Adaptivity & Control of Resources in Embedded Systems

D1f

COOPERATION

Interface Specification

Responsible: **TU Kaiserslautern** (TUKL)

Karl-Erik Årzén (ULUND), **Pascal Faure** (AKA), **Gerhard Fohler** (TUKL), **Marco Mattavelli** (EPFL), **Alexander Neundorf** (TUKL), **Vanessa Romero** (ULUND)

Project Acronym: **ACTORS**
Project full title: **Adaptivity and Control of Resources in Embedded Systems**
Proposal/Contract no: **ICT-216586**
Project Document Number: **D1f 1.0**
Project Document Date: **2009-01-31**
Workpackage Contributing to the Project Document: **WP1**
Deliverable Type and Security: **R-PU**

Contents

Chapter 1

Introduction

This deliverable is the result of task 1.5 "Requirements/QoS Interface", which is part of workpackage 1 "Specification and design of actor components and networks of actors" of the ACTORS project. Right now this deliverable is an intermediate version and there will be a final version towards the end of the ACTORS project.

This deliverable presents the interface of the ACTORS resource manager from the point of view of the applications using it. The resource manager is introduced in deliverable D3a "State Abstractions", which is also available as an intermediate version now. The purpose of the resource manager is to distribute available resources in such a way among applications that an overall maximum service quality of the system is achieved. In order to have flexibility in assigning different amounts of resources to applications, the application must be flexible, i.e. they must be able to adapt to different availability of resources. This deliverable shows how applications can support different service levels, and how this can be represented using the application interface of the resource manager. Deliverable D4b "RBS Specification" on the other hand presents the interface of the resource manager to the other side, the operating system. The results of this deliverable are necessary for all tasks of workpackage 3 since they are all closely related to the resource manager, and for task 2.2. In task 2.2 a runtime system for CAL applications will be developed. This runtime system has to provide the possibility to run system actors. System actors are the components which will actually implement the interface from the application side, i.e. they will be the ones who connect to the resource manager.

Since deliverable D3a is also an intermediate version, we also present the current state of the application interface of the resource manager here in chapter two. Chapter three gives an introduction into MPEG4 AVC [1] and its extension SVC [2]. These extensions of MPEG4 add scalability features both in the temporal and spatial domain. This makes them very interesting as candidates for the video decoding applications necessary for the ACTORS demonstrators, since these scalability features correspond to the need for adaptive applications. Without applications being adaptive the resource manager has much less flexibility in distributing resources among them.

So while SVC provides very useful features for supporting multiple service levels in video decoders, chapter four presents how an MPEG4 Simple Profile decoder can support the application interface of the resource manager by offering multiple quality levels. We chose MPEG4 Simple Profile instead of SVC here for two reasons: even if Simple Profile MPEG streams don't have built-in scalability features it can still be useful to provide different service levels, i.e. reduced CPU requirements when decoding them, and right now we actually have a Simple Profile decoder written in CAL available.

The fifth and last chapter shows how a different kind of application, a control application can support the application interface of the resource manager, and by that shows that the proposed interface should be flexible and powerful enough for very different types of applications. In the coming iterations of the project modifications and improvements of the interface will of course most probably be necessary.

Chapter 2

The Application Interface of the Resource Manager

2.1 Overview

The interface of the resource manager will be fully documented in the final version of the deliverable D3a, here only an overview will be presented.

In an ACTORS system there will be an operating system with support for resource reservations, one global resource manager running in userspace, and a number of applications. Applications participating in the resource management provided by ACTORS register with the resource manager and publish their quality levels and associated resource requirements. It is then the task of the resource manager to determine the service levels for the applications and a distribution of resources, mainly CPU time among the applications which leads to a satisfying behaviour of all applications in the system. In general we expect that applications providing more detailed information about their behaviour will get a reservation which fits their needs more closely than applications providing less detailled information, which includes applications not participating in the resource management at all. The interface shall be generic enough to support not only applications written using CAL, but also in any other language.

The applications can be split into two groups, applications with discrete quality levels and applications with continuous mapping between resource usage and achieved quality. The resource requirements published by the applications shall be platform independent, e.g. specifying a CPU bandwidth requirement as percentage of the total available CPU bandwidth would not satisfy this criteria, since it would depend on the performance of the concrete CPU. On the other hand specifying e.g. a CPU time granularity of 40 ms for a video stream is platform independent since the frame rate and by this the deadlines will always be the same, independent from the actual hardware platform which is used.

2.2 D-Bus

The resource manager will be an application running in userspace on a Linux system. It will communicate with the set of running ACTORS-aware applications. There are different options for implementing the communication that could be used, such as shared memory, pipes, or local sockets could be used, remote procedure call methods like ONC RPC [3], Java RMI [4] or CORBA [5] could be used, but we are proposing to use D-Bus [6] instead. D-Bus is a message bus system, which enables applications on one computer to talk to each other. It uses a star topology

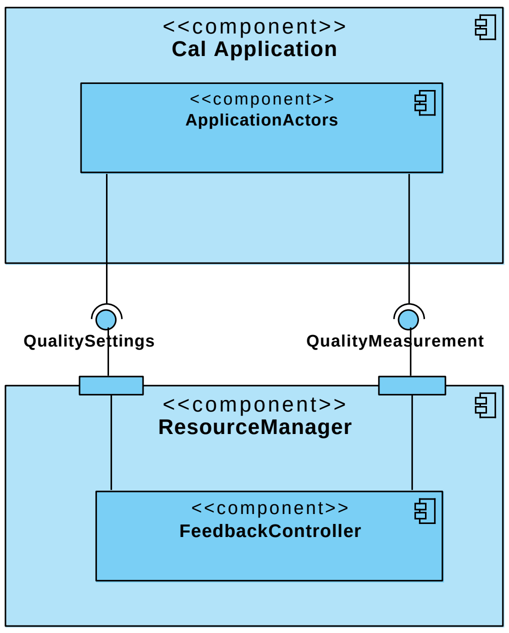

Figure 2.1: Resource Manager Applications Interface

with a central D-Bus daemon to which all applications connect. To talk to another application they send a message to the D-Bus daemon which forwards it to the receiver application, this constitutes the so-called bus. There can be multiple of those buses running on a system, both systemwide buses, i.e. for all users, and "local" buses for users or specific uses. In the last few years D-Bus has established itself as the standard way for communicating between applications on Linux systems. On all recent Linux distributions by default a system bus and a per-user session bus are running. The system bus is used for system wide events, which can e.g. originate from the kernel and in this way be sent to any user application. The per-session bus is used e.g. as the main communication medium of the two major desktop environments for Linux, KDE 4 [7] and GNOME [8]. This widespread deployment and the availability of bindings for many programming languages makes it the obvious candidate for implementing the interface for ACTORS. The D-Bus library is dual licensed under GPL version 2 [9] and the Academic Free License 2.1 [10], so it can be used freely both for free software projects as well as for closed software projects.

Compared to shared memory it has a higher overhead since the messages are serialized, read and written multiple times until they are available in the receiver. Since we expect that changes of the resource reservations and service levels of the application will happen infrequently, this shouldn't be a problem.

Using D-Bus cannot really be compared to using pipes or local sockets, since it is actually the same. D-Bus uses internally local sockets for communication. So using pipes or local sockets directly for the interface would have meant that we would have to create an ACTORS-specific protocol to use on top of these sockets, and in the end we would probably end up with something like a reduced version of D-Bus, but with less features, less testing and therewith more bugs.

Compared to CORBA, D-Bus is easier to work with and it is much less complex, and it comes already with every Linux installation. Java RMI is not a candidate because it is bound to Java as implementation language. ONC RPC is network-

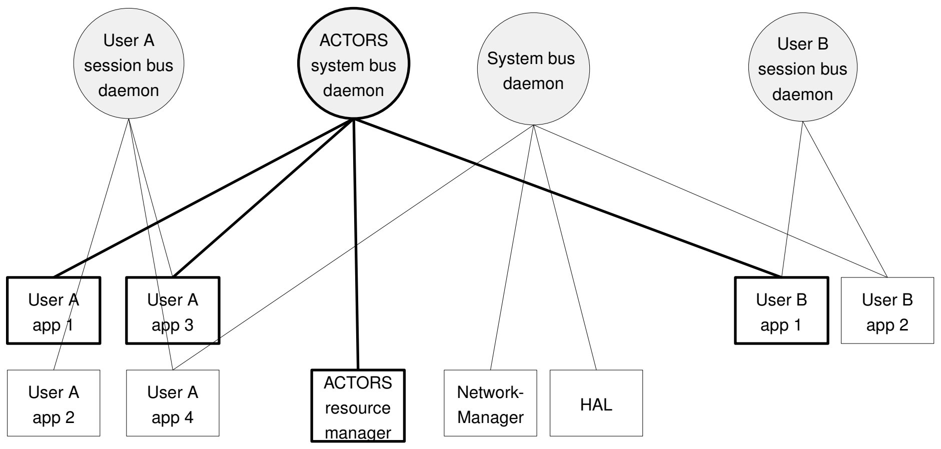

Figure 2.2: Possible D-Bus bus setup in ACTORS

centric and less flexible, e.g. it doesn't support sending signals as D-Bus does.

In ACTORS we will start by using the per-user session bus, since this is the easiest way to get something working. In the final system we will use a system-wide bus since the resource manager potentially has to coordinate applications from multiple users. We will investigate whether this will be the default D-Bus system bus or whether we need a separate ACTORS system bus daemon. For the resource manager and the applications the only difference will be the address used for identifying the bus to connect to. A possible scenario using a separate bus for ACTORS can be seen in Fig. 2.2. The interface will be implemented completely in the side of the resource manager, the participating applications are only users of that interface, they don't have to provide any D-Bus interface by themselves, this makes adding support for the ACTORS resource manager to applications easier.

2.3 Interface Specification Format

The resource manager will provide the application interface via D-Bus. D-Bus provides introspection functionality so that the interface of D-Bus service providers can be discovered at runtime. Introspection is provided by D-Bus objects by implementing the standard interface `org.freedesktop.DBus.Introspectable`. This interface has just one method, `org.freedesktop.DBus.Introspectable.Introspect (out STRING xml_data)`. This method returns the string `xml_data`, which contains the description of the interfaces of this object. The XML format is documented in [6] and there is a formal DTD[1] [11] for it, so the XML interface description can be verified for correctness.

Using this DTD it is possible to describe which methods are available and which arguments they support. This is enough information to document the "wire-format", but it is not expressive enough to be able to understand the semantics of the interface, e.g. composite type arguments are anonymous and this is not sufficient for code generators, so more information has to be added. The Telepathy project [12]

[1]Document Type Definition, see e.g. http://www.w3schools.com/DTD/dtd_intro.asp

extended the XML format to allow for more information in the interface specification [13]. This extended D-Bus introspection format will be used to specify the resource manager interface. Details about it can be found in the appendix.

