Interactive Realtime Multimedia Applications on Service Oriented Infrastructures

ICT FP7-214777

WP 4 and other partners

A Blueprint for Creating Interactive Applications for Virtualized Real-time Service Infrastructures

IRMOS Application Blueprint

Scheduled Delivery: 01.12.2009
Actual Delivery: 02.12.2009
Version 1.0

Project co-funded by the European Commission within the 7th Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Responsible Partner: DTO

Revision history:

Date	Editor	Status	Version	Changes
15.05.2009	Ralf Einhorn	Draft	0.1	initial draft
19.06.2009	Lars Fürst	Draft	0.11	simple example added (chapter 5)
21.07.2009	Ralf Einhorn	Draft	0.2	incorporating partner's comments
23.09.2009	Ralf Einhorn	Draft	0.3	common update
14.10.2009	Ralf Einhorn	Draft	0.4	incorporating partner's contribution
20.10.2009	Ralf Einhorn	Draft	0.5	cleanup for internal review version
22.11.2009	Ralf Einhorn	Draft	0.6	addressed reviewer's comments
01.12.2009	Ralf Einhorn	Draft	0.7	removing internal terms
02.12.2009	Ralf Einhorn	Release	1.0	release version

Authors

Ralf Einhorn, Lars Fürst (DTO), Michael Braitmaier, Dominik Lamp (USTUTT), Kleopatra Konstanteli, George Kousiouris, Andreas Menychtas, Dimosthenis Kyriazis (NTUA), Eduardo Oliveros (TID), Neil Loughran (SINTEF), Bassem I. Nasser (IT Inn)

Internal Reviewers

Dimosthenis Kyriazis (NTUA), Dominik Lamp, Georgina Gallizo (USTUTT)

Acknowledgements

The research leading to these results has received funding from the EC Seventh Framework Programme FP7/2007-2011 under grant agreement n° 214777

More information

The most recent version of this document and all other public deliverables of IRMOS can be found at http://www.irmosproject.eu.

Glossary of Acronyms

Acronym	Definition
AC	Application Component
ACC	Application Client Component
AD	Application Description
ASC	Application Service Component
ASCD	Application Service Component Description
D	Deliverable
EASC	External Application Service Component
EC	European Commission
EE	Execution Environment (ISONI)
fps	frames per second
FS	Framework Services
FSC	Framework Service Controller
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IRMOS	Interactive Realtime Multimedia Applications on Service Oriented Infrastructures
ISONI	Intelligent Service Oriented Network Infrastructure
NFS	Network File System
OS	Operating System (e.g. Linux)
QoS	Quality of Service
SC	Service Component (general)
SOA	Service Oriented Architecture
VM	Virtual Machine
VMU	Virtual Machine Unit
VSN	Virtual Service Network
VSND	Virtual Service Network Description
WEE	Workflow Enactment Engine
WP	Work Package

Table of Contents

List of Figures

List of Tables

1. Introduction

The intention of this document is to deliver a "blueprint", serving as a reference guide (or "cookbook") for application and application component developers. It will provide a guide for engineering applications on the IRMOS Service Oriented Infrastructure. To this direction, it contains definitions and a validation of a methodology and can also be regarded as an advance version of a prospective "IRMOS Methodology White Paper".

In the framework of the IRMOS project a number of useful tools will be developed in order to assist developers during various phases of the application development process. This document gives an overview of these tools and presents a methodology on how to use them.

1.1. About this document

This guide should make clear what an application developer needs to do in order to put her/his application to the IRMOS virtualized service platform. Therefore it can be regarded as a document *for* application adaptors. But of course the experiences *from* application adaptation on how to do so (originating from the demonstrator scenarios) have been used during the creation of this guide.

It should *not* repeat the content of any existing IRMOS deliverables but rather reference them where needed. However, it should provide the whole picture (and hence, partly be a guide through the respective deliverables) and fill potential gaps.

Chapter 2 gives an overview on characteristics of potential IRMOS applications and first generic (i.e. independent of the IRMOS framework) steps to be done for preparation.

Chapter 3 focuses on the steps needed for integration and adaptation to IRMOS and available tools helping the developer on this task.

Chapter 4 dives a bit deeper in technical aspects by describing the interfaces used in these processes functional and descriptive as well as off-line and on-line.

1.2. Basic terminology

The following list contains only the basic terms used in IRMOS along with *brief* descriptions which are needed to ease readers' understanding of this guide. A comprehensive overview on IRMOS terminology can be found in the related deliverables.

- *Application* – the software to run on IRMOS. An IRMOS application consists of application components (ACs) and is described by an *Application Description* (AD).
- *Application component* (AC) – software component actually providing functionality. In IRMOS there are three kinds of ACs:
  - *Application Service Component* (ASC) that runs autonomously and is deployed inside ISONI, e.g. an image renderer.
  - *External ASC* (EASC) – same as ASC but runs outside ISONI because it offers non-standard, "non-virtualizable" functionality, e.g. a GPU processing node.
  - *Application Client Component* (ACC) – the software used by an end-user (client) to access the IRMOS application – always running outside ISONI.

- *Application Service Component Description* (ASCD) – a document describing everything that is needed (e.g. resource needs and functional parameters) to run the ASC on IRMOS.
- *Workflow* – Application with temporal constraints (e.g. certain ASCs of an application may appear and disappear during execution).
- *Framework service* (FS) – offer meta-functionality for administration – the glue in between ACs, ISONI and the different actors.
- *Intelligent Service Oriented Network Infrastructure* (ISONI) – the virtualised resource infrastructure used for executing IRMOS applications.
- *IRMOS* – the whole framework including ACs, FSs, ISONI.

2. The Foundation: An Application

Basically there are two possibilities to create an application that will run on the IRMOS platform.

- Pre-existing applications can be adapted to run on IRMOS ("adaptation").
- New applications developed tailored to the IRMOS platform ("green field development").

Both cases are valid and are examined in IRMOS: the virtual augmented reality application is focussed on adaptation, while the post production and e-learning ones contain development from scratch. [1]

Instead of running on dedicated physical hosts, the service parts of the applications run on Virtual Machine Units (VMUs) within ISONI.

2.1. Appropriate Applications

In principle, all server applications are suitable to be executed in the IRMOS environment as they are, without any kind of adaptation or modification. But the concrete characteristics of IRMOS, mainly focused on providing infrastructure with QoS guarantees, make applications that require live interactions among their users or which are infrastructure demanding (in terms of CPU, network or storage usage) more appropriate for IRMOS.

Applications that require scaling resources or have changing use patterns over time require adaptation of the infrastructure according to the specific usage periods and reservation of more or less resources depending on the load in each moment.

In other cases, there are applications that not only have important requirements in terms of resource usage and depend on reliable resources but also need QoS guarantees that no other platform offers with the level of customization available in IRMOS.

IRMOS platform is considered to be suitable for the aforementioned applications by allowing both for interactivity but also for QoS guarantees across a virtualized infrastructure. The IRMOS design provides certain advantages in relation to different aspects related with security, as for instance: the isolation during execution from other applications (that caused by misbehaviour or malicious software) could try to monopolise all resources in the machine, damaging other applications executing over the same physical machine. There is also a complete isolation at network level that prevents applications (like sniffers) running inside IRMOS to access data of other surrounding applications. One of the fundamental aspects of IRMOS is the ability of providing storage with quality characteristics, beyond size space or limited network traffic; users can specify more demanding requirements like throughput or latency in the access. Moreover, IRMOS allows the concurrent access of the storage by different components of an application or by different applications at the same time.

2.2. Preparation

In this section we describe the steps that are required in order to prepare an application to be executed on IRMOS platform (see 1.2 for terminology). These steps are the following:

- *Step 1 - Application Components Creation:* The application has to be split into application components (ACs) at least separating the user interface part (ACC) from the computing part(s) (ASC(s)). This can also be done by adding appropriate interfaces and is independent of the IRMOS framework. For partitioning think of each component running on a different physical host. Think of this separation as a transformation of the application from a monolithic one towards a distributed program that is able to run on distributed systems [3], [4]. *How* the application is separated depends on the specific application logic and what should be achieved. While it may be convenient for one application to be split in a large number of modules as the module specific tasks have a high degree of reusability (think of the topic of componentized development and service-oriented applications), it might be different for another application that has specific subtasks which have to be componentized and put on, e.g. highly demanding computational resources. The conclusion is that "componentizing" your application is the way to unlock the power of the IRMOS platform by making use of scalable QoS-guaranteed resources in an "on-demand"-fashion.
- *Step 2 - Components Interfaces Creation:* Components must be equipped with interfaces to the IRMOS framework:
  - ASCs need a run-time interface to be configured, controlled and monitored by the framework. The purpose of this interface is to act as a mediator between the AC and the IRMOS framework services, represented on the application component's node by an IRMOS framework service that gathers the aforementioned information. However the amount of information might vary according to the information made available by the AC. The more information an AC reveals to the IRMOS framework the better IRMOS can deal with QoS issues for that component.
  - ACC need to be started with specific parameters required for accessing the application service running on IRMOS. The configuration information provides the necessary details to allow the ACC to access the various components (ASCs) on the IRMOS platform that need to have direct connections to an ACC.
- *Step 3 - Components Packaging:* ASCs must be packaged for deployment.
- *Step 4 - Components Description:* ASCs must be described, especially resource-wise.
- *Step 5 - Application Description:* The application (as a whole) must be described.

The latter steps (Steps 4 and 5) are facilitated through various tools provided by IRMOS and modular approaches e.g. for wrapping ACs. These tools are described in section 3.4 of this document.

Figure 2-1 shows a typical IRMOS application. The application runs distributed on different hosts whereas the Intelligent Service Oriented Network Infrastructure (ISONI) as the execution environment used in IRMOS behaves transparently.

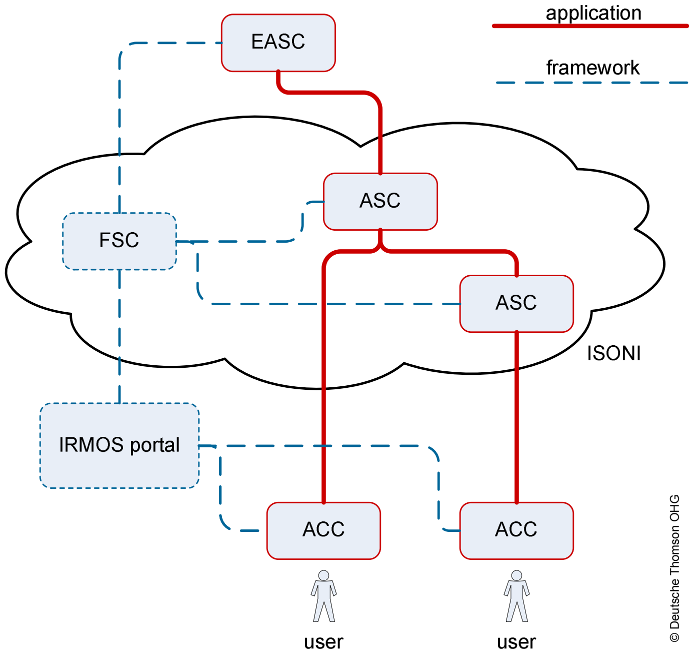

Figure 2-1 Sample IRMOS Application

Application related communication and data transfer like passing image data from one component to another is done independently from IRMOS (however its parameters have to be described for resource reservation). Framework related communication e.g. for controlling and monitoring ASCs is limited to the minimum (for having as little impact as possible on the actual application).

3. From ACs to an IRMOS Application

As described in section 2, an IRMOS application is based on application components (ACs). Figure 3-1 depicts the phases of an AC. This chapter goes through the phases relevant for the application component developer as well as the application designer (the two upper boxes). However processes happening during the "use" phase like ASC configuration and execution have to be taken into account for creating ASCs.

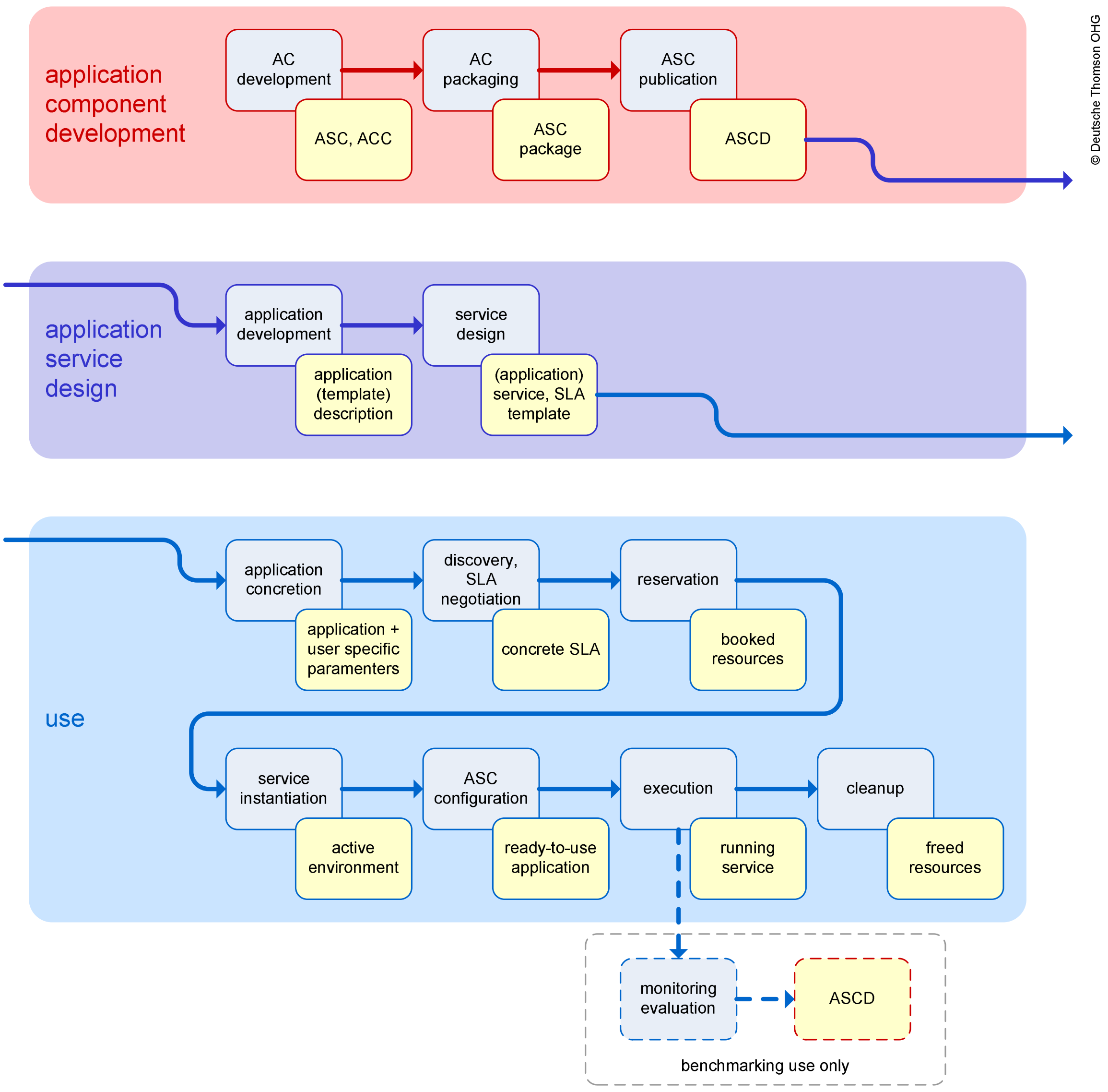

Figure 3-1 IRMOS Application Phases

Figure 3-2 shows three concrete use cases for the application phases depicted above:

1. Each ASC and each IRMOS application has to be created off-line (related to the actual use).
2. The applications are set up by assembling ("interconnecting") ACs resulting in an application description (AD).
3. Finally these applications can be used multiple times ("on-line").

For the AC developer we will focus on the first use case which is however the most complex as it comprises all phases. Ideally everything will be done by the Framework Services by the beginning of the "(benchmark) use".

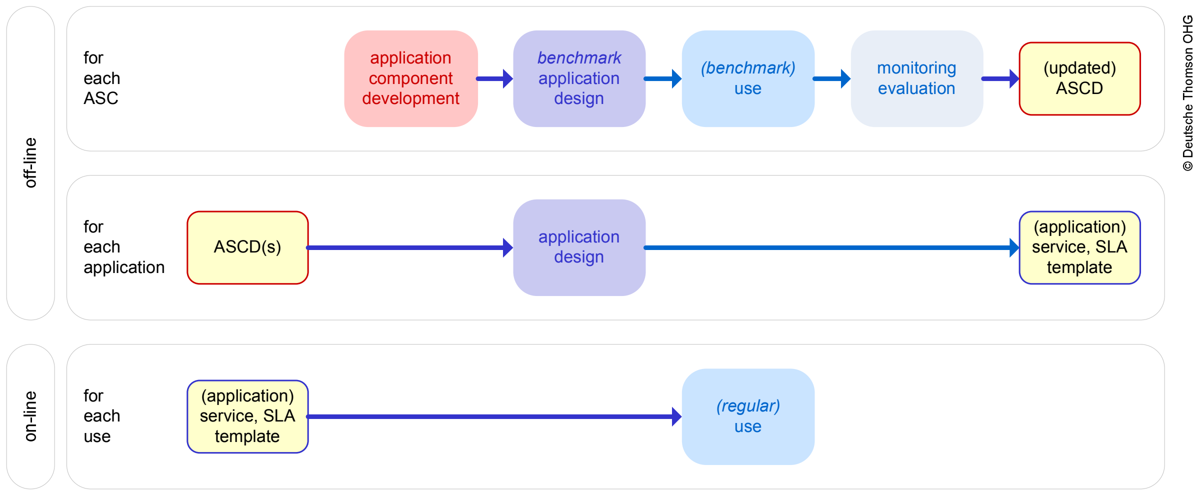

Figure 3-2 Different Use Cases of the Application Phases

3.1. AC Development

The first step in the process of adapting an application to run on the IRMOS platform refers to the development of the application components. Ideally the created ACs may also be used in a non-IRMOS environment. Communication between the ACs (including control and pay load data transfer) must be implemented – if not already in place.

Splitting up a monolithic application into components can also be realized by keeping the application core as it is but rather adding functionality (e.g. interfaces) which enables the core to behave as a specific component with dedicated functionality. Hence the same binary can be used e.g. as a GUI as well as a service part.

(E)ASCs must be able to run without manual interaction. They are started and controlled similar to UNIX-like services. More details regarding this on-line interface are described in section 4.2.

The application level adaptation for ACCs is similar to ASCs. For seamless integration an ACC should be prepared to be started via an IRMOS ACC launcher which means that the current configuration can be passed to the ACC upon start-up. For details see chapter 4.3.

3.2. AC Packaging

Like any other software ACs have to be packaged for deployment.

- EASCs are ACCs deployed independent of the IRMOS EE. Therefore the packages may be of any format and are not further discussed here.
- ASCs are to be deployed in the IRMOS' ISONI EE – in defined package format(s).

In the current state the ASC developer can expect to find a basic Fedora Core 9 operating system. The following formats are accepted for deployment.

The operating system is provided as a turn key solution by the ISONI layer of IRMOS. Basically, a customized base installation is generated, i.e., parameters like IP address, hostname, etc. are set to customer-requested values.

The ASC to be deployed inside the VMU has to be provided as a standard .rpm package. Particularly, the dependencies have to be specified completely. The standard mechanisms for recursive dependencies apply. All packages that are included in the standard Fedora repository can be specified as dependency and are automatically installed.

The ASC .rpm and all non-standard .rpm files that it depends on are wrapped in a .zip container and uploaded to ISONI.

3.3. ASC Publication

ASCs must be published to the IRMOS framework which includes the following two actions:

1. Publish a description of the ASC to the IRMOS Provider domain. This task implies that the ASCD has already been created (cf. section 4.1). This ASCD is then uploaded to the ASC repository, a dedicated repository for storing data related to the ASCs. Note that this procedure only applies to ASCs, since ACCs are deployed separately (i.e. independent form the IRMOS framework). For EASCs an ASCD is needed as well, but deployment takes place framework independent.
2. The ASC publication process also includes the publication of the corresponding ASC binary to the IRMOS ASC repository. Among several other items the ASCD also contains a link to the binary ASC package.

The above two processes take place at the same step as depicted in Steps 1 to 3 in the publication sequence diagram (Figure 3-3). The Application Provider uploads a directory of predefined structure that includes both the ASCD as well as the binary, therefore the link to the binary ASC package that each ASCD carries, is actually a relative path pointing to the location of the binary inside its directory.

These steps are repeated for each ASC and each time the Application Provider obtains a link to the specific location where the information about each ASC is stored in the ASC repository.

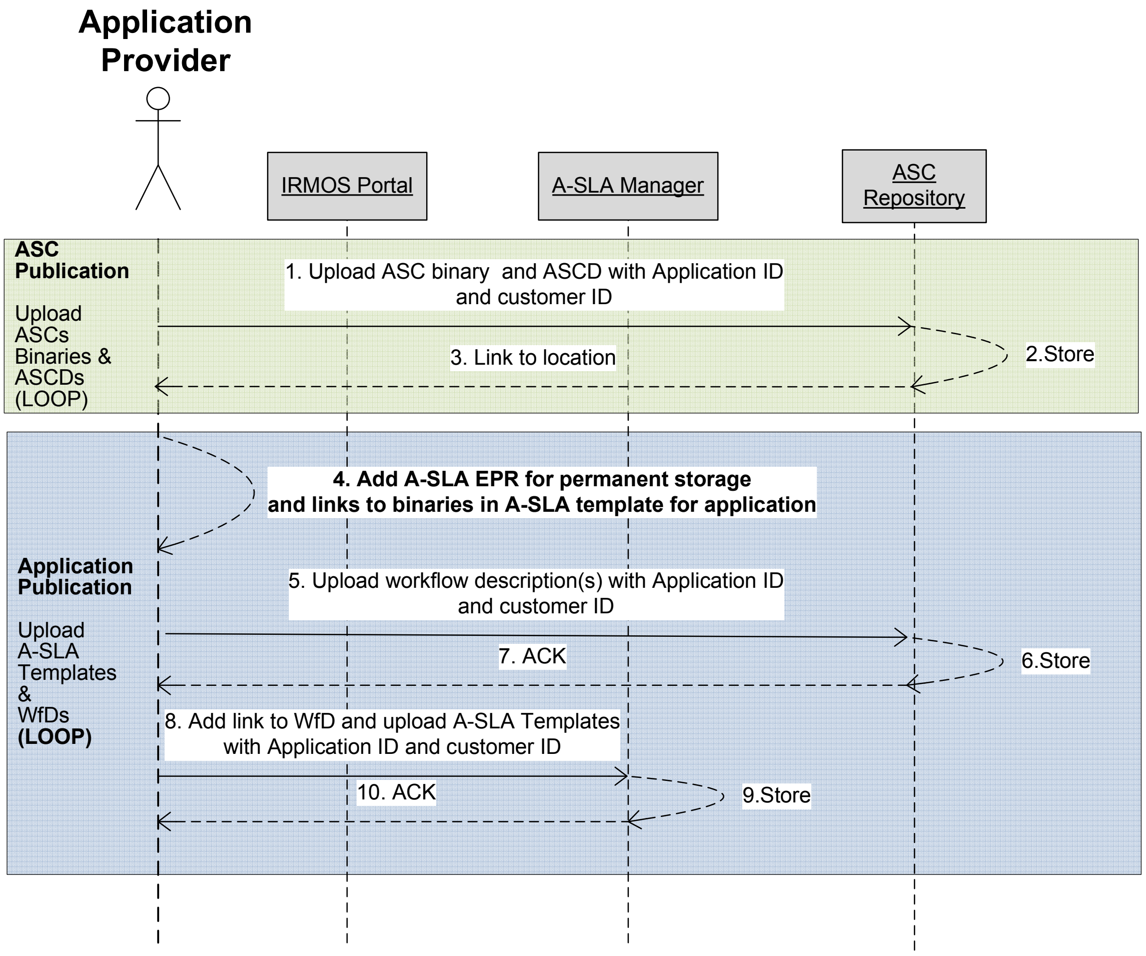
Application Provider
IRMOS Portal
A-SLA Manager
ASC Repository
ASC Publication
Upload ASCs Binaries & ASCDs (LOOP)
1. Upload ASC binary and ASCD with Application ID and customer ID
2.Store
3. Link to location
4. Add A-SLA EPR for permanent storage and links to binaries in A-SLA template for application
Application Publication
Upload A-SLA Templates & WfDs (LOOP)
5. Upload workflow description(s) with Application ID and customer ID
6.Store
7. ACK
8. Add link to WfD and upload A-SLA Templates with Application ID and customer ID
9.Store
10. ACK

Figure 3-3 ASC and Application Publication Sequence

3.4. Application Development

When creating an application the developer uses the IRMOS Papyrus[1] bundle which contains a profile for modelling ASCs. As there is a lot of detail involved in this we will only describe the process from a high level point of view.

It should be noted that while we refer to the Papyrus tool in developing applications the approach is generic and thus the profile and process can be utilised in any UML design tool which supports the UML2 meta-model.

Similar to single components the entire application(s) derived from the components needs to be described. This means that components are selected and virtually interconnected. Several values for high level parameters may be pre-selected or its ranges may be restricted. However an application description (AD) remains a template where several parameter values can be selected by the user. As applications are derived from ASCs there parameters (the ones visible from outside) are also taken from the corresponding ASCDs.

[1] Papyrus is an open source graphical UML modelling tool; see http://www.papyrusuml.org. A bundle containing Papyrus as well as IRMOS specific extensions has been created.

There are several elements involved in creating an application:

1. Creation of ASC descriptions (using UML classes)
2. Creation of instances of these descriptions (using UML class instance specification)
3. Creation of links between instances (using UML composite structure diagram)
4. Creation of a workflow which describes the interaction between the ASC instances (using UML activity diagram)
5. Creation of an instance of the application (this is essentially an instance of all the above)

In the following sub-sections we will briefly cover these different elements.

3.4.1. ASC descriptions

An ASCD can be described as the development of a set of properties which are specific to the domain in which the ASCD is to be used. For example in the film domain the developer may wish to describe parameters relating to video, production schedules and so on, while in another domain the set of parameters can be entirely different. However, each ASCD also includes a set of common parameters which are part of the ASCD profile (e.g. benchmarking properties, etc.). Therefore, the developer will typically set these values with concrete values in this description using the properties window. The descriptions are developed using UML class diagrams with the UML4ASCD profile. Figure 3-4 illustrates a typical ASCD for a `SimpleImageReszier` application. It simply defines a class which has the ASCD profile assigned to it. Within an ASCD a number of properties are described which are also stereotyped with elements from the ASCD4UML profile (e.g. `<<aSCParameter>>`, `<<benchmark>>`, etc.).

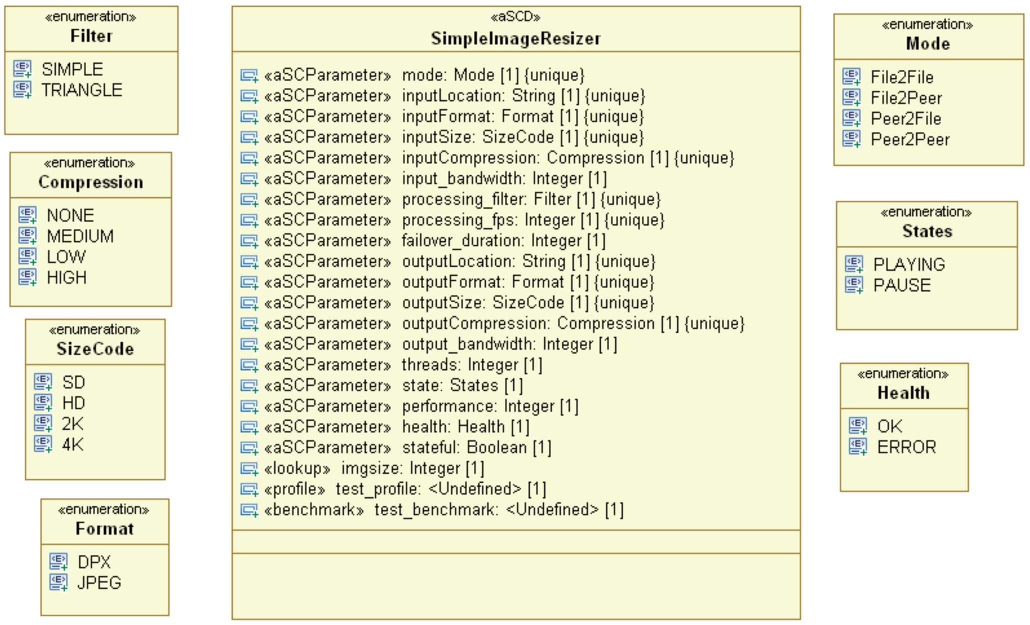
«enumeration»
Filter
SIMPLE
TRIANGLE
«enumeration»
Compression
NONE
MEDIUM
LOW
HIGH
«enumeration»
SizeCode
SD
HD
2K
4K
«enumeration»
Format
DPX
JPEG
«aSCD»
SimpleImageResizer
«aSCParameter» mode: Mode [1] {unique}
«aSCParameter» inputLocation: String [1] {unique}
«aSCParameter» inputFormat: Format [1] {unique}
«aSCParameter» inputSize: SizeCode [1] {unique}
«aSCParameter» inputCompression: Compression [1] {unique}
«aSCParameter» input_bandwidth: Integer [1]
«aSCParameter» processing_filter: Filter [1] {unique}
«aSCParameter» processing_fps: Integer [1] {unique}
«aSCParameter» failover_duration: Integer [1]
«aSCParameter» outputLocation: String [1] {unique}
«aSCParameter» outputFormat: Format [1] {unique}
«aSCParameter» outputSize: SizeCode [1] {unique}
«aSCParameter» outputCompression: Compression [1] {unique}
«aSCParameter» output_bandwidth: Integer [1]
«aSCParameter» threads: Integer [1]
«aSCParameter» state: States [1]
«aSCParameter» performance: Integer [1]
«aSCParameter» health: Health [1]
«aSCParameter» stateful: Boolean [1]
«lookup» imgsize: Integer [1]
«profile» test_profile: <Undefined> [1]
«benchmark» test_benchmark: <Undefined> [1]
«enumeration»
Mode
File2File
File2Peer
Peer2File
Peer2Peer
«enumeration»
States
PLAYING
PAUSE
«enumeration»
Health
OK
ERROR

Figure 3-4 Example ASC Description

3.4.2. ASCD Instances

ASCD instances are essentially developed using UML class instance specification diagrams. Using these diagrams it is possible to develop instances which are typed based on the names of the ASCD descriptions. Figure 3-5 illustrates a class instance which uses the aforementioned ASCD description. The instance simply provides a concrete value to the mode property. For purpose of brevity we only show one property with a value although the instance would typically contain several properties and assigned values.

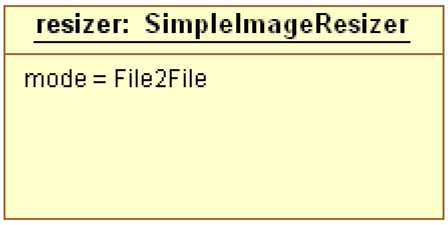

Figure 3-5 Example of ASC Instance

3.4.3. Linking ASC Instances

An application description is typically made up of several interconnecting ASCD instances. In order to model this we use the UML composite structure diagram. Composite structure diagrams are used for modelling the static structure of the application, i.e. the components involved and how they are connected. Figure 3-6 illustrates the previous resizer instance of `SimpleImageResizer` connecting with other ASCDs.

