast<%age>	Percentage of total traffic expected to be of multicast type	Application Layer	• Apply knowledge about intensity of multicast traffic, when available, to drive sleep-wake duty cycles at MAC layer and routing layer (and RF transmissions)
Traffic _Type_ShortMsg<%age>	Percentage of total traffic expected to be of short message type (e.g., C2, PLI, etc.)	Application Layer	• Apply knowledge about traffic type (short messaging, in particular), when available, to drive sleep-wake duty cycles at MAC layer (and in turn RF transmissions)

Control Knob	Description	Network (OSI) Layer	Notes
Mobility<High, Medium, Low>	Frequency of changes to node locations; supply thresholds for user to categorize as "high/medium /low"	Application Layer (information based on Mission Phase, CONOPs)	• Apply knowledge of anticipated mobility, when available, to drive routing protocol timers and topology control algorithms, and in turn RF transmissions/network interface power settings
Topology_Density<Low,Medium,High>	Number of 1-hop (directly connected) RF neighbors	Application Layer (info based on CONOPs, Mission Phase)	• Apply knowledge of topology density, when available, to set RF Tx power level on interface to connect with varying numbers of 1-hop RF neighbors

Table 1 - Sample networking-related control knobs presented as APIs

3.7.1 EM Network control features: Control Descriptions, Arguments and Return Values

3.7.1.1 Neighbor Discovery Timer

Description

Control knob to set frequency of 1-hop neighbor discovery broadcast messages while using proactive link-state routing in order to control sleep-wake duty cycles at routing layer (and in turn frequency of RF transmissions at PHY layer).

Argument

Parameter Name	Unit	Description
Neighbor_Discovery_Timer	Time in secs	Timer used by proactive routing protocol to launch 1-hop neighbor discovery messages (e.g., HELLO messages)

Return Value:

Boolean: <Success, Failure> indicating whether the "set" operation was successful or not.

3.7.1.2 Topology Message Dissemination Timer

Description

Control knob to set frequency of flooding topology information messages while using proactive link-state routing in order to control sleep-wake duty cycles at routing layer (and in turn frequency of RF transmissions at PHY layer).

Argument

Parameter Name	Unit	Description
Topology_Message_Timer	Time in secs	Timer used by proactive routing protocol to flood topology information messages (e.g., TC (Topology Control) messages)

Return Value:

Boolean: <Success, Failure> indicating whether the "set" operation was successful or not.

3.7.1.3 Routing Path Flush & Re-computation Timer

Description

Control knob to set frequency of refreshing source-destination routes while using reactive (on-demand, distance vector) routing in order to control sleep-wake duty cycles at routing layer (and in turn, frequency of RF transmissions at the PHY layer).

Argument

Parameter Name	**Unit**	**Description**
Route_Flush_Recompute_Timer	Time in secs	Timer used by reactive routing protocols to flush nodes' route-caches and re-compute new paths

Return Value:

Boolean: <Success, Failure> indicating whether the "set" operation was successful or not*9.3.4 Channel Busy/Idle Sensing Timer*

Description

Control knob to set frequency of sensing the state (busy, idle) of the channel by when employing a contention based (e.g., CSMA (carrier sense multiple access)) MAC protocol in order to control sleep-wake duty cycles (and in turn RF transmissions).

Argument

Parameter Name	**Unit**	**Description**
Channel_Busy_Idle_Timer	Time in secs	Timer used by CSMA MAC to place packets on the RF channel

Return Value:

Boolean: <Success, Failure> indicating whether the "set" operation was successful or not.

3.7.1.4 Number of IDLE slots

Description

Control knob to set the number of IDLE slots when employing a non-contention time division multiple access (TDMA) media access control (MAC) protocol in order to control sleep-wake duty cycles (and in turn RF transmissions).

Argument

Parameter Name	**Unit**	**Description**
Num_IDLE_Slots	Integer	Number of slots during which the MAC will remain idle

Return Value:

Boolean: <Success, Failure> indicating whether the "set" operation was successful or not.

3.7.1.5 Multicast Traffic Type by Percentage

Description

Control knob to input information about the percentage of user traffic expected to be of multicast type. In turn, this knob will be used to drive path (route) selection algorithms and to perform selective control of sleep-wake duty cycles.

Argument

Parameter Name	**Unit**	**Description**
Traffic_Type_ShortMsg	Percentage value between [0, 100]	Percentage of application traffic expected to be of multicast type.

Return Value:

Boolean: <Success, Failure> indicating whether the input ("write") operation was successful or not.*9.3.7 Short Message Traffic Type by Percentage*

Description

Control knob to input information about the percentage of user traffic expected to be of "short messaging" type (e.g., Position Location Information (PLI), Command & Control (C2) message, etc.). In turn, this knob will be used to drive path (route) selection algorithms and to perform selective control of sleep-wake duty cycles.

Argument

Parameter Name	**Unit**	**Description**
Traffic_Type_Multicast	Percentage value between [0, 100]	Percentage of application traffic expected to be of type "short messaging".

Return Value:

Boolean: <Success, Failure> indicating whether the input ("write") operation was successful or not.

3.7.1.6 Degree of Mobility

Description

Control knob to input information about the degree of mobility anticipated in the given network (subnetwork). In turn, this knob will be used to drive routing protocol timers and topology control algorithms to control RF transmissions (energy expenditure) at the nodes.