2.4 Formal Resource Manager Interface Specification

Here follows the specification of the D-Bus interface of the ACTORS resource manager using the extended D-Bus introspection format. It should meet the requirements described above and in D3a. Due to the ongoing work in the project it will be necessary to update it from time to time.

```
<?xml version="1.0" ?>
<!-- DOCTYPE node PUBLIC
    "-//freedesktop//DTD D-BUS Object Introspection 1.0//EN"
    "http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd" -->

<node xmlns:tp="http://telepathy.freedesktop.org/wiki/DbusSpec#extensions-v0">
  <interface name="eu.actors-project.ResourceManagerInterface">

     <tp:enum name="ResourceId" value-prefix="Resource" type="u">
        <tp:enumvalue suffix="None" value="0" />
        <tp:enumvalue suffix="CPUBandwidth" value="1" />
        <tp:enumvalue suffix="CPUGranularity" value="2" />
        <tp:enumvalue suffix="MaxParallelity" value="3" />
        </tp:enumvalue>
     </tp:enum>

     <tp:enum name="FunctionType" value-prefix="Function" type="u">
        <tp:enumvalue suffix="None" value="0" />
        <tp:enumvalue suffix="Linear" value="1" />
        <tp:enumvalue suffix="Quadratic" value="2" />
     </tp:enum>

     <tp:enum name="Category" value-prefix="Category" type="u">
        <tp:enumvalue suffix="None" value="0" />
        <tp:enumvalue suffix="Low" value="1" />
        <tp:enumvalue suffix="Medium" value="2" />
        <tp:enumvalue suffix="High" value="3" />
     </tp:enum>

     <tp:mapping name="ResourceDemand">
        <tp:member type="u" tp:type="ResourceId" name="Key"/>
        <tp:member type="u" name="Value"/>
     </tp:mapping>

     <tp:struct name="QualityLevel" array-name="QualityLevelList">
        <tp:member type="u" name="quality" />
        <tp:member type="a{uu}" tp:type="ResourceDemand" name="resourceDemands" />
     </tp:struct>

    <method name="registerApp">
      <arg type="i" direction="out" />
      <arg name="applicationId" type="s" direction="in" />
    </method>

    <method name="announceCPUCategory">
       <arg type="i" direction="out" />
       <arg name="applicationId" type="s" direction="in" />
       <arg name="category" type="u" tp:type="Category" direction="in" />
    </method>

    <method name="announceQualityLevels">
       <arg type="i" direction="out" />
       <arg name="applicationId" type="s" direction="in" />
       <arg name="qualityLevels" type="a(ua{uu})"
            tp:type="QualityLevel[]" direction="in" />
    </method>

    <method name="announceContinuousQualityFunction">
      <arg type="i" direction="out" />
      <arg name="applicationId" type="s" direction="in" />
      <arg name="type" type="u" tp:type="FunctionType" direction="in" />
      <arg name="param1" type="d" direction="in" />
      <arg name="param2" type="d" direction="in" />
      <arg name="param3" type="d" direction="in" />
      <arg name="param4" type="d" direction="in" />
      <arg name="param5" type="d" direction="in" />
    </method>

    <method name="reportHappiness">
      <arg name="applicationId" type="s" direction="in" />
      <arg name="happiness" type="u" direction="in" />
    </method>

    <signal name="changeQualityLevel">
      <arg name="applicationId" type="s" direction="in" />
      <arg name="newLevel" type="u" direction="in" />
    </signal>

    <signal name="changeContinuous">
      <arg name="applicationId" type="s" direction="in" />
      <arg name="newValue" type="i" direction="in" />
    </signal>

  </interface>
</node>
```

Since the XML specification is quite verbose, below the interface can be found translated to a pseudo-C++-like language, which is more concise (but doesn't compile). Our intention is to have an interface similar to this automatically generated from a code generator for use in the resource manager and potentially also in the client applications.

```
enum ResourceId
{
   ResourceCPUBandwidth = 1,
   ResourceCPUGranularity = 2,
   ResourceMaxParallelity = 3,
   ResourceEnergy = 4
};

enum FunctionType
{
   FunctionNone    = 0,
   FunctionLinear = 1,
   FunctionQuadratic = 2
}

enum Category
{
   CategoryNone = 0,
   CategoryLow = 1,
   CategoryMedium = 2,
   CategoryHigh = 3
}

struct QualityLevel
{
   unsigned int quality;
   map<ResourceId, unsigned int> resourceDemands;
};

class eu.actors-project.ResourceManagerInterface
{
   public:
      int registerApp(string applicationId);
      int announceCPUCategory(string applicationId,
                              Category category);
      int announceQualityLevels(string applicationId,
                                list<QualityLevel> qualityLevels);
      int announceContinuousQualityFunction(string applicationId,
                                            FunctionType type,
                                            double param1,
                                            double param2,
                                            double param3,
                                            double param4,
                                            double param5);
      void reportHappiness(string applicationId, unsigned int happiness);
   signals:
      void changeQualityLevel(string applicationId, unsigned int newLevel);
      void changeContinuous(string applicationId, int newValue);
};
```

2.5 Documentation of the Interface eu.actors-project.ResourceManagerInterface

This section contains the description of the interface methods of the resource manager. They are designed in such a way that the clients are purely passive, they don't have to provide any interface. They can make use of the resource manager, but they don't have to. It is also completely agnostic to the fact whether the client applications are CAL applications or applications written in any other language.

The documentation uses the format shown below:

Method <method_name>(<parameter_name>: <type>) → <return_type>

The types are either standard D-Bus types, see Table A.1 in the appendix, or custom project-defined types. These custom types are composed also of the standard D-Bus types and documented after the methods. This is done in a similar way for the documentation of the method parameters. Here, if a parameter is a custom type, also its D-Bus type signature is shown:

`<parameter_name>: <custom_type><type>` <documentation>

Method registerApp (applicationId: s) → i

Each application which participates in the resource management has to register with the resource manager. The applicationId used here will be used later on by the resource manager to address this application.

Parameters

`applicationId:s` This string has to be generated by the application and must be systemwide unique. A combination of application name, process id and thread id may be used.

Returns

`i` Integer return value, 0 on success, an error code otherwise.

Method announceCPUCategory (applicationId: s, category: Category) → i

Called by client applications to indicate how much CPU bandwidth they might require. There are only three different categories, so this gives only a very rough grouping. These categories will be used by the resource manager to calculate an initial resource distribution. An example for an application of CategoryHigh would be a high quality video encoder, an example for CategoryLow would be a basic control application or a wav-file player.

Parameters

`applicationId:s` The unambiguous name under which the application registered using registerApp().

`category:Category (u)` Specifies the overall CPU requirement category for this application.

Returns

`i` Integer return value, 0 on success, an error code otherwise.

Method announceQualityLevels (applicationId: s, qualityLevels: QualityLevel[]) → i

Called by client applications to make their supported quality levels known to the resource manager. Any previously announced quality levels or quality functions for this client are discarded.

Parameters

`applicationId:s` The unambiguous name under which the application registered using registerApp().

`qualityLevels:QualityLevel[](a(ua{uu}))` This is one of the most important data structures of the interface. This is the list of the quality levels supported by the application. Each quality level has an associated quality indicator and the set of resource requirements for this level, e.g. the time granularity. All resource requirements should be hardware/platform independent.

Returns

`i` Integer return value, 0 on success, an error code otherwise.

Method announceContinuousQualityFunction (applicationId: s, type: FunctionType, param1: d, param2: d, param3: d, param4: d, param5: d) → i

Called by client applications which don't offer discrete quality levels. Here they inform the resource manager how their achieved quality depends on the available resources. There are some types of functions supported, each function can be parametrized using up to 5 (number arbitrary chosen) parameters.

Parameters

`applicationId:s` The unambiguous name under which the application registered using registerApp().

`type:FunctionType (u)` Selects the appropriate function type to characterize the behaviour of this application.

`param1:d` Parameter 1 for the given function type. Used by FunctionLinear and FunctionQuadratic.

`param2:d` Parameter 2 for the given function type. Used by FunctionLinear and FunctionQuadratic.

`param3:d` Parameter 3 for the given function type. Used by FunctionQuadratic.

`param4:d` Parameter 4 for the given function type. Currently unused.

`param5:d` Parameter 5 for the given function type. Currently unused.

Returns

`i` Integer return value, 0 on success, an error code otherwise.

Method reportHappiness (applicationId: s, happiness: u) → nothing

This function is called by the client to report how well it is currently achieving the assigned quality.

Parameters

`applicationId:s` The unambiguous name under which the application registered using registerApp().

`happiness:u` An integer value between 0 and 100 expressing how well the assigned quality is actually achieved. So if the application is told to run at the lowest level and is able to do this perfectly, it should report 100, if it is told to run at the highest level but can do only 50 % of that it should report a lower value, e.g. 50.

Signal changeQualityLevel (applicationId: s, newLevel: u)

This signal is emitted to notify an application about the quality level it should run on.

Parameters

`applicationId:s` The unambiguous name under which the application registered using registerApp().

`newLevel:u` The index of the quality level at which the application should run on.

Signal changeContinuous (applicationId: s, newValue: i)

This signal is emitted to notify an application which announced a continuous resources-to-quality mapping function about the new amount of resource available for it.

Parameters

`applicationId:s` The unambiguous name under which the application registered using registerApp().

`newValue:i` The new amount of resources available to the application.

Enum ResourceId

The resource id is used to identify the different resources managed by the ACTORS resource manager. More resource ids may be added later if necessary.

Values

`ResourceNone=0` Invalid resource.

`ResourceCPUBandwidth=1` The CPU bandwidth resource. This value will be interpreted relative to the maximum CPU bandwidth value for this application.

`ResourceCPUGranularity=2` The CPU granularity resource. This will be an absolute time value in microseconds which specifies during which time the requested CPU bandwidth should be provided.

`ResourceMaxParallelity=3` This specifies how many threads in parallel make sense for this application. For a single threaded application this would be 1, for an application running two threads it would be 2, for an application running multiple threads it will probably be higher.

Enum FunctionType

The function type is used to specify the continuous function which describes the relation between available CPU bandwidth and achievable service quality for applications which don't have discrete quality levels.

Values

`FunctionNone=0` Invalid function.

`FunctionLinear=1` The relation between the CPU bandwith and the achieved quality can be described using a linear function.

`FunctionQuadratic=2` The relation between the CPU bandwith and the achieved quality can be described using a quadratic function.

Enum Category

Categories can be used to group items according to some property.

Values

`CategoryNone=0` Invalid category.

`CategoryLow=1` A generic "Low" category.

`CategoryMedium=2` A generic "Medium" category.

`CategoryHigh=3` A generic "High" category.

Struct QualityLevel - (ua{uu})

A struct representing one quality level of an application, i.e. the quality and the requried resources. It is used in announceQualityLevels(). In bindings that need a separate name, arrays of QualityLevel should be called QualityLevelList.

Members

`quality:u` This one indicates the quality of this level. It is an integer value ranging from 0 (worst) to 100 (best).

`resourceDemands:ResourceDemand[] (a{uu})` This is the map of resource identifications to their required amount, e.g. `ResourceCPUBandwidth` or `ResourceCPUGranularity`.

Map ResourceDemand - a{ Key: u → Value: u }

A mapping from resource ids to an amount (absolute or relative) of them. The amount must be an integer value.

2.6 Information Flow

The overall flow of information from and to the resource manager is depicted in Fig. 2.3.

In the center the resource manager is located. An application which wants to have its resources managed by ACTORS has to actively register itself with the resource manager. This is done by connecting to the respective D-Bus bus and calling the `registerApp()` method of the resource manager. The next step is to announce the overall CPU bandwidth requirements category of the application using `announceCPUCategory()`. Then, depending on the type of application, the application has to announce either its supported quality levels or its continuous function which maps the available resources to achievable quality by calling the methods `announceQualityLevels()` or `announceContinuous()` respectively. Each quality level consists of a numerical value which gives an indication for the quality of this level, `QualityLevel::quality`, together with a set of resource requiments for this level stored in the map `QualityLevel::resourceDemands`.

