

# An Erlang-based hierarchical distributed VoD System \*

Juan J. Sánchez, José L. Freire, Miguel Barreiro  
Víctor M. Gulías, Javier Mosquera

University of Coruña  
Computer Science Department, LFCIA Lab  
Campus de Elviña – 15071 A Coruña (SPAIN)  
e-mail: {juanjo, freire, enano, gurias, mosky}@lfcia.org

## Abstract

Video on Demand (VoD) is a service that enables users to request any multimedia content at any time, without being constrained by any pre-established scheduling. Current commercial solutions tend to have two main problems: lack of flexibility, and high cost for a large scale deployment.

The VoD server described in this paper is based in a hierarchical architecture, implemented using a functional programming language (Erlang) and built over a cluster-based architecture (Beowulf).

The described system can be adapted with great flexibility to the underlying network topology, and scaled in order to support a large and growing number of concurrent users, all with a low cost solution, and using commodity hardware whenever desired.

After an initial system design work, where the hierarchical structure was static and composed of three specialized levels (streaming, cache and massive storage), and after the first implementations and testing, the architecture has been evolved to an arbitrary number of levels, made of modules with a known interface, thus achieving high levels of adaptability and versatility.

**Keywords:** Functional Programming, Distributed Programming, Cluster, Linux, Erlang, Video on Demand.

## 1 Introduction

A Video on Demand server (VoD), is a system that provides video services, allowing users to request VOs (Video Objects) at any time, without pre-established time restrictions.

---

\*Partially supported by FEDER TIC-1FD97-1759, Xunta de Galicia PGIDT99COM10502 and UDC 2000-5050252026

Services like movie-on-demand (MoD), distance learning tools, or interactive multimedia information services, offering personalized news based on user profiles, are just some multimedia applications examples that may use this kind of servers.

VoD systems must satisfy several key requirements, including:

- Huge storage capacity: in most applications, the system should be able to offer users a big number of multimedia objects, that should be stored in some way at the server. Even though storage capacity grows steadily for a given cost, so do user expectations.
- Support for a great amount of concurrent users: the system should be able to manage a great amount of VO requests. A single user may potentially perform several concurrent requests (for instance, multi-camera movie or conference archival). VOs may also be requested at high rate by automated tools to perform processing.
- High bandwidth: the high bandwidth usage characteristic of multimedia contents service, together with the need of serving objects to a large amount of users, makes of this one of the main requirements: the server must be able to provide a high total throughput.
- Predictable response time: when a user asks for a video object, the system should be able to give – performing statistical estimations that take the system state as input – an approximation of the waiting time. Besides, the server should try to minimize the waiting time for any user.
- Fault tolerance: with 24x7 expected uptime, there's a need for some kind of mechanisms, both hardware and software, to keep working at least in a degraded mode when some kind of fail happens. Users will expect the same kind of uptime as conventional TV (versus: they are used to some downtime on common internet-based services).

In addition to the traditional requirements for this kind of servers, we added three more, that put stricter conditions on the design decisions:

- Upwards and also downwards scalability: the system must be able to service a reduced number of users in a simple, scaled-down environment, and also capable of growing by increasing the used resources, in order to support a very large amount of concurrent users. It should be usable on very economical, commodity hardware as well as on high-end server farms and large SMP systems.
- Adaptability: a system able of adapting to the underlying topology, making an efficient use of the available network bandwidth. The topology of current DOCSIS cable and ADSL networks results in each section of the customer - provider link offering vastly different characteristics, thus there is a pressing need to optimize network resource usage to avoid saturating the slowest paths.
- Low cost, taking into account the total cost of deploying, managing and operating the system.

## 2 State of the Art

There are currently several video streaming server solutions available from RealNetworks, Microsoft, Apple, Cisco, Philips, IBM, Oracle, Kasenna and others.

While most of them are well suited for low volume video streaming on the Internet and some for corporate video services on a LAN, few try to address the difficulties in serving high volumes of multimedia contents on a large, heterogeneous network.

In general, the detailed analysis of these products, gives the conclusion that most of them represent very expensive, closed, non-scalable and non-adaptable solutions. They however tend to be turnkey solutions, ready to plug and easy to run on a small deployment.

A more detailed study of some of these solutions can be found at [10].

## 3 VoDKA: a hierarchical distributed functional VoD server

In order to cope with the previous requirements, an innovative solution, based in a distributed and hierarchical storage system [3], built over Linux clusters composed by commodity hardware [2], is proposed: VoDKA (*Video on Demand Kernel Architecture*).

The hierarchical architecture allows the division of the system functionality among the different levels of the hierarchy, giving a bigger specialization, which makes viable the satisfaction of the “a priori” incompatible requirements. For example, the complex relation between high capacity, low cost, and a very short response time, finds a solution in the layering that is described in detail in section 3.1.

The conclusions obtained after the system implementation with this first proposal, have produced an architectural design refinement, resulting in a more general solution, whose main features are detailed in section 4.

In the analysis and design stages of the system development, design patterns [7] with their adaptations to the programming language are used, and *behaviours* [6] (which represent the equivalent to the design patterns but at a lower abstraction level).

The programming language in which most development has been made is Erlang/OTP [1], which has ideal features for the implementation of monitoring, control and scheduling subsystems. In some of the I/O modules, whose performance is critical for an optimal performance of the system, C has been used instead, after an initial phase of Erlang/OTP prototypes.

In every moment, emphasis has been made in reuse, by using OpenSource tools (Linux, Erlang/OTP), or through the adoption of certain subsystems from open solutions for special functions inside the system, like protocol adaptation or working with special file formats (Quicktime), and subsystems independization (OpenMonet [8], Inets `mod_xsl` [9]). Additionally, the use of OpenSource tools has allowed for easy operating system and hardware neutrality.

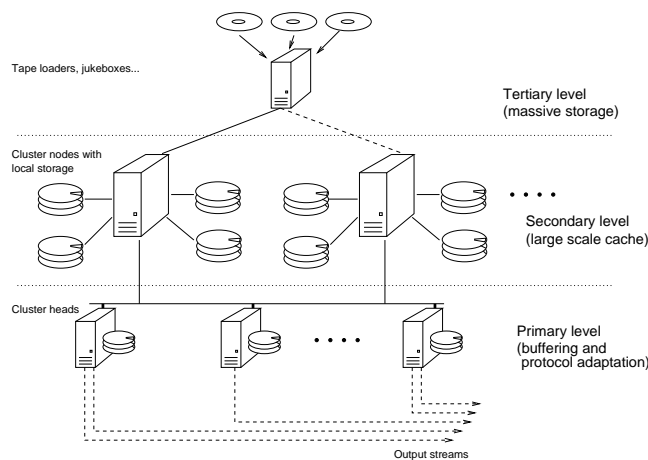


Figure 1: Initial hierarchical structure

### 3.1 Initial design proposal

The hierarchy in the initial design proposal (figure 1) was composed of three specialized levels, described bottom-up in the following paragraphs:

- **Tertiary Level (massive storage level):** massive storage level has different requirements in terms of response time, compared with the ones in the higher levels. Its main goal is to store all the available video objects in the VoD server with tape chargers, disk arrays, or any other massive storage system.

The server global performance depends on this level in terms of load time, latency, throughput, etc. Even though it's interesting to optimize these quantitative parameters, the higher level can alleviate their weights in the global performance measurements, because the secondary level acts as a cache for the tertiary one.

- **Secondary Level (cache level):** it's composed of a set of nodes, each of them with enough capacity for storing at least a complete video object. The object, read from the tertiary level, is stored temporarily in some node before being striped in the primary one. An appropriate scheduling policy should decide which are the videos that should be maintained in the cache, and for how long, in order to satisfy the future user requests and limit accesses to the massive storage level, if possible.
- **Primary Level (streaming level):** it's composed of a set of nodes in charge of protocol adaptation and sending the final video stream in the right format to the client. The interest in putting the sufficient number of nodes in this level is due to performance as well as to the need of quick reaction after a fail in any of the nodes, sending the tasks dynamically to another node, limiting in this way the losses in the quality of service. This level has important requirements both in bandwidth and in response time.

## 3.2 Key technologies

Besides the innovative architecture presented, two of the most important and differentiating features of the proposed solution are the use of a functional language and the adaptation to a cluster-based architecture.

### 3.2.1 Erlang/OTP

The used language, Erlang, has been designed and used at Ericsson for programming distributed control systems. The combination of the functional paradigm and parallel computing gives a declarative language, without side effects, and with a high level of expressiveness, abstraction and ease of prototyping.

Erlang is specially suitable for distributed, fault tolerant, soft real-time systems. It is a language based in asynchronous message passing, transparent transference of values, and higher order communications, that has the capacity of supporting a high number of concurrent processes.

The language is suited for the development of distributed systems, permits the transparent location of processes in different nodes. It also includes primitives for the support of fault tolerance and provides facilities for the replacement of code without having to stop the system.

The proposed solution also uses extensively the libraries and distributed design patterns of the Open Telecom Platform (OTP), including generic servers, supervision mechanisms, a distributed database (Mnesia) with location transparency, fragmentation, replication, and integration with the language, and a lot of useful integration libraries: SNMP, ASN.1, C Interface, Corba, Java, HTTP.

SNMP, the Inets HTTP server, the SASL support libraries, the EVA and MESH alarm and measurement handling applications, and Mnesia are used by the monitoring subsystem. The C interface, the TCP and UDP libraries and others are extensively used in the I/O and streaming layer.

There are also additional modules in development (LDAP administrative interface, user application gateway) that make extensive use of ASN.1, and the Java interface, among others.

Erlang/OTP not only provides many useful libraries and applications; there is also a rather homogeneous philosophy underlying all of OTP, so application design and interfaces is naturally coherent. A high degree of reusability and high programmer efficiency are also encouraged and made possible.

All the cited features makes of Erlang a very well suited language for the development of the proposed system.

### 3.2.2 The Linux Cluster

The use of Beowulf<sup>1</sup> clusters in the proposed solution proposed in this paper (Linux-based low cost distributed system) has been another of the keys that make proposed server an innovative architecture. The distributed memory architecture complements itself perfectly with the message passing philosophy of the chosen language. While the VoD system can run on other architectures as well (SPARC/Solaris, PowerPC/AIX), a linux cluster provides an ideal environment.

---

<sup>1</sup>Note the somewhat liberal use of the term *Beowulf*; in strict sense, the way we use Linux clusters does not make them Beowulf-class

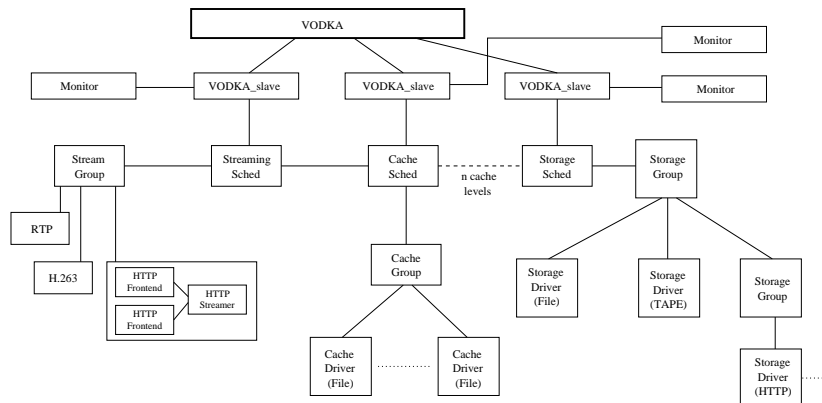


Figure 2: VoDKA Server with  $n$  levels

Some of the advantages derived of the utilization of this technology are:

- The wide experience in the OpenSource community and former work with high speed networks, distributed systems and clustering over Linux.
- Source code availability: modification of any part of the software for adapting, correcting, locating problems or instrumentation.
- Homogeneous license (GPL): easy legal treatment (versus, e.g., current MPEG4 situation, where each component has a different license, and their interaction is sometimes annoying and even contradictory).
- Compatibility: code developed with Linux can be ported without problems to Solaris, AIX, Tru64, IRIX, etc. Respect of standards (POSIX 1003.\*, SVID, 4.xBSD). Progressive commitment of commercial Unix vendors to further interoperability (SGI and IBM embracing Linux, AIX 5/L providing Linux interfaces).
- Good performance
- Wide availability of development tools
- Support for different hardware platforms (x86, Alpha, SPARC, ARM, S/390 (zSeries), IA64, SH3, MIPS, Power...)

## 4 Design refinement

The initial design ideas had to be modified in order to satisfy the needs of a production system. In particular, a design generalization was needed, because the three level fixed hierarchical architecture (streaming, cache and massive storage) can be too complex for very small installations, and too rigid for complex network topologies like the ring-based backbone that interconnects some metropolitan cable networks.

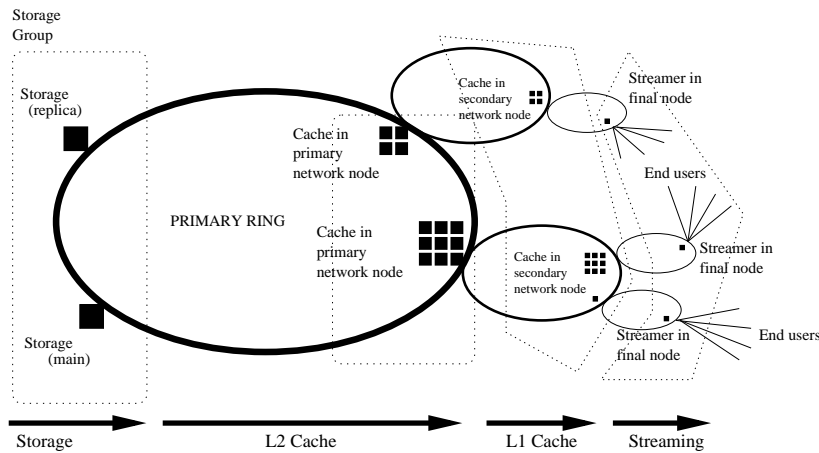


Figure 3: Configuration of the server over a complex network topology

This re-definition of the hierarchical architecture gives a new one, divided in a variable number of specialized levels, all of them sharing the same standard interface (figure 2).

This way, for example, the cache level can be absent in a given installation, or can be augmented producing a complex multilayer cache adapted to the underlying topology (figure 3) or configuring different massive storage levels physically distributed. The usual setup for a DOCSIS based cable network provides a restricted, if sufficient under normal circumstances, bandwidth from the customer to the cable headend, but plenty of available bandwidth through high speed SDH fiber rings among cable headends and the provider central switching offices. In some cases there are additional switching levels to the customer. So, in order to accommodate this situation a series of distributed storage and cache servers are deployed throughout the provider network, optimizing link characteristics.

Even, a server could be used as a storage subsystem of a different server, giving a VoD meta-server.

Besides the flexibility related to the physical distribution of the system, its necessary the system to be able to, inside each of the levels and among different ones, interact using heterogeneous storage and transference protocols. This flexibilization is based in identifying the communication patterns between the different levels, factoring them out as functional abstractions (*pipes*, for transference among the nodes) parametrized by functional closures which encapsulate the particular protocols and are established by schedulers that collaborate doing the tasks related to some work. This generalization in each level controllers ensures the utilization of generic monitorization mechanisms and reflective programming for discovering the features of each level.

It is very important to maintain the system independence of individual digital multimedia content distribution formats, currently in constant evolution, to the front-end of the VoD system, where the content distribution is actually done (streamer), providing different protocols adaptation: HTTP, RTP, etc.

The first logic layering in the VoD server is the one differentiating the video

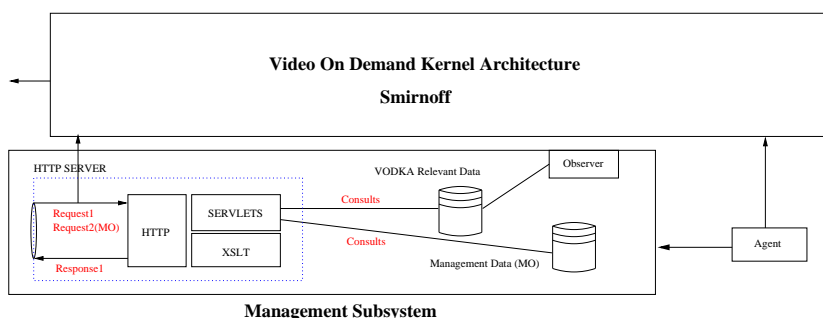


Figure 4: Separation between the VoD server and the management web application

server kernel implemented mainly with distributed functional technology and built over Beowulf clusters, and the different application servers that, using the video distribution capacities offered by the VoD server, give to the end user final services (figure 4).

The applications, developed using conventional technologies, interact with the video server redirecting the VO requests with the adequate parameters; besides, applications use the information obtained by the monitoring subsystem about the state of the system at any moment. Typical applications are cinema and television on demand, distance learning, e-commerce, interactive news, etc.

Figure 5 shows an example of how the media object flow among storage levels works until the actual streaming of the object in a simplified system configuration where the cache has been removed. In this case, the client request is received by an HTTP front-end, which defines the adaptation protocol required for a distribution of type *progressive download* over HTTP of a multimedia object. The front-end interacts with its scheduler (*Streaming Sched*) through the group in which is integrated (*Sched Group*), and decides the way in which the video stream is actually going to be distributed (DD1, in this example, instantiated to an HTTP adaptation in a port negotiated with the client). The streaming level scheduler propagates the media object request of the to its successor in the responsibility chain [7], incorporating the protocol that the storage should use for the transference (DD2, in this example a TCP/IP communication). The successor, the storage level scheduler (*Storage Sched*), propagates the request towards a storage multiplexor (*Storage Group*), connected to different storage systems: a mounted file system (*File Storage Driver*), a tape robot (*Tape Storage Driver*). The scheduler mission is to decide which source is going to be used for obtaining the media object (in the example the *File Storage Driver*, building a *pipe* that connects that data source with the transference protocol suggested by the streaming scheduler, that creates a new *pipe* for taking the storage transference and sending it using the destination suggested by the HTTP adapter.

In addition to the improvements described, the second prototype also introduced reflective generic servers and *Vodka Explorer*, a GUI tool for navigating the logical server topology and reading or changing its state.

During the I/O layer rewrite, it was noted that VoDKA performance bottlenecks were often not CPU, but purely I/O bound, thus it was decided that

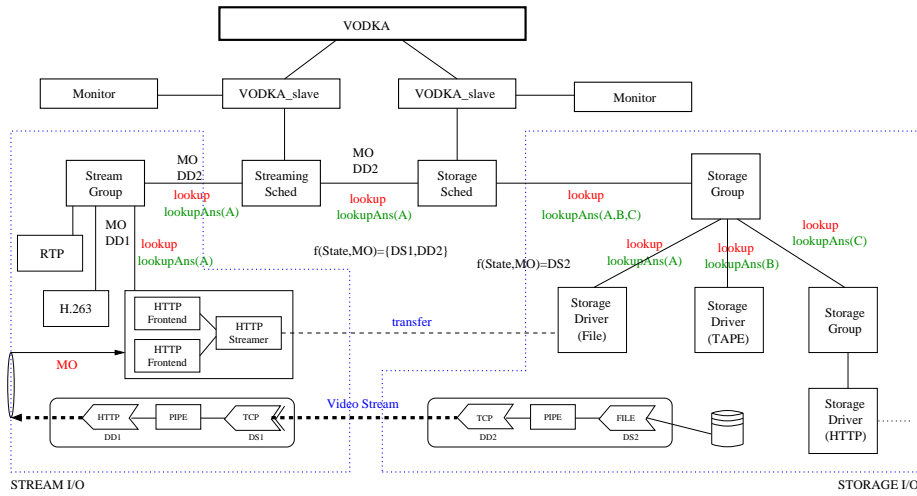


Figure 5: Example of transference in a system configuration without cache

rewriting these modules in C was not worth the effort: Erlang was fast enough, given proper care was taken.

#### 4.1 Example VoD deployment

As an example of the VoDka server configuration flexibility, figure 6 shows a schematic of the responsibilities distribution among nodes in our cluster *Borg*.

- The massive storage is in the node `borg25`, that also has an associated scheduler with two storage controllers (CD unit and tape robot with 0.5 TB of capacity).
- The storage scheduler is the successor of the cache scheduler, that is running in the node `borg24` in the responsibility chain. The cache scheduler uses as cache the aggregated bandwidth of the local cache controllers of the nodes `borg1...borg23`.
- The node `borg24` itself hosts an streaming scheduler whose successor is the cache scheduler, supporting a progressive download HTTP adapter, that gives video service to the lab using the department switched 10Mbps network.
- The server `covas` hosts a cache scheduler, whose successor is the `borg24` cache scheduler (two cache levels), and a local cache controller. Besides, the server contains an streaming scheduler fed by the cache scheduler, and supporting an progressive download HTTP adapter, using the ATM adapter that is directly connected to the university backbone.
- `borg0`, the *Borg* cluster front-end, is used for the system monitorization.
- One of the nodes is a SPARCstation running Linux, while the rest of the nodes are x86 machines also running Linux and `covas` is a Sun UltraEnterprise 3000 running Solaris, demonstrating system portability.

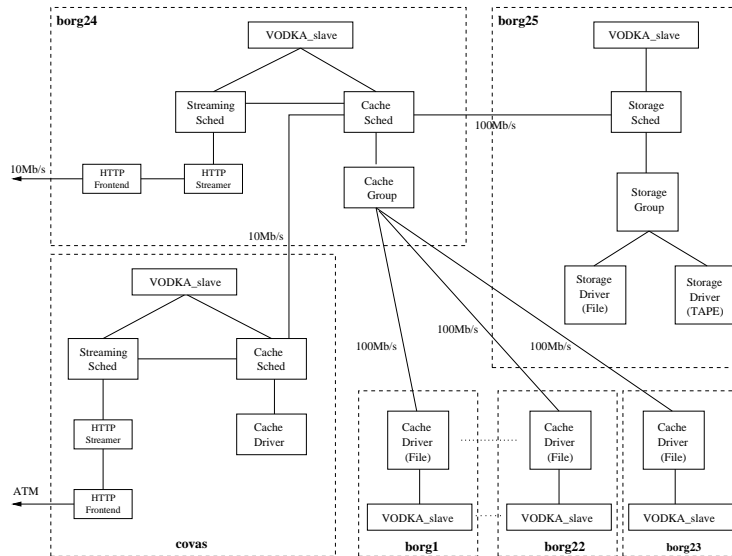


Figure 6: *Borg* configuration in the University Campus

## 5 Conclusions

A video on demand server that meets the basic requirements of such a system has been developed: great storage capacity, high bandwidth, predictable response time, large number of concurrent users and fault tolerance.

Evolving this system, in a second stage, the solution meets now other important features: scalability in both directions, adaptability to different network complexities and topologies, and low cost. This has been possible in part by using key technologies like the Erlang programming language, design patterns knowledge, and Beowulf clusters.

Currently, the system is being refined, paying special attention to the scheduling subsystem, and the addition of new modules for the support of different formats and storage protocols and distribution of multimedia objects. An important amount of work is also being done in the implementation of external applications, that are going to interact with the end user and the server.

## References

- [1] J. Armstrong, R. Virding, C. Wikström, M. Williams. *Concurrent Programming in Erlang*. Second Edition, Prentice-Hall. 1996.
- [2] M. Barreiro, V. M. Gulías, Cluster setup and its administration. In Rajkumar Buyya, editor, *High Performance Cluster Computing*, Vol. I. Prentice Hall, 1999.
- [3] S. G. Chan and F. Tobagi Hierarchical storage systems for interactive Video-on-demand Technical Report, Stanford University, Computer Systems Laboratory, Number CSL-TR-97-723, p. 84. 1997

- [4] Cisco Systems, Inc. *A Distributed Video Server Architecture for Flexible Enterprise-Wide Video Delivery* en [http://www.cisco.com/warp/public/cc/pd/mxsv/iptv3400/tech/dvsa\\_wp.htm](http://www.cisco.com/warp/public/cc/pd/mxsv/iptv3400/tech/dvsa_wp.htm), White Paper, 2000.
- [5] D. Du, J. Hsieh, J. Liu, *Building Video-on-Demand servers Using Shared-Memory Multiprocessors*. Distributed Multimedia Research Center and Computer Science Department, University of Minnesota, and Ronald J. Vetter, Computer Science Department, North Dakota State University. 1996.
- [6] U. Ekström, *Design Patterns for simulation in ERLANG/OTP*. Master Thesis, Uppsala University, Sweden, 2000
- [7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison Wesley, Reading, 1996.
- [8] LFCIA, OpenMonet Project. <http://sourceforge.net/projects/openmonet>
- [9] LFCIA, Erlatron Project. <http://sourceforge.net/projects/modxsl>
- [10] J.J. Sánchez, V.M. Gulías, A. Valderruten, J. Mosquera. *State of the Art and Design of VOD Systems*. Proceedings of the *International Conference on Information Systems Analysis, SCI'00-ISAS'00*. ISBN 980-07-6694-4, Orlando, USA, July 2000.