Adaptable Resource Management Based on the Virtual Resource Tree Model

Marcelo Ferreira Moreno  Sérgio Colcher  Luiz Fernando G. Soares
Department of Informatics – Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
moreno@inf.puc-rio.br  colcher@inf.puc-rio.br  lfgs@inf.puc-rio.br

©2006 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE. The original publication is available at http://ieeexplore.ieee.org - http://dx.doi.org/10.1109/ICNICONSMCL.2006.40
Adaptable Resource Management Based on the Virtual Resource Tree Model

Marcelo Ferreira Moreno  Sergio Colcher  Luiz Fernando G. Soares
Department of Informatics – Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
moreno@inf.puc-rio.br  colcher@inf.puc-rio.br  lfgs@inf.puc-rio.br

Abstract
This paper presents an abstract model for resource management mechanisms, called Virtual Resource Tree (VRT). The VRT model is developed aiming at meeting different end-to-end Quality of Service (QoS) requirements demanded by multimedia applications. Scalability plays an important role in this scenario, bringing new mandatory features for resource management, such as the distributed management of heterogeneous resources and the support for adapting the different underlying management algorithms. In addition, this paper discusses the main issues towards the definition of the VRT file system (VRT-FS). VRT-FS is a framework that shows how, with a simple and familiar design, the functionalities of the VRT model can be ported to network operating systems.

1. Introduction

Distributed multimedia applications usually demand end-to-end Quality of Service (QoS) management. Briefly, end-to-end QoS provisioning begins by discovering which resources will participate in the process and which will be their individual QoS responsibilities. This “QoS orchestration”, should take into account possible cross-relationships among the chosen resources and take profit of admission control mechanisms to reserve part of the capacity of each involved resource. Resource scheduling techniques can then be applied in order to share the resource capacity and honor the admitted reservation [1].

In general, resource management usually has to deal with resources that can differ in: (i) nature (e.g. CPU, disks, network links), (ii) technology (e.g. ATM links, Ethernet links, etc) or (iii) type (fiber, coaxial cable, etc.). The complexity involved in the management of such a heterogeneous scenario can be relieved if resource management mechanisms are handled uniformly. Uniform mechanisms mean that the whole set of management operations is wrapped around by a standardized interface in order to offer the same access and a predefined well-known semantic, no matter which specific interface a resource presents, or how complex it may be.

QoS provisioning has become harder since new requirements have emerged. In fact, the fast and inexpensive deployment of services with new QoS requirements has become an essential ability, especially to telecommunications providers. In this context, services should be flexible enough to accommodate new QoS demands through smooth adaptations in their communication and processing infrastructure [2]. The support for new services may involve the choice of new scheduling, admission and classification algorithms. For this sake, many different high-level adaptability abstractions have been proposed [3].

Despite of its benefits, the deployment of a uniform, adaptable resource management mechanism can generate an undesirable or unacceptable overhead. Therefore, the mechanism should also be scalable itself and, again, adaptable on its own scalability.

The first goal of this paper is to present a generic resource management model, focused on end-to-end QoS provisioning and maintenance. The definition of such an abstract model should allow a uniform treatment of QoS aspects. The model is based on the Virtual Resource Tree (VRT) concept that allows the concurrent use of different management policies.

The paper also discusses how the VRT model can be easily instantiated using well-known mechanisms and data structures currently available in operating systems. The so-called VRT file system (VRT-FS) instantiation provides a virtual data structure and a programming interface that, differently from other file system-based resource management implementations, has its main focus on system adaptability.

---

1 This work was partly supported by the “National Telecommunication Development Fund” (FUNTTEL) and the “National Council for Scientific and Technological Development” (CNPq), which are maintained by the Brazilian Government
2. Background work

The VRT concept was firstly introduced as part of the SCM (Service Composition Model) [1]. Based on SCM, a family of frameworks was developed to model QoS provisioning mechanisms in generic environments. The family is organized in functional subsets: Service Parameterization, Resource Sharing and Resource Orchestration Frameworks.

