An Operating System Suited for Integrated Broadband Communications Networks

T. Vardanega

formerly
Intecs Sistemi
Operating Systems Section
Via Fratti, 14 - 56125 Pisa
Italy

currently
ESA/ESTEC
WDI Section
Postbus 299 - 2200 AG Noordwijk
The Netherlands

Abstract

Broadband communications technologies are close to maturity. Substantial changes can be therefore anticipated in the software architecture of next-generation computer networks. The consequences will be numerous. Amongst them, the classic separation between local- and wide-area networks is bound to become impractical under the future circumstances. This paper pursues this argument by introducing the design of an innovative operating system suited for future broadband networks. Special focus lays on properties of distribution, openness and scalability.

1. Introduction

For the purposes of this paper, we assume an on-line environment to be a telecommunications-oriented operating system equipped with special tools, for use in the control elements of integrated broadband communications (IBC) networks. The IBC network is a telecommunications network based on the asynchronous transfer mode (ATM) that promises very high transmission speed and bandwidth. IBC networks are seen as the evolution from the Integrated Services Digital Networks (ISDN) and their broadband enhancement (B-ISDN). The motivation behind the step to IBC network stays in the increasing dependence of human activities on intensive use and interchange of a growing variety of communications services. High performance networks therefore are real prerequisite for future competitive activities. Programmes which aim at the exploitation of broadband technologies are being sponsored by a number of countries, bodies and organisations, world-wide, including the United States, the Commission of the European Communities and Japan (see for example [7] and [8]).

The transition to the IBC network is fairly complex however. It involves a number of issues ranging from pure questions of hardware and software technology to the definition and introduction of suitable standards. The topics addressed by this paper basically fall in the area of the software technology. In this area, numerous evolutionