A METHODOLOGY FOR ENGINEERING REAL-TIME INTERACTIVE MULTIMEDIA APPLICATIONS ON SERVICE ORIENTED INFRASTRUCTURES

Dimosthenis Kyriazis[1], Ralf Einhorn[2], Lars Fürst[2], Michael Braitmaier[3], Dominik Lamp[3], Kleopatra Konstanteli[1], George Kousiouris[1], Andreas Menychtas[1], Eduardo Oliveros[4], Neil Loughran[5] and Bassem Nasser[6]

[1]*National Technical University of Athens*
[2]*Tixeltec*
[3]*University of Stuttgart*
[4]*Telefonica Investigation e Disarollo*
[5]*SINTEF*
[6]*University of Southampton IT Innovation Centre*

ABSTRACT

Future Internet applications raise the need for environments that can facilitate real-time and interactivity without major modifications in the application domain. Such environments should be able to efficiently adapt resource provisioning to the dynamic demands of the applications, the majority of which tends to be real-time and interactive. In principle, all applications are suitable to be executed in service oriented environments as they are, without any kind of adaptation or modification. Nevertheless, the concrete characteristics of emerging infrastructures, mainly focused on guaranteeing the offered quality of service, require specific steps to be performed by the application developers and adopters in order to exploit the maximum of the infrastructure offerings. In this paper, we present a methodology for creating or adapting real-time interactive multimedia applications for service oriented infrastructures.

KEYWORDS

Service-oriented Infrastructures, Cloud computing, Quality of Service, Real-time

1. INTRODUCTION

Service Oriented Architectures (SOAs) [1] refer to a specific architectural paradigm that emphasizes implementation of components as modular services that can be discovered and used by clients. Infrastructures based on the SOA principles are called Service Oriented Infrastructures (SOIs). Through the agility, scalability, elasticity, rapid self-service provisioning and virtualization of hardware, Service Oriented Architecture principles are reflected into Clouds, which provide the ability to efficiently adapt resource provisioning to the dynamic demands of Internet users. Many architectural paradigms from distributed computing such as service-oriented infrastructures, Grids and virtualisation are incorporated into Clouds. There are three main classes in the cloud services stack which are generally agreed upon [2]:

- *Infrastructure as a Service* (IaaS), which refers to the provision of 'raw' machines (servers, storage, networking and other devices) on which the service consumers deploy their own software (usually as virtual machine images).
- *Platform as a Service* (PaaS), which refers to the provision of a development platform and environment providing services and storage, hosted in the cloud.
- *Software as a Service* (SaaS), which refers to the provision of an application as a service over the Internet or distributed environment.

In this paper we focus on the SaaS class, presenting a methodology for adapting real-time interactive multimedia applications for cloud-based service oriented infrastructures. Before we continue, let us clarify the term "real-time". Traditionally, 'real time' refers to hard real-time systems, where even a single violation

of the desired timing behaviour is not acceptable, for example because it leads to total failure, possibly causing loss of human lives. However, there is also a wide range of applications that also have stringent timing and performance needs, but for which some deviations in Quality of Service (QoS) are acceptable, provided these are well understood and carefully managed. These are soft real-time applications and include a broad class of interactive and collaborative tools and environments, including concurrent design and visualization in the engineering sector, media production in the creative industries, and multi-user virtual environments in education and gaming. In particular, we focus on interactive soft real time applications where one or more users interact with the application and with each other.

Soft real-time applications are traditionally developed without any real-time methodology or run-time support from the infrastructure on which they run. The result is that either expensive and dedicated hardware has to be purchased to ensure good interactivity levels and performance, or that general-purpose resources are used as a compromise (e.g. commodity operating systems and Internet networking) with no way to guarantee or control the behaviour of the application as a result. In this paper, we present a methodology, being developed in the European Commission supported IRMOS project [3], for application developers describing how to use specific tools during various phases of the application development process. The outcome of this process is soft real-time applications that can be executed on Service Oriented Infrastructures (SOIs) with guaranteed quality levels. The proposed adaptation enables the PaaS providers to utilize techniques for modelling, predicting, provisioning and monitoring resource and QoS requirements commitments and thus allowing real-time interactive multimedia applications to be executed on SOIs.

The aforementioned applications consist of *Application Components* (ACs) and are described by an Application Description (AD) that includes their functional parameters. The ACs are software components actually providing functionality and can be: a) *Application Service Components* (ASCs) that run autonomously and are deployed inside the virtualized infrastructure, e.g. an image renderer, b) *External ASCs* (EASCs) – same as ASCs but run outside the virtualized infrastructure because they offer non-standard, "non-virtualizable" functionality, e.g. a GPU processing node, c) *Application Client Components* (ACCs) – the software used by an end-user (client) to access the application – always running outside the virtualized infrastructure. The ACs are described by a document containing everything that is needed (e.g. resource needs and functional parameters) to run the ASC on the SOI. This is called *Application Service Component Description* (ASCD).

The remainder of the paper is structured as follows: Section 2 gives an overview on characteristics of potential applications and first generic (i.e. independent of the SOI) steps to be done for preparation, while Section 3 focuses on the steps needed for integration and adaptation the applications to a real-time aware cloud-based platform (e.g. IRMOS) and available tools helping the developer on this task. The functional and descriptive interfaces used in these processes are described in Section 4. The paper concludes with a discussion on future research and potentials for the current study.

2. THE FOUNDATION: AN APPLICATION

Basically there are two possibilities to create an application that will run on a service oriented infrastructure:

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

Instead of running on dedicated physical hosts, the service parts of the applications run on Virtual Machine Units (VMUs) within a virtualized infrastructure provided by the IaaS provider, an example of which is Intelligent Service Oriented Network Infrastructure – ISONI [4].

In principle, all server applications are suitable to be executed in the cloud-based environment as they are, without any kind of adaptation or modification. But the concrete characteristics of platforms that facilitate real-time and interactivity, 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 such platforms.

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. 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 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 the environment to access data of other surrounding applications.

Following, we describe the steps that are required in order to prepare an application to be executed on a SOI. 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)). 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 [5], [6]. 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 utilize the SOI 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 SOI:
  - 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 infrastructure framework services, represented on the application component's node by a 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 SOI the better the infrastructure can deal with QoS issues for that component.
  - ACC need to be started with specific parameters required for accessing the application service running on the SOI. The configuration information provides the necessary details to allow the ACC to access the various components (ASCs) on the infrastructure 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 and modular approaches e.g. for wrapping ACs.

The following figure (Figure 1) shows a typical service oriented application. The application runs distributed on different hosts, while application related communication and data transfer like passing image data from one component to another is done independently from the platform (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).

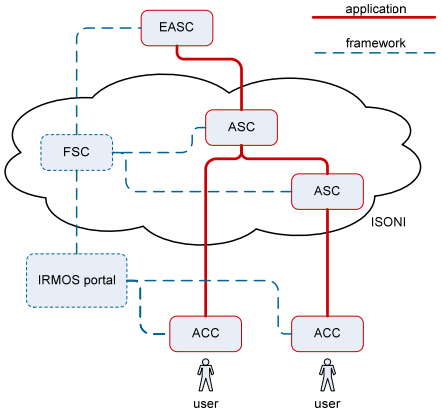

Figure 1. Sample application

3. FROM APPLICATION COMPONENTS TO AN APPLICATION

As described in Section 2, a service oriented application is based on application components (ACs). Figure 2 depicts the phases of an AC. This section 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.

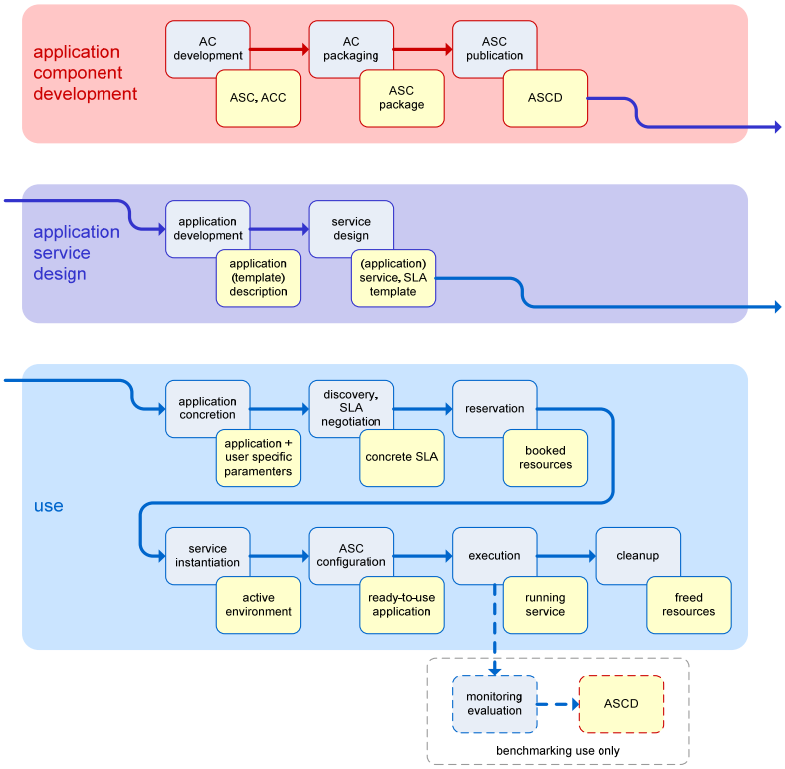

Figure 2. Application phases

3.1 AC Development & Packaging

The first step in the process of adapting an application to run on a service oriented environment refers to the development of the application components. Ideally the created ACs may also be used in a non-SOI. 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.

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

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

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

3.2 ASC Publication

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

1. Publish a description of the ASC to the PaaS domain. This task implies that the ASCD has already been created. 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 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 an 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 4). 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.

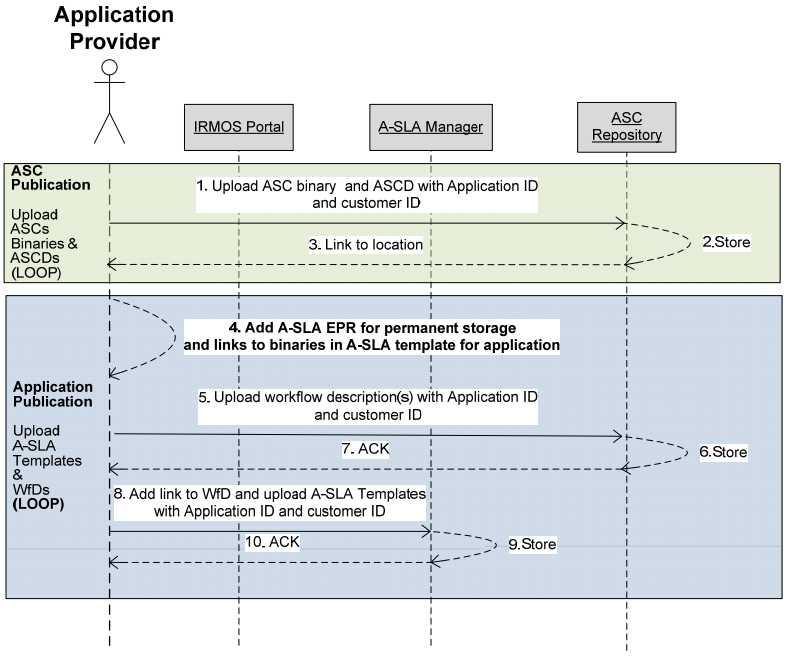

Figure 4. ASC and application publication sequence

3.3 Application Development

When creating an application the developer uses a service modeling environment (e.g. Papyrus [7]) that 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.

There are several elements involved in creating an application:

3. Creation of ASC descriptions (using UML classes)
4. Creation of instances of these descriptions (using UML class instance specification)
5. Creation of links between instances (using UML composite structure diagram)
6. Creation of a workflow which describes the interaction between the ASC instances (using UML activity diagram)
7. 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.

3.3.1 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. The following figure (Figure 5) 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.).

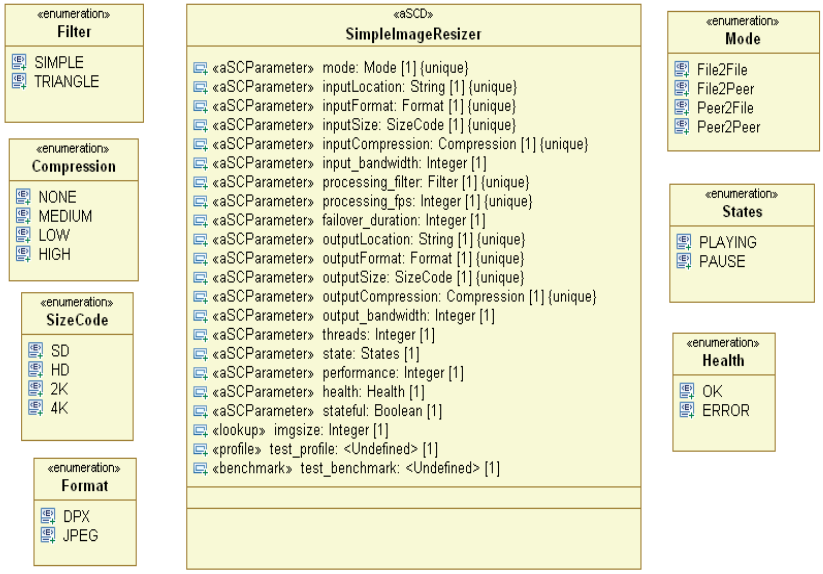

Figure 5. Example ASC description

3.3.2 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 6 depicts 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.

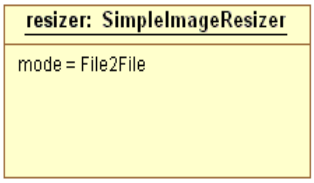

Figure 6. Example of ASC instance

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. As such, Figure 7 illustrates the previous resizer instance of SimpleImageResizer connecting with other ASCDs.

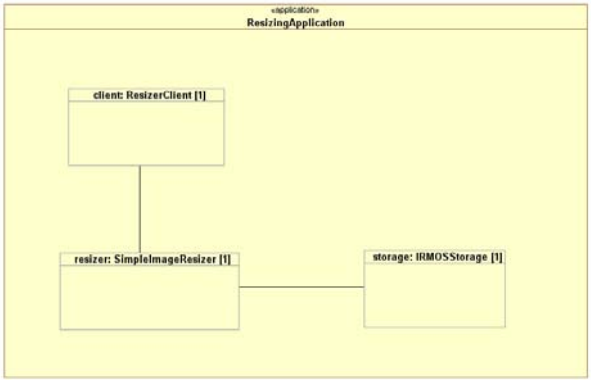

Figure 7. Example of linking ASC instances

3.3.3 Application Instance

The previous 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 8.

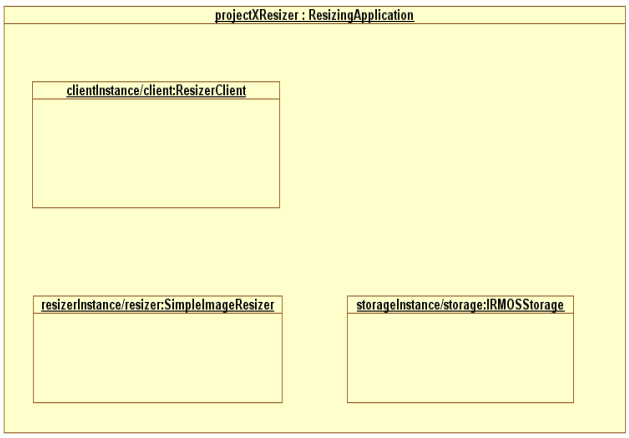

Figure 8. Application instance example

3.4 Application Publication

Having published the ASCs (Section 3.2) 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 (SaaS) Provider publishes these A-SLA templates to the PaaS 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 4). 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 Section 3 more compact. The following figure (Figure 9) shows the different AC-relevant interfaces both descriptive and functional.

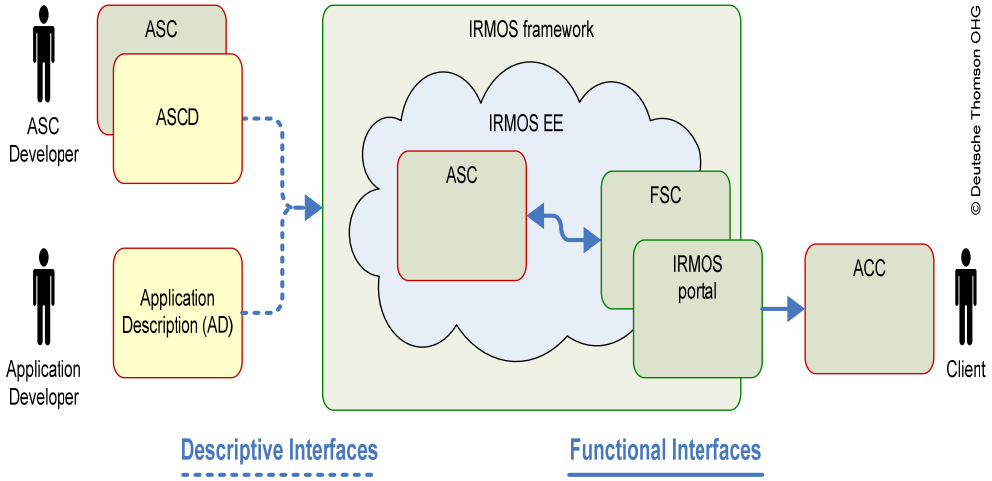

Figure 9. Interfaces

An ASCD thoroughly describes an ASC and represents the only 'off-line' interface between the application and the SOI. Hence it is an essential document containing the entire set of information letting the environment use the ASC regarding resource reservation, configuration, control, monitoring, benchmarking and modelling. An ASCD is built using a UML design tool and the associated ASCD4UML profile (Figure 10). For simplicity we have chosen to use this profile within the Papyrus UML design tool 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 1.

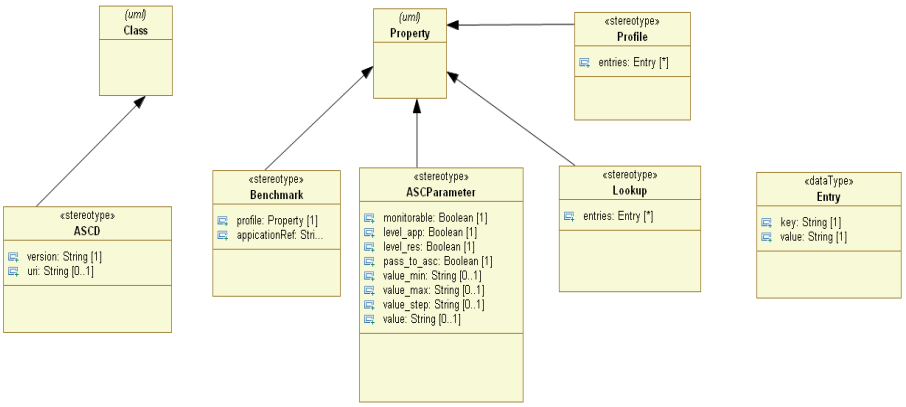

Figure 10. UML profile for ASCD

Table 1. Description of UML4ASCD stereotypes

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

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, we can also provide values to different properties which are contained within the profile elements.

4.1 Defining Resources on High and Low Level

One of the main purposes of the ASCD is to let the PaaS provider generate a description of the required resources for executing an ASC on the virtualized infrastructure. The aforementioned description 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 virtualized infrastructure without any actual values (as these are calculated by the PaaS provider).

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.

4.2 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 description of the required resources. Others (IP addresses) will directly be used for both, description of the resources and ASC configuration.

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.

4.3 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. Benchmarking is usually 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 PaaS 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., 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.

4.4 Providing Information for Modelling

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

5. CONCLUSION

Current approaches on Service Oriented Architectures focus on designing and implementing a rich set of services to efficiently operate, manage and reconfigure computing, storage and network resources under real-time conditions, providing to end users and to the associated applications the appropriate and required level of QoS. All Platform and Infrastructure capabilities are offered as on-demand services, although the architecture of the media applications varies from traditional n-tier enterprise applications to service-oriented workflows. Thus emerging cloud-based platforms and service oriented infrastructures face the challenge of providing QoS guarantees by proper real-time CPU scheduling [8] in order to facilitate real-time and interactivity as requested by Future Internet Applications.

In this paper we presented a methodology on how to engineer real-time interactive multimedia applications on service oriented infrastructures. The methodology describes the steps needed for integration and adaptation of the applications as well as their functional and descriptive interfaces to SOIs. Given the multi-tenanted development tools provided by PaaS providers (e.g. Facebook), such methodologies are considered essential not only for a limited number of application developers but also for consumers who also tend to become application developers.

ACKNOWLEDGEMENT

This work has been supported by the IRMOS project and has been partly funded by the European Commission's IST activity of the 7th Framework Programme under contract number 214777.

REFERENCES

[1] T. Erl, "Service-oriented Architecture: Concepts, Technology, and Design", Upper Saddle River: Prentice Hall PTR, ISBN 0-13-185858-0, 2005.

[2] The NIST Definition of Cloud Computing, Peter Mell and Tim Grance, Version 15, http://csrc.nist.gov/groups/SNS/cloud-computing, 2009

[3] The IRMOS Project, www.irmosproject.eu

[4] Whitepaper, "Intelligent Service Oriented Network Infrastructure Whitepaper", 2009.

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

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

[7] 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.

[8] Fabio Checconi, Tommaso Cucinotta, Dario Faggioli, Giuseppe Lipari, "Hierarchical Multiprocessor CPU Reservations for the Linux Kernel," in Proceedings of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009), Dublin, Ireland, June 2009