Context-aware Resource Management for Service Oriented Applications Based on Cross-Layer Monitoring over Virtual Communities

Shudong Chen, Johan Lukkien, Richard Verhoeven

***Abstract*— Distributed services hosted by electronic devices can be composed into applications using external service orchestration. Since the collaboration involves multiple parties and happens over the Internet, without access control and proper quality of service (QoS) management this type of application is unacceptable. This paper presents VICSDA, a context-aware resource management middleware for service oriented applications, which aims to secure service discovery and access, and to handle the inherent dynamics of services and the network. Devices and services are organized in virtual communities. All activities happen within a virtual community. Only authenticated users are capable to access services and resources. VICSDA accepts application-oriented QoS, gathers statistics of required services and their dynamic resource utilization through a cross-layer monitoring architecture, and updates the service coordination decisions according the real time states of the environment of a running application. For these, two novel services are designed. One is the resource manager which anticipates the performance of both required services and the network and be aware of the environment changes. Another is a device manager which runs on each device and has the fully control of services and their underlying resources. A 3D video streaming application is chosen to prove the feasibility of VICSDA. The system adapts the video encoding parameters and the delivery protocol to the observed capability of the end device and the bandwidth of the channel. Concrete results showed that based on measurable and manageable services, VICSDA is context-aware and can optimize the overall performance of service oriented applications.**

***Index Terms*—Context-aware resource management, Virtual community, Cross-layer monitoring, Service orientation**

I. Introduction

There is a rapidly increasing amount of applications in the ubiquitous computing area, like content, functionalities and resources sharing. Resource constrained devices, e.g. Personal Digital Assistants (PDA), Consumer Electronic (CE) equipments, expose their capabilities as networked services and applications are achieved through the composition of services. Security is of high importance in this type of applications. Firstly, mechanisms of secure discovery and access to services are required to prevent services and devices from being abused. Secondly, communications which happen over networks need protection from being overheard. Therefore, how to protect the ownership of services and devices, and how to maintain the confidentiality becomes crucial. Moreover, applications that depend on shared services need to cope with issues like heterogeneity of the networks, capabilities of devices, notorious wireless channels and mobility, etc. For example, a video streaming application over an in-home network has to deal with the heterogeneity of the underlying network technologies. The video can be smoothly streamed to a PC with a 100Migabit speeds (Mbps) Internet connection. However, when the streaming is switched to a PDA, the video may suffer from the error-sensitive wireless connection and the constrained resources of the PDA if there is no dynamic resource management provided. It becomes even more challenging when we consider co-existence of different traffic, for instance, the video provider has to respond to multiple invocations simultaneously. This demands the development of new QoS-enabled mechanisms and architectures to efficiently monitor and manage the distributed resources, such as CPU, memory, and network bandwidth.

This paper presents a context-aware resource management middleware, VICSDA, designed for service oriented applications, supporting secure service discovery and access, dynamic cross-layer application QoS adaptation in heterogeneous networking environments. VICSDA was developed for the purpose of the I-Share project [1], [2] that focuses on the secure and efficient sharing of multimedia through forming applications from distributed services in virtual communities. More specifically, the proposed VICSDA middleware aims at:

--1. Providing a secure service discovery, access and collaboration environment to both service providers and service users;

--2. Assisting end users to form applications using an external orchestrator which allows binding services at the run time;

This work is supported by the research project of Freeband I-Share: Intelligent Middleware for Sharing Resources for Storage, Communication and Processing of Multimedia Data, supported by the Dutch government, 2005-2008.

--3. Supporting application-oriented end-to-end QoS optimization through cross-layer performance monitoring and predicting of both services and the network, performing adaption or taking remedial actions in the case of context alteration of the application execution environment and service degradation;

--4. Assisting service providers to optimize the usage of the available local resources and network.

We assume that service providers and service users trust each other and are expected to communicate honestly within a membership boundary. Based on this trust, plain services hosted by devices are organized into virtual communities (VC) and become community services by being added security enhanced functionalities. Only authenticated users are capable to access community services and resources. Service discovery, access, and collaboration all happen in the scope of a VC.

Additionally, the focus of forming a VC in I-Share is to enlarge the notion of external orchestration with a secure enriched service composition [3]-[5]. This external orchestration provides end users with a high flexible and adaptable implementation of applications by connecting services at runtime. Shared VC services can be bound by an *Orchestrator* at runtime to accomplish applications. These applications are achieved based on service collaboration over heterogeneous networking environment. Targeting at optimizing application-oriented end-to-end QoS, VICSDA has been expended with a context-aware resource management functionality which can adapt the service coordination decisions to the real state of the environment [6]-[8]. A central concept is to gather the dynamic performance statistics of services and the network through a cross-layer monitoring, and then make decision adaption. In principle, a manner to manage services and resources which belong to different providers requires the highest priority to be achieved. Therefore, a device manager service is implemented [9]. It locates at each device and deals with service registration, resource allocation and monitoring etc. Next, with measurable and manageable services and resources, a resource manager service predicts the performance of required services and the network, and makes service coordination decisions for the Orchestrator, for instance, service selection recommendations. On service disruption and environment alternation, adaption information will be provided.

The remainder of this paper is organized as follows. Section II presents the virtual community based secure service discovery and access control and the design of VICSDA. The context-aware resource management based on cross-layer monitoring is detailed in Section III. Section IV describes a 3D video streaming prototype to validate the feasibility of the proposed middleware. Finally, we discuss some related work and draw conclusions in Sections V and VI, respectively.

II. VIRTUAL COMMUNITY BASED SECURE SERVICE DISCOVERY AND ACCESS CONTOL

Within the I-Share project, we have designed a service oriented virtual community overlay for secure service discovery and access, i.e. VICSDA [3]. It is a dynamic contract-based aggregation whose members have commonalities and interact via shared services by means of a digital network like the Internet. It has rules that each member has to follow. It provides services to members and it has the potential to develop different applications through an external service orchestration. It guarantees desired QoS to members.

In the rest of this paper, when we discuss VCs we are not referring to any aggregation of people, but to the communication among them which is done through the sharing and collaboration of services hosted by electronic devices.

A user can apply to join a VC and become a member. Members should obey the contracted management policy of that VC, for instance, they should trust each other and provide the promised QoS. Their behavior is monitored to guarantee a contracted QoS provision and relate to their reputation. In case of misbehavior, they will pay some form of penalty. Each member is autonomous which means he has the right to determine what services that he owns can be shared and which member can access his shared services. Furthermore, a member is free to decide to deregister from a VC at any time.

Plain services can become community services and shared among members by being added additional functionalities. From this point of view, a VC can be envisaged as an overlay network for the existing services. In this overlay, a service has enhanced functionalities: it is only exposed to authenticated members; it can filter access requests using its access control policy; and all the exchanged messages are encrypted. All service activities including publish, discovery, access and collaboration are executed within the scope of a VC. Meanwhile, a community service still has the properties of a plain service: its external network interface still uses the Service Oriented Architecture (SOA) [10]-[12] service interface; SOA standards such as Simple Object Access Protocol (SOAP) [13] and Hypertext Transfer Protocol (HTTP) [14] protocols are still suitable for service collaboration [15], [16].