In summary, the Service Parameterization Framework models generic data structures for service characterization based on parameters organized in service categories. By its turn, the Resource Sharing Frameworks are based on the Virtual Resource concept to model resource allocation and scheduling mechanisms. The VRT model described in Section 3 is an enhancement to these frameworks. Finally, the Resource Orchestration Frameworks are in charge of partitioning QoS provisioning responsibilities among the involved subsystems.

QoSOS architecture [1] was then designed as a specialization of the QoS frameworks, in order to describe adaptable QoS mechanisms for operating systems, despite their location (end nodes, routers, etc). Among several improvements, QoSOS introduced the Service Adaptation Framework, which describes “meta-mechanisms” that automate system adaptations and thus the provision of new services and the replacement of existent QoS policies.

The QoSOSLinux 0.1, the first implementation for QoSOS, used a very simple VRT structure, partially based on third party resource schedulers, like LinuxTC [4] and DSRT2 [5]. Some absent mechanisms in these schedulers were developed to complete and validate the frameworks and the VRT concept. At present, QoSOSLinux is being improved by using a more complete and adaptable resource management system, based on the VRT file system, as discussed in Section 4.

3. The Virtual Resource Tree Model

The advantages of the hierarchical resource scheduling are well known and its benefits have been widely discussed in the literature [6][7][8]. On the other hand, applying hierarchical admission control mechanism on a resource that is being hierarchically scheduled is a less familiar matter. A structure that allows hierarchical scheduling in conjunction with admission control, providing adaptation on both mechanisms is still less common.

The VRT Model goes beyond these features by supporting resource composition and thus end-to-end resource management, as will be discussed in Section 3.2. Moreover, all these facilities are obtained through a homogeneous behavior. The following sections describe the main components of the model.

3.1 Virtual Resource Trees

A virtual resource tree (VRT) is an abstraction that denotes the hierarchical division of the capacity of one resource or a group of resources managed together. The root node represents the main (virtual) resource (or group of resources) being divided with respect to its scheduling and allocation (reservation) mechanisms, as shown in Figure 1. Different management policies can be applied on the same resource and thus offering a wide and flexible set of services.

![Figure 1. A generic Virtual Resource Tree](image)

Each node in the tree is called a virtual resource and represents a partition on the capacity of its parent node. A scheduler is associated to each virtual resource, thus distributing the resource capacity among its child nodes. What exactly is the “capacity” being distributed may vary according to the resource. For example, if the resource is a CPU (in a CPU VRT), processing capacity may be shared among its children.

From the VRT root to its leaves, the virtual resource capacity is divided and finally allocated to flows. Flows can be regarded as final users of a virtual resource tree. For example, in a CPU VRT an instruction flow may be the user of one of its leaves, while in a network-interface VRT, the final user may be a data flow. When flow units arrive at a resource, a classifier, associated with the corresponding virtual resource tree, analyzes them to determine the correct VRT leaf where they will be queued. Usually, flow identification is based on rules that must be set to the VRT leaves when they are created.

A VRT-leaf scheduler chooses leaf users from the leaf queue. Scheduling strategies are responsible for deciding which child virtual resource user will hold the parent-node capacity each moment. Decoupling scheduling strategies from schedulers themselves
allows offering adaptable points (also called *hot spots*) that can be completed at instantiation time, or even modified during system operation time (by means of an adapting meta-service).

Usually, resources are scheduled in time or space. A single resource may be scheduled in both dimensions. The VRT model is generic enough to support both dimensions. When a resource needs to be scheduled in both dimensions, two VRTs (one for each dimension) must be associated with it. An example of such resource is the disk, which must be scheduled in time (for access) and in space (for storing).

Except for the VRT leaves, each virtual resource in a VRT is associated with an admission controller. Admission controllers check if a request for a new child virtual resource creation is *allowed* and *feasible*. A creation is allowed if its demands comply with all admission policies associated with the parent virtual resource. Admission policies are a set of restrictions rules that must be observed whenever a new request is being analyzed. An admission strategy algorithm verifies the feasibility of a virtual resource creation taking into account the current already admitted requests. Similar to scheduling strategies, admission policies and admission strategies are also kept apart from the VRT mechanisms to achieve the required flexibility with regards to adaptation.