When multiple applications have registered in this way with the resource manager, the resource manager will determine a distribution of resources which leads to the desired overall system behaviour. This includes determining the service level and the parameters of the reservation for each application. The applications will be notified via the `changeQualityLevel()` and `changeContinuous()` D-Bus signals about their assigned service levels. Here signals are used because otherwise the applications would have to provide a D-Bus interface themselves, which would increase the amount of modifications required to add support for the resource manager to applications.

When the application receives this notification, it shall adapt its inner algorithms in some way, so that it uses the specified resources while producing the promised quality. This may not always be successful, for instance due to varying computational demands from the work load, e.g. in an MPEG stream [14]. It is an important information for the resource manager whether the application is able to achieve the promised quality of the assigned service level. It is up to the applications to determine an integer value between 0 and 100 which expresses how well it currently achieves the designated service level, we'll call this number the *Happiness* of an application. Each application will inform the resource manager by calling the `reportHappiness()` method about its current Happiness. In task 3.2 we will investigate whether this Happiness number has to be sent periodically or event-based.

Additionally to these Happiness numbers the resource manager will be provided with important values for the applications. They are defined by the user or

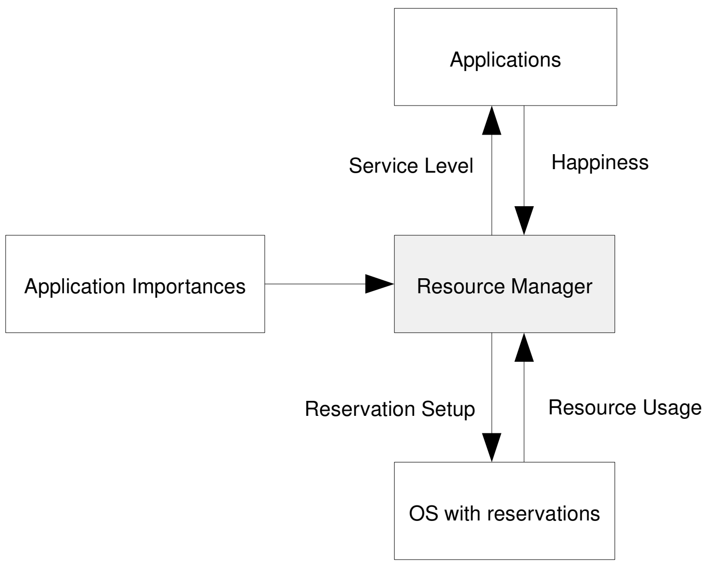

Figure 2.3: Information flow from and to the resource manager

system integrator, and serve as hint to the resource manager which applications it should prefer and which can degrade first. Primary functions of the system like e.g. phone calls for a mobile phone should be assigned a higher importance than add-on functionality like e.g. an animated effects in the user interface. Practically the importance values could be provided using an XML-file which is read by the resource manager.

When the resource manager has determined a distribution of resources, it will set up the reservations accordingly. This will be done by specifying bandwidth and period or equivalent parameters as e.g. (α, Δ) as introduced in deliverable D3a, for the reservations and sending them to the operating system. Then the applications can run within these reservation. The operating system should also report back some resource usage information, i.e. whether deadlines are missed or to which percentage the reservations are used by the applications.

The resource manager will take its decisions based on this information together with the resource usage information from the operating system.

2.7 Example Session with the Resource Manager

Fig. 2.4 shows the communication between two applications *App1* and *App2* with the resource manager as a sequence diagram. At the beginning the reource manager is already running. Then App1 starts and connects to the resource manager, this is done in steps 1 to 3. After that the resource manager determines the distribution of resources, which should be easy since there is only one application. In step 4 it notifies App1 about the quality level at which it should run by sending a D-Bus signal. At some point App1 reports its happiness back to the resource manager, as can be seen in step 5. This can happen once as in the example or multiple times.

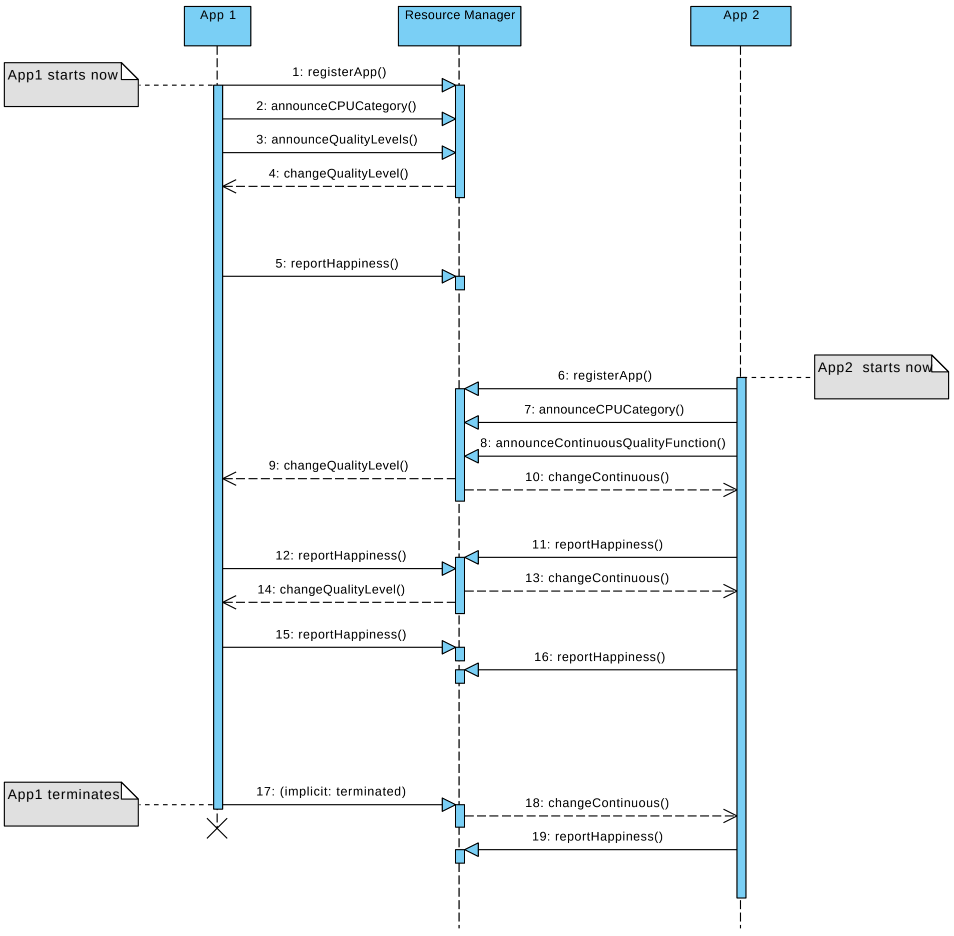

Figure 2.4: Example communication sequence between the ACTORS resource manager and two applications

Then App1 runs for some time without significant events happening, after some time application App2 starts and registers with the resource manager (steps 6 to 8). App2 doesn't support discrete quality levels but instead announces a continuous quality function. With now two applications the resource manager finds a new distribution of resources and quality levels, and notifies both applications about this (steps 9 and 10). Both applications report their happiness (steps 11 and 12) and based on this information the resource manager changes the reservation and service levels of the applications (steps 14 and 15). Again App1 and App2 report their happiness back (steps 16 and 17) but this time the resource manager doesn't change the setup. After some time App1 terminates. The resource manager detects this at step 17 and can give more resources to App2 (steps 18 and 19).

Chapter 3

Scalable Video Processing Impact on Resource Management

3.1 Overview of the MPEG H.264/SVC Standard

H.264/AVC scalable video coding (SVC), also denoted as the scalable extension of H.164/AVC, has been recently standardized by The Joint Video Team of the ITU-T VCEG and the ISO/IEC MPEG [2]. SVC provides the functionalities to allow for encoding multiple scalable representation in a single bit stream. The scalability applies in temporal, spatial, SNR quality and the combined scalability of them. Such functionalities facilities the applications such as video adaptation, transmission, and storage, without much compromise of the coding efficiency [2].

Basic concepts for extending H.264/AVC towards a scalable video coding standards

1. The Temporal Scalability:

   The temporal scalability in SVC is achieved through the Hierarchical B pictures which restricts the temporal prediction order for motion-compensated prediction process [15], [16]. Comparing with the scalability of the previous standards, H.264/AVC provides a significantly increased flexibility through the adaptive control from the memory management control operation (MMCO) commands and the arbitrary selection via the reference pictures list re-ordering (RPLR) commands. These tools were originated from the H.264/AVC which implies SVC's good extensibility. The dyadic hierarchial B pictures is illustrated in Fig. 3.1 from [17]. Besides the dyadic temporal prediction sturcture, non-dyadic hierarchical prediction structure is also possible for low delay coding requirement [2].

2. The Spatial Scalability:

   For spatial scalable coding, SVC defines several inter-layer prediction mechanisms such as the inter-layer intra prediction, inter-layer motion and residual prediction [18]. With Inter-layer Motion and Residual Prediction, for each inter-coded macroblock within an Enhancement P (EP) or Enhancement B (EB) slice, the motion information may be predicted from a corresponding inter-coded block in the lower spatial reference layer. The residuals may also be predicted from that block. With Inter-layer Intra Prediction, each macroblock within an Enhancement I (EI), Enhancement P (EP) or Enhancement B (EB) slice could be intra predicted from the corresponding block in the lower

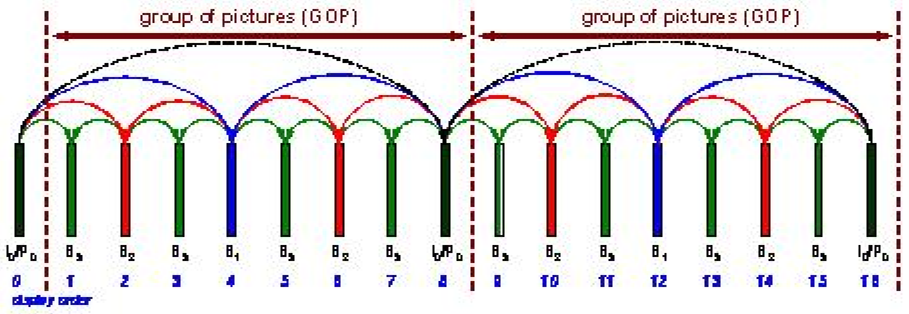

Figure 3.1: Temporal scalability with dyadic hierarchical B pictures

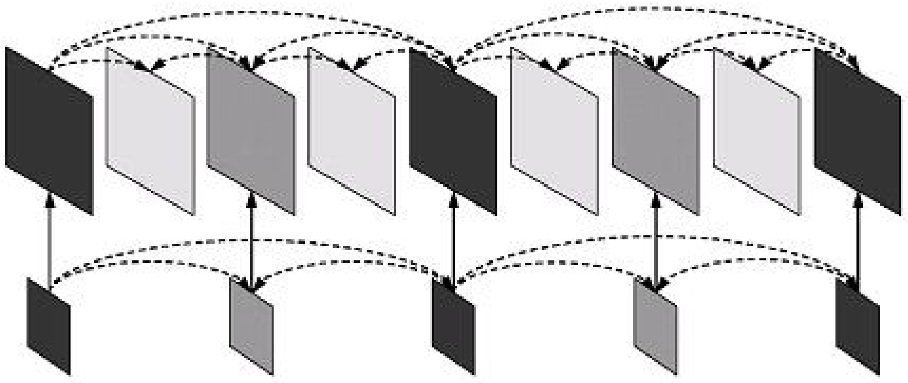

Figure 3.2: Spatial scalability with inter-layer prediction

reference layer if that block is intra-coded, as illustrated in Fig. 3.2 from [2]. The motivation of the inter-layer prediction is to enable the usage of as much lower layer information as possible for improving rate-distortion efficiency of the enhancement layers [2].

3. The SNR Quality Scalability:

   For SNR scalability, the degree of the granularity can be achieved through coarse-grain quality scalable coding (CGS), medium-grain quality scalability (MGS), and fine-grain quality scalability (FGS) via the slice type and the signaling information. Coarse-grain quality scalable coding (CGS) is a special case of the spatial scalability with identical picture sizes for base and enhancement layer. The difference is the quantization step size used in each layer. Medium-grain quality scalability (MGS) is enabled to address the problem of CGS such as the limited numer of rate points. Rather having only one refinement for single spatial layer, MGS provides multiple quality layers for each spatial layer. Therefore, MGS enables more operation points for stream extraction. Fine-grain quality scalability (FGS), finally has been considered over-engineered and might be considered as the future exploitation topic for SVC [17].

4. The Combined Scalability:

   In the SVC extension of H.264/AVC, the basic concepts for temporal, spatial, and quality scalability as described are combined, as in Fig. 3.3.

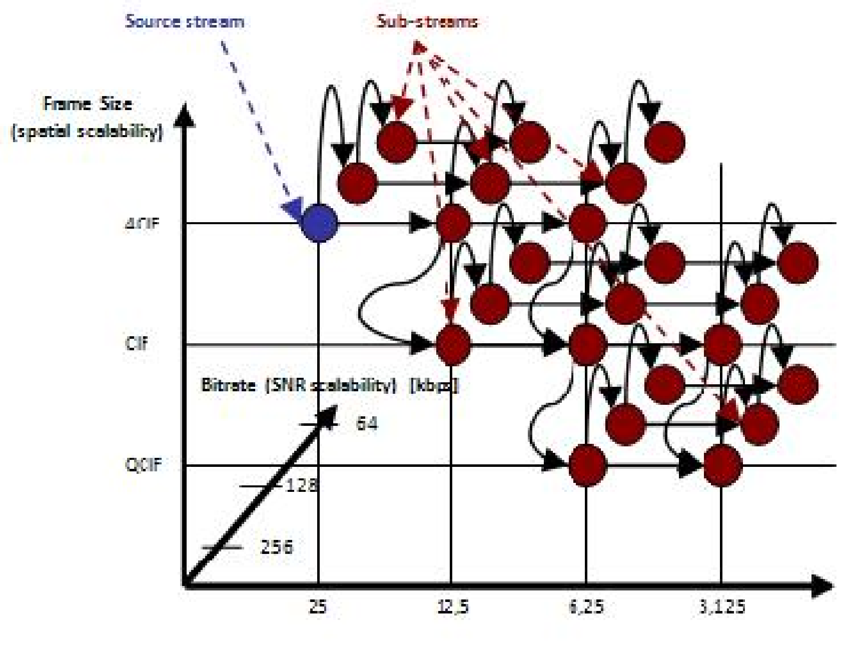

Figure 3.3: Combined scalability with temporal, spatial and quality scalability

The Reference Software JSVM

Besides the document, the JVT team also provides the reference software JSVM [17], which enables the concept-to-approve for the implementations. Fig. 3.4 shows an example encoder structure with two spatial layers.

3.2 Challenges and Opportunities with the SVC

The Challenges of Resource Management for SVC

The flexibility of SVC facilitates the adaptability and the scalability in video processing. However, it also poses many challenges for efficient resource management schemes to maximize the overall user-perceived output quality of the system, given the availability of only a limited amount of resources, like the CPU bandwidths, energy, etc. during an operation/mission. Such performance optimization problems should be carefully analyzed and addressed [19].

Problem Formulation

The notations and assumptions used here are defined as follows. The limited resources which could be examined are Memory (M), processing Time (T), CPU bandwidth (B), maximum Parallelism (P), Energy (E). We consider at the frame level, each frame $f(k)$ is associated with its own frame size s_k , displaying time deadline $tPlay_k$, distortion d_k. Each frame has its own dependencies in the video sequence in temporal t, spatial d, SNR scalability q respectively. For the video sequence, We assume full-pixel accuracy for the motion compensation. we assume that the pixel i in a P-block in P-frame n is predicted from pixel j in frame $n-2$, and pixel i in a B block in frame n is predicted from pixel j_{-1} and j_1 within frames $n-1$ and $n+1$, respectively. However, our investigation does not rely on this assumption.

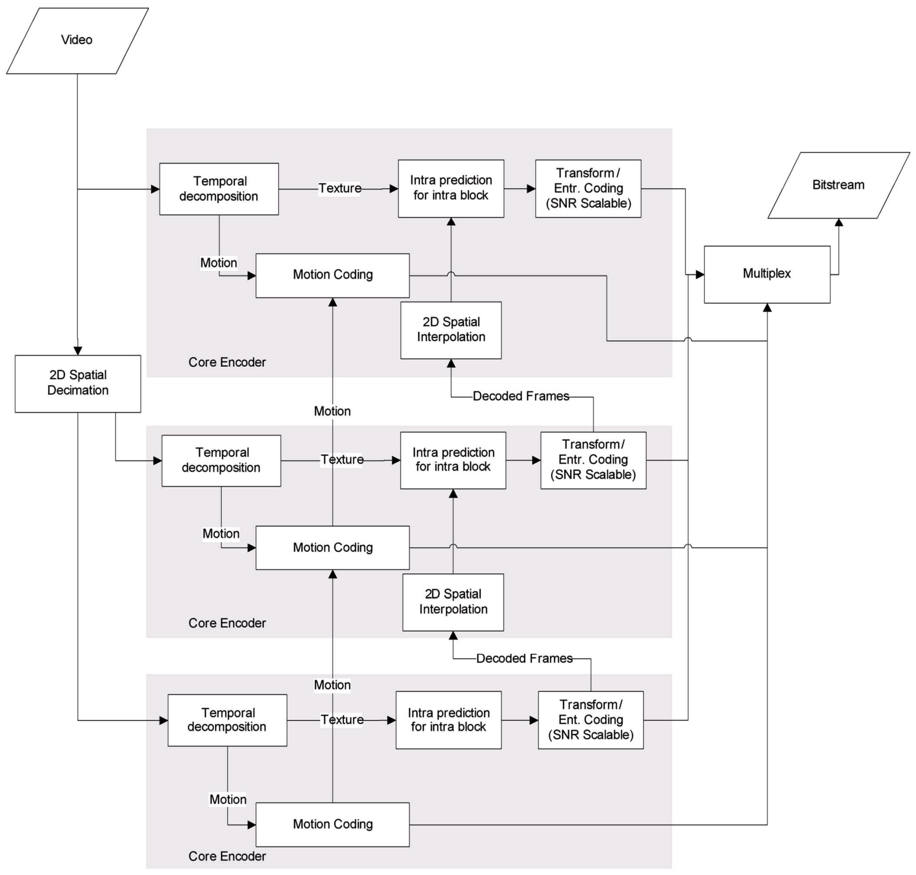

Figure 3.4: SVC reference encoder JSVM structure

The Rate-Distortion problem could be formulated as follows.

> *Given the rate, which could be processing Time T, Memory (M), CPU Bandwidth (B), Energy (E), how to process the bit stream of video sequence S with* $f_1, f_2, ..., f_k, ...$ *for the various scalability layers* (d, t, q) *to minimize the total distortion D of the sequence to achieve the best decoding quality.*

The video quality could be measured in various ways. Here we are using the objective perceptual quality by Mean Square Error (MSE) or Y-PSNR, and the same approach could be extended to the subjective perceptual quality metrics with various human visual system (HVS) in the literature. The variance of the video distortion could be considered as different categories of QoS, such as the high, medium or low QoS levels.

Preliminary Investigation

Generally, there are two approaches for estimating the R-D curve of the video in a specific application: the empirical approach and the analytical approach. The empirical approach is to reconstruct the R-D curve based on interpolating between several sample values of rate and distortion. The analytical approach is to build a mathematical model of the application by analyzing the statistical properties of the video data. The advantage of the empirical approach is that it is generally easy to apply, however, it does not give us much insight into the video coding process and its high computational requirements during streaming typically place a burden on streaming servers. Here, we try some analytical approaches.

Analytical Distortion Modeling for SVC

1. The Distortion Modeling for Temporal Scalability: IDR pictures, key pictures and hierarchal B pictures.

   - For a remaining I-macroblock, the decoder can reconstruct the pixels exactly. Hence,

   $$e(n,i) = \widehat{f}(n,i) - \widetilde{f}(n,i) = 0.$$

   - For a remaining P-macroblock, the decoder adds the received difference signal $\widehat{d}(n,i)$ to the motion compensated prediction signal

   $$\widetilde{f}(n,i) = \widetilde{f}(n-2,j) + \widehat{d}(n,i).$$

   the error at the decoder can be written as

   $$\begin{aligned} e(n,i) &= \widehat{f}(n,i) - \widetilde{f}(n,i) \\ &= (\widehat{f}(n-2,j) + \widehat{d}(n,i)) - (\widetilde{f}(n-2,j) + \widehat{d}(n,i)) \\ &= \widehat{f}(n-2,j) - \widetilde{f}(n-2,j) \\ &= e(n-2,j) \end{aligned}$$

   this implies that no new errors introduced, but past errors propagate.

   - For a remaining B-macroblock, the decoder has

   $$\widetilde{f}(n,i) = (\widetilde{f}(n-1,j_{-1}) + \widetilde{f}(n+1,j_1))/2 + \widehat{d}(n,i).$$

   The error at the decoder can be expressed as

   $$e(n,i) = (e(n-1,j_{-1}) + e(n+1,j_1))/2$$

   Again, no new errors introduced, but errors from the references frames propagate.

   - For a dropped macroblock, the decoder will use the error concealment. We assume the decoder uses the one of the previous frame $\widetilde{f}(n-t)$for error concealment.

   $$\begin{aligned} e(n,i) &= \widehat{f}(n,i) - \widetilde{f}(n-t,k) \\ &= (\widehat{f}(n,i) - \widehat{f}(n-t,k)) - (\widehat{f}(n-t,k) + \widetilde{f}(n-t,k)) \end{aligned}$$

   If we define

   $$e_0(n,i,t) = \widehat{f}(n,i) - \widehat{f}(n-t,k)$$

   then

   $$e(n,i) = e_0(n,i,t) + e(n-t,k)$$

   the second term is the propagation of errors from previous frame errors. the first term is the new error created due to the previous error frame.

   In summary, the MSE error propagation is as follows.

$$e^2(n,i) = \left\{ \begin{array}{cc} 0, & ia \\ e^2(n-2,j), & iia \\ e^2(n-r,j_r), & iiia \\ (e^2(n-1,j_1)+e^2(n+1,j_{-1}))/2, & iva \\ e_0^2(n,i,t)+e^2(n-t,k) & va \end{array} \right\}$$

Here, condition (ia)corresponds to an I-block which is still remaining, condition (iia) corresponds to a P-block which is still remaining, condition (iiia) corresponds to a B block when only one of the reference frame has error, condition (iva) corresponds to a B block when both of the reference frame has errors, condition (va) corresponds to the block was dropped.

In SVC, the temporal scalability is realized by the Hierarchical B pictures. If the frame $f(d,t_i,q)$is dropped, then it will affect the pictures of the lower temporal id and the error will propagate. So the distortion will be

$$D = D_0 + \sum_l D_l$$

where D_l is the affected frames in the length of the temporal dependencies.

2. The Distortion Modeling for Spatial Scalability: In SVC, inter-layer prediction is used for the spatial scalability. the macroblock of $\widehat{f}(d_{higher},t,q)$ could be either Intra encoded or predicted from the base layer $\widehat{f}(d_{base},t,q)$. Suppose there are $s\%$ macroblock predicted from the base layer upsampled with the scale ratio R. For the dyadic spatial scalability, the scale ratio factor is 2. so $\widetilde{f}(d_{high},t,q)$=$s\% \times R \times \widetilde{f}(d_{base},t,q)$+ $\widehat{d}(n,i)$.
   then the errors of the propagation of inter-layer prediction is

$$\begin{aligned} e(d_{high}) &= \widehat{f}(d_{high},t,q) - \widetilde{f}(d_{high},t,q) \\ &= (s\% \times R \times \widehat{f}(d_{base},t,q) + \widehat{d}(n,i)) - (s\% \times R \times \widetilde{f}(d_{base},t,q) + \widehat{d}(n,i)) \\ &= (s\% \times R \times (\widehat{f}(d_{base},t,q) - \widetilde{f}(d_{base},t,q)) \\ &= (s\% \times R \times e(d_{base}) \end{aligned}$$

   Therefore, the error of the base layer propagates. The MSE is

$$e^2(d_{high}) = (s\%)^2 \times R^2 \times e(d_{base})^2$$

3. The Distortion Modeling for SNR Scalability: The SNR scalability in SVC has 3 levels: coarse granularity scalability(CGS), medium granularity scalability(MGS)and fine granularity scalability(FGS). the final adopted are CGS and MGS, while the FGS is considered over-engineered and might be one of topics in the future SVC activites [17]. The SNR scalability is achieved by the progressive refinement quantization and associated coding. The basic progressive refinement quantization principle is illustrated in Fig. 3.5 (from [20]).

$$\left\{ \begin{array}{c} P = Q_0^{-1}(C_0) \\ P = Q_1^{-1}(C_1) + C_0 \\ ... \\ P = Q_n^{-1}(C_n) + C_0 + C_1 + ... + C_{n-1} \end{array} \right.$$

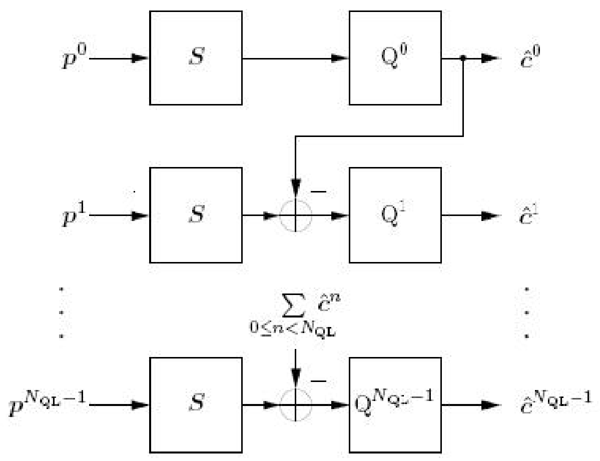

Figure 3.5: Progressive refinement quantization

Therefore, if progressive refinement $f(d, t, q_i)$ is dropped, the corresponding distortion is

$$e(q) = P - P_{loss} = C_i$$

Therefore the MSE is

$$e^2(q) = C_i^2$$

4. The Distortion Modeling for the Combined T/D/Q Scalability: The loss distortion modeling for each scalability have been described as above. The loss distortion of the combined scalability need to be carefully investigated and is still in progress.

Given the above distortion model, the solution of the problem is finding the processing policy vector π that minimizes the expected Lagrangian

$$J(\pi) = D(\pi) + \lambda R(\pi)$$

Work To Do

The current investigation on resource management for SVC is still in a preliminary phrase and much work remain to be done. We indicate the future plan in the following

1. complete the derivation of the loss distortion model for SVC, especially regarding the combined salabilities and the distortion propagation over the dependencies

2. derive the relevant application-specific parameters to allow such processing in a good trade-off among the model's accuracy, complexity, and real-time usability.

Chapter 4

Example Application with Discrete Quality Levels: An MPEG-4 Simple Profile Decoder

4.1 Alternatives for Making a Simple Profile MPEG4 Decoder Adaptive

An example application which can have multiple discrete quality levels is an MPEG video decoder. Different quality levels of a video decoder can e.g. concern the frame rate, the resolution of the decoded image or the number of skipped frames. We are currently working on adding support for multiple quality levels to the Simple Profile MPEG decoder coming with Cal2C [21]. The flattened (or "elaborated") actor network of this decoder is shown in Fig. 4.1.

The MPEG stream is read from a file, which is done by an actor in the I/O-area. The stream data is then transferred via FIFOs to parser actors. The parser actors split the stream into its three Y, U and V components. For these three components three similar subnetworks can be found, each one for processing the respective component. The results then go back via FIFOs to the I/O area where finally a display actor displays the video on the screen. As can be seen, the Simple Profile decoder requires an already quite complex network of actors. This also clearly shows the great potential for parallelization when using the CAL language, since each of the actors could be potentially executed in parallel on a separate core.

The Simple Profile of MPEG-4 doesn't include the features supporting adaptivity present in higher MPEG-4 profiles, e.g. it doesn't support B-, SP- and SI-frames, streams in multiple solutions, feedback to the encoder or other advanced features of higher profiles [22].

Nevertheless we are investigating options on how to make also a Simple Profile MPEG-4 decoder adaptive. There are multiple options, they are introduced in the following sections. To implement these or one of these options, we will add at least one system actor, which announces the service levels to the resource manager. The same actor will also receive the also receive the decisions from the resource manager on which service level to run and "translate" this to tokens which will be sent to the appropriate actors.

Frame Skipping

Quality Aware Frame Skipping (QAFS) [23] is a technique focussed on MPEG2 video streams. The main idea is to extract meta data from the video stream and use

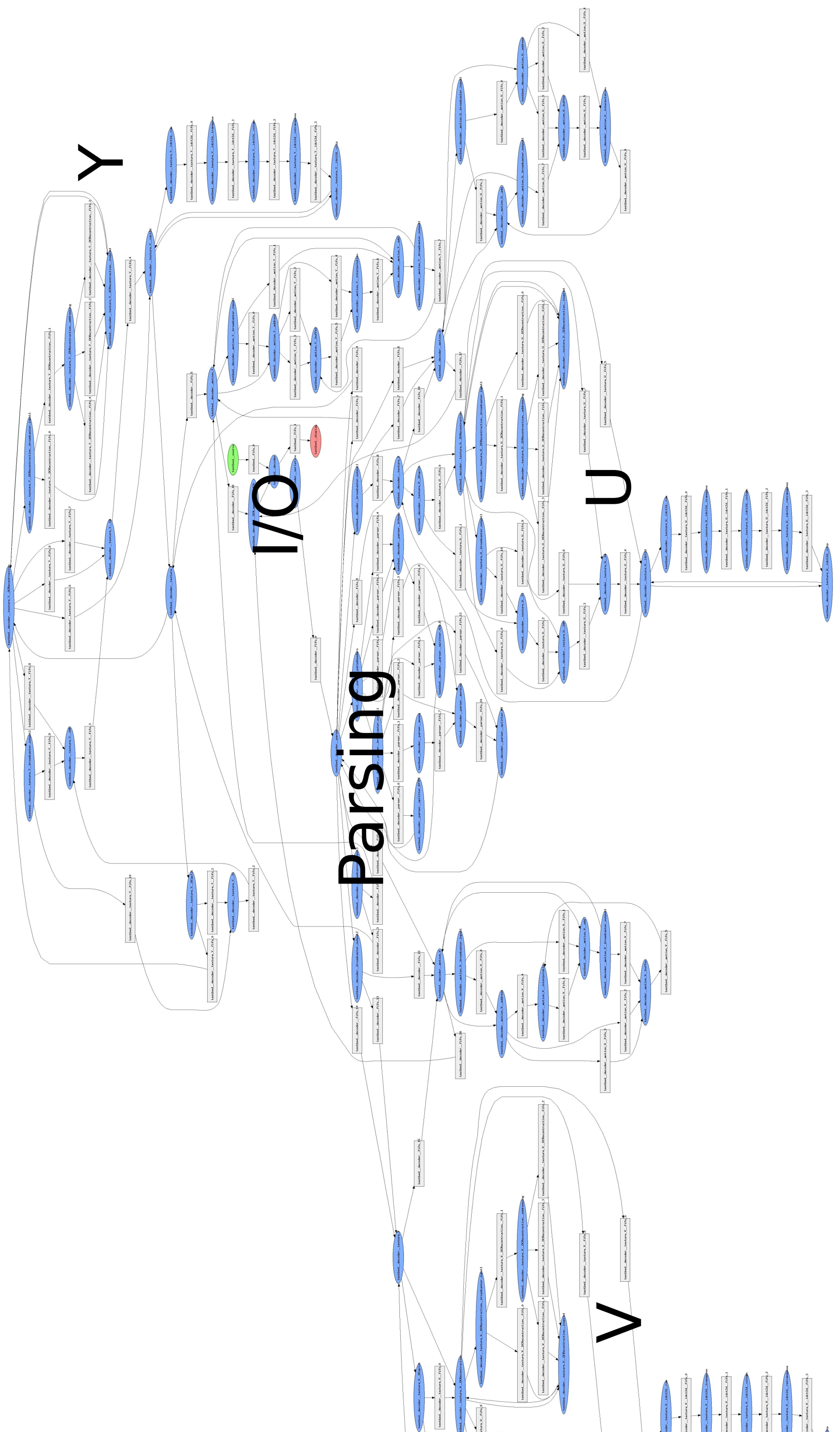

Figure 4.1: The flattened actors network of the Simple Profile MPEG4 decoder

this information to take sensible decision about which frames to skip in the case that not enough CPU time is available to decode all frames. This method requires gathering information from multiple frames, in order to be able to select one or more of those for skipping. In short, a GOP is considered, and then frames which contribute the least to the video quality are discarded first. This means B-frames are the first “victims” of this algorithm, and just after all B-frames in a GOP are discarded, P-frames are considered.

The MPEG decoder from Cal2C doesn’t buffer multiple frames, and since it is a Simple Profile decoder, it also doesn’t support B-frames. This means that the QAFS algorithm cannot simply be applied to this decoder. We are looking into other (simpler) ways of frame skipping to reduce the CPU demands and also output quality of the decoder.

Macro Block Skipping

Alternatively to skipping whole frames we consider skipping the decoding of macro blocks within a frame. An advantage compared to skipping whole frames is that it is not necessary to buffer multiple frames, introduced by the need to buffer a whole GOP in QAFS. If a macro block is skipped, the content from the previous frame will be displayed in the area this macro block encoded. We don’t have any experience with this yet, so we are curious to see how it works, how the performance savings are and how the impact on the perceived quality will be. We envision different strategies for skipping macro blocks. It could be naive, e.g. "skip the first 10 macro blocks", or based on the sizes of the macro blocks, e.g. "skip all macroblocks bigger than X" or based on the position of the macroblocks, e.g. "start skipping macroblock at the edges of the image".

DCT Coefficient Skipping

Another alternative we consider looking into is to use only the lower frequency DCT coefficients when decoding and skip the higher frequency ones. Different Quality levels could be achieved by giving the number of DCT coefficients to use, which should range from 1 to 64. This way 64 quite fine grained quality levels should be achievable. This should save some decoding time but produce an image which is less sharp or has visible square artifacts.

4.2 Assumed Quality Levels of the Simple Profile MPEG4 Decoder

We are not that far with our work yet so that we can define the quality levels which will be available. Also we can’t say yet how much CPU bandwidth requirements the decoder will have. Still we can fit it into a CPU requirements category. It is an MPEG video decoder, so it requires a significant amount of computations for producing a decoded frame, therefore it is not category *Low*. But it is a decoder, not an encoder and it supports only the Simple Profile, obviously there are many applications which will have significantly higher CPU requirements, so it is also not category *High*. So we put it into category *Medium*. We also assume that for this decoder example we will be able to provide the following quality levels:

Index	Quality	CPU bandwidth	CPU time granularity	Max. parallelity
0	100	100	40 ms	16
1	90	80	40 ms	16
2	50	60	40 ms	16
3	10	35	120 ms	16

Table 4.1: Set of quality levels of the Simple Profile MPEG4 decoder. It will be assigned the category *Medium*.

Quality Level 0

The decoder works with maximum quality, nothing is skipped. It has maximum CPU bandwidth demands, i.e. the application cannot consume more CPU time than in this mode. The frame rate is 25 fps, which gives a time granularity of 40 ms. The decoder consists of 73 actors, which potentially can all run in parallel. So we conclude that a high maximum parallelity is given here and set this value to 16.

Quality Level 1

This quality level is not implemented yet. It should consume a bit less CPU than quality level 0 and produce still good quality. The decoding algorithm could be modified like this: all I frames are decoded completely. In all P frames 20 % of the macroblocks are skipped, preferring those at the edges of the frames.

With a modified algorithm like this the quality of the produced video frames will be lower than at level 0. We will define a method how to measure the quality. For now we assume that the quality at this level will be 90 % of the maximum quality. Since for all P frames 20 % of the macro blocks will be skipped, we assume that at this level the decoder will demand around 80 % of the CPU bandwidth required for level 0. Still each frame will be decoded, so the time granularity is also 40 ms. The actors network hasn't changed, so the maximum parallelity stays the same.

Quality Level 2

This quality level also is not implemented yet. Running at this level the decoder should consume significantly less CPU and produce still acceptable quality. A possible algorithm could be : All I frames are decoded completely, in P frames 40 % of the macroblocks are skipped, preferring those at the edges of the frames, additionally always the lowest 16 DCT coefficients are skipped in all P frames.

Again right now we can only guess the resulting quality and CPU bandwidth requirements. As can be seen in Table 4.1 we assume a reduced quality of 50 % and a required CPU bandwidth 60 % of that of level 0.

Quality Level 3

This level is also not implemented yet. We assume that for this level only I frames are decoded. This should save a lot of CPU time but result in a signigicantly reduced video quality. Assuming that the stream consists of at most 1/3 I frames, we guess a quality of only 10 % by a CPU bandwidth demand reduced to 35 % of level 0.

The characteristics of these quality levels are summarized in Table 4.1.

Chapter 5

Example Application with Continuous Quality Levels: A Feedback Controller

5.1 Introduction

As an example of how a CAL application with continuous quality levels can interact with the resource manager a feedback controller is used. The example is intentionally kept very simple in order to emphasize the interaction with the resource manager.

The aim of the control system is to control the position of a rolling ball on a beam. The so called ball and beam process, hence, consists of a horizontal beam and a motor that controls the beam angle. The measured signals from the process are the beam angle relative to the horizontal plane and the position of the ball. The process is shown in Fig. 5.1.

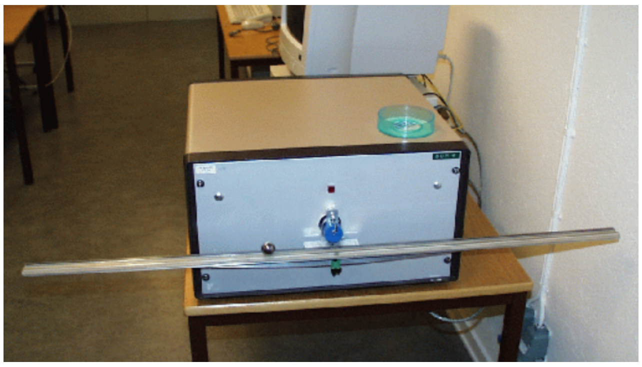

Figure 5.1: The Ball and Beam process

The dynamic model from the motor to the ball position consists of two transfer function blocks connected in series, in which the beam angle appears as an intermediate output signal, see Fig. 5.2. Processes of this type are often controlled by cascade controllers where an inner controller controls the dynamics of the first block and an outer controller controls the overall dynamics. Often PID controller are used both for the inner and outer controllers. The cascade control structure is shown in Fig. 5.3. In cascade structures the output (i.e. control signal) of the outer control loop is used as the reference signal for the inner controller. Cascade

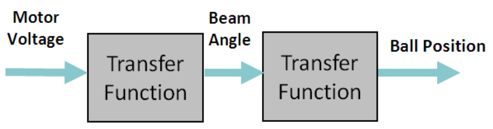

Figure 5.2: Ball and Beam Model Structure

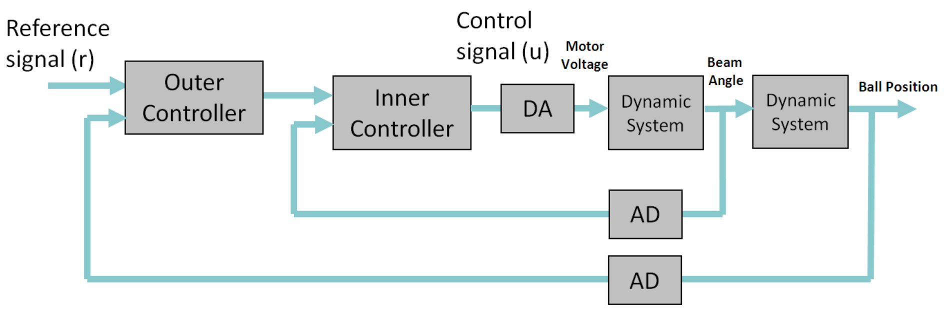

Figure 5.3: Cascade control structure for the Ball and Beam process

control structures, often with more than two layers, are very common in industrial practice, e.g., in automotive combustion engine control.

The PID controller is the most common controller type in industry, the "textbook" version of the PID-controller can be described by the equation

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int^t e(s)\,ds + T_d\,\frac{de(t)}{dt}\right)$$

where the error e is the difference between the reference signal r (the set point) and the process output y (the measured variable). K is the *gain* or *proportional gain*, T_i the *integration time*, and T_d the *derivative time* of the controller.

However, some modifications are necessary in order to get good control performance. A pure derivative cannot, and should not be, implemented, because it will give a very large amplification of measurement noise. The gain of the derivative must thus be limited. This can be done by approximating the transfer function sT_d as follows:

$$sT_d \approx \frac{sT_d}{1 + sT_d/N}$$

The transfer function on the right approximates the derivative well at low frequencies but the gain is limited to N at high frequencies.

It is also advantageous not to let the derivative act on the reference signal and to let only a fraction b of the reference signal participate in the proportional part. The PID-algorithm then becomes

$$U(s) = K\left(bR(s) - Y(s) + \frac{1}{sT_i}\Big(R(s) - Y(s)\Big) - \frac{sT_d}{1 + sT_d/N}\,Y(s)\right) \tag{5.1}$$

where U, R, and Y denote the Laplace transforms of u, r, and y.

A controller with integral action combined with an actuator that saturates can give some undesirable effects. If the control error is so large that the integrator saturates the actuator, the feedback path will be broken, because the actuator will remain saturated even if the process output changes. The integrator, being an unstable system, may then integrate up to a very large value. When the error is finally reduced, the integral may be so large that it takes considerable time until the integral assumes a normal value again. This effect is called *integrator windup*.

There are several ways to avoid integrator windup. A common method for *antiwindup* is illustrated by the block diagram in Figure 5.4. In this system an extra feedback path is provided by using the output of the actuator model and forming an error signal e_s as the difference between the estimated actuator output u and the controller output v and feeding this error back to the integrator through the gain $1/T_t$. The error signal e_s is zero when the actuator is not saturated. When the actuator is saturated the extra feedback path tries to make the error signal e_s equal to zero. This means that the integrator is reset, so that the controller output is at the saturation limit. The integrator is thus reset to an appropriate value with the time constant T_t, which is called the tracking-time constant.

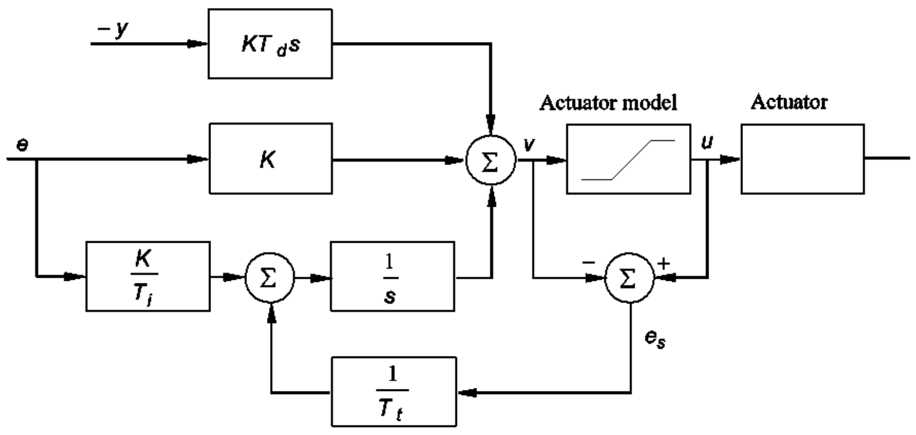

Figure 5.4: PID controller with antiwindup. The actuator output is estimated from a mathematical model of the actuator.

In order to implement a digital PID controller the PID algorithm must be discretized. This can be done in several ways. The following is one common way of doing this. The proportional part

$$P(t) = K\Big(br(t) - y(t)\Big)$$

requires no approximation because it is a purely static part. The integral term

$$I(t) = \frac{K}{T_i}\int^t e(s)\,ds$$

is approximated by a forward approximation, that is,

$$I(kh + h) = I(kh) + \frac{Kh}{T_i}\,e(kh)$$

The derivative part given by

$$\frac{T_d}{N}\,\frac{dD}{dt} + D = -KT_d\,\frac{dy}{dt}$$

is approximated by taking backward differences. This gives

$$D(kh) = \frac{T_d}{T_d + Nh}\,D(kh - h) - \frac{KT_dN}{T_d + Nh}\,\Big(y(kh) - y(kh - h)\Big)$$

The control signal is given as

$$u(kh) = P(kh) + I(kh) + D(kh) \tag{5.2}$$

In order to implement tracking anti-windup the update equation for the integral part is modified slightly. The resulting pseudo-code for a PID controller looks as follows:

```
y = yIn.get();
e = r - y;
D = ad * D - bd * (y - yold);
v = K*(b*r - y) + I + D;
u = sat(v,umax,umin)}
uOut.set(u);
I = I + (K*h/Ti)*e + (h/Tt)*(u - v);
yold = y
```

Here `ad` and `bd` are pre-calculated parameters representing the parameters in the discretization of the derivative part. The saturation function, `sat`, limits `v` between the end values, i.e. it corresponds to the internal actuator model.

A too long delay between the input and the output has a negative effect on control performance. The code above is therefore structured so that the computational delay between the input of the measurement signal (the sampling) and output of the control signal (the actuation) is as small as possible. Therefore, the update of the state variables `I` and `yold` are performed after the actuation. To split the code in two parts, commonly called `CalculateOutput` and `UpdateState` is very common in control implementation. It also influences the execution order in cascade controller structures. In order to minimize the input-output latency for a cascade controller it is important to execute the parts of the algorithm in the following order:

```
Sampling
Outer.CalculateOutput
Inner.CalculateOutput
Actuation
Inner.UpdateState
Outer.UpdateState
```

5.2 Basic CAL Model

A first CAL model of the cascade controller for the Ball and Beam process is shown in Fig. 5.5. The model contains two instances of a PID CAL actor, one clock system actor, two input system actors, and one output system actor. The PID actor contains two actions, one in which the `CalculateOutput` part of the algorithm is performed and one in which the `UpdateState` part is performed. The state variables in the controller and the variables that have to been saved in between the invocations of the two actions are represented as state variables in the actor. In the example we assume that the reference value for the outer controller is constant and contained within the corresponding actor.

The clock actor periodically generates a trigger token. The input and output actors are the interface between the CAL application and the external IO. This interface can be implemented in a variety of ways. The semantics is, however, that, an input actor should produce a current sample each time it receives an input trigger token and that the output actor should send the value contained in the input token to the actuator.

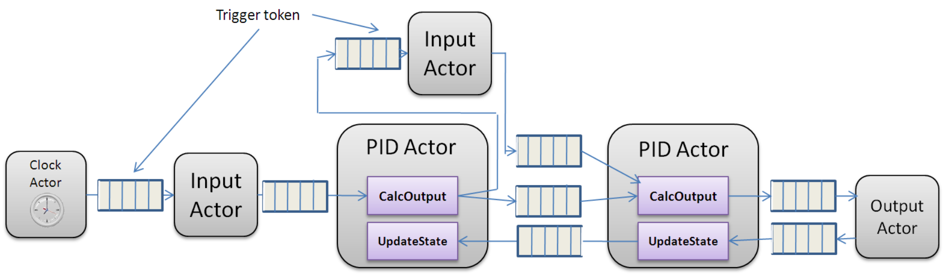

Figure 5.5: CAL Model of the cascade controller.

The PID actors and its actions constitute an SDF model and can, hence, be merged into a statically schedulable region as indicated in Fig. 5.6 by the model compiler. The resulting dataflow graph is shown in Fig. 5.7.

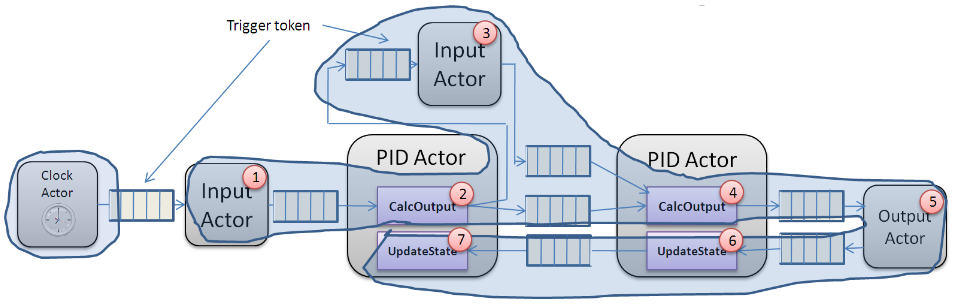

Figure 5.6: CAL Model of the cascade controller with statically schedulable regions.

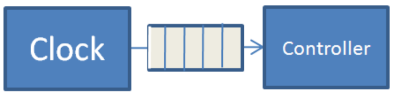

Figure 5.7: The resulting dataflow graph

5.3 The Continuous Quality Function

As described in Deliverable D3a control applications naturally have a continuous mapping from resource usage to quality. A controller is typically designed for a nominal desired sampling period. By increasing the sampling period the consumed computing resources decrease and also the control performance. This relationship between sampling period and performance is often linear or quadratic. If, in addition to changing the sampling period, also the controller parameters are updated in accordance with the new sampling period, the performance decrease becomes smaller.

A natural candidate as a performance measure is a continuous-time quadratic cost function defined as

$$J = \lim_{T\to\infty} \frac{1}{T} \int_0^T \begin{bmatrix} x(t) \\ u(t) \end{bmatrix}^T Q \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} dt$$

where Q is a positive semi-definite matrix, $x(t)$ is the state variable vector, and $u(t)$ is the control signal vector. An advantage with the quadratic cost function definition of control performance is that for a large class of systems it is possible to calculate the expected value of the cost function analytically off-line using, e.g., the Jitterbug tool [24]. The output of this tool would then typically be a graph that relates the control cost (the inverse of the performance) with the sampling period and which in most cases could be well approximated by either a linear or quadratic function.

The cost could be expressed as a function of the sampling period, h either as

$$J(h) = \alpha + \beta h^2$$

or as

$$J(h) = \alpha + \gamma h$$

One could combine this with a maximum value of h for which the achieved performance is considered acceptable, i.e., the resource manager should ensure that the control application always receives at least the corresponding amount of resources.

Instead of representing the quality mapping as a function from sampling period to cost the mapping could be transformed into a mapping between resource requirements and quality. The nominal sampling period would then correspond to the largest resource requirement and the maximum acceptable sampling interval would correspond to the smallest resource requirement.

The cost function can also be used as a quality sensor. In this case the cost is calculated on-line by the control application and compared to the nominal value. However, since the original cost function is expressed in continuous-time this cost function first has to be discretized. The difference between the nominal cost at the current service level and the measured cost can be used as a basis for calculating a happiness value that is periodically sent back to the resource manager.

5.4 Extended CAL model

In order to be practically useful the simple model in Fig. 5.5 has to be extended. For example, it should be possible on-line to change the controller parameters from, e.g., some user interface. This requires that the PID actors are extended with extra inports and actions through which new controller parameters can be sent. To simplify the presentation we will assume that CAL supports structured tokens, i.e. a new set of controller parameters are sent as a single token to the PID actor. When a parameter token is available a separate action, `updatePars`, is triggered that updates the controller parameters. The situation is shown in Fig. 5.8. The same approach could be used to handle on-line changes of the reference signal. Parameter updates take place asynchronously from the execution of the control algorithm. Hence, the `UpdatePars` action cannot be part of the same schedulable region as the other PID actions.

The parameters associated with the quality function would be registered with the resource manager at the initialization of the application by executing a special system actor. This system actor would call the interface methods `registerApp()` and `announceContinuousQualityFunction()` with the associated parameters as arguments. The call to the system actor could either be included in the CAL application or one could use some other means to ensure that this system actor is called

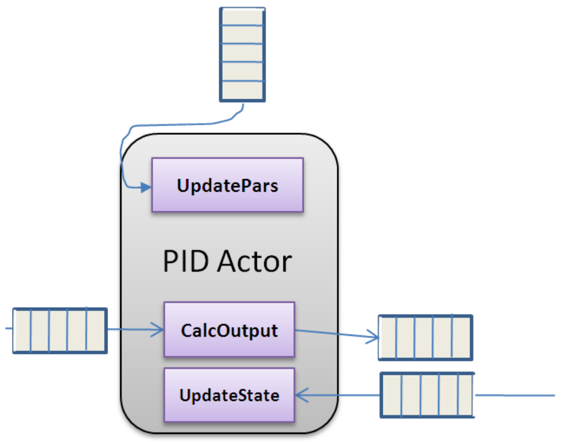

Figure 5.8: PID actor with parameter update action.

once during initialization. An example of where it is included in the CAL model is shown in Fig. 5.9. The One-Shot actor is an ordinary CAL actor that only emits an

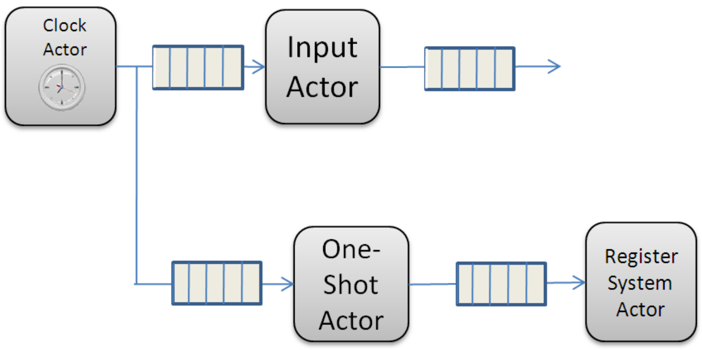

Figure 5.9: System actor for announcing the quality function.

output trigger token the first time it receives an input token. The Register System Actor contains a call to the Resource Manager interface. The service level information that would be transfered to the resource manager for a typical control application are shown in Chapter 5 in D3a, although in that case the continuous quality functions have been discretized.

Once the Resource Manager has distributed the resources it reports back to the application how much resources it has been assigned. This is done through the `changeContinuous()` signal. This signal is catched by a special system actor that listens for this particular signal and when it arrives sends the amount of resources to an ordinary CAL actor that translates the integer amount of resources to the corresponding sampling period and recalculates the controller parameters. The new sampling period is sent to the clock actor and the new parameters are sent to the two PID actors., see Fig. 5.10.

The on-line quality measurement could be implemented by an ordinary CAL actor that based on the control signal and the measurement signal updates the cost function and every n'th time sends the value to another actor where the happiness value is calculated and finally sent over to the resource manager by a special sys-

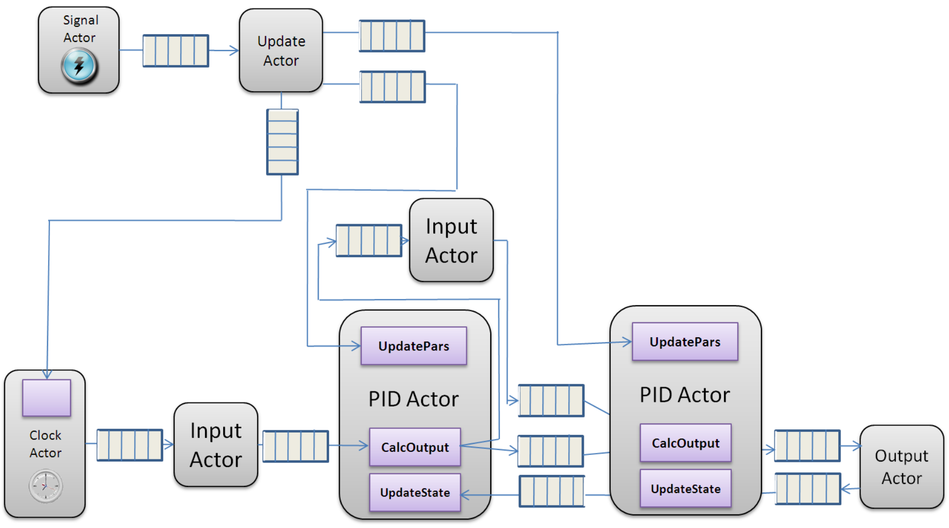

Figure 5.10: Receiving new quality function arguments. The Signal actor is a system actor that receives the new quality level. This is forwarded to the Update actor where the new controller parameters are computed. The new parameter sets are forwarded to the PID actors and the new sampling period is forwarded to the clock actor.