VICSDA is a facility implemented to provide the following functionalities: i) formation and maintenance of a VC; ii) achieve applications with guaranteed QoS [3]. Basic functionalities of a VC, for example, member de/registration, service de/registration, authentication and encryption, service collaboration, reputation and recommendation, and fault recovery are done by different services depicted in Fig. 1.

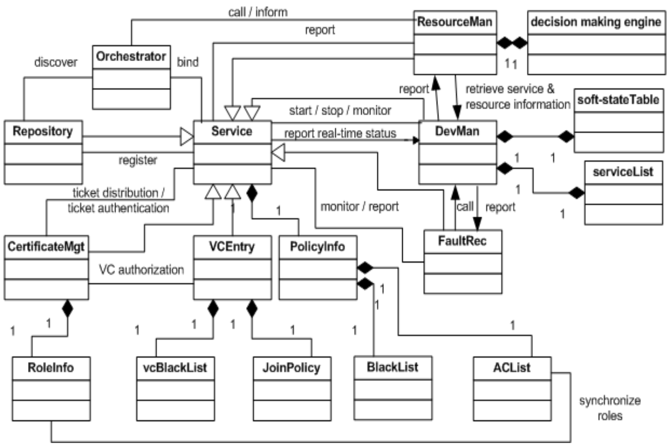
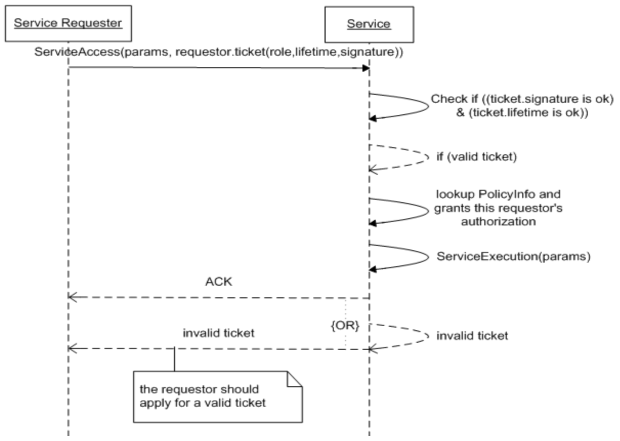


Fig. 1. Logical architecture of VICSDA


Service Requester
Service
ServiceAccess(params, requestor.ticket(role,lifetime,signature))
Check if ((ticket.signature is ok) & (ticket.lifetime is ok))
if (valid ticket)
lookup PolicyInfo and grants this requestor's authorization
ServiceExecution(params)
ACK
{OR}
invalid ticket
invalid ticket
the requestor should apply for a valid ticket
Figure 2. Secure service access in VICSDA

A service provider needs to become a member before it registers services into a *Repository*. With successful accessing a *VCEntry*, a service provider will become a member and be granted with specific roles and a certificate (ticket) signed by a *CertificateMgt*. A member can execute actions corresponding to his role while carrying a valid ticket. The Repository caches service registrations, receives service discovery queries, and performs matching between queries and registrations. In I-Share, we classify service registration into two types: active and passive. That a service registers itself at a VC Repository is called an active registration [17]. If the registration is done by its host device's management service, *DevMan*, then this is a passive registration. Protocols designed for these two types of service registration will be detailed in the next section where the DevMan is presented.

Access to a service is restricted to authorized members. Different access actions are granted to different members according to their registered roles. A service can be registered into multiple VCs. Only services in the same VC can be shared. A service user is required to provide a valid ticket of that VC. This ticket shows which VC this user belongs to and what kind of roles she can play there. A service first validates this ticket and then grants capabilities according to that user's roles. Using the authentication and authorization of a VC, un-trusted or malicious access requests to services can be filtered. Additionally, all exchanged messages in the scope of a VC are encrypted. Asymmetric cryptography [18] is used for communication secrecy. The cryptographic key pair for encryption and decryption is distributed by CertificateMgt. Fig. 2 illustrates the design of this access control approach which applies to all community activities.

VICSDA supports the use of fine-grained VC control policies while leaving ultimate control of the local access to services at service providers. A community service will be added an *ACList* and a *BlackList* as community properties. Service providers can specify each service's local access control policy by defining capabilities of different roles and blocking malicious service users by editing the BlackList.

The focus of forming a VC is to enlarge the notion of external orchestration with a secure enriched service collaboration and composition. Shared services can be bound together for achieving different application aims by an external *Orchestrator*. The Orchestrator is responsible for discovering which service can provide a required interface and what protocols should be used for the communication between two services in the workflow chain, and binding them together at runtime. Detail working principles of the Orchestrator will be introduced in the following sections.

III. CONTEXT-AWARE RESOURCE MANAGEMENT BASED ON CROSS-LAYER MONITORING

We have accomplished building applications through external service orchestration with open source code available at [2]. Most of these examples are multimedia applications, for instance video streaming over heterogeneous networks where the environment is inherently dynamic. The availability of required services, resources of underlying devices, or the network bandwidth can change at any time. Without proper management the QoS of this type of application is unacceptable. This requires VICSDA to provide the adaptive resource management functionality which can adapt the application QoS to the actual state of the environment. Aiming at this, VICSDA takes sophisticated cross-layer QoS monitoring and prediction actions and provides dynamic adaption with respect to satisfy the expected end-to-end QoS specified by end-users. Additionally, VICSDA can dynamically redistribute reserved resources within application activities to meet applications' requirements when unexpected perturbations lead to resource scarcity.

In this section, we first present the context-aware resource management model, followed by a description of the respective components.

A. *Context-aware Resource Management model*

We assume that the Orchestrator has discovered all required services from the VC Repository and it needs to know the best composition and collaboration between services in order to provide a high application QoS.

Fig. 3 illustrates how VICSDA supports the context-aware resource management. The central concept is as follows. With knowing the application-oriented QoS requirement specified by the Orchestrator, VICSDA first interprets it into performance-oriented QoS, monitors and predicts the performance of required services and of the network, and makes coordination decisions dynamically. These decisions are returned to the Orchestrator.

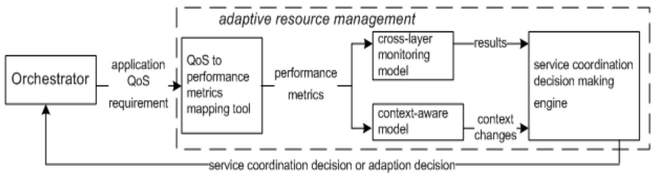

adaptive resource management
Orchestrator
application QoS requirement
QoS to performance metrics mapping tool
performance metrics
cross-layer monitoring model
results
context-aware model
context changes
service coordination decision making engine
service coordination decision or adaption decision

Fig. 3. Context-aware dynamic resource management model

Since we want to support a wide range of applications, we defined an application-independent representation expressed in Backus-Naur form [19] to express QoS requirements and service coordination decisions. In this representation, the description of a subject *S* consists of one or more sequences of symbols: topic *T*, content *C*, and boundary *B*:

$$S = (T, C, B)$$

For example, the Orchestrator needs the best possible service combination between two types of services, a video streaming service and a display service. In this case S is the Orchestrator's requirement *APP*, T can be symbolized as *SRVCOMPO*, B is specified as the type of two services *iStreamer* and *iDisplay*, and C is defined as *1 to 1 binding from iStreamer to the iDisplay* to express their association. This requirement is shown below.

```
INPUT
  SRVCOMPO
    COMPOTYPE TYPE = binding
    MAPTYPE TYPE = 1 TO 1
    SOURCE TYPE = iStreamer
    DEST TYPE= iDisplay
```

A corresponding return expression might be as follows.

```
RETURN
  SRVCOMPO
    COMPOTYPE TYPE = binding
    MAPTYPE TYPE = 1 TO 1
    SOURCE ID = 200001 # a streamer service #
    DEST ID = 200030     # a display service #
```

A QoS requirement can be highly application-oriented, for instance, the QoS requirement of the video streaming application might be *to render a MPEG-4 video with a 25fps fame rate*. This abstracted QoS requirement needs to be translated into QoS requirements of each required service, for instance, *a display service is required to provide 200MB memory and 30 million cycles per second CPU processing resources* to execute this video rendering task. In this way, VICSDA know what performance metrics it should estimate. This translation is done by the tool that maps the application-oriented QoS to performance-oriented QoS.

One of the main goals of VICSDA is to be aware of changes during the application execution and to adapt the application QoS accordingly. Therefore, a context-aware model is designed which uses a publish/subscribe scheme [20] to notify context changes. Service can publish specific events according to the requirements of different applications. Other services can subscribe to their interested events. For instance, a video display service publishes a *resource scarcity* event which will be generated when it cannot process arrived video within a given time constraint. The *ResourceMan* can subscribe to this event and are subsequently asynchronously notified when this event occurs. Context-aware adaption based on events notification is like an open loop control. These events may arrive at the subscriber at anytime. Subsequently, VICSDA will predict relevant metrics and make decision adaptations. The Orchestrator will accordingly adjust the current service coordination.

B. *Obtaining Resources Controllability*

In principle, in order to execute resource management a manner to manage services and resources which belong to different providers requires the highest priority to be achieved. Therefore, a device management service, *DevMan*, is implemented [17]. It is located at each device. It can activate and deactivate services hosted by that device, register these services into multiple VCs, and manage the magnitude of resources used by that service in each registered VC. Using this enabled full control of services and underlying resources, the performance of the distributed services can be estimated. Also, through the DevMan, the Orchestrator can execute service QoS adaptations according to dynamic service coordination decisions.

As mentioned, service registration in VICSDA is classified into two types: active and passive. In order to monitor service running states and manage resources usages, it is defined that services are entirely registered passively by the DevMan. However, in order to be inline with the nature of service registration which is that services are autonomous entities, we designed a novel service registration protocol to make these two registration types be compatible with each other. In the case of service active registration, a *serviceSelfRegistered* message will be sent to the DevMan and the DevMan will add that service's information into its serviceList. Details of this protocol can be referred to Fig. 4 and Fig. 5.

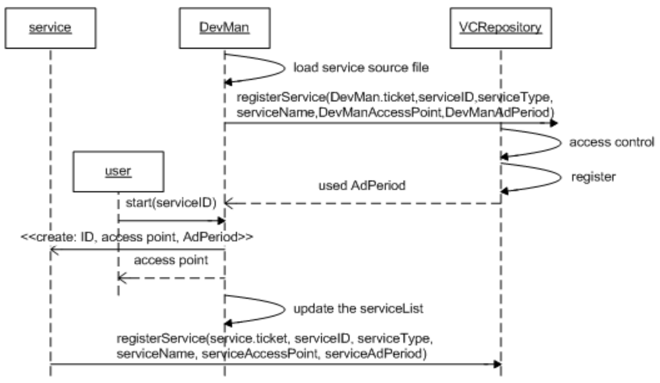

service
DevMan
VCRepository
load service source file
registerService(DevMan.ticket,serviceID,serviceType, serviceName,DevManAccessPoint,DevManAdPeriod)
access control
register
user
used AdPeriod
start(serviceID)
<<create: ID, access point, AdPeriod>>
access point
update the serviceList
registerService(service.ticket, serviceID, serviceType, serviceName, serviceAccessPoint, serviceAdPeriod)

Fig. 4. Passive service registration done by the DevMan.

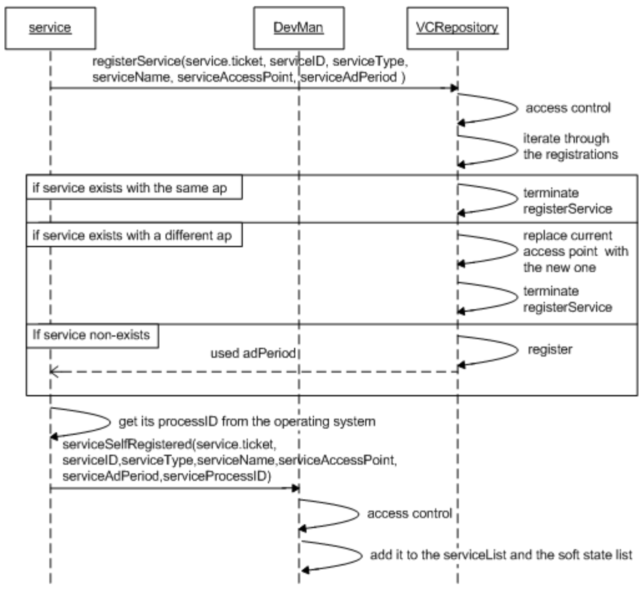

service
DevMan
VCRepository
registerService(service.ticket, serviceID, serviceType, serviceName, serviceAccessPoint, serviceAdPeriod)
access control
iterate through the registrations
if service exists with the same ap
terminate registerService
if service exists with a different ap
replace current access point with the new one
terminate registerService
If service non-exists
register
used adPeriod
get its processID from the operating system
serviceSelfRegistered(service.ticket, serviceID,serviceType,serviceName,serviceAccessPoint, serviceAdPeriod,serviceProcessID)
access control
add it to the serviceList and the soft state list

Fig. 5. Active service registration done by services.

