INTERNATIONAL STANDARD

ISO/IEC 23004-4

First edition
2007-09-15

Information technology — Multimedia Middleware —

Part 4: Resource and quality management

Technologies de l'information — Intergiciel multimédia —

Partie 4: Management des ressources et de la qualité

Reference number
ISO/IEC 23004-4:2007(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 23004-4 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 23004 consists of the following parts, under the general title *Information technology — Multimedia Middleware*:

— *Part 1: Architecture*

— *Part 2: Multimedia application programming interface*

— *Part 3: Component model*

— *Part 4: Resource and quality management*

— *Part 5: Component download*

— *Part 6: Fault management*

— *Part 7: System integrity management*

Introduction

MPEG, ISO/IEC JTC 1/SC 29/WG 11, has produced many important standards (MPEG-1, MPEG-2, MPEG-4, MPEG-7, and MPEG-21). MPEG feels that it is important to standardize an application programming interface (API) for Multimedia Middleware (M3W) that complies with the requirements found in the annex to the Multimedia Middleware (M3W) Requirements Document Version 2.0 (ISO/IEC JTC 1/SC 29/WG 11 N 6981).

The objectives of Multimedia middleware (M3W) are to allow applications to execute multimedia functions with a minimum knowledge of the middleware and to allow applications to trigger updates to the middleware to extend the middleware API. The first goal can be achieved by standardizing the API that the middleware offers. The second goal is much more challenging, as it requires mechanisms to manage the middleware API and to ensure that this functions according to application needs. The second goal can support the first, by reducing the needed standard APIs to those that provide middleware management. Consequently, applications can use these standard management APIs to generate the multimedia system they require.

The aim of M3W is to define a component-based middleware layer in high-volume embedded appliances that enables robust and reliable operation. These types of product are heavily resource constrained, with a high pressure on silicon cost and power consumption. In order to be able to compete with dedicated hardware solutions, the available resources will have to be used very cost-effectively, while enabling robustness and meeting stringent timing requirements imposed by high-quality elements such as digital audio and video processing.

High-volume embedded appliances are considered business-critical devices. Their failure can have important economic implications for the producing company. Hence, they have stringent and demanding quality requirements. Unfortunately, in the software industry misbehaving products are commonplace. However, this is not the case with consumer electronics, home appliances and mobile devices. Users are accustomed to robust and well-behaved devices.

In consumer electronics, there are demanding quality requirements and a need to use the available hardware resources cost-effectively. The term "Application" is used to refer to the software entity that (indirectly) provides certain functionality to an end-user; the Application may include Service Instances that are bound to it. In order to ensure that an Application provides the correct service, it needs to be assigned the hardware resources it requires. This can be achieved by assigning it the "worst case" resources required. However, with many applications, the worst-case resource usage is much less than the mean case. If this worst-case approach is followed, hardware resources will be wasted. On the other hand, problems may arise if the running applications require more resources than can be made available. If the platform follows a typical fair policy for the assignment, the behaviour of the system will be unpredictable, especially for applications that must provide results according to some specified time constraints.

The goal of Resource Management is to assign budgets or resource reserves to Applications. These budgets are guaranteed, so that they are available in any situation. This approach helps to enhance system robustness, because an Application is unable to affect the resource reserves of others.

In order to benefit from Resource Management, Applications participating in Resource Management should be resource-aware (RA). This means that they are aware of the resources they need during their execution and should adapt their behaviour to the resources which are made available. This ensures that applications can function correctly without exhausting all of the system resources that have been allocated to them.

Quality-Aware (QA) Applications are RA applications that are aware of the quality of service that they deliver. Typically, they are capable of providing different quality levels. They are characterized by the quality they provide and the resources needed for this purpose. Usually the higher the quality level, the higher the resource needs. This type of Application is able to dynamically change the provided quality level, depending on the assigned budgets.

Often, QA and RA applications are real-time applications. The assigned budgets allow them to provide a suitable output within some time interval. A hard real-time application can be viewed as QA with only two quality levels: maximum quality or nothing. The Resource Management framework can also deal with Non Resource-Aware Applications (NRA) in the sense that they are assigned a fixed budget all together.

The relation between the Resource Management framework and QA applications is based on a **contract model**. The system provides resources and the Applications commit to generate the required results (outputs) with a specific and stable quality. Budget assignment must be obtainable, which means that it should not be possible to assign more resources than those actually available. Contracts are negotiated with the Applications with the goal of maximizing overall system quality, as perceived by the user. In this process, the "importance" of the Applications is a primary parameter for this operation.

Power is considered to be one of the most important resources to be managed in the next generation of consumer electronics. Its importance is clear for mobile devices, where the goal is to maximize battery life. In stationary devices, it is also relevant for environmental conditions, fan-less operation and to increase the lifetime of the silicon devices. In addition, power management is related to heat control, so that the temperature of different parts of the device can be maintained within a certain threshold.

The basic goals in M3W with respect to power management are as follows.

— Reduce power consumption.

— Increase battery life, for mobile devices.

— Ensure a system-wide limit for heating: If it is detected that the temperature of the device it is too high, it may be required to reduce it.

— Take advantage of power-aware hardware and M3W components.

— Ensure that the system and relevant Applications are always executed, in spite of the power saving policy selected.

Power management is very much related to resource management. A number of techniques for reducing power consumption are based on moving hardware components to less power-consuming modes of operation, which implies reducing/modifying the available resources (CPU capacity, memory size, bandwidth, etc.). This is the case, for example, with CPU voltage and frequency scaling, which can reduce power consumption and consequentially, CPU computational power. For this reason, it is desirable to integrate power management with quality and resource management.

The Resource Management Framework is in charge of achieving the functionality that has been sketched in this introductory clause. Its functions and architecture are defined in this part of ISO/IEC 23004.

Information technology — Multimedia Middleware —

Part 4:
Resource and quality management

1 Scope

This part of ISO/IEC 23004 defines the Resource and Quality Management framework of the MPEG Multimedia Middleware (M3W) technology.

2 Organization of this document

This part of ISO/IEC 23004 has the following high level structure:

— Clause 1 defines the scope of this part of ISO/IEC 23004.

— Clause 3 gives an overview of documents that are indispensable for the application of this part of ISO/IEC 23004.

— Clause 4 gives the terms and definitions used in this part of ISO/IEC 23004.

— Clause 5 provides an overview of and introduction to the Resource and Power Management Framework.

— Clause 6 contains a detailed specification of the interfaces of the Resource and Power Management Framework that are part of the M3W API.

— Clause 7 gives an overview of the realization of the Resource and Power Management Framework.

This part of ISO/IEC 23004 has the following annexes:

Annex A, Dynamic view of the Resource and Power Management Framework, describes the interactions between the different entities of the Resource and Power Management Framework.

Annex B, CPU chief details: The Resource and Power Management Framework distinguishes four different roles that are together responsible for Resource and Power Management:

1) resource chief;

2) resource manager;

3) quality chief;

4) quality manager.

The CPU chief is discussed in this annex. The CPU chief is a specialization of a resource chief. This is the chief responsible for monitoring and enforcing the CPU budgets.

Annex C, Approach to power management: Power is a resource as well. This annex discusses managing this specific resource.

Annex D, Composition of quality information example: Often we need to manage the Resource Consumption and delivered Quality of a composition of entities instead of a single atomic entity. This annex discusses how to compose the Quality and Resource information in such cases.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 23004-1, *Information technology — Multimedia Middleware — Part 1: Architecture*

4 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 23004-1 apply.

5 Overview of interface suites

5.1 Introduction

This is an informative clause that gives an overview of the API specification of the resource management framework.

5.2 Resource management

The optional M3W Resource Management Framework provides timely, guaranteed and protected access to system resources. These functions allow Applications to specify their resource demands, leaving the Resource Management Framework to satisfy those demands using multimedia platform specific resource management schemes. In this situation, two models (or views) can be defined:

— Resource specification models: Applications and Services must specify the budget or resource share they need. For this purpose, the runtime provides a certain model and an associated interface. If it is feasible, a given resource share can be committed to these entities.

— Resource management models: The runtime must provide a means for accounting, enforcing and monitoring resource usage. In this way, it is possible for the runtime to guarantee the committed budgets and to provide useful information to applications so that they can adapt their behaviour to their current resource shares / budgets.

These two models are somewhat independent. For a given specification model, a number of management models can be implemented to satisfy it, and vice versa.

The resource specification model is based on assigning resource shares or budgets to the Applications. As an example, the specification model for handling memory and CPU is now described:

— Memory: The corresponding entities should make a claim for a certain amount of memory. The manager should respond whether the grant can be approved or not. Ideally, these entities should request all the memory they will need for their entire execution lifetime. If they dynamically claim memory, it could be that in the middle of its execution there is no additional free memory, and the program is then unable to continue with its execution. On the other hand, it may be difficult for an application to know in advance the exact memory needed, and it may ask for much more than it really needs. In any case, memory feasibility tests are very simple and can be made frequently, with little overhead.

— CPU: The specifying model is based on requesting a certain CPU share (percentage) for a given period of time. This budget is refilled periodically. CPU budgets can either be allocated to a thread or to a group of threads (also named a cluster). For example, a thread may be allowed to use the CPU for 1ms every 10ms (budget is 1 ms and the refill period is 10ms). In the context of this specification, a thread is the unit of concurrency.

Resource shares are assigned by the Resource Management framework and shall be enforced during system operation. For this purpose, it is necessary to account for resource usage and take some action if a budget user tries to use more resources than budgeted. If an Application requests more resources, it should be checked whether this is possible, given the current system situation. If there are available resources, the request may be satisfied. Otherwise, it should be denied or a new system assignment should be made.

QA Applications and Services have static information on their quality levels and their associated resource needs. In the case of the CPU, the required budget is an estimation that is e.g. calculated by measurements taken when running the program with a meaningful set of input streams and data. In some domains, e.g. multimedia, the real resource needs depend on the type of data, so it is difficult to provide accurate estimations. Applications may dynamically adapt and update their required budget estimations for use in future negotiations. The same considerations can be made with respect to memory, although typically in this case the required memory is subject to less variation, and estimations can be made more accurately.

An Application or Service may provide the required quality level with different sets of resources. For example, it can use a lot of CPU and little memory or only a little CPU but with plenty of memory to deliver the same quality level. In the context of the Resource Management Framework the set of resources that an Application requires for a certain quality level is called a configuration. A quality level can have several configurations associated. The Resource Management framework is in charge of deciding which configuration will be selected, depending on the needs of the applications and the available resources in the system.

In a M3W system, Resource Aware and Non-Resource Aware applications can co-exist. The approach for dealing with this situation is to divide the system resources into two partitions:

— RA: This partition is composed by all Resource Aware Applications and Services. The QM has assigned them a certain budget, which shall be guaranteed, for the execution (until a re-negotiation).

— NRA: This partition includes the rest of resources and it is used for the execution of the Non-Resource Aware Applications and Services.

One possible way of implementing this division is by assigning priorities to RA threads (those that belong to RA Applications or RA Services) that are higher than those assigned to NRA elements. When the budget of a RA element is exhausted, its priority could be reduced. When the budget is replenished, the priority is raised. A real-time operating system, with appropriate scheduling policies, is required for the execution of the RA threads. The scheduling of NRA threads can be done using a classical quantum scheduler.

Power management is an important aspect of resource management, as the available system resources depends on the power settings of the corresponding hardware components. If the power settings are changed, (for example to reduce consumption, then the negotiated resource shares cannot be kept, and the contract may be broken. For this reason, the approach is to consider power as another resource, although with special characteristics, and it is managed together with the rest of the system resources. More details on power management can be found in Annex C.

5.3 Quality management

The purpose of quality management is to drive the interaction with the QA Applications, so that the quality of the overall system is maximised. A system may be composed of a set of QA Applications. Where each application can provide a set of quality levels. A quality level is characterized by a description of the provided quality and by the required resources for this purpose. The problem is to decide the quality level for each Application, and as a consequence the resource assignment, so that system quality is maximized and the resource assignment is feasible.

The criteria for making this selection vary from system to system. The information that has to be considered includes:

— Importance of Applications: There are applications that are fundamental in a particular device and should always provide some minimal quality, such as the handling of incoming calls in a mobile phone. Information from the execution context is also useful for identifying the important Applications. In a system with multiple windows, it is usually the case that the Application running in the window with the user focus is the most important.

— User settings: The goal is to maximize user satisfaction. Hence, it is important to consider user settings.

— Domain knowledge: This information is used in the decision process, e.g. to decide which Application is most important and has to receive more resources.

The quality management functions do not finish when a contract is negotiated. It is necessary to monitor the system to handle situations where the system is overloaded or has free resources. The resources required by Applications are usually estimations. In multimedia applications, the required resources strongly depend on the input data. In particular, the computation time may depend on whether the scene is static or with a lot of action and on the compression algorithm used. Hence, it can be that an Application requires more resources or has more than needed. In this situation, there should be mechanisms for its correction. As a consequence, the resources needed for the quality level may change and the contracts must be (at least partially) renegotiated.

The operations that quality management provides can be described as follows:

— Reservation is the most basic mechanism for providing QoS guarantees. It consists of assigning shares of resources (budgets) to Applications and guaranteeing them based on runtime monitoring and enforcement of their usage. Most of the functions related to reservation are performed as part of resource management.

— Negotiation can be defined as the operation for creating a new contract (or modifying the current one) between the system software and the different Applications. This long-term contract determines a set of minimum requirements for both sides. The system guarantees a minimum budget, and the Application agrees on executing the functionality required by the user providing, at least, a minimum utility. The process is based on having a dialogue between both domains to check whether new settings are feasible in the system. Differentiating negotiation (feasibility test) from the actual settings into two different phases is useful when we want to check the feasibility of several mutually exclusive configurations, and then select one of them. Negotiation can be executed at Application (or Service) start-up, or dynamically when it is detected that the current contract is no longer valid. This renegotiation may be in response to either external or internal events. External events are those signalled externally to the system, while internal events are those detected by runtime monitoring.

— Setting is the operation of making the system configuration resulting from a negotiation operational. As part of this procedure it is necessary to communicate to the Applications their executing quality level and to set the resource budgets, by using the facilities provided by the resource manager. This configuration change has to be done following a certain procedure that depends on how abrupt the changes can be made. Some Applications require a smooth change, so it is necessary to follow a protocol that ensures this requirement. For example, Applications that reduce their quality level may have to change before others raise theirs, in order to ensure that budgets are guaranteed during the setting operation.

— Monitoring is the process in charge of obtaining information about the execution of Applications and their threads, in order to know how they behave. This information includes (i) resource usage made by Applications and threads, (ii) progress monitoring (usually in the form of milestones reached or percentage of processing finished), (iii) adaptation measurements (performance achieved by threads, and data dropping made by Applications), and (iv) application domain metrics (utility provided as a function of provided QoS parameters and QoS violations).

— Adaptation is the operation of responding to transient overloads, detected at runtime, by performing some short-term changes. The goal is to adapt the processing algorithm in order to have a near-constant

computational complexity. The main characteristic of adaptation is that the current contract between the system domain and the Application domains must continue to be fulfilled.

— Optimization is the process of achieving the best configuration for all the entities in the system, and hence achieving the best use of available resources at each point in time. Negotiation determines a contract between the system domain and the Application domains that defines the minimum requirements for them, and optimization can make small changes to current configurations but without changing the contract. The dynamic nature of multimedia systems makes that a certain configuration that is optimal for the current state it is no longer optimal when the situation changes. Optimization tries to anticipate the need for resource reallocation, enhancing the medium and long-term behaviour of services. It is closely related to adaptation and negotiation, but the main difference is the mechanism that triggers their execution. Negotiation is executed when the new situation (for example, when the user launches a new application) requires the modification of the current contract between the system domain and the application domains. Optimization is executed periodically by the Resource Management framework's control system to try to make small changes to improve the behaviour of the system (without changing the contract). Finally, adaptation is triggered by transient overload situations detected at runtime.

The QA Applications have to follow a certain procedure for interacting with the Resource Management framework. First of all, they have to be implemented in such a way that the contract is met. They should not try to use more resources than budgeted and have to provide the committed quality. QA Applications have to provide some operations related to the interaction with the QoS management facilities:

— Provide quality information. This includes the quality levels and the resources required for each quality level. This information is basic for the negotiation process. Applications have to update this information whenever it changes. This may be caused by changes in the estimations of resources needed, quality levels that can or cannot be provided, for example due to the nature of the input data, etc.

— Follow the assigned settings. The Application receives a command to set the quality level to be executed after a negotiation. It has to change its execution mode accordingly. Applications also have to notify the completion of the change to the Resource Management framework.

— Notify information and events. It is useful to provide information on the behaviour of the Application to the QoS manager, so that it can use this information to optimize system operation. Events should also be notified to the QoS manager. Some relevant events are fault detection, request for changing the contract, etc.

Quality management includes power management. Information on the current power status and user power settings must be taken into account in the operations specified above, and especially with negotiation. The available resources depend on the selection of the power settings. Hence, it is needed to find a suitable power setting that meets user power requirements and that allows for providing the minimum global quality required by the user.

In addition, it is interesting to be able to take advantage of Services with algorithms that are specifically designed for reducing power consumption. A Quality-Aware Service may have different resource configurations for providing a quality level. Some of these configurations may include power-saving algorithms. In order to include this type of support in the Resource and Power Management Framework, it is required to be able to:

— Include information on the power consumption level of the configurations associated to a quality level.

— Manage the power consumption level information in the quality manager: the quality manager has to be aware of this qualification and take advantage of it. For example, it could select configuration of quality levels that requires low energy, when the system is in a low-power mode.

6 Resource management interface suite

6.1 Overview

In this clause the main types and functions are commented, to facilitate understanding of the API. This description deals initially with the data types and then the logical components are described. Following that, a description of the roles is given. Finally, the Interfaces and their operations are described.

6.2 Types and constants

6.2.1 RcQoSEventId

Signature

```
enum RcQoSEventId
{
    QOS_EVENT_OVERRUN,
    QOS_EVENT_BUDGET_TOO_LOOSE,
    QOS_EVENT_BUDGET_TOO_LOW,
    QOS_EVENT_UNKNOWN,
    MAX_QOS_EVENTS
};
```

Qualifiers

None

Description

Resource management related events.

6.2.2 RcQoSEventTime

Signature

```
typedef Int32 RcQoSEventTime;
```

Qualifiers

None

Description

Time that a event occurred.

6.2.3 RcQoSEventData

Signature

```
typedef pVoid RcQoSEventData;
```

Qualifiers

None

Description

Event specific data.

6.2.4 RcQoSEventInfo

Signature

```
struct RcQoSEventInfo
{
    RcQoSEventTime eventTime;
    RcQoSEventData eventData;
};
```

Qualifiers

None

Description

Information associated with an event.

6.2.5 RcQoSImportance

Signature

```
typedef Int8 RcQoSImportance;
```

Qualifiers

None.

Description

This represents how important the execution of an application is for the user of the device. Important applications are favoured in the resource assignment and the QoS-Manager tries to execute them with a high quality level.

6.2.6 RcQoSApplicationId

Signature

```
typedef Int8 RcQoSApplicationId;
```

Qualifiers

None.

Description

This is the Resource Management Framework application identifier. Its value is provided by the Resource Management framework.

6.2.7 RcQoSResourceId

Signature

```
enum RcQoSResourceId
{
    RID_CPU
    RID_NETWORK_BASE,
    RID_NETWORK_TOKENBUCKET,
    RID_POWER,
    QOS_MAX_NUM_RESOURCES
};
```

Qualifiers

None.

Description

This ID describes the type of resource.

Values

Table 1

Name	Description
• RID_CPU	A 'CPU' resource
• RID_NETWORK_BASE	A 'network base' resource
• RID_NETWORK_TOKENBUCKET	A 'network token bucket' resource
• RID_POWER	A 'power' resource
• QOS_MAX_NUM_RESOURCES	The maximum number of resources

6.2.8 RcQoSDescription

Signature

```
typedef String RcQoSDescription;
```

Qualifiers

None.

Description

Type used to describe entities in the Resource Management and Power Management framework. For example this type is used to describe Quality Levels, as well as components.

6.2.9 RcQoSBudgetId

Signature

```
Int8 RcQoSBudgetId;
```

Qualifiers

None.

Description

This is the identifier of the budget.

6.2.10 RcQoSBudgetUser

Signature

```
typedef pVoid RcQoSBudgetUser;
```

Qualifiers

None.

Description

This is the user of a budget.

6.2.11 RcQoSBudgetInfo

Signature

```
typedef pVoid RcQoSBudgetInfo;
```

Qualifiers

None.

Description

Describes a budget. For example this is used to express required budgets by a logical user for a certain Quality Level.

6.2.12 RcQoSLogicalUserName

Signature

```
typedef String RcQoSLogicalUserName;
```

Qualifiers

None.

Description

String that contains the name of a logical user. Budgets are assigned to logical users. A logical user can consist of a number of budget users.

6.2.13 RcQoSQualityLevelId

Signature

```
typedef Int8 RcQoSQualityLevelId;
```

Qualifiers

None.

Description

This is the identifier of the quality level. It is the identifier that is provided in the description of the quality level of an application. Higher Level Id's mean better output quality

6.2.14 RcQoSQualityInformation

Signature

```
struct RcQoSQualityInformation
{
    RcQoSApplicationId appId;
    UUID               compUuid;
    RcQoSDescription   componentDescription;
    RcQoSQualityLevel  availableQualityLevels[MAX_AVAILABLE_QUALITY_LEVELS];
};
```

Qualifiers

None.

Description

This describes the quality information of an application.

6.2.15 RcQoSQualityLevel

Signature

```
struct RcQoSQualityLevel
{
    RcQoSQualityLevelId qualityLevelId;
    RcQoSDescription    qualityLevelDescription;
    RcQoSConfiguration  qualityLevelConfigurations[MAX_CONFIGURATIONS_NUMBER];
};
```

Qualifiers

None.

Description

This describes a quality level that an application can provide.

6.2.16 RcQoSConfigurationId

Signature

```
typedef Int8 RcQoSConfigurationId;
```

Qualifiers

None.

Description

This is the identifier of the configuration.

6.2.17 RcQoSConfiguration

Signature

```
struct RcQoSConfiguration
{
    RcQoSConfigurationId                configurationId;
    RcQoSRequiredResourceSpecification requiredResources[QOS_MAX_NUM_RESOURCES];
};
```

Qualifiers

None.

Description

This describes a resource configuration required for providing a quality level.

6.2.18 RcQoSRequiredResourceSpecification

Signature

```
struct RcQoSRequiredResourceSpecification
{
    RcQoSResourceId     resourceId;
    RcQoSRequiredBudget requiredBudgets[MAX_REQUIRED_RESOURCE_BUDGETS];
};
```

Qualifiers

None.

Description

Data structure used to specify the requirements on a specific resource (identified by resourceId). These requirements can be budgets for a number of different logical users.

6.2.19 RcQoSRequiredBudget

Signature

```
struct RcQoSRequiredBudget
{
    RcQoSLogicalUserName logicalUserName;
    RcQoSBudgetInfo      budgetInfo;
};
```

Qualifiers

None.

Description

This structure is used to specify the required budget for a specific logical user. This is usually part of the requirements on a specific resource for a specific Quality Level.

6.2.20 RcQoSAssignedBudget

Signature

```
struct RcQoSAssignedBudget
{
    RcQoSLogicalUserName logicalUserName;
    RcQoSBudgetId        budgetId;
};
```

Qualifiers

None.

Description

Structure that contains the assignment of a budget to a logical user.

6.2.21 RcQoSAssignedBudget

Signature

```
typedef RcQoSAssignedBudget RcQoSAssignedBudgetsSet[MAX_ASSIGNED_BUDGETS];
```

Qualifiers

None.

Description

Structure that contains all the assigned budgets.

6.2.22 RcQoSBudgetStatus

Signature

```
struct RcQoSBudgetStatus
{
    Float availableBudgetPercentage;
    Int8 status;
};
```

Qualifiers

None.

Description

Structure that contains the status of a budget.

6.2.23 RcQoSBudgetMonitoringInfo

Signature

```
typedef pVoid RcQoSBudgetMonitoringInfo;
```

Qualifiers

None.

Description

Contains information about the usage of a budget, such as overruns and underruns.

6.2.24 RcQoSApplicationInfoList

Signature

```
typedef RcQoSApplicationInfo RcQoSApplicationInfoList[MAX_QOS_APPLICATIONS];
```

Qualifiers

None.

Description

Contains list of identifiers of applications. This structure is returned when the registered applications are requested.

6.2.25 RcQoSCPUBudgetUser

Signature

```
typedef Int32 RcQoSCPUBudgetUser;
```

Qualifiers

None

Description

The ID of the CPUBudgetUser.

6.2.26 RcQoSCPUBudgetInfo

Signature

```
struct RcQoSCPUBudgetInfo
{
    Int32 refillPeriod;
    Int32 priority;
    Int32 budget;
};
```

Qualifiers

None

Description

Information about a CPU budget contains budget (amount), priority and refill period.

6.2.27 RcQoSBudgetCPUMonitorizationInfo

Signature

```
struct RcQoSBudgetCPUMonitorizationInfo
{
    RcQoSResourceId resourceId;
    unsigned Int32     maxUsagePercentage;
    unsigned Int32     minUsagePercentage;
    unsigned Int32     meanUsagePercentage;
    unsigned Int32     numberRefillPeriods;
    unsigned Int32     numberOverruns;
};
```

Qualifiers

None

Description

Structure that contains the results of monitoring the budget usage.

6.2.28 RcQoSNetworkRateUnit

Signature

```
typedef Int8 RcQoSNetworkRateUnit;
```

Qualifiers

None

Description

BITS = 1; BYTES = 8;

KILOBITS = 1000; KILOBYTES = 8000;

MEGABITS = 1000000; MEGABYTES = 8000000;

6.2.29 RcQoSNetworkBudgetUser

Signature

```
struct RcQoSNetworkBudgetUser
{
    String senderAddress;
    String receiverAddress;
    Int8 senderPort;
    Int8 receiverPort;
    Int8 protocol;
};
```

Qualifiers

None

Description

The network budget user, socket addresses and protocol

6.2.30 RcQoSNetworkBasicBudget

Signature

```
struct RcQoSNetworkBasicBudget
{
    RcQoSNetworkRateUnit rateUnits;
    Double bandwidth;
};
```

Qualifiers

None

Description

The basic network budget type, bandwidth

6.2.31 RcQoSNetworkTokenBucketBudget

Signature

```
struct RcQoSNetworkTokenBucketBudget
{
    RcQoSNetworkRateUnit rateUnits;
    Double averageRate;
    Double peakRate;
    Int32 bufferSize;
};
```

Qualifiers

None

Description

The token bucket budget type, average and peak rates and buffer size, i.e. the maximum amout of data received at peak rate.

6.2.32 RcQoSPowerPolicyId

Signature

```
typedef Int8 RcQoSPowerPolicyId;
```

Qualifiers

None

Description

Identification of a power policy.

6.2.33 RcIQoSPowerPolicyList

Signature

```
typedef RcIQoSPowerPolicy RcIQoSPowerPolicyList[QOS_MAX_POWER_POLICIES];
```

Qualifiers

None

Description

Array of Power policies

6.2.34 RcQoSUserPowerSetting

Signature

```
enum RcQoSUserPowerSetting
{
    MAXIMIZE_PERFORMANCE,
    MINIMIZE_POWER_CONSUMPTION,
    BALANCED_PERFORMANCE_POWER_CONSUMPTION,
    USER_DEFINED_1,
    USER_DEFINED_2,
    USER_DEFINED_3,
    USER_DEFINED_4
};
```

Qualifiers

None.

Description

Enumeration of possible power settings.

6.2.35 RcQoSPowerPolicyDescription

Signature

```
typedef String RcQoSPowerPolicyDescription;
```

Qualifiers

None

Description

Description of power policies.

6.2.36 RcQoSCPUFrequencyType

Signature

```
typedef Float RcQoSCPUFrequencyType;
```

Qualifiers

None

Description

CPU frequency.

6.2.37 RcQoSCPUVoltageType

Signature

```
typedef Float RcQoSCPUVoltageType;
```

Qualifiers

None

Description

CPU voltage.

6.2.38 RcQoSWlanMode

Signature

```
enum RcQoSWlanMode
{
    ACTIVE,
    POWER_SAVE
};
```

Qualifiers

None

Description

Power mode of WLAN.

6.2.39 RcQoSPowerDeviceID

Signature

```
enum RcQoSPowerDeviceID
{
    CPU_FREQUENCY,
    CPU_VOLTAGE,
    WLAN_MODE,
    QOS_MAX_POWER_DEVICES
};
```

Qualifiers

None.

Description

Identifies the type of power device.

6.2.40 RcQoSPowerDeviceStatus

Signature

```
struct RcQoSPowerDeviceStatus
{
    RcQoSPowerDeviceID deviceID;
    native status;
};
```

Qualifiers

None

Description

Contains the power status of a device.

6.2.41 RcQoSPowerDeviceSettings

Signature

```
typedef RcQoSPowerDeviceStatus RcQoSPowerDeviceSettings[QOS_MAX_POWER_DEVICES];
```

Qualifiers

None

Description

Contains the power settings for a device, which is the power status of all power devices.

6.2.42 RcQoSPowerStatus

Signature

```
struct RcQoSPowerStatus
{
    Float batteryLevelPercentage;
    Bool online;
    RcQoSPowerDeviceSettings deviceSettings;
};
```

Qualifiers

None

Description

Contains power status.

6.3 Logical component

6.3.1 Interface-Role Model

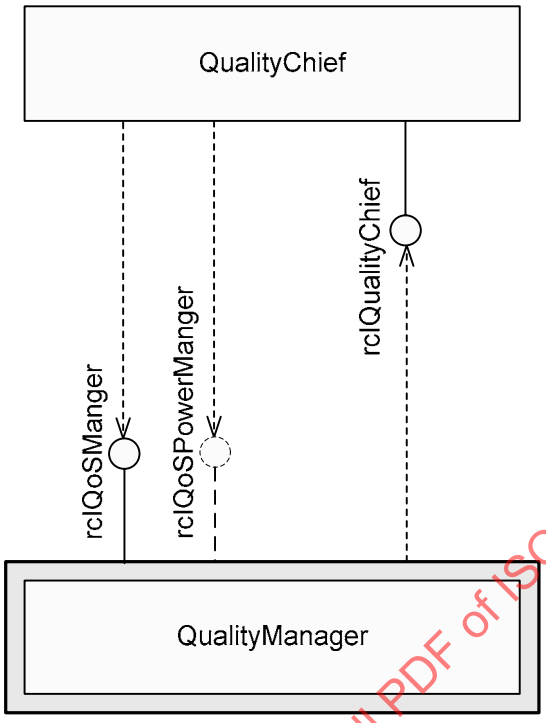

Figure 1 — Interface Role Model

`rclQoSPowerManager` is optional an interface needed for power management.

6.3.2 Diversity

The following table lists the required and provided interfaces:

Table 2

Role	Interface	Presence
QualityManager	rclQualityManager	Mandatory
QualityChief	rclQualityChief	Optional

6.4 Roles

6.4.1 QualityManager

Description

Its main goal is to manage system global quality. It assigns resources to Applications and Service Instances. See subclause 8.2.1 for the full details.

Instantiation

There is always one QualityManager instantiated in the system. The QualityManager manages the Quality for the complete system.

6.4.2 QualityChief

Description

The quality chief is part of the application. It is the counterpart of the QM. It is in charge of interacting with the QM and driving the execution of the Application, according to the agreed quality level. See subclause 8.2.2 for the full details.

Instantiation

The QualityChief role is part of the application. Each QA application has to instantiate its own QualityChief.

6.5 Interfaces

6.5.1 rclQosManager

Description

This interface is implemented by a Quality Manager and offers operations for the (un)registration of Quality Chiefs, updating and extration of Quality Information, requesting renegotation of the budgets, setting priorities of Quality Chiefs, etc.

Interface ID

```
{5f935698-1eb9-40e5-aeba-f3cd4366ed0a}
```

6.5.1.1 registerQualityChief

Signature

```
RcResult registerQualityChief(
    [in]  rcIQoSQualityChiefRef qChiefIf,
    [in]  UUID                  compUuid,
    [in]  RcQoSDescription      applicationDescription,
    [out] RcQoSApplicationId    applicationId);
```

Description

This function should be executed by an application to notify the Quality Manager that it is QA. As a result, the quality information of the application is stored in the Resource Management framework. In this call, the application provides its quality information.

6.5.1.2 unregisterQualityChief

Signature

```
RcResult unregisterQualityChief(
    [in] RcQoSApplicationId applicationId);
```

Description

This function is called when application that is registered within the QM, wants to remove this relationship. As a result, the quality of the application is no longer managed.

6.5.1.3 requestNegotiation

Signature

```
RcResult requestNegotiation(
    [in] RcQoSApplicationId      applicationId,
    [in] RcQoSQualityInformation qualityInfo,
    [in] RcQoSQualityLevelId     defaultQualityLevel);
```

Description

This operation is called whenever an application wants to request a new negotiation effort. This can happen when an application is going to start to execute or when due to some event, the application wants to renegotiate the contract.

6.5.1.4 notifyEvent

Signature

```
RcResult notifyEvent(
     [in] RcQoSApplicationId   applicationId,
     [in] RcQoSEventId         eventId,
     [in] RcQoSEventInfo       eventInfo);
```

Description

Generic operation to notify the Quality Manager of QoS related events.

6.5.1.5 setQualityInfo

Signature

```
RcResult setQualityInfo(
    [in] RcQoSApplicationId          applicationId,
    [in] RcQoSQualityInformation     applicationQualityInformation);
```

Description

This function allows the QoS-aware application to update the quality information that is stored in the QoS Manager.

6.5.1.6 endQualityLevelSetting

Signature

```
RcResult endQualityLevelSetting(
    [in] RcQoSApplicationId applicationId);
```

Description

This function is used to communicate to the QoS Manager that the applications has finished the setting of the new quality level.

6.5.1.7 setPowerUserSetting

Signature

```
RcResult setPowerUserSetting(
    [in] RcQoSUserPowerSetting userSetting);
```

Description

Sets the power settings according to the given user setting for example to maximize performance or minimize power consumption.

6.5.1.8 getPowerUserSetting

Signature

```
RcResult getPowerUserSetting(
    [out] RcQoSUserPowerSetting userSetting);
```

Description

Gets the current power settings.

6.5.1.9 getPowerStatus

Signature

```
RcResult getPowerStatus(
    [out] RcQoSPowerStatus status);
```

Description

Gets the power status.

6.5.1.10 setApplicationImportance

Signature

```
RcResult setApplicationImportance(
    [in] RcQoSApplicationId applicationId,
    [in] RcQoSImportance importance);
```

Description

This function allows setting the importance of a QoS-aware application.

6.5.1.11 getApplicationImportance

Signature

```
RcResult getApplicationImportance(
    [in] RcQoSApplicationId applicationId,
    [out] RcQoSImportance importance);
```

Description

This function allows getting the importance of a QoS-aware application.

6.5.1.12 getRegisteredApplications

Signature

```
RcResult getRegisteredApplications(
    [out] RcQoSApplicationInfoList appInfoList);
```

Description

This function provides the identifier of the registered applications.

6.5.1.13 setBudgetUser

Signature

```
RcResult setBudgetUser(
    [in] RcQoSResourceId resourceId,
    [in] RcQoSBudgetId   budgetId,
    [in] RcQoSBudgetUser budgetUser);
```

Description

This operation serves to identify the user of a budget. It specifies the entity that is going to use it. After the execution of this operation, the budget is made available for the registered entity, it is enabled.

6.5.1.14 removeBudgetUser

Signature

```
RcResult removeBudgetUser(
    [in] RcQoSResourceId resourceId,
    [in] RcQoSBudgetId   budgetId);
```

Description

This function removes the Budget-User binding. After the execution of this operation, the budget has no associated user and hence, it is not enabled.

6.5.1.15 getBudgetStatus

Signature

```
RcResult getBudgetStatus(
    [in]  RcQoSResourceId   resourceId,
    [in]  RcQoSBudgetId     budgetId,
    [out] RcQoSBudgetStatus budgetStatus);
```

Description

Gets the budget status for a certain budget of a resource.

6.5.1.16 getBudgetMonitoringInfo

Signature

```
RcResult getBudgetMonitoringInfo(
    [in]  RcQoSResourceId           resourceId,
    [in]  RcQoSBudgetId             budgetId,
    [out] RcQoSBudgetMonitoringInfo budgetMonitoringInfo);
```

Description

Gets the budget monitoring info for a certain budget of a resource.

6.5.1.17 resetBudgetMonitoringInfo

Signature

```
RcResult resetBudgetMonitoringInfo(
    [in] RcQoSResourceId resourceId,
    [in] RcQoSBudgetId budgetId);
```

Description

This function resets the monitoring information for the budget.

6.5.1.18 activateBudgetMonitoring

Signature

```
RcResult activateBudgetMonitoring(
    [in] RcQoSResourceId resourceId,
    [in] RcQoSBudgetId budgetId);
```

Description

This function causes the QualityManager to start getting monitoring information on the associated budget.

6.5.1.19 deactivateBudgetMonitoring

Signature

```
RcResult deactivateBudgetMonitoring(
    [in] RcQoSResourceId resourceId,
    [in] RcQoSBudgetId budgetId);
```

Description

This function causes the QoS Manager to stop getting monitoring information on the usage of a budget.

6.5.2 rclQoSPowerPolicy

Description

Interface that a power policy has to provide

Interface ID

```
{82aa78bf-1ff8-4eef-b276-f4c8ab09b2ed}
```

6.5.2.1 getPowerSettings

Signature

```
RcResult getPowerSettings(
    [in] RcQoSPowerDeviceSettings  minimumSettings,
    [in] RcQoSUserPowerSetting     userSetting,
    [out] RcQoSPowerDeviceSettings deviceSettings);
```

Qualifiers

None

Description

Returns the Power settings for the device for this policy

6.5.2.2 getPolicyDescription

Signature

```
RcResult getPolicyDescription (
    [out] RcQoSPowerPolicyDescription policyDescription);
```

Qualifiers

None

Description

Returns the description of this policy.

6.5.3 rclQosPowerManger

Description

Interface provides the operations to control the power management

Interface ID

```
{b60e9b74-cafa-4f2a-992e-9c970420deb9}
```

6.5.3.1 addPMGlobalPolicy

Signature

```
RcResult addPMGlobalPolicy(
    [in] RcIQoSPowerPolicy   powerPolicy,
    [out] RcQoSPowerPolicyId policyId);
```

Qualifiers

None

Description

Add a Power policy to the Power Manager.

6.5.3.2 removePMGlobalPolicy

Signature

```
RcResult removePMGlobalPolicy(
    [in] RcQoSPowerPolicyId policyId);
```

Qualifiers

None

Description

Remove a power policy from the power manager

6.5.3.3 setPMGlobalPolicy

Signature

```
RcResult setPMGlobalPolicy(
    [in] RcQoSPowerPolicyId policyId);
```

Qualifiers

None

Description

Select (for use) a power policy.

6.5.3.4 getCurrentPMGlobalPolicy

Signature

```
RcResult getCurrentPMGlobalPolicy(
    [out] RcIQoSPowerPolicy currentPolicy);
```

Qualifiers

None

Description

Return the power policy that is currently used.

6.5.3.5 getPMGlobalPolicy

Signature

```
RcResult getPMGlobalPolicy(
    [in] RcQoSPowerPolicyId policyId,
    [out] RcIQoSPowerPolicy powerPolicy);
```

Qualifiers

None

Description

Get Power policy based on policy id from the Power Manager.

6.5.3.6 getPMGlobalPolicies

Signature

```
RcResult getPMGlobalPolicies(
    out RcIQoSPowerPolicyList policies);
```

Qualifiers

None

Description

Get all Power policies of the Power Manager.

6.5.4 rcIQosQualityChief

Description

This interface is implemented by Quality Aware entities and enable retrieval of the Quality Levels that can be provided by the entity and the required budget for that Quality Level as well as selection of a Quality Level.

Interface ID

```
{785c8eba-55c3-475c-9d27-00cc87325792}
```

6.5.4.1 setQualityLevel

Signature

```
RcResult setQualityLevel (
    [in] RcQoSQualityLevelId     qualityLevelId,
    [in] RcQoSConfigurationId    qualityConfigurationId,
    [in] RcQoSAssignedBudgetsSet assignedBudgets);
```

Description

This operation is called by the QM or by a QC in order to command the application or component to execute according to the quality level specified as parameter. The list of assigned budgets is a set of pairs logicalUserName and budget identifier.

6.5.4.2 getQualityInfo

Signature

```
RcResult getQualityInfo (
    [out] RcQoSQualityInformation qualityInfo);
```

Description

This operation is called in order to get the quality information of the corresponding component or application.

6.5.4.3 getBudgetUser

Signature

```
RcResult getBudgetUser (
    [in]  RcQoSLogicalUserName logicalName,
    [out] RcQoSBudgetUser      budgetUser);
```

Description

This function allows the functional code to get the budget user of a certain budget logical name. In this way, it is possible to register the budget user and to perform operations on the budget, such as getting its status, monitoring information, etc.

6.5.4.4 notifyEvent

Signature

```
RcResult notifyEvent(
    [in] RcQoSApplicationId applicationId,
    [in] RcQoSEventId       eventId,
    [in] RcQoSEventInfo     eventInfo) ;
```

Description

This function allows notification of QoS related events.

7 Overview of realization

This is an informative clause. The realization is described in Clause 8 and addresses the following topics:

— Overview of the roles used for Resource and Quality Management

— Realization of Resource Chiefs

— Realization of Chiefs and Management for specific resources

  — CPU

  — Network

  — Power

— Informative discussion on the composition of Quality Information

8 Resource management

8.1 Overview

The description of the roles in subclause 8.2 present the main functionality of the Resource Management Framework from a conceptual point of view. The deployment of this framework has been done taking into account two main issues. First, some of the roles are outside of the RRE and the M3W framework itself. In particular, the Resource Chiefs are usually very dependent on the underlying operating system and, in some cases, they are included in the OS kernel. So, the interfaces of these entities have not been specified. Second, there is no need to make all the operations that the Resource Manager and the Quality Manager perform public. For this reason, these roles are encapsulated in a QoS Manager, which exports their public Interface. In this way, the user has a cleaner vision of the QoS Manager and the implementer has more freedom for designing and implementing it.

The QoS Manager is a singleton Service. In a Device, there can only be one Service Instance running, to ensure proper system operation. This approach allows interacting with the Resource Management Framework using the standard mechanisms of the M3W Component Model, which facilitates the development of QA Services. A Quality Chief can ask the RRE for the QoS Manager Instance and start the interaction with it, as shown in Figure 2.

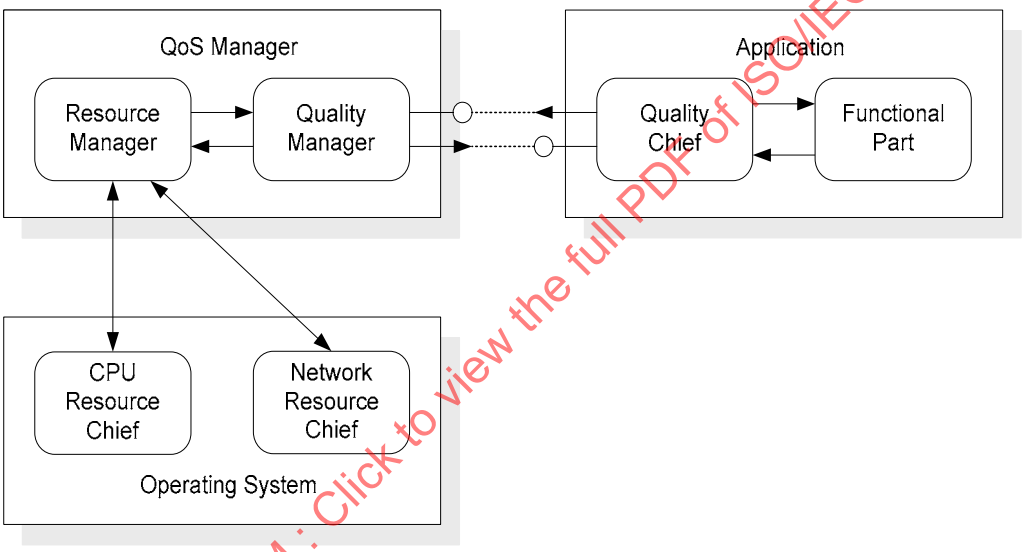

Figure 2 — QoS Manager and Application

In Annex A, the dynamic behaviour of this system is shown.

8.2 Responsibilities of roles in the Resource Management Framework

The purpose of this clause is to describe the architectural structure of the Resource Management Framework. The main roles and their basic functionality are described.

8.2.1 Quality manager (QM)

Its main goal is to manage system global quality. It assigns resources to Applications and Service Instances. The interaction between the QM and the Applications or Service Instances is based on the mentioned contract model. The QM follows a protocol for negotiating this contract. Additional interaction should be performed for re-negotiating, due to the Application needing more resources, not fulfilling the required quality, etc. In summary, the QM is responsible of most of the functionality described in subclause 5.3.

8.2.2 Quality chief (QC)

The quality chief is part of the application. It is the counterpart of the QM. It is in charge of interacting with the QM and driving the execution of the Application, according to the agreed quality level. The complexity of the quality chief varies. Its basic functions are negotiating with the QM, setting the resulting quality levels, and handling overruns and other potential requests from the QM. Advanced functions include Application monitoring, adaptation, management of the Application's resources, etc. The QC can take advantage of domain semantic information for carrying out these operations, which should imply that they are performed more efficiently.

8.2.3 Resource manager (RM)

The resource manager is in charge of performing the platform and device independent actions for managing resources, as described in subclause 5.2. In particular, the main responsibilities of this role are:

— Setting the budgets assigned to each of the threads (or clusters of threads)

— Account for resource usage

— Enforce budgets

— Keep statistical information

— Provide information on available resources to Applications

It closely interacts with the resource chiefs (discussed in the following subclause) for doing these activities.

In the negotiation phase, it uses the admission test to check whether there is sufficient capacity for satisfying a tentative assignment. The admission test can be done for each resource individually or in an integrated way. The second option is the most convenient, as it can consider dependences on the resource usage. This is the reason why this responsibility has not been deferred to the resource chiefs.

8.2.4 Resource chief (RC)

The resource chief has specific knowledge of a particular resource. It is in charge of supporting the Resource Manager to acount and enforce the budgets of this resource. Its main responsibilities are to:

— Isolate platform dependent issues,

— Manage its corresponding resource(s),

— Interact with the Resource Manager for accounting and enforcing purposes.

8.3 Realization of a resource chief

The resource chiefs are in charge of handling the details of the implementation of the budget. They perform the low level operations of the resource manager. They are apart of this because they depend on the operating system and hardware where the RRE is running. In this way, it is possible to isolate the RMF from the platform dependent details. It is expected that there is one chief for each managed resource.

The model of budgets differentiates budget from budget users. Initially, the RMF creates a budget for a particular resource. Although the associated resource share is reserved, it is not already usable. The budget needs to have a user, which can be different entities depending on the resource and the management approach. In the case of the CPU, the budget user is a thread or a group of threads (cluster).

The resource chiefs have to provide a number of basic functions for the interaction with the resource manager:

— Budget handing: The chief exports operations for creating and destroying budgets.

— Budget user handling: The chief exports operations for assigning and removing a user from a budget.

— Status operations: The application, in general, and the budget user, in particular, requires knowing the available budget. An operation is provided for retrieving this information.

— Monitoring operations: The chief provides operations for activating and deactivating the capture of statistical information. It also provides operations for getting this information.

The internal structure of a resource chief can be quite variable, depending on the nature of the managed chief.

8.3.1 CPU management

Some resources, such as CPU, are allocated to threads or clusters (group of threads). This is done when the corresponding Application negotiates with the QoS Manager. As part of the deployment of the negotiated settings, some threads may be created and their execution parameters are set. Alternatively, a thread may be already created and as a result of the negotiation, some of its parameters may need to be changed. In any case, for threads in resource-aware Applications or Services, negotiation is prior to thread execution.

The creation of threads could be done either by Applications or by the runtime (QM or RM). In the Resource Management Framework it is left to the Applications. As part of its code, the Application creates the required threads. However, some characteristics of these threads must be done by the runtime. In particular, it is needed to set some parameters, such as thread priority and thread budget. These two parameters are selected as part of the negotiation. In this case, the Resource Manager is the element that is in charge of setting these parameters. In this way, it is possible to make sure that the parameters set are those that resulted from the negotiation, and the Application or Service could not preclude the correct behaviour of the system, by using wrong values.

In some cases, it may be important for the Application to set the relative priorities for its threads. This is allowed, as far as these priorities are only relative to the threads in the Application. The QM or RM must take this information into consideration for assigning the final priorities to the threads

As an example, some details of the CPU chief are provided in Annex B.

8.3.2 Network QoS management

The quality of network service consists of different characteristics such as throughput, latency and reliability. Applications' demands in respect to these characteristics differ from one Application to another. For example, interactive Applications require low latency and high reliability whereas streaming media applications have higher demands on throughput and jitter. The term Quality of Service in the context networking domain means a set of technologies to manage the effects of congestion on application traffic by using network resources optimally rather than continuously adding capacity.

The default service offering associated with the Internet is characterized as a best-effort variable service response. Within this service profile the network makes no attempt to actively differentiate its service response between the traffic streams generated by concurrent users of the network. As the load generated by the active traffic flows within the network varies, the network's best effort service response will also vary.

8.3.2.1 Network Quality of Service

The objective of various Internet Quality of Service (QoS) efforts is to augment this base service with a number of selectable service responses. These service responses may be distinguished from the best-effort service by some form of superior service level, or they may be distinguished by providing a predictable service response which is unaffected by external conditions such as the number of concurrent traffic flows, or their generated traffic load.

The Integrated Services (IS,RFC1633) model relies on the assumption, that resources need to be explicitly managed in order to meet application requirements to achieve predictable and controlled network service. It has some emphasis on real-time multimedia applications, for which the current Internet service is not always sufficient. Integrated Services does not rely on the current Internet infrastructure, it needs to be extended to support additional traffic control. Integrated Services relies on a signalling protocol to make resource reservations.

The Differentiated Services (DS,RFC2475) architecture takes a "network boundary"-centric approach to provide service quality. The traffic classification and conditioning is done only at the boundary nodes of a Differentiated Services network. Individual flows are aggregated into classes of flows having similar resource needs.

Regarding the Resource Management framework, network connectivity as a resource differs from other resources in the device, such as CPU or memory. The resource cannot be explicitly managed by the device itself because there are external factors affecting the quality. Therefore the support for network QoS means supporting the two architectures presented above.

In practice, support for the QoS mechanisms would be provided by the operating system. The Resource Management Framework should provide a standard network QoS interface for the Applications, and mappings to the mechanisms of the operating system.

8.3.2.2 Network QoS management

The difficulty in managing network QoS is that the quality of a connection is influenced by a number of factors external to the terminal. This implies that the acceptance test is directly coupled with the actual resource reservation. There is no way to separate one from another due to varying external conditions. If a bandwidth test is accepted, the resource needs to be reserved at the same time.

The unit subject to resource reservation is called "a flow". It consists of a group of packets that have the same source and destination. It is typically identified by its endpoints, i.e. network (IP) address, port number and protocol identifier (e.g. TCP/UDP). The relation between flows and Applications is one to many: An application can have several flows, but a flow belongs to a single Application. In many cases an Application does not know the full identification for the flows that are subject to QoS management upon invocation. For example, a video stream player typically receives a location of the stream source from the user. This information usually requires some translation before it is usable by the QoS management system, e.g. parsing an URL, mapping a hostname to an IP address, etc. The actual resource requirements can also depend on the content of the flow.

8.3.3 Power management

There are an increasing number of approaches and techniques to power saving in the literature. An important number of them rely on low level hardware settings or on special software coding or compilation. However, we believe that these approaches are useful but it is not appropriate to base the power management in M3W in these types of techniques in isolation.

In order to make a meaningful power management it is necessary to consider the state of the whole system. Low level decisions on power saving may preclude the fulfilment of functions that are fundamental for system operation. The approach to power management is based on considering the global system state in power system level decision making. In particular, the decision on the power saving strategy and initial power settings will be decided by the Resource Management Framework taking into account the following information:

— User preferences and settings. The final goal of this type of systems is to satisfy the user, so it is clear that his preferences are of primary importance. For this purpose, functions should be included in the API to let him state his preferences with respect to power saving. Indirectly he also can set the feasible power strategy by setting which are the Applications he wants to execute and how important they are

— System work load. The current system work load can be used to determine the power settings, which could be the least power consuming configuration that still provides the required hardware needs for the Applications currently running. In a situation where it is advisable to reduce power consumption even more, it could be analysed which of these Applications are really important and renegotiate the contract. Anyhow, there are some system functions and user Applications that are critical and that should be executed in any case.

— Temperature, battery level, power source. These states should also be considered to decide whether a more or less aggressive strategy is going to be used in the future.

— Power-aware Services. There could be Services that may provide quality levels with power saving algorithms or that use less power consuming hardware devices. The Resource Management Framework should select the appropriate quality level, according to the system power status and the active power policy.

At a lower level, it may be possible to perform additional power saving actions, such as turning off some hardware device that is not being used. In any case, this lower level power savings should respect the constraints imposed by the Resource Management Framework and ensure that the budgets assigned to Applications are met.

Finally, as has been mentioned above, there is a large number of techniques for power saving. The ones that we have found to be relevant in the context of this project are:

— Dynamic frequency and voltage scaling

— Different power consumption states in hardware devices

— System congruent operating states

The decision on the power setting is ultimately done by the QM, taking into account the previously described information. For this purpose, it needs to rely on some entity that knows about the power management and advises the QM on the power settings to use. Moreover, there can be different algorithms to use, depending on battery status, the context, etc. In the context of the Resource Management Framework, this entity is called the Power Policy Manager (PPM). In order to improve the flexibility of this approach, the Resource Management framework allows the dynamic change of policies, to allow for an optimal selection.

8.4 Composition of quality information

This is an informative subclause. The composition of quality information is required when a QA component is using other QA components. According to the definition of this entity, each of them has a number of quality levels. It is obvious that a QA component can command the used ones to provide it a certain quality level. It is also clear that during the negotiation it is necessary to assign resources for all of them. The problem is how to perform the negotiation in this situation.

There are two main jobs to carry out:

— Quality level composition: a component provides a number of quality levels. However, the availability of some of them may depend on the quality of services that need to be provided by other components. For example, a component can only provide high precision mathematical results, if a high precision square root is available. At a particular time, the provision of this high precision depends on whether the required services are in the system. In consequence, it is necessary to calculate which quality levels a component provides at a certain point in time, taking into consideration the quality levels provided by the components it requires for its execution.

— Resource usage composition: it is necessary to know the resource demand of the quality level that a component provides, taking into account the QA components that it needs. For this purpose, it is necessary to compose the resource needs for each of them and assign them to the quality level. The procedure to perform this computation depends on the specific resource and the characteristics of the budget. In the next clause, this procedure is illustrated with an example for the CPU and memory.

The above operations can be done in different ways:

— Centralized: In this case, it is the QM that composes the quality information. For this purpose, it needs that all the QA components are registered in the RMF. In addition, they have to state the components that they need for executing. It is up to the QM to combine the quality and resource usage information.

— Distributed: Each component composes the quality information of those that it uses. At the same time, it provides this information to the components that may use it. If this component is the one providing the global service (usually called in the context of the RMF application), it is in charge of providing the composed information to the RMF.

These approaches have advantages, which are in most cases complementary. However, the second choice was finally selected for implementation. The reason is that the composition of the quality levels requires domain specific knowledge. How a generic RMF can know which quality levels of an application are valid or invalid, depending on the quality level of the required components? It is much simpler to let the application use this specific knowledge for determining the feasible quality level, given the QA components available.

More information can be found in Annex D.

Annex A
(informative)

Dynamic view of the Resource and Power Management Framework

In this annex, some sequence diagrams are given, which illustrate the dynamic behaviour of the resource management framework.

Figure A.1 shows the negotiation protocol. In this example, an Application is started and the Quality Chief collects the information on its quality and resource needs, taking into account the information from the Service Instances it uses. Then it requests the Quality Manager to find a suitable configuration, which starts a process of evaluation of possible configurations and their viability. A configuration at this level consists of a particular quality level for each of the Applications and some resources for non resource-aware Applications. This process finishes when a configuration is found that is feasible and that maximises the global system quality. Then, the selection is communicated to the Quality Chief of the Applications, who commits to provide the corresponding quality.

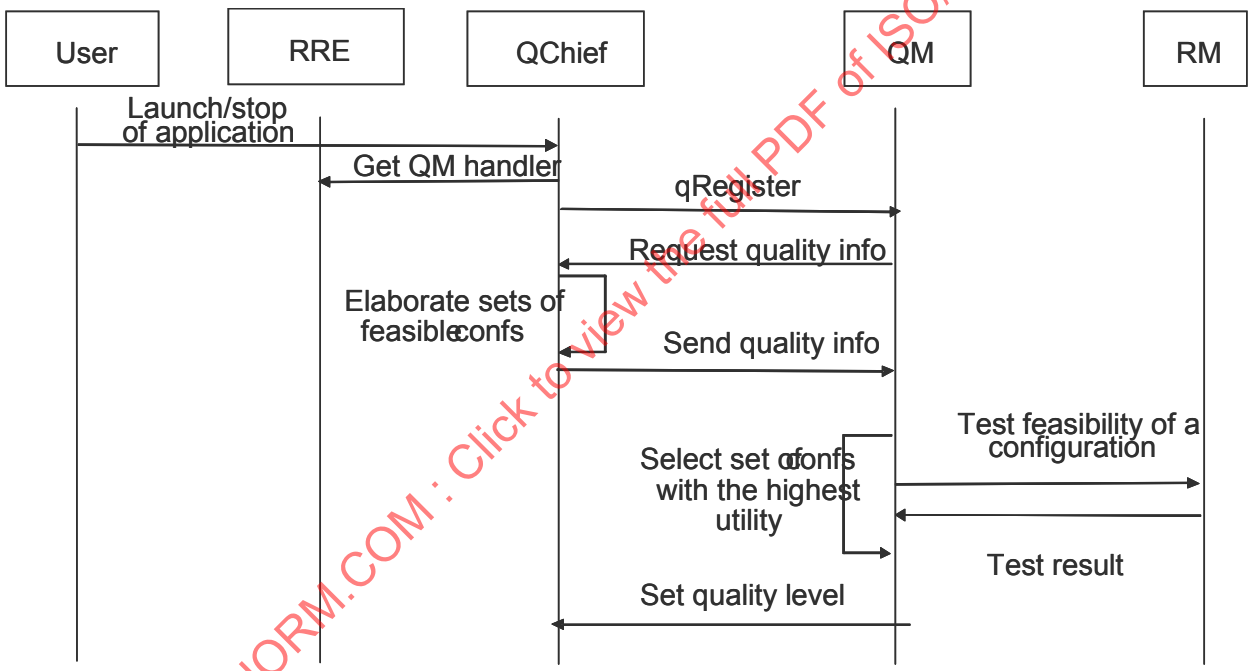

Figure A.1 — Negotiation Protocol

Figure A.2 — Setting Process shows the setting process. Once a system configuration has been negotiated, it is time to effectuate the negotiated configuration. All the Applications may have to change their operation mode accordingly. It is necessary to ensure that the system behaves provides the correct functionality during this re-configuration change. For this purpose, it should be ensured that the switching to certain Quality Modes by the Applications follow a certain order. This order is dictated the Quality Manager. The Quality Manager requests each of the changing Applications to adapt its execution to the committed quality level. For this purpose, it may be necessary to change the set of executing threads, or some of their characteristics (the code it executes, time requirements, etc.). To realise this, it may use the operating system (which includes the resource chiefs). The budgets assigned by each of the threads or clusters are set by the Resource Manager after the corresponding threads have been created and notified. Finally, the Application (re)starts its execution and notifies the Quality Manager of the end of the setting process.

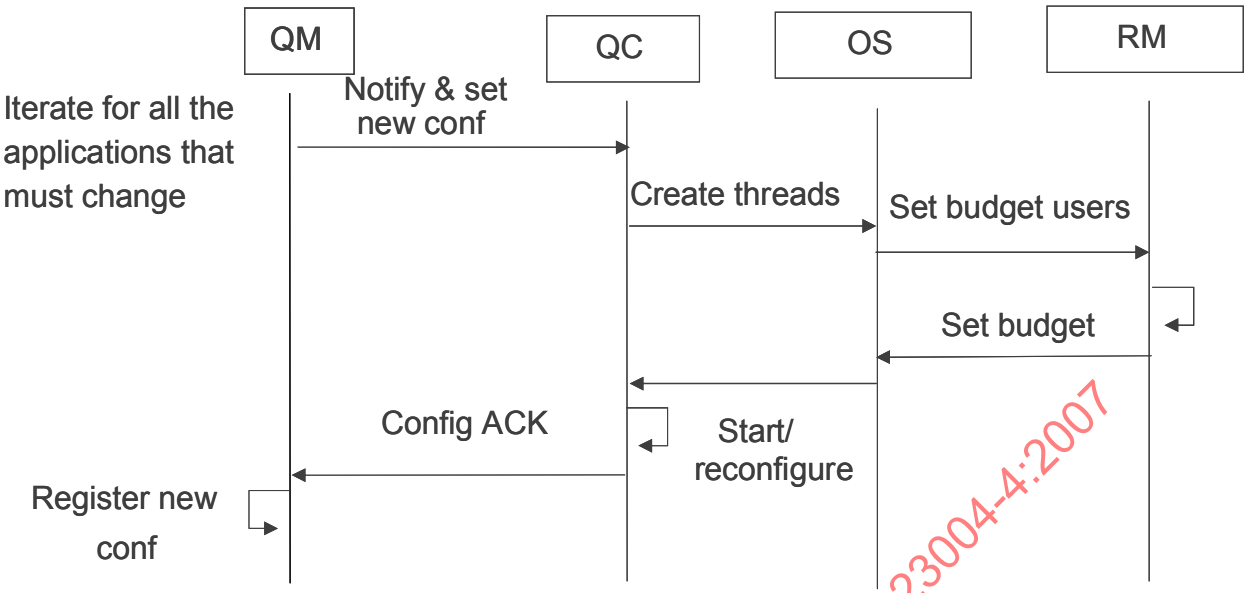

Figure A.2 — Setting Process