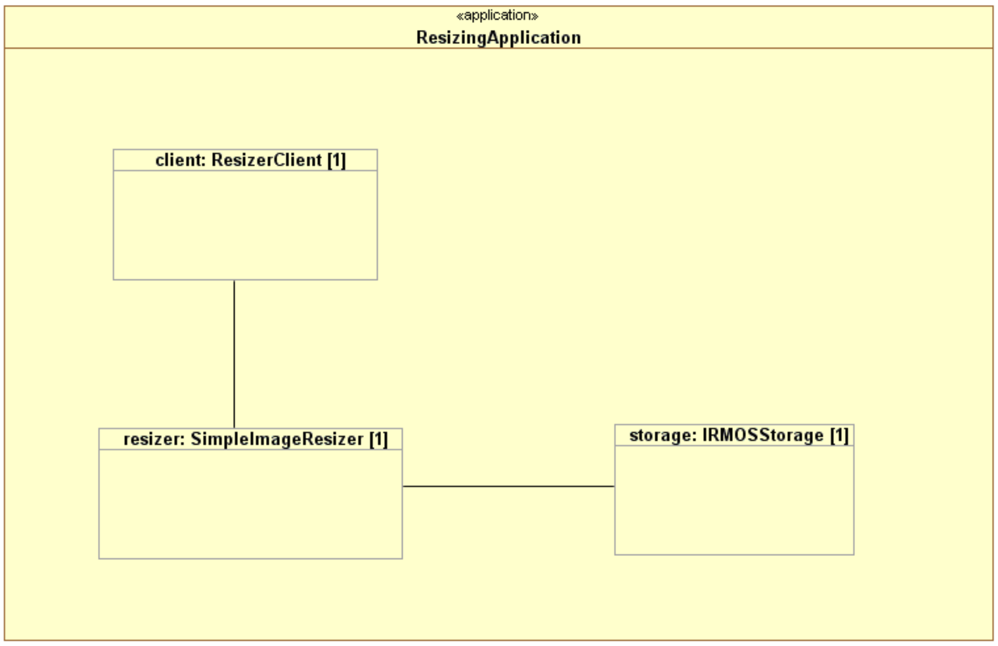
«application»
ResizingApplication
client: ResizerClient [1]
resizer: SimpleImageResizer [1]
storage: IRMOSStorage [1]

Figure 3-6 Example of linking ASC Instances

3.4.4. Workflow

An application description contains a workflow describing how the different ASCD instances interact with one another. In IRMOS the workflow of an application is defined using a UML activity diagram. UML allows for classes, such as the `ResizingApplication`, to have attached behaviours in different forms one of which is UML activities.

3.4.5. Application Instance

The previous chapter sub-sections illustrated how to define an application composed of components. This gives the structure and the types of the application, but it does not provide the actual configurations for the different ASCDs. As mentioned these configurations are in the ASC instances. So the task is to fill an instance of the application class with the ASCD instances. To do this we use the UML composite structure diagram and draw some 'slots' then assign them with ASCD instances as shown in Figure 3-7.

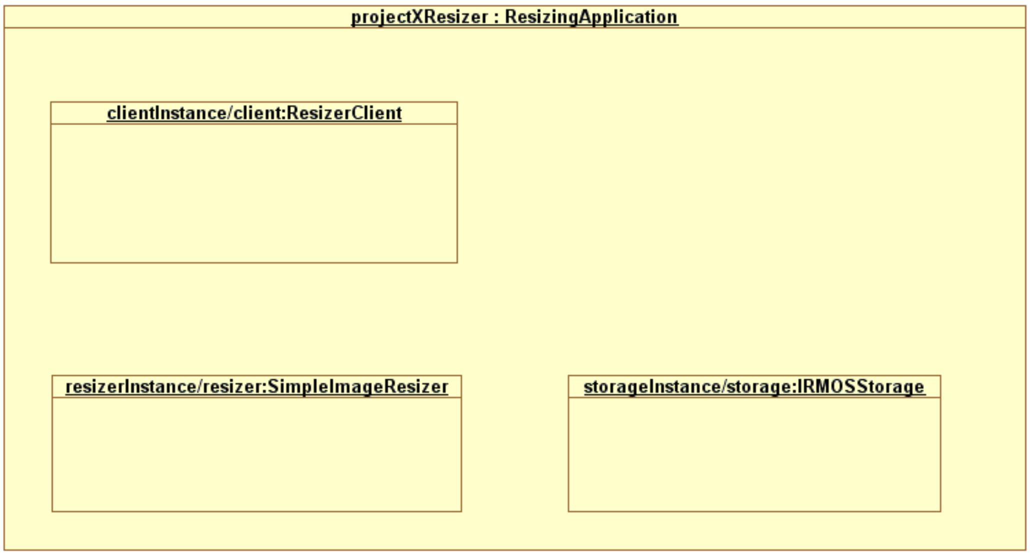
projectXResizer : ResizingApplication
clientInstance/client:ResizerClient
resizerInstance/resizer:SimpleImageResizer
storageInstance/storage:IRMOSStorage

Figure 3-7 Application Instance Example

3.5. Application Publication

Having published the ASCs (Section 3.3) and created the A-SLA templates that correspond to different ways of using the same application (different workflow and/or different level of QoS), the Application Provider publishes these A-SLA templates to the IRMOS Provider domain by uploading the corresponding descriptive files to the dedicated repository (via the A-SLA Manager), as depicted at steps 4 to 10 in the publication sequence diagram (Figure 3-3). It should be stressed at this point that each A-SLA template builds heavily on the ASCDs and also includes a link to the workflow description of the application it represents as well as a reference to the permanent storage that is shared by all customers of the application. In order to be coherent with what is described in the ASC publication, the workflow description could also be stored inside the application's corresponding folder in the ASC repository.

4. The Interfaces

Although the interfaces are an essential part of the ASC development (i.e. they are used during development) they and their usage are described separately in order to keep chapter 3 more compact. Figure 4-1 shows the different AC-relevant interfaces both descriptive and functional.

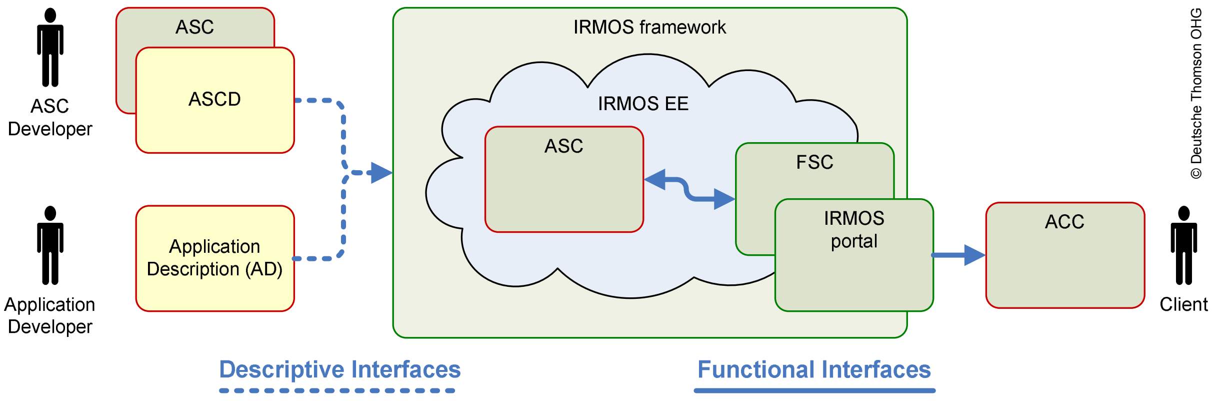

Figure 4-1 IRMOS Interfaces

4.1. Creating the ASCD

An ASCD thoroughly describes an ASC and represents the only 'off-line' interface between the application and the IRMOS framework. Hence it is an essential document containing the entire set of information letting IRMOS use the ASC regarding resource reservation, configuration, control, monitoring, benchmarking and modelling. The ASCD (content, purpose etc.) is described in [1].

An ASCD is built using a UML design tool and the associated ASCD4UML profile (Figure 4-2). For simplicity in IRMOS we have chosen to use this profile within the Papyrus UML design tool as this is accessible to everyone due to its open source nature. However, as UML is an adopted standard there is nothing to stop the developer using any UML design tool of their choice along with the previously mentioned ASCD4UML profile. The profile is purposely designed to be easy to use within any UML design tool such as Papyrus (which is the tool used in this guide). The profile consists of a number of different stereotypes, as illustrated in Table 4-1.

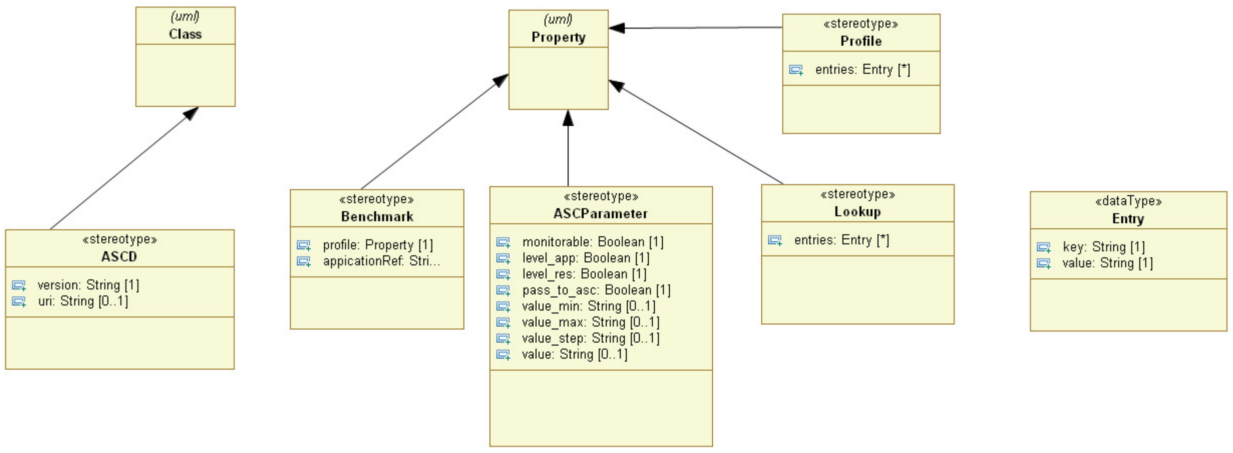

Figure 4-2 UML profile for ASCD

Stereotype	Description
`<<aSCD>>`	Identifies a class as being an ASCD
`<<aSCParameter>>`	Identifies a property as being an ASC parameter within an ASCD
`<<lookup>>`	Identifies a property as a lookup table within an ACSD
`<<profile>>`	Identifies a property as profile within and ASCD
`<<benchmark>>`	Identifies a property as a benchmark within an ASCD

Table 4-1 Description of UML4ASCD stereotypes

The main idea is to allow different stereotypes to be applied to specific properties we want to define in our intended application. When defining an ASCD, as described in Section 3.4.1, we can also provide values to different properties which are contained within the profile elements.

4.1.1. Defining Resources on high and low Level

One of the main purposes of the ASCD is to let the FS generate a VSND (i.e. a description of the required resources) for executing an ASC on the ISONI EE. The VSND requires values for low level parameters (e.g. "CPU frequency") completely independent of a specific application. Therefore application ("high") level parameters values (e.g. "frames per second") must be translated ("mapped") to low level ones.

The ASC developer has to select which low level parameters must be provided for her/his ASC to run as well as the high level parameter for configuration of the ASC – which also might have an impact on performance.

Basically two tables of parameters have to be created.

- The high level parameter table contains all parameter inputs on application level. These parameters must contain the range of the possible values of the parameter or an enumeration of all possible values that these parameters may take if they have discrete values. It is best that these values are numerically represented, e.g. possible resolutions which have discrete allowed values (800x600, 1024x768 etc.) can be added as numbers of pixels (480000 pixels, 786432 pixels).

- The low level parameter table contains all parameters needed for executing the ASC on the ISONI EE without any actual values (as these are calculated by the FS). If these low level parameters are not provided by the default ISONI monitoring infrastructure, as it is the case in application specific parameters, the ASC must provide internal mechanisms for self-monitoring of the latter. These monitored parameters are not obligatory for the execution of the ASC, but they are needed in the case that they are inserted as a QoS metric in the ASLA.

Additionally parameters may be marked as `pass_to_asc` to let the FS know that the parameter along with its current value has to be passed to the ASC upon configuration (see 4.2), e.g. `compression_type` or `peer_ip`.

The ASC developer must specify what inputs he wants to map with what outputs. A flag in the ASCD can be used for this purpose. These outputs can then be used in the ASLA for determining the QoS levels of the ASC or the application in general.

4.1.2. Defining functional Parameters

Besides parameters relevant for resource dimensionality there are a couple of parameters necessary for the ASC to run, e.g. an IP address of a peer node. Some parameters may not be relevant to the FS at all but just have to be passed (e.g. an operation mode of the ASC selected by the user). Others have impact on calculating low level parameters and will also be used (indirectly) for the VSND. Others (IP addresses) will directly be used for both, VSND and ASC configuration.[1]
Hence functional parameters can be both – high level or low level. However their values are simply defined without the need of processing them for resource estimation.

4.1.3. Providing Information for Benchmarking

In order to let the FS determine resource mappings (i.e. low level resource values for specific high level application parameters) an ASC needs to be benchmarked. Benchmarking requires a dedicated set of parameter (profiles) as well as data for input and output. In terms of IRMOS an entire application – dedicated for benchmarking – needs to be defined.

In IRMOS, benchmarking will be conducted as a special case of the normal execution phase. The reason for this is twofold. First of all, extra delays that are inserted by the virtualization layer in the infrastructure and by the rest of the IRMOS platform services (like the Framework Services) need to be measured. By benchmarking on realistic infrastructures the data set will be more precise and so will be the estimations. Furthermore, the Framework Services do not have the necessary infrastructure to perform benchmarking, from physical machines to specialized monitoring services, according real-time schedulers, intelligent network links etc., like the ISONI infrastructure, since this is not their purpose in the overall architecture.

In order for this to be implemented, the ASC needs to follow all the steps of a normal application design (like workflow description and modelling), in order to be able to be executed as a standalone application, for the aforementioned reasons. This “benchmarking workflow” will contain the specific ASC plus several other elements that are required for it to be executed.

The ASC developer must also provide the high level values for which to benchmark and the according input files (e.g. he must specify that he will run the video encoding ASC with a video of 25 fps, and provide an according video). These values for which to benchmark can be inserted into the ASCD or in a better fashion the ASC developer can edit a specific purpose ASLA for benchmarking, so that he can also control the cost of the benchmarking phase.

4.1.4. Providing Information for Modelling

In order to let the FS improve estimations for resources (and consequently their exploitation) a model of the ASC has to be provided.

In IRMOS application modelling aims at understanding the application's behaviour in terms of key performance indicators, i.e. workload completion time, mean time to failure, mean time for recovery from failure, availability.

The application model refers to one or more ASCs, which are then combined to determine the behaviour of an application as a whole. The application developer has to specify this combination which is referred to as Application Description (AD).

The application performance estimation builds on the application model to estimate required resources to be allocated. This requires the following information to be available in the ASC model:

1. Workload Features: these are the characteristics of the workload that the customer is allowed to submit e.g. average video length, video format.
2. ASC Interrupt Events: specification of the interactions with the application that this customer is allowed to do e.g. stop/pause the application.
3. ASC FSM: the FSM describing the ASC. This includes the different states and transitions (in terms of probability values) that affect the ASC execution. It also includes resource failure models that may affect the ASC runtime e.g. link failure, bandwidth drop below certain limit, etc.
4. ASC Normal Operation Time Estimator: this is the tool (e.g. neural-network) to be used for estimating the ASC uninterrupted fault-free completion time. This requires selecting a representative benchmark suite tests with which the ASC is benchmarked.

4.2. Adapting to the FS-ASC-Wrapper

Each ASC provides a run-time interface to the framework services (FS) used for configuring, controlling and monitoring the ASC. To facilitate implementation a generic wrapper is provided which is responsible for the communication to the FS. The wrapper calls three different executables (most likely: scripts) which act as bridges to the ASC core (Figure 4-3).

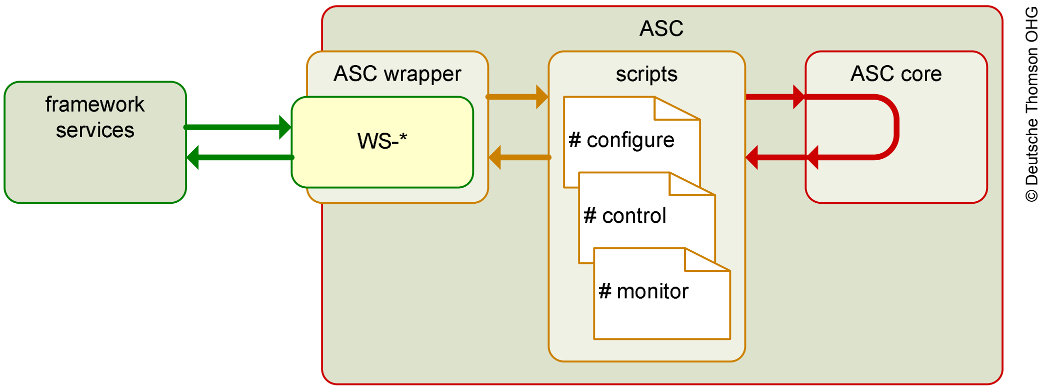

Figure 4-3 FS-ASC-Interface

The ASC developer must provide these three scripts and make the ASC core work with them. While this is rather trivial for 'configure' and 'control', the ASC has to provide monitoring information during run-time. E.g. the ASC writes current fps-values to a log-file. The 'monitor'-script parses this log-file and translates this output into a given format which is passed back to the wrapper which itself passes it back to the FS.

The FS-ASC interface is described in more detail in [1]. For deployment these three scripts will be named by convention and placed in a dedicated location in order to make them accessible by the ASC wrapper.

4.3. Integrating ACCs

The GUI component (ACC) needs to know some basic information for accessing an IRMOS service application. This meta information can be acquired via the IRMOS portal (a web based portal). It is passed to an ACC launcher which starts the actual ACC core either directly or via another start script (Figure 4-4).

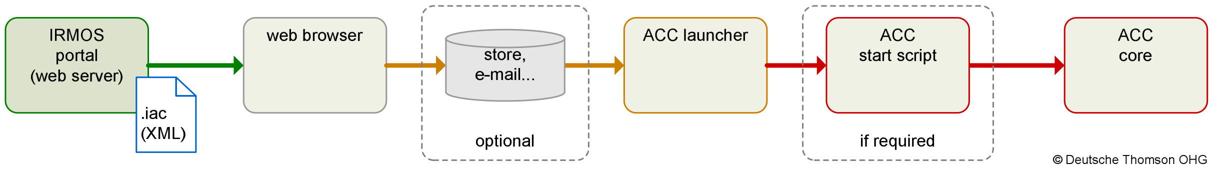

Figure 4-4 ACC Launcher

The ACC developer needs to provide an ACC start script which gets the configuration from the ACC launcher, translates this information in a format suitable for the ACC core (e.g. by writing a configuration file or constructing an appropriate command line statement). Alternatively the ACC core is prepared to be directly started from the ACC launcher.

The ACC-IRMOS interface is described in more detail in [1].

5. Next Steps

During the time of writing this document some of the tools and methods described here are still under development and improvement. Therefore it can be regarded as a snapshot covering the status in the middle of the project. By the end of this project an updated version could describe the final status of these methods.

Being used and extended by the developers of the current IRMOS prototype applications it can help future application developers in their task of creating new applications to be deployed on the IRMOS service platform. Moreover it may help the developers of the framework services and execution environment to spot issues that still could be improved to simplify use from the applications' perspective.

Regarding all these aspects serve to raise the attractiveness of the IRMOS platform to make it successful.

6. References

[1] IRMOS Deliverable: D4.1.1 - Definition and implementation of the three scenarios and its real time requirements, http://www.irmosproject.eu/Deliverables/Download.aspx?ID=31

[2] IRMOS Deliverable D4.2.1 - Interface Definition to the IRMOS SOI, http://www.irmosproject.eu/Deliverables/Download.aspx?ID=36

[3] Ghosh, Sukumar (2007), *Distributed Systems – An Algorithmic Approach*, Chapman & Hall/CRC, ISBN 978-1-58488-564-1

[4] Lynch, Nancy A. (1996), *Distributed Algorithms*, Morgan Kaufmann, ISBN 1-55860-348-4