Argument

Parameter Name	**Unit**	**Description**
Mobility_Degree	<High, Medium, Low>	Intensity of mobility based on a priori defined thresholds, categorized as "High, Medium, Low".

Return Value:

Boolean: <Success, Failure> indicating whether the input ("write") operation was successful or not.*3.9 Desired Topology Density*

Description

Control knob to input information about the desired density of network connectivity (i.e., 1-hop RF neighbors) for the given network (subnetwork). In turn, this knob will be used to drive topology control algorithms and set the RF Tx power levels on interfaces to realize the desired degree of connectivity in order to control RF power expenditure at the nodes.

Argument

Parameter Name	**Unit**	**Description**
Topology_Density	<High, Medium, Low>	Density of topology in terms of 1-hop RF neighbors based on a priori defined thresholds, categorized as "High, Medium, Low".

Return Value:

Boolean: <Success, Failure> indicating whether the input ("write") operation was successful or not.

3.8 EM Interactions with the Platform and External Systems

3.8.1 EM Management Exchanges Interfaces with External Devices

External interfaces to an SDR radio can significantly impact energy consumption. Some devices may draw prime power from the radio, for example, to maintain its own battery state. Digital interactions through the interfaces will draw power and in addition will keep the processors busy performing interactions with external peripherals. Certain interfaces may be able to have associated sleep intervals. For example, it may be possible to define a period for sampling the state of a sensor and sleeping in between sampling intervals. Similarly, a display update may be periodic with sleep intervals between updates. To complicate this concept, the periodicity for one peripheral may differ from the periodicity of a different peripheral.

Energy consumption management of peripherals can be similarly accommodated to the fashion in which processors are managed:

- SET Peripheral (PeripheralID, active/inactive, duration)

3.8.2 Advanced Energy Management System

The ensemble of controls within the SDR radio are able to be managed as a collection of presets for waveform types, current link conditions, applications supported and the platform energy resource.

However at the system level, the platform, and external applications may have performance requirements that the radio and the radio network are not aware of. Similarly, the radio may have requirements that the platform or other applications may not be aware of. For example, an application may wish to specify a rigid time or a time window at which a message should either be sent or received from a given radio, as an associated sensor will report at that time. A platform may have a need to keep a prescribed latency of connectivity with a set of other platforms. Or a radio may need to indicate it is low on battery and wishes to not be a member of a relay function. Thus a variety of energy management requests must be adjudicated amongst radio, network, applications and platforms. Each of these have priority levels, including variability as a function of remaining battery capacity and mission criticality.

Consequently to respond to energy management in a fully comprehensive fashion, it may be necessary to implement an advanced energy management system capable of adjudicating priorities, assessing schedules, deadlines, time windows with hard and soft time constraints, and the corresponding timelines and associated levels of activity.

Appendix B
Definitions

SDR – Software Defined Radio – a Radio in which the majority of the functionality is accomplished by software or a combination of software and firmware.

CR – Cognitive Radio – A cognitive radio is a radio in which communication systems are aware of their internal state and environment, such as location and utilization of RF frequency spectrum at that location. It can make decisions about radio operating behavior by mapping that information against predefined objectives. A cognitive radio is further defined by many to utilize Software Defined Radio, Adaptive Radio, and other technologies to automatically adjust its behavior or operations to achieve desired objectives. The utilization of these elements is critical in allowing end-users to make optimal use of available frequency spectrum and wireless networks with a common set of radio hardware.

PHY – The PHY layer means the physical layer of the ISO stack. This usually consists of a specification of modulation, and is often referred to as the modem, but must also cover all functionality accomplished by physical RF circuitry, filters, A/D and D/A converters, power amplifiers and other specialized circuitry of the RF front end including the antenna.

Link - The link layer consists of functionality associated with exchanging binary information from the modem and error correcting it to interface with higher levels of network functions

MAC – The Media Access Control layer concerns how multiple radios share the RF spectrum, usually consisting of some combination of time division, frequency division, code division and spatial division.

API - Application Programming Interface – A set of well-defined interfaces for various software functions that control a software function across a specified boundary. If these are formed clearly, then software developers do not need to deeply understand how the function is accomplished in the software in order to make use of the function.

MIMO – Multiple Input and Multiple Output is a way of transmitting and receiving with multiple antennas such that multipath artifacts are put to advantage to increase data rate, reduce fade impacts, and potentially even to focus transmitted energy towards an intended receiver.

FEC – Forward Error Correction is a method of adding some amount of redundancy so that a small percentage of bit errors in the communication channel can be automatically recognized and corrected at the receiver.

Network Coding – This type of error correction is intended to be able to make up for dropped packets that did not successfully get relayed from a source through a network to the intended destination. As such it is another layer of forward error correction.

Knobs – A parameter used to cause adaptivity in the software. In the context of this document, it is one of many parameters the software can adapt and pass to other software APIs in order to manage energy efficiency of the SDR radios and at the system level.

Meters – A parameter that can be sensed to control software adaption mechanisms. In the context of this document, many RF circuitry, modulation, and application properties can be sensed and presented to the energy saving logic, which will then adapt the energy saving knobs of the SDR software through the relevant APIs.