tem actor that internally calls the `reportHappiness()` method in the interface. The approach is illustrated in Fig. 5.11. Not shown in the figure is the connection be-

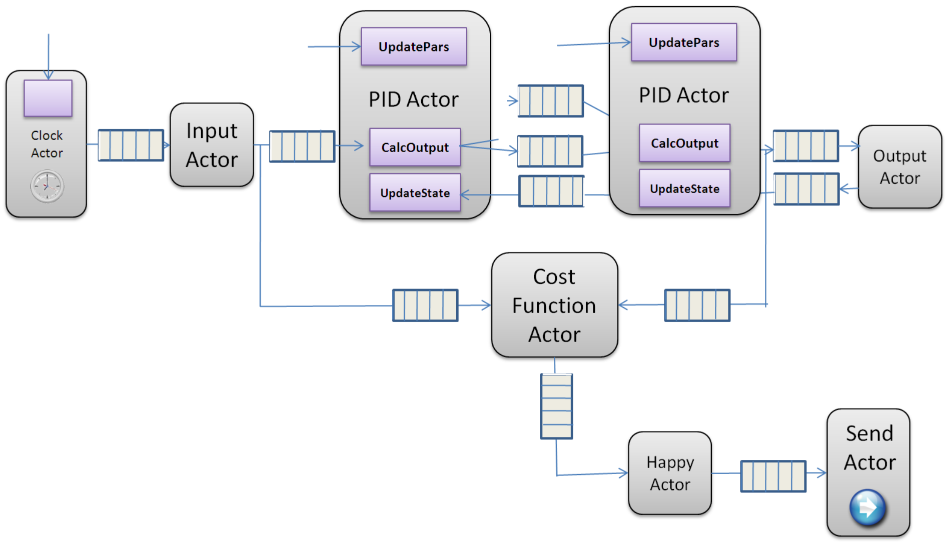

Figure 5.11: Calculation of obtained quality. A cost function actor updates the cost function. In the happy actor this value is converted to a happiness value that is sent to the resource manager in the system actor Send Actor.

tween the Update Actor and the Cost Function Actor where the current sampling interval is propagated. This is necessary in order to be able to compute the cost function correctly.

Appendix A

The Extended D-Bus Introspection Format

A.1 The D-BUS Introspection DTD

Below you can find the current DTD for the D-Bus introspection format. It defines the basic XML format used for D-Bus interfaces.

```
<!-- DTD for D-BUS Introspection data -->
<!-- (C) 2005-02-02 David A. Wheeler; released under the D-BUS licenses,
         GNU GPL version 2 (or greater) and AFL 1.1 (or greater) -->

<!-- see D-BUS specification for documentation -->

<!ELEMENT node (interface*,node*)>
<!ATTLIST node name CDATA #REQUIRED>

<!ELEMENT interface (annotation*,method*,signal*,property*)>
<!ATTLIST interface name CDATA #REQUIRED>

<!ELEMENT method (annotation*,arg*)>
<!ATTLIST method name CDATA #REQUIRED>

<!ELEMENT arg EMPTY>
<!ATTLIST arg name CDATA #IMPLIED>
<!ATTLIST arg type CDATA #REQUIRED>
<!-- Method arguments SHOULD include "direction",
     while signal and error arguments SHOULD not (since there's no point).
     The DTD format can't express that subtlety. -->
<!ATTLIST arg direction (in|out) "in">

<!ELEMENT signal (arg,annotation)>
<!ATTLIST signal name CDATA #REQUIRED>

<!ELEMENT property (annotation)> <!-- AKA "attribute" -->
<!ATTLIST property name CDATA #REQUIRED>
<!ATTLIST property type CDATA #REQUIRED>
<!ATTLIST property access (read|write|readwrite) #REQUIRED>

<!ELEMENT annotation EMPTY> <!-- Generic metadata -->
<!ATTLIST annotation name CDATA #REQUIRED>
<!ATTLIST annotation value CDATA #REQUIRED>
% \end{verbatim}
```

A.2 The Extended D-Bus introspection format

Using the D-Bus DTD it is possible to describe which methods are available and which arguments they support. This is not expressive enough to be able to extract semantics from it, since e.g. the composite type arguments are still anonymous.

E.g. for code generators more information has to be added. The Telepathy [12] project extended the XML format to allow for more information in the interface specification. The basic and extended elements are documented below.

<node name="/org/freedesktop/sample_object"> A node describes a D-Bus object. A D-Bus object has a name and it can support multiple D-Bus interfaces.

<interface name="org.freedesktop.SampleInterface"> A D-Bus interface is similar to a class in C++ or Java. It can contain methods, signals and properties.

<method name="Frobate"> A method is a member function of an interface which can be called by other D-Bus clients. It can take input and output arguments.

<arg name="foo" type="i" direction="in"/> <arg> is a method argument. The attribute direction specifies whether it is an input or output argument. The type attribute specifies the type of the argument. D-Bus supports basic types as int or string and also composite types as struct, array and map. The complete list of types is in Table A.1.

<signal name="changeContinuous"> A <signal> is a message which is sent out by an interface and which can be received by interested other objects on the bus. It can have arguments the same way as methods.

<property name="Bar" type="y" access="readwrite"/> Properties are basically public member data, which can be read and mofified.

<tp:docstring> Encloses HTML-formatted documentation for the parent tag.

<tp:enum name="ResourceId" value-prefix="Resource" type="u"> Allows definition of an enum type, i.e. an integer type with a restricted set of values, which are also named. This is then typically mapped to an unsigned integer type, e.g. UINT32. The attribute value-prefix should be used by code generators as a prefix for all values of this enum.

<tp:enumvalue suffix="None" value="0" /> <tp:enumvalue> is only valid inside enclosing <tp:enum> tags. It defines one of the valid values for this enum type. The attribute suffix will be appended to the value-prefix from the enclosing <tp:enum>.

<tp:mapping name="ResourceDemand"> This defines the name of a map from one type to another type. Used together with <tp:member>

<tp:struct name="QualityLevel" array-name="QualityLevelList"> This gives an struct defined using (...) a name. A separate name for an array of these structs can be specified. It is used together with <tp:member>

<tp:member type="u" tp:type="ResourceId" name="Key"/> When used inside <tp:mapping>, there should be always exactly two, one for the key and one for the value of the map. The names don't matter much then.

When used inside <tp:struct>, they define an individual member of that struct.

The tp:type attribute This attribute can be used everywhere additionally to the type attribute. It allows to define the type in more detail, e.g. to restrict the possible values of an integer to those of an enum, or to give members of a struct names.

Conventional Name	Code	Description
INVALID	NULL	Not a valid type
BYTE	’y’	8 bit unsigned integer
BOOLEAN	’b’	Boolean, 0 is FALSE, 1 is TRUE
INT16	’n’	16 bit signed integer
UINT16	’q’	16 bit unsigned integer
INT32	’i’	32 bit signed integer
UINT32	’u’	32 bit unsigned integer
INT64	’x’	64 bit signed integer
UINT64	’t’	64 bit unsigned integer
DOUBLE	’d’	IEEE 754 double
STRING	’s’	null-terminated UTF-8 string
OBJECT_PATH	’o’	Name of an object instance
SIGNATTURE	’g’	A type signature
ARRAY	’a’	An array of the type which follows
STRUCT	’(...)’	A struct consisting of the enclosed types
VARIANT	’v’	A variant type, the type is part of the value itself
DICT_ENTRY	’{...}’	A map or dictionary mapping from the first enclosed type to the second

Table A.1: D-Bus Type Signatures

There is no “official“ formal specification of these extensions yet, so documents cannot formally be verified for correctness. Due to the deficiencies of the DTD language producing a formal specification for the format of D-Bus interfaces using XML Schema or RelaxNG would help in this regard.

Bibliography

[1] JVT, "Advanced video coding for generic audiovisual services." ITU-T Rec. H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), ITU-T and ISO/IEC JTC 1, Version 7, 2007.

[2] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the h.264/avc standard," *IEEE Trans. on Cir. Sys. Vid. Technol.*, vol. 17, no. 9, pp. 1103–1120, 2007.

[3] Sun Microsystems, Inc., "RFC 1057, RPC: Remote Procedure Call Protocol Specification, Version 2." http://www.ietf.org/rfc/rfc1057.txt, 1988.

[4] Sun Microsystems, Inc., "Java Remote Method Invocation." http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp.

[5] Object Management Group, Inc., "Common Object Request Broker Architecture (CORBA) Specification, Version 3.1." http://www.omg.org/technology/documents/corba_spec_catalog.htm.

[6] H. Pennington, A. Carlsson, and A. Larsson, "D-Bus Specification." http://dbus.freedesktop.org/doc/dbus-specification.html.

[7] "The K Desktop Environment." http://www.kde.org.

[8] "GNOME: The Free Software Desktop Project." http://www.gnome.org.

[9] Free Software Foundation, Inc., "GNU General Public License, version 2." http://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

[10] "The Academic Free License 2.1." http://open2.mirrors-r-us.net/licenses/afl-2.1.php.

[11] D. A. Wheeler, "DTD for D-Bus Introspection data." http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd, 2005.

[12] "Telepathy: Flexible communications framework." http://telepathy.freedesktop.org.

[13] "Specification D-Bus introspect format extensions, Version 0." http://telepathy.freedesktop.org/wiki/DbusSpec#extensions-v0.

[14] D. Isovic, G. Fohler, and L. Steffens, "Some misconceptions about temporal constraints of mpeg-2 video decoding," in *23rd IEEE Real-Time Systems Symposium*, 2003.

[15] H. Schwarz, D. Marpe, and T.Wiegand, "Hierarchical b pictures," *Joint Video Team, Doc. JVT-P014*, July 2005.

[16] H. Schwarz, D. Marpe, and T.Wiegand, “Analysis of hierarchical b-pictures and mctf,” in *Proc. ICME*, (Toronto, Canada), 2006.

[17] J. Reichel, H.Schwarz, and M.Wien, “Joint scalable video model jsvm-12 text,” *Joint Video Team, Doc. JVT-Y202*, Oct. 2007.

[18] H. Schwarz, D. Marpe, and T.Wiegand, “Svc core experiment 2.1:inter-layer prediction of motion and residual data,” *ISO/IEC JTC 1/SC 29/WG 11, Doc. M11043*, July 2004.

[19] C.Wust, “Intelligent control for scalable video processing,” *PhD Thesis, Philips Research Lab, Eindhoven University of Technology*.

[20] T. Rusert and J. R. Ohm, “Backward drift estimation with application to quality layer assignment in h.264/avc based scalable video coding,” *ICASSP, Hawaii, USA.*, 2007.

[21] M. Wipliez, M. Raulet, G. Roquier, J.-F. Nezan, and O. Deforges, “A c-code generation of rvc mpeg4-sp decoder using cal2c.” http://pastel.paristech.org/3899/02/Raulet_CAL2C.pdf.

[22] I. E. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia.* Wiley, 1 ed., August 2003.

[23] D. Isovic and G. Fohler, “Quality aware MPEG-2 stream adaptation in resource constrained systems,” in *16th Euromicro Conference on Real-time Systems (ECRTS 04)*, (Catania, Sicily, Italy), 2004.

[24] A. Cervin and B. Lincoln, “Jitterbug 1.1—Reference manual,” Tech. Rep. ISRN LUTFD2/TFRT--7604--SE, Department of Automatic Control, Lund Institute of Technology, Sweden, Jan. 2003.