Once a VCRepository receives a service registration call, it first examines its tables. If a service with the same literal information and access point exists in the registration, this process will be terminated to avoid storage redundancy; if a registered service has a different access point but the same, this means that either this service is registered as inactive by its DevMan or this is a delayed service entry. The existing access point will be replaced by the new arrival. Finally, if there is no record for this service, the VCRepository will simply store this service registration. In this way, the replacement of a DevMan's access point with the one of the actual running service can be solved by calling the same *registerService* method. In the previous VICSDA service registration protocol and also in other existing ones, the active service registration will terminate here. However, in this paper, an extra step is to notify the DevMan with the service's VC related information, for instance, the corresponding *processID* and the *access point*, for the sake of later observation.

In principle, service registration is mainly done passively by the DevMan in order to control the resources usage of services for VCs. This updated protocol guarantees VC related information of a service can also be traced by the DevMan even when the service registers itself.

Two components, a *serviceList* and a *soft-stateTable* are designed to maintain the information of the services on a device, where all active and inactive services are kept in the serviceList while the soft-stateTable caches the resources usage by services invocations coming from VCs in order to deal with simultaneous invocations. The structure of the serviceList is shown below.

```
[{serviceName, serviceType, serviceID, vcs[{vcName,
serviceAccessPoint, processID, adPeriod,
resources[{resourceID,maxAmount}]}]}]
```

An example given below assumes that a device has registered a *photo sender service* in two VCs, *SAN* and *VCA*.

```
[{photosender,phototransfer,100000, vcs[
{SAN,http://www.win.tue.nl/san/amosa/photosender:88,javas
ender1,50s,  resources[{memory,200MB}{cpu,20%}]}
{VCA, http://www.win.tue.nl/san/amosa/photosender:99,
javasender2,60s, resources[{memory,150MB}{cpu,15%}]}]}
```

Using this serviceList, DevMan has the knowledge about which VC a service has registered (*vcName*), which process is running as a service instance for a VC (*processID*), how much *resources* can be used maximally by that process. If a service is not deployed or is deactivated, values of *vcs* will be set as null. *adPeriod* is used for keep the freshness of this service. Details about the update mechanism will be addressed later in this section.

With the information of dynamic resources utilization by service in VCs logged in the *soft-stateTable*, resource reservation, scheduling, and even dynamic reallocation can be achieved. For instance, when a device is heavily loaded, the DevMan can hang up a *task*, also as known as *process* in multitasking operating systems, which is handling a resource consuming service method call to release some amount of resources, or deactivate a service to withdraw all reserved resources when resources are excessively used by it. This resource scheduling decision is made with the aim of providing users a high overall performance of a device. The structure of this soft-stateTable is represented as below.

```
[{resourceID, vcs[{vcName,maxAmount}], tasks
[{taskID,taskType,serviceID,vcName, currAmount,
flag}]}]
```

resourceID represents different types of resources, e.g. CPU, physical memory, virtual memory; *vcs* describes the maximum amount of resources a device can use for each VC. Each task entry expresses that this type of resource is allocated to which task (*taskID*), the type of this task (*taskType*), serving for which service (*serviceID*) in which VC (*vcName*) and current usage amount (*currAmount*). Tasks, viz. processes for service invocations, are independent and compete for resources. Possible status of a task can be *running*, *ready*, *blocked*, or *completed*, *failed*, *cancelled*. Correspondingly, the resources they are competing for can also have different status. We tag assigned resources with different *flags* to represent their status. The mapping between the resources status (*flag*) and the tasks status is defined in formula (1).

$$flag = \begin{cases} reserved & taskStatus \in \{ready, blocked\} \\ u \sin g & taskStatus = running \\ released & taskStatus \in \{completed, failed, cancelled\} \end{cases} \qquad (1)$$

An example is given below. It shows that a device is running a multimedia task for a service with a serviceID *200031* which consumes *250MB memory* and *50% CPU cycles*. Meanwhile, a photo transfer task to service *200020* is waiting in the task ready list with *100MB memory* and *10% CPU cycles* being reserved. From the *pdfreader* task entry, we can deduce that was a document browsing task coming from the VC *VCA*. It may have been executed successfully or fail, or been hung up by the DevMan due to resource scarcity reasons.

```
[{memory, vcs[{SAN,700MB}{VCA,300MB}], tasks[
{100859,multimedia,200031,SAN,250MB,using}
{100000,phototranser,200020,VCA,100MB,reserved}
{1000888,pdfreader,200036,VCA,50MB,released}]}

{cpu, vcs[{SAN,60%}{VCA,20%}], tasks[
{100859,multimedia,200031,SAN,50%,using}
{100000,phototransfer,200020,VCA,10%,reserved}
{1000888,pdfreader,200036,VCA,5%,released}]}]
```

Working together with the serviceList, this soft-stateTable helps the DevMan to prevent excessive resource use by one service which would result in a low overall performance of a device.

Soft-state is characterized as data with a limited validity period [21],[22]. Examples include the state of short-lived user sessions, caches, stored aggregates and transformations on large datasets, etc. In VICSDA, we use soft-state for service and resource observation and management. The idea is using soft-state to keep track of the services state with high accuracy including service registrations freshness and resources usage by different services for different VCs.

Fig. 6 bellow illustrates the process of the registration update. This process is triggered by a timer with *adPeriod* as the period, at the DevMan, at the service and at the VCRository asynchronously. A service sends the renewal message to the VCRepository; the VCRepository updates it registrations; the DevMan observes the process where a VC service instance is running and then updates the state in the serviceList.

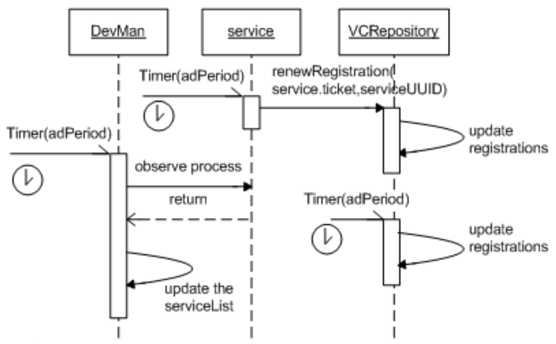
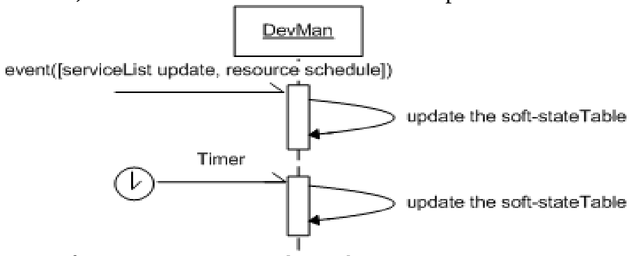


Fig. 6. Soft-state registration update at the VCRepository and the DevMan.

In order to provide resource usage information as fresh as possible, two methods are used to update this soft-stateTable, shown in Fig. 7 The update process can be triggered by a timer periodically or by events, for instance, when the DevMan monitors the aliveness of a process for the serviceList update, the result can be reused for the soft-stateTable update; or when the DevMan updates a *task* entry's *flag* based on a scheduling decision, the entire soft-stateTable will be updated.


DevMan
event([serviceList update, resource schedule])
update the soft-stateTable
Timer
update the soft-stateTable
Fig. 7. Soft-state resource usage update at the DevMan.

C. Cross-layer Monitoring Architecture

In service oriented applications, the end-to-end QoS delivery from the service provider to the service user may span over different types of networks and can be therefore divided into the node level, network level and application level. Devices that host services belong to the node level and are expected to manage its local resources for handling invocations to its services and providing the expected QoS. At the network level, the capability of the delivery channel needs to be probed as one of the factors to make resource management decisions and adaptations. Application-oriented QoS requirements and received QoS feedback from end users will arrive at the application level.

This monitoring is achieved by two distinct services, the *ResourceMan* and the *DevMan*. There is one ResourceMan service running at one time while multiple DevMans are located on each device.

With respect to meeting the expected end-to-end QoS, VICSDA takes cross-layer QoS monitoring and prediction actions. At the node level, devices that host services are expected to manage their resources to provide the expected QoS. At the network level, the capacity of the delivery channel is probed. Application-oriented QoS requirements and delivered QoS feedbacks from end users will arrive at the application level. There is one central ResourceMgt service running while each device runs a single DevMan service. They function at the node level and the network level respectively. The cross-layer monitoring architecture is depicted in Fig. 8 and described hereafter.

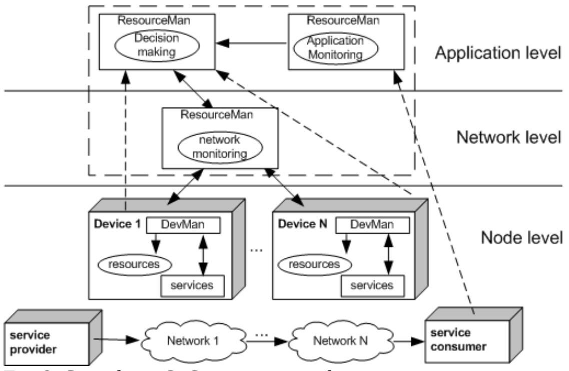


Fig. 8. Cross-layer QoS monitoring architecture

1) *Node level monitoring*

The DevMan monitors each hosted service by retrieving statistics from the serviceList and the soft-stateTable. With the current resource utilization information, the DevMan can manage the resource allocation to services in order to provide a high overall performance of the device. For instance, it can deactivate some services which are not serving any applications to release resources for another resource consuming service invocation.

Moreover, in order to make the ResourceMan aware of changes within the environment, the DevMan will notify the ResourceMan when resource scarcity or service degradation occurs. Subsequently, the service coordination adaptation will be triggered.

2) *Network level monitoring*

Network capacity is crucial to service oriented applications. Especially in time sensitive multimedia applications, packet loss and jitter will severely impact the video arrival rate at the destination service and the perceived QoS by end users. Therefore, the capacity of the underlying networks needs to be probed as another input parameter provided to the ResourceMan service. We use existing work on network performance probing [23],[24] to measure the bandwidth of the delivery channel and anticipate network latency.

3) *Application level monitoring*

The functionality of the ResourceMan service at the application level is to process the gathered statistics from lower levels and to make service coordination decisions according to the required application QoS. The mapping between the application-oriented QoS requirements into the performance-oriented resource metrics is introduced at the application level. Additionally, it would be useful when service users report the delivered QoS, such that the service coordination decisions can be more efficient. Therefore, the ResourceMan is designed to be able to receive feedback information at the application level.

D. *Service Coordination Decision Making Engine*

The network condition and resource usage of the required services gathered through the cross-layer monitoring, together with the context change events will be sent to the decision making engine of the ResourceMan to make dynamic service coordination decisions. Service coordination decisions include the services compatibility and the capability of a service to execute a specific task.

Compatibility of services means the matching between their supported protocols, for instance, the encoding protocol of a video streaming service and the decoding protocol of a display service. These protocols are described in the service description as part of the service properties. We have defined the schema of a VC service description to store the static information of a service [4], for instance, required memory, supported protocols, etc. The ResourceMan can retrieve this static information from the service description file to check service compatibilities.

The ResourceMan performs a schedulability test to estimate the capability of a device to execute a specific task of a service, for instance, whether a display service can decode received bit streams within a given time constraint. For this, the ResourceMan needs to launch this video decoding task to that display service. In order to get a high precision, a soft state approach is used to avoid a high kernel load. The test is executed at a soft state level instead of directly at an operating system kernel. Operating system interrupts are also eliminated. The ResourceMan uses the current scheduling algorithm of the underlying node to test the schedulability of a task. For example, the host node of the display service is using the Earliest-Deadline-First (EDF) [25] as the dynamic real-time scheduling algorithm to manage the principle resource CPU. With the knowledge of the frame rate of a video, the CPU requirement to decode it, and the current CPU usage retrieved from the soft-StateTable, the ResourceMan can use the following formula (2) to perform the schedulability test:

$$U = \sum_{\forall i \in task} \frac{Ei}{Pi} \quad (2)$$

U is the CPU utilization, E_i is the execution time of task$_i$, and P_i is the required period of task$_i$. If U is smaller than 1, the boundary value of the EDF scheduling, this device is capable to schedule this task such that all the running and ready tasks in its task pool will all complete by their deadlines. Otherwise, it is unable to schedule this task.

E. *Cross-layer Signaling Protocol*

Messages for the resource management are exchanged in either a pull or push model, depicted in Fig. 9. When the ResourceMan needs to anticipate the QoS of a service, it pushes relevant metrics to the DevMan of that service. In the communication between a DevMan and its services which run inside the same device, the DevMan pulls data from the serviceList and the soft-stateTable periodically. Environment change events are pushed to the ResourceMan using the publish/subscribe scheme.

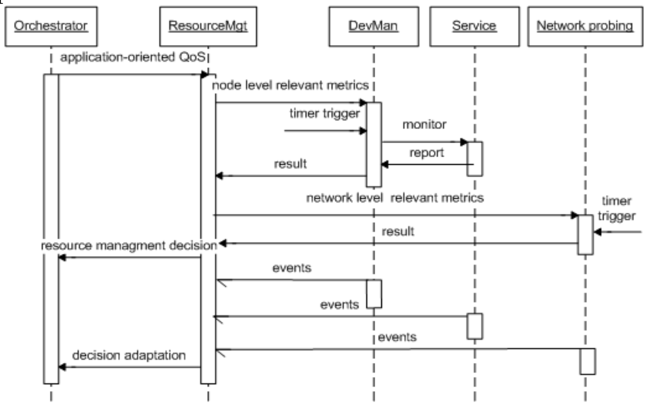


Fig. 9. Cross-layer signaling protocols

IV. PROTOTYPE

A test-bed has been set up to demonstrate the presented middleware. We describe here an interactive free view point 3D video streaming application to address the feasibility of VICSDA. This application demonstrates within a VC, a 3D video that is interpolated between four cameras and rendered either on a high performance PC or a PDA. During the rendering an end user can select an arbitrary view point by moving or clicking the mouse which controls the display. The displayed content will automatically adjust to that view point. Moreover, users' intention to change the display is tracked. The video can be redirected to another display when that display's controlling mouse is double clicked. The mouse-s, video streaming and display services are available as services and connected by an orchestrator. The encoding and delivery protocol of the video is adaptive to the capacity of the delivery channel and of the display device. For instance, when the video is redirected from the PC to the PDA, the resolution of the video will be decreased from 800x600 pixels to 320x240 pixels. The video streams will be truncated from a 3D video format to a 2D video format to assure the resource constraint PDA can decode all received frames in time and provide the end user a desired perceived QoS.

For the realization, a 3D video streaming software [26], which encodes and compresses the 3D video, has been wrapped as a 3D video streaming service and is registered at the VCRepository. First, the Orchestrator discovers the required services from the VCRepository: the 3D video streaming service, the display services, and the controlling mouse services. Next, it binds them together based on the service coordination decision made by the ResourceMan. The ResourceMan checks the availability of required services (active states, capability of a service user), the compatibility between them, and provided performance (capability to render the 3D video). Later, during the streaming, the ResourceMan monitors the capability of the display service and the delivery channel and makes service coordination adaptations if necessary. Consequently, the 3D video streaming service will adapt its encoding and delivery protocol for the next frame.

The following digital devices are used: four cameras (to shot a 5 minutes long video from different view point), two intermediate PCs (to generate the 3D video, and to host the 3D video streaming service), a PC (host of one display service), a PDA (host of another display service), and two control devices (one PC mouse and one PDA touch screen). The physical deployment and the system deployment of this application are given in Fig. 10 and Fig. 11 respectively.

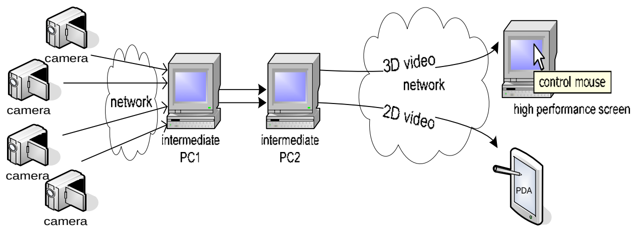


Fig. 10. Physical deployment of the 3D video streaming prototype

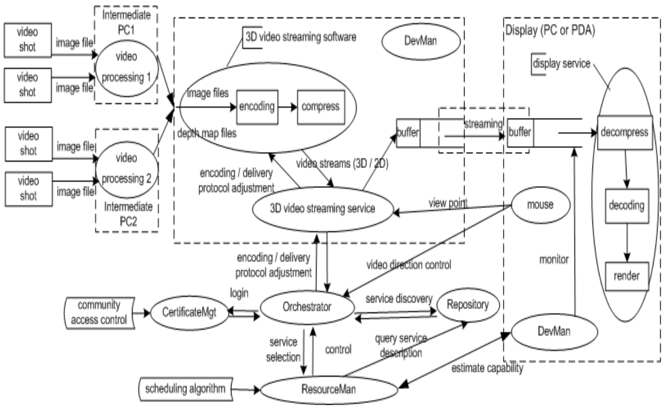


Fig. 11. System deployment of the 3D video streaming prototype

We designed several test cases. In the VC access control test, we first let a user invoke a VC service without a ticket and second let a VC member invoke a service registered at a different VC. Both of these two invocations failed because of the lack of valid tickets. This shows that only an authenticated user can access services in VCs. In the context-aware adaption test, we captured snapshots of a DevMan's soft-stateTable. Analysis shows that the service coordination decision adaptation was triggered by context change events.

With the satisfactory video rendering as a concrete result, we can conclude that the encoding and delivery protocol of the video can adapt to the capacities of the service and the channel.

V. Related Work

In the field of collaborative computing, there have already been some virtual community research activities.

Virtual community concept in peer-to-peer (P2P) systems, for example BitTorrent [27] and Tribler [28] focus on the P2P overlay network. Users with similar interests will be grouped into a virtual community automatically to speedup the download and solve the network bandwidth restriction problem. Members in the same virtual community will share resources and files. Here, resources include CPU, memory, hard disk storage, and network bandwidth. The objective of BitTorrent is that through virtual community formation the performance of the whole community is improved. Trible, based on social phenomena such as friendship and trust, can help to automatically build a robust semantic and social overlay on top of BitTorrent and can yield good cooperative downloading performance with respect to existing solutions. However, the anonymity in this P2P community forming limits the scope of applications. Although a potential use of P2P networks is to sharing video streams in real time, presented by [29], where peers in a P2P overlay can employ Application Level Multicasting (ALM) trees to receive the same interested video stream, the main purpose of P2P networks is still off-line file sharing. Moreover, secure resource sharing is always an issue in P2P systems.

A Personal Network (PN) [30], [31] aims to achieve seamless communication between electronic devices in an ad-hoc fashion. A PN enables a user remotely access to any of his personal service and content as if they were physically present in his vicinity. A PN supports applications based on the sharing resources and takes context and location information into account. Architecture of PNs, resource discovery, self-organization, routing, and security are addressed. However, the authentication of access to PN devices is still a security issue. And functionalities like context awareness, service discovery and resource management need to be realized.

VICSDA provides the secure service sharing within a VC and the denial of access to community service from outside a VC. This revives the concept of Authorized Domains, as proposed by the Marlin [32] and Coral [33] Digital Rights Management (DRM) based platforms. These platforms are built for sharing multimedia content across multiple devices in an in-home network. Multimedia content is protected using a governance rule specified by users. Protected content issued for a specific domain can be consumed on any device that has joined this domain. We claim that these are VC related work applied in different fields without service orientation concept and leaving the membership withdrawal issue open.

We also compare our work with several platforms on resource management at inter-domain level, which based on monitoring and measurement [34]-[39].

Paper [34] presented a scalable monitoring platform for the high-speed Internet, SCAMPI, which can provide monitoring of high-speed Internet circuits. It uses a monitoring adapter at 10 Gigabit speeds (Gbps), which is a programmable hardware monitoring adapter with built-in monitoring functionality. Using passive monitoring, which analyzes existing traffic in the network, SCAMPI developed several measurement tools to be used in several applications, e.g. Quality-of-Service monitoring, threshold alerting for traffic engineering, billing and accounting. Paper [35] based on passive monitoring sensors, addressed an Internet traffic monitoring infrastructure at speeds starting from 2.5 Gbps and possibly up to 10 Gbps using locality buffering. This work aims to prohibit unauthorized tampering with original traffic data with traffic monitoring tools. IST-AQUILA project [36] is developing inter-domain QoS-metrics measurement mechanisms to enable measurement based admission control in large-scale IP environment. For instance, the paper [37] addressed an Inter-domain resource control method. Based on the BGRP protocol which provides sink-tree based aggregation of resource reservations for the delivery of end-to-end QoS to applications across multiple separately administered domains, it can limit the signaling load and efficiently handle the reserved resources between Domains. These proposed systems assume that a centralized manager negotiates monitoring operations with each domain along the service / resource delivery path. This results in a scalability problem when the inter-domain network expands. VICSDA provides a solution for measuring the QoS which eliminates the inter-domain issues.

Work in [38], [39] focuses on QoS provision in multimedia content delivery. A service oriented monitoring system designed for use in multi-domain heterogeneous networking environment for the purpose of supporting cross-network audiovisual service offering is described. The proposed QoS monitoring system aims at providing service performance verification with respect to the QoS guarantees specified in contractual agreements between providers and users. This is achieved by monitoring services inside wired and wireless access/core networks and also at the customer side. Monitoring information is provided to service providers for providing quantified QoS-based services and service assurance and to network providers for managing network resources. However, it requires the media content to support an MPEG-21-based codec. Using service orientation technology and the context-free QoS representation VICSDA supports QoS assurance for a great variety of applications.

VI. Conclusion

This paper presents VICSDA, a context-aware dynamic resource management middleware for service oriented applications, which aims to handle the inherent dynamics of the local devices and the network. Based on a secure service cooperation environment provided by forming services into virtual community, a cross-layer monitoring architecture is designed to gather the performance statistics of services and the network. The ResourceMan service can interpret application-oriented QoS requirements into performance-oriented metrics. Being aware of the changes in the environment of an application through an event publish/subscribe scheme, it can anticipate the performance of required services and the network by making service coordination decisions. Service performance prediction is executed at a soft state level to avoid heavy kernel load. With the full control of VC services and their underlying resources achieved by a DevMan service, we can optimize the overall performance of a device and in turn of the application. The resource management service is platform independent. Benefit from the context-free representation method adopted for characterizing various parameters, the resource management service is flexible over a great variety of applications.

A 3D video streaming application is implemented to test the feasibility of VICSDA. Concrete outcome shows that this context-aware dynamic resource management middleware optimized the displayed video quality over adapting the 3D video’s encoding and delivery protocol based on the capacity of the delivery channel and of the display devices. Secure service discovery and access control were achieved. Conclusions can be drawn that VICSDA supports the end-to-end QoS provision to extensive service cooperation while keeping the authority, confidentiality, and full control of each service provider over its underlying resources.

Fault detection and recovery of the system will be addressed in the future to facilitate enhanced reliability and robustness of VICSDA.

Appendix

A. Syntax of the application-independent context-free representation

The description of a subject *S* consists of one or more sequences of symbols: topic *T*, content *C*, and boundary *B*:

$$S = (T, C, B)$$

So far, we have confined

$$S = \{INPUT, RETURN\}$$

$$T = \left\{serviceSelection, resourceScheduling\right\}$$

For input data *INPUT*, when the topic is *serviceSelection*, the rules of C are described as:

```
<Content_specification> ::=
INPUT<srvCompo><QoS><Boundary>
```

When the topic is *resourceScheduling*, the rules of C are described as:

```
<Content_specification> ::=
INPUT<task_descriptor><Boundary>
```

The expressiveness of *srvCompo*:

```
<srvCompo> ::=
SRVCOMPO<compoType><mapType><compoSource>
<compoDestination>
<compoType> ::=COMPTYPE <compoTypeValue>
<mapType> ::= MAPTYPE <mapTypeValue>
<compoSource> ::=SOURCE <srvID> | <srvType>
<compoDestination> ::= DEST <srvID> | <srvType>
<compoTypeValue> ::= TYPE = <string>
<mapTypeValue> ::= TYPE = <integer> TO <integer>
<srvID> ::= ID = <integer>
<srvType> ::= TYPE = <string>
```

The expressiveness of *QoS* is:

```
<QoS> ::= QOS <service><property><input><output>
<service> ::= SRV <srvID> | <srvType>
<property> ::= PROP <propType>
<input> ::= IN <context>
<output> ::= OUT<context>
<srvID> ::= ID = <integer>
<srvType> ::= TYPE = <string>
<propType> ::= TYPE = <string>
<context> ::=
(<string><operator><string>)|
(<string><operator><integer>)
operator = {>, >=, =, <=, <}
```

The expressiveness of *task_descriptor* is:

```
<task_descriptor> ::=
TASK<taskID><taskType><srvID>
<vcName><resourceRequirement>
<taskID> ::=TID = <string>
<taskType> ::= TYPE = <string>
<srvID> ::= ID = <string>
<vcName> ::= VC = <string>
<resourceRequirement> ::=REQ <context>
<context> ::=
(<string><operator><string>)|
(<string><operator><integer>)
operator = {>, >=, =, <=, <}
```

The expressiveness of *B* is:

```
<B> ::= BOUNDARY <context>
<context> ::=
(<string><operator><string>)|
(<string><operator><integer>)
operator = {>, >=, =, <=, <}
```

Comments are allowed to be inserted anywhere into this S specification using the following syntax:

```
<comment> ::= ( # <string> \n) | ( # <string> #)
```

Similar with the input data, the syntax of the return data is:

```
<Content_specification> ::=RETURN <srvComp><QoS>
<srvComp> ::=
SRVCOMP<compType><mapType><compSource>
<compDestination>
<QoS> ::= QOS <service><property><input><output>
```

Besides the service combination requirement of the Orchestrator as an example given in section III, we give another example which is used to express the video streaming task description generated by ResourceMan for the acceptance test on a display service. ResourceMan characterizes this task based on the display service's inherent properties, run time status and the underlying device configuration.

```
INPUT
  TASK
     TID = 100859
     TYPE = multimedia
     ID = 200030    # a display service #
     VC = SAN
     REQ
        memory >= 200MB
        CPU processing usage = 40%
     BOUNDARY    #  resource scarcity event triggers  #
         frameRate <= 20fps
        memory <150MB
        CPU utilization >50%
```

Correspondingly, the rescheduling approach returned from ResourceMan is as follows.

```
RETURN
   QOS
      SRV ID = 200030     # the display service #
  OUT
      fileFormat = FGS     # selected encoding protocol #
     frameRate = 15fps    # accepted video frame rate #
```

B. Prototype video

A video showing the prototype and the 3D video streaming demo is available from:

http://www.win.tue.nl/san/amosa/download.php

ACKNOWLEDGMENT

We would like to thank our anonymous reviewers. Their invaluable feedback will help substantially in improving the quality of the paper. Our thanks are also to Remi Bosman, Melissa Tjiong, Goran Petrovic, Peter de Width, Jurjen Middendorp, and Pim Vullers for their technical feedback at the implementation stage of this work.

REFERENCES

[1] IShare Freeband website. http://www.freeband.nl/project.cfm?id=520&language=en. 2008
[2] IShare SAN. http://www.win.tue.nl/san/amosa/ishare/intro.php. 2008
[3] S.Chen, J.J.Lukkien, I. Radovanovic. Freeband I-Share D1.5. 2007. Project Deliverable. http://www.win.tue.nl/san/publications/
[4] S.Chen, J.J.Lukkien, et al. Freeband I-Share D1.16. 2007. Project Deliverable. http://www.win.tue.nl/san/publications/
[5] S.Chen, J.J.Lukkien, et al. VICSDA: Using Virtual Communities to Secure Service Discovery and Access. In the Proc. of the Fourth International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, Vancouver, Canada, August, 2007.
[6] H.A.Duran-Limon, G.S. Blair, G. Coulson. Adaptive Resource Management in Middleware: A Survey. IEEE Distributed System Online. pp.1541-4922, IEEE Computer Society, Vol. 5, No. 7, July 2004.
[7] I. Cardei , et al. Hierarchical Architecture for Real-Time Adaptive Resource Management. In the Proc of IFIP/ACM Middleware Conference, Springer-Verlag, pp. 415-434, 2000.
[8] Dynamic Scheduling, final adopted specification, ptc/01-08-34, Object Management Group, 2001.
[9] S. Chen and J.J. Lukkien. Secure Resource Control in Service Oriented Applications. In the Proc. of the 6th Annual IEEE Consumer Communications & Networking Conference (CCNC 09), Las Vegas, USA, 2009.
[10] SOA. http://www.service-architecture.com/index.html
[11] Ethan Cerami. Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. First Edition February 2002. ISBN: 0-596-00224-6, 304 pages. Publisher: O'Reilly
[12] UPnP Device Architecture. Version 1.0. UPnP Forum. 2000
[13] SOAP Version 1.2. Published Specification. http://www.w3.org/TR/soap/
[14] HTTP protocol. http://www.w3.org/Protocols/
[15] Biplav Srivastava and Jana Koehler. Web Service Composition - Current Solutions and Open Problems. Online document. http://www.zurich.ibm.com/pdf/ebizz/icaps-ws.pdf
[16] W. Zhang, F. Liu, S. Chen and F. Ma. Automatic Services Composition in the Grid Environments. In the Proc. of ICCS 2006, pp. 1004-1007, Reading, UK, May, 2006.
[17] S. Chen and J. J. Lukkien. Obtaining Resource Controllability in Service Cooperation Environments. In the Proc. of the 7th International ACM Conference on Mobile and Ubiquitous Multimedia (MUM 08), Umea, Sweden, 2008.
[18] Housley, R., W. Ford, W. Polk and D. Solo. Internet X.509 Public Key Infrastructure: Certificate and CRL Profile. http://tools.ietf.org/html/rfc3280
[19] BNF. Online document. http://foldoc.org/foldoc.cgi?Backus-Naur+Form
[20] P. Eugster, P. Felber, et al, "The Many Faces of Publish/Subscribe", ACM Computing Surveys, Vol. 35, No. 2, June, 2003, page 114-131.
[21] X. Zhang, M. A. Hiltunen, K. Marzullo, and R. D. Schlichting.: Automizable Service State Durability for Service Oriented Architectures. In: Sixth European Dependable Computing Conference, pp. 119--128 (2006)
[22] B. C. Ling, E. Kiciman, and A. Fox.: Session State: Beyond Soft State. In the Proc. of the Symposium on Networked Systems Design and Implementation, pp. 22—22, 2004.
[23] D. Q. Liu and J. Baker. Streaming Multimedia over Wireless Mesh Networks. I. J. Communications, Network and System Sciences, 2008, 2, pp. 105-206.
[24] C.F. van Antwerpen. Interface Selection Layer Improving QoS using Interface Pair Selection. Master's Thesis of Eindhoven University of Technology.
[25] J.A.Stankovic, M. Spuri, K. Ramamritham, G.C. Buttazzo, "Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms", The Springer International Series in Engineering and Computer Science, Vol. 460, 1998.
[26] G. Petrovic, P. H. N. de With, "Near-future Streaming Framework for 3D-TV Applications", In the Proc. of the IEEE International Conference on Multimedia & Expo (ICME) 2006, page 1881-1884
[27] Cohen, B. Incentives to Build Robustness in BitTorrent. In the Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, 2003.
[28] J. A. Pouwelse, P. Garbacki, et al. Tribler: A Social-based Peer-to-Peer System. In the Proc. of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06). February, 2006, Santa Barbara, USA.
[29] J. D. Mol, D. H. P. Epema, and H. J. Sips. The Orchard Algorithm: Building Multicast Trees for P2P Video Multicasting without Free-Riding. IEEE Transactions on Multimedia, VOL. 9, NO. 8, December 2007, pp. 1593-1604.
[30] I.G. Niemegeers and S.M. Heemstra de Groot, "From Personal Area Networks to Personal Networks: A user oriented approach", Wireless Personal Communications, vol. 22, pp. 175-186, August 2002.
[31] F.T.H. den Hartog, M.A. Blom, C.R. Lageweg, et al. First experiences with Personal Networks as an enabling platform for service providers. In the Proc. of the Second International Workshop on Personalized Networks (PerNets 2007), August, 2007, Philadelphia, USA.
[32] Marlin Developer Community. http://www.marlin-community.com/
[33] Coral Consortium Cooperation. http://www.coral-interop.org/
[34] J. Coppens, E.P. Markatos, J. Novotny, M. Polychronakis, V. Smotlacha & S. Ubik; SCAMPI - A Scaleable Monitoring Platform for the Internet; Proceedings of the 2nd International Workshop on Inter-Domain Performance and Simulation (IPS 2004), Budapest, Hungary, 22-23 March 2004.
[35] Peter I. Politopoulos, Evangelos P. Markatos and Sotiris Ioannidis. Evaluation of Compression of Remote Network Monitoring Data Streams. In the Proc. Of the 6th IEEE Workshop on End-to-End Monitoring Techniques and Services (E2EMon 2008), April 2008, Salvador, Brazil.
[36] IST-LOBOSTER, European IST research projects, http://www-st.inf.tu-dresden.de/aquila/
[37] Peter Sampatakos, Eugenia Nikolouzou, Iakovos Venieris, "Applying the BGRP concept for a Scalable Inter-Domain Resource Provisioning in IP Networks", in the Proc. of the Sixth International Symposium on Communications Interworking), Perth, Australia, October 13-16, 2002.
[38] A. Mehaoua, T. Ahmed, H. Asgari, M. Sidibé, A. NAFAA, G. Cormentzas, and T. Kourtis, "Service-driven Inter-domain QoS Monitoring System for Large-scale IP and DVB Networks" in ELSEVIER's Computer Communication Journal, Special Issue on Monitoring and Measurements of IP Networks, Volume 29, Number 10, page 1687-1695 - june 2006
[39] M. Sidibe and A. Mehaoua. Service Monitoring System for Dynamic Service Adaptation in Multi-domain and Heterogeneous Networks. In the Proc. of the Ninth International Workshop on Image Analysis for Multimedia Interactive Services, May 2008, Klagenfurt, Austria.