The creation of a child virtual resource is performed by a virtual resource creator, which is also associated with the parent virtual resource. Briefly, based on service parameters supplied to the admission controller, the virtual resource creator must update the parent resource scheduling and classification rules; create the new virtual resource; and configure the new child resource scheduler and classification rules with default management policies.

### 3.2 Composite VRTs

In the simplest case, a VRT root may represent a single real resource. For example, the VRT root may be associated with a CPU or a communication channel. A VRT root can represent as well a set of resources to be managed as a single one. For example, a system with more than one CPU may have a single CPU VRT for managing them as a single resource. In both previous cases, the VRT is said to be a *primitive VRT*.

However, a VRT root may also represent the composition of different virtual resources that are managed independently. In this case, the tree is called *composite VRT*. Usually, when end-to-end QoS is required many resources are involved, as for example, network links, CPUs on end-user equipments, CPUs at intermediate systems, etc. It is desirable if, at some *level of abstraction*, this whole path of resources could be viewed and managed as a single resource. This abstraction is provided by composite VRTs.

A composite VRT arises as a result of a *resource orchestration* applied on a set of VRTs (including primitive and composite VRTs) that compounds a *resource forest* (for example, representing an operating system, a network, etc.). Forests can be nested, that is a forest can contain VRTs or other forests. When a service is requested to a forest, a specific system component verifies the feasibility of its admittance taking into account the current utilization of resources represented by the forest and the new request proposed load. This component is also called an admission controller, although it performs in a higher level of abstraction.

The forest admission controller starts its job by calling one of its child negotiator components. The called negotiator starts the orchestration process that first identifies a subset of child VRTs and forests that would be involved in the service provisioning.

The orchestration continues assigning portions of the QoS provisioning responsibility to each chosen VRT or forest, and activating mapper components, which are launched to translate the requested service category (and its associated parameters) to service categories directly related to each assigned component. Three types of QoS parameter translations are common:\[2\]

- **additive**: the sum of all individual QoS portions assigned to each chosen VRT or forest is equal to the requested QoS parameter;
- **multiplicative**: the product of all individual QoS portions assigned to each involved VRT or forest is equal to the requested QoS parameter;
- **concave**: each QoS portion assigned to each chosen VRT or forest is equal to the requested QoS parameter.

After mapping, admission controllers associated to each VRT and forest are called. In the case of forest its admission controller repeats all the above steps, recurrently, until primitive VRTs are reached, when their admission controllers are launched.

If all started admission controllers of a forest return affirmative answers, an affirmative answer is also passed to the higher-level admission controller that requested the forest services. This procedure repeats until the admission controller which has started the whole process is reached. Then, all resource

\[2\] Delay parameter is an example for the first type, reliability for the second type, and bandwidth is an example for the third type.
reservations made during the admission control phase are committed, that is, virtual resources are finally created in all involved primitive VRTs. Also, a composite VRT root is created from each forest, representing all the forest resources that were allocated. In any other case, the request is denied.

The composite-VRT root created by the highest-level admission controller can divide its capacity by creating child virtual resources, which in turn may divide its own capacity and so on, like in a primitive VRT. However, all other composite-VRT roots created in the process cannot be shared, since they are virtual resources completely allocated to other composite VRTs. In effect, the intermediary composite VRTs will only contain the root node.

As a simple example of the orchestration process, consider the QoSOS scenario illustrated in Figure 2. The composite VRT (1) represents the orchestration among two primitive resources: CPU and NIC (link) bandwidth. Each resource is managed by its respective primitive VRT (2 for CPU and 3 for NIC). The dotted lines in Figure 2 illustrate the resource orchestration that originated the root of the composite VRT.

![Figure 2. A composite VRT](image)

Before ending this section, several points should be stressed. First, orchestration strategies are also hot spots, and thus target of adaptations. Second, the admission controller used to create composite VRTs is not associated with a specific node of a tree or with a specific tree. Actually, it is associated with the resource forest. Moreover, it must be noted that a VRT may belong to more than one forest. Finally, since the composite-VRT root represents a special virtual resource (indeed a composition of heterogeneous virtual resources), composite VRTs are said to represent higher abstraction levels than primitive trees.

### 3.3 VRT Management Operations

The VRT management tasks can be summarized in four operations: Splitting, Merging, Traversing, and Adapting.

Splitting (a resource) comprises the creation of a new virtual resource in a VRT. Splitting triggers the admission control process that will verify the compliance and feasibility of the request. If the admission is successful, a virtual resource is created.

Merging comprises the creation of a new composite VRT from a set of VRTs. The new composite-VRT root is created by the orchestration process associated with a forest.

The traversing operation consists on walking through the VRT structure, visiting parent or child nodes from an initial node. Traversing also allows walking from a composite-VRT root to one of the merged virtual resources the root represents (that is, the virtual resources created during the orchestration processes), and vice versa. Hence, traversing can be done inside a VRT or between VRTs of different abstraction levels, following the orchestration path.

Finally, the adapting operation performs the insertion or replacement of resource management policies associated with an existent virtual resource in any VRT or associated with a forest.

### 4. The VRT File System

An instantiation of the VRT model demands the implementation of a complex data structure that must be flexible enough to support management policy adaptations as described in previous sections. Paradoxically, VRT implementations should be simple, since many resource orchestrations may involve computational systems with limited processing power. Thus, scalability is an important design issue in order to not constrain VRT features.

Among the several solutions we have investigated, special file systems became an interesting design solution. First, the file system abstraction enforces the hierarchical organization of data, is easy to be traversed, and has ready-to-use naming conventions. Second, file system operations form a well-known and standardized interface, with functions like `open`, `close`, `read`, `write`, `chdir` and `mkdir`, which can be easily mapped to the VRT operations, avoiding the creation of new system calls [9]. Third, special file systems can represent more than just a mechanism for maintaining files and directories. They can become more attractive when they offer APIs for user and system interaction, allowing to change internal system modules and providing powerful management information bases (MIBs). At last, the file system abstraction is found in any operating system, even in lightweight implementations, with the same regular semantic. Most of these systems allow special file system structures,
where the behavior of each operation may be redefined and where files may represent kernel memory regions.

The VRT model instantiation based on a special file system is called the VRT-FS. Its specification can be regarded as a framework for VRT implementations, since many mechanisms are intentionally left open to provide adaptability, scalability and portability. Figure 3 presents most of the VRT-FS structure.

The root of VRT-FS is the /vrt directory. There are four directories under /vrt:

- /vrt/primitive: groups primitive VRTs representing resources of the local system;
- /vrt/export: groups virtual resources of the local system that are available for participating in composite VRTs;
- /vrt/forest: groups virtual resources trees or other forests that are available for composite VRT creation;
- /vrt/composite: groups composite VRTs created from requests coming from the local system.

The following sections are dedicated to the discussion of these directories.

![Figure 3. VRT-FS structure (/vrt/primitive)](image)

4.1 Primitive VRTs in VRT-FS

Since VRTs may be scheduled in time or space, there are the /vrt/primitive/temporal and /vrt/primitive/spatial directories, respectively. Both directories have exactly the same structure, placing the primitive VRT roots as their subdirectories. Examples of such VRTs are:

- /vrt/primitive/temporal/cpu for CPU VRT;
- /vrt/primitive/temporal/nic1 for a network interface VRT;
- /vrt/primitive/spatial/mem for memory VRT;
- /vrt/primitive/temporal/hd1 and /vrt/primitive/spatial/hd1 for a hard disk, and so on.

Let us take the /vrt/primitive/temporal/cpu directory as our explanation example. Each directory created for representing virtual resources, including the root, has the following files and subdirectories:

- ./vrparams: maintains information about the virtual resource. The read operation returns the resource type and its capacity (combined with chdir, this is part of the traversal operation). The write operation in a non-root virtual resource modifies its allocated capacity, demanding a (re)admission control, as discussed further on.
- ./schedstrat; ./admpol: these are the files with the implementation of the scheduling strategy, admission strategy, and admission policies, respectively. They contain the basic resource management policies assigned to the virtual resource. Their content format may vary depending on the operating system, since they may contain programming code or advanced parameters for dynamic algorithms [10]. For example, these files could be object files to be inserted as a module into the kernel. They can also be microkernel services, software components, etc. The read operation returns the file content, and the write operation saves the file and demands the inclusion of the new management policy into the system (the adapting operation).
- ./vr1; ./vr2: these are subdirectories created to represent child virtual resources. A child virtual resource is created by means of an mkdir call (the splitting operation). The name of the new subdirectory is automatically created, following the format vrX, where X is an integer number not in use for subdirectories in the current directory. The QoS parameters for virtual resource creation are passed as mkdir parameters, and will be saved in a vrparams file inside the new subdirectory. They must be in accordance with the parameters expected by its parent resource management policies. The vrX subdirectory is created only if the admission controller associated with the parent answers affirmatively.

In fact, the admission control in VRT-FS is a two-step process (admit/commit). The admit request is triggered by the mkdir operation. If the admission controller answers affirmatively, the vrX subdirectory is created containing a single vrparams.tmp file, which contains the parameters supplied. The commit request is done renaming this file to vrparams, meaning that the virtual resource was created.

3 Higher level code for specifying the behavior of such policies is also possible (and more secure) [8][11][12].
successively. The commitment is time limited: if the file rename does not occur in a certain time interval, the vrX subdirectory will be removed. The admission process based on two steps is useful to some types of distributed QoS negotiation schemes, and avoids deadlocks on concurrent resource reservations [5].

We have aforementioned the usage of the write operation over the vrparams file. This can be a useful mechanism usually requested by a tuning process. The operation renames the file to vrparams.tmp, and then changes it writing the new parameters. While the vrparams file does not exist, the virtual resource will not be scheduled. The commitment will be done as in the previous case of admission control, by renaming the file back to vrparams. As before, the commitment is time limited. This operation can only be done over a resource if the new requested capacity is greater than the sum of the capacities of its virtual resource children. If a timeout occurs, the virtual resource must be removed, as in the previous case, but only if it has no previous child. If it is not the case, the vrparams file will have its original parameters back.

As stated in VRT model, leaves do not have resource management policies, except for the simple scheduling strategies. So, a leaf directory needs just the vrparams file. An mkdir operation in a leaf directory must return an error. A leaf can be turned into an intermediary virtual resource if the files schedstrat, admstrat and admpol are created in its directory.

4.2 Distributed orchestration in VRT-FS

The virtual resources that could participate in a composite VRT must be placed in /vrt/export.

Most file systems have the symbolic link abstraction (shortcuts can also be used), which allows the representation of a file or directory already placed in the directory tree. Thus, each virtual resource that could be exported may be represented by a symbolic link named lkX, where X is an integer number not in use for others links in the same directory. In the case of /vrt/export, links always point to another directory; a virtual resource in any VRT; or a forest. In the case of a forest, its components have to be virtual resources of the local system. Figure 4 illustrates the /vrt/export directory. The dashed lines show the path to linked virtual resources or forests.

Grouping virtual resources under a single directory facilitates the work of the file server, since shared folders are resident in a single place.

All resource forest managed by the local system must be placed in /vrt/forest. Each directory created for representing a resource forest (let us take the /vrt/forest/frt1 directory as our explanation example) has the following files and subdirectories:

- ./admstrat:./admpol: these are the files with the implementation of admission strategy, and admission policies, respectively. They contain the basic resource management policies assigned to the forest, similar to those discussed in Section 4.1. The read operation returns the file content, and the write operation saves the file and demands the inclusion of the new management policy into the system (the adapting operation).
- ./lkX: references, using the link abstraction, to another directory representing (i) a primitive or (ii) a composite VRT or (iii) another forest.

![Figure 4. VRT-FS structure (/vrt/export and /vrt/forest)](image)

Note that the linked components may be located outside the local system, so a distributed file system protocol must be used in this case.

4.3 Composite VRTs in VRT-FS

Unlike the primitive VRTs, which have their roots predefined in the VRT-FS, composite VRTs must have their root directory explicitly created below /vrt/composite.

The mkdir operation in this directory triggers the admission control, similar as in Section 4.1, but now representing an orchestration. The QoS parameters for virtual resource creation as well as the reference to the resource forest from which the composite VRT will be created are passed as mkdir parameters, and will be saved in a vrparams file inside the new subdirectory. The new created subdirectory is named cmpX, where X is an integer number not in use by other composite VRTs in the same directory. The admission control defined by the referenced resource forest is then called. The admission is also a two-step process, which in the end creates a list of virtual resources that the new composite VRT represents.

Using once more the link abstraction, each virtual resource listed by the orchestration will be represented
by a symbolic link named lkX, where X is an integer number not in use for others links in the same cmpX subdirectory. Links always point to another directory (another virtual resource in a composite or primitive VRT). Figure 5 illustrates typical composite VRTs in VRT-FS. The dashed lines show the path to linked virtual resources.

![Figure 5. VRT-FS structure (/vrt/composite)](image)

Note in Figure 5 that /vrt/composite/cmp1 represents the root of a composite VRT with no children. An mkdir operation below cmp1 will not succeed, since this root is also a leaf and does not have the required admission and scheduling mechanisms associated. On the other hand, the /vrt/composite/cmp2 directory is the root of a composite VRT with a child virtual resource (vr1). Note that it has the associated resource management policies, inserted by the creation of admpol, admstrat and schedstrat files, similar to the case of primitive VRTs. The lk1 link points to a remote virtual resource, while lk2 points to the root virtual resource of the cmp1 composite VRT.

There are some restrictions on link creation, imposed by the VRT model. A link in a composite VRT must always point to a leaf virtual resource. After a composite VRT is created, the linked leaf virtual resources cannot become an intermediary node (split). Thus, a mechanism for verifying the participation of virtual resources in existent compositions is required.

### 4.4 Other remarks on VRT-FS

The VRT-FS inherits all features of the VRT model. They provide different resource scheduling behaviors for each intermediary virtual resource, avoiding the drawbacks related to the use of a single scheduling algorithm [7]. Furthermore, with the successive divisions in capacity portions of the resource, the VRT and VRT-FS guarantee overload protection among virtual resources. Fair resource sharing with overload protection are benefits that compensate the overhead introduced by hierarchical scheduling, which have already been discussed in the literature [6][7]. Moreover, note that these benefits are not just for resource scheduling, but for a complete set of adaptable resource management policies built into the hierarchical end-to-end structure.

We have introduced more complexity in VRT-FS using file naming conventions and distributed file systems. As an alternative, these mechanisms could be provided by directory services, for resource naming and localization, and network protocols, for resource orchestration. The use of VRT-FS is more attractive, because its single structure and the well-known file system abstractions (and operations) perfectly match the requirements of the VRT model.

### 5. RELATED WORK

Eclipse/BSD [9] is an extension of a mainstream operating system aiming at incorporating “proportional resource sharing”. It proposes the /reserv file system as a uniform API for a hierarchical scheduling. Eclipse/BSD provides only static QoS policies, for scheduling and admission control. These policies can not be changed during runtime. Moreover, each resource is managed by just one scheduling strategy; and applications must know low-level parameters to request reservations. VRT-FS is in some way inspired in Eclipse/BSD in using file systems for hierarchical resource management. However, VRT-FS changes the focus to distributed resource management and adaptability. The features of VRT model override the above drawbacks.

CKRM-Linux [13] is a work in progress that intends to provide class-based resource management in Linux kernels. It proposes the /rcfs file system and the use of Workload Managers (WLMs) to define high-level goals to be satisfied. The model considers the extension of Class-aware patches, such as resource schedulers, but this feature is not implemented in the /rcfs file system. VRT comes out as a more complete resource management model, including the possibility of defining abstract resource levels (through using orchestration mechanisms), QoS mappers among resources of different levels, and admission control for resource reservation. CKRM does not clearly declare if its target is QoS provisioning or just workload differentiation, since there is no mention about admission control to prevent violations.

The “path” abstraction in Scout operating system [14] is similar to VRT resource orchestration concept. “Paths” allow an application to control resource consumption for a given communication path at all levels of an operating system. However, a path is
limited to a routing graph defined at kernel compilation
time. VRT resource orchestration presents more
dynamism since it can be performed during the system
runtime.

[6], [7] and [8] propose hierarchical CPU
scheduling in order to improve fairness when
applications with different needs are concurrently
sharing the resources. Hierarchical scheduling allows
the coexistence of different schedulers for each
identified application class. VRT is more generic since
it extends hierarchical scheduling with the association
of other resource management policies, allowing a
uniform management of heterogeneous resources.

VRT-FS main contributions come from its end-to-
end uniform model and from its adaptable and scalable
instantiation.

6. CONCLUSIONS AND FUTURE
WORK

In this paper, we have described an abstract model
for generic and adaptable resource management, based
on the Virtual Resource Tree (VRT) concept. The VRT
model provides uniform, end-to-end and adaptable
mechanisms for resource management. The model has
been continuously refined through its instantiation for
several different platforms. The paper also presented
the VRT-FS, a framework for VRT-model
instantiations in network operating systems. VRT-FS is
a special file system that offers an API for resource
management operations, such as the creation of virtual
resources; the combination of independent virtual
resources; the access to a management information
base (MIB); and the modification of resource
management policies in internal system modules.

This work opens many questions and leaves space
for further investigation. Adaptation tasks in VRT-FS
are susceptible to security and consistency issues that
must be investigated. Ideally, the solutions should be
integrated to the file system security model and
admission policies. Another point that deserves
attention is the interface and event definition for
scheduling and admission strategies. The design of
domain-specific languages (DSL) for creating such
resource management policies is a very interesting area
that may also improves security. Finally, a complete
implementation of VRT-FS into the QoSOSLinux
kernel is under development, integrated with the
QoSOS middleware. This effort is being used for
model validation and performance measurements.

7. REFERENCES

provisioning in network operating systems”. Journal of the
Brazilian Telecommunications Society, Special Issue, v.18,

creation environments”. 2nd International Workshop on
Applied Formal Methods in System Design, p. 133 - 137,
1997

Networks”. ACM SIGCOMM Computer Communications

Implementation Overview”, 1999, available online at
ftp://icaftp.epfl.ch/pub/people/almesber/pub/
tcio-current.ps.gz

Resource Management for Distributed Multimedia
Applications”, Journal on High-Speed Networking. Special

Proceedings of 2nd Symposium on Operating Systems Design
and Implementation (OSDI’96), 1996.

scheduler for multimedia operating systems”. Proceedings of
2nd Symposium on Operating System Design and
Implementation (OSDI’96), 1996.

Soft Real-Time Applications in General-Purpose Operating

Silberschatz. “Retrofitting Quality of Service into a Time-
Sharing Operating System”. Proceedings of the 1999

Dependent Scheduling Algorithm” in Proc. IFIP ATM & IP

for Implementing Process Scheduling Hierarchies”. ACM
SIGPLAN 2004 Symposium on Partial Evaluation and

Real-Time Resource Management”. 14th EuroMicro
International Conference on Real-Time Systems, June 2002

Kashyap and H. Zheng, “Improving Linux resource control
using CKRM”. Proceedings of the 2004 Linux Symposium,
v.2, 2004

the scout operating system. Proceedings of 2nd Symposium
on Operating System Design and Implementation (OSDI’96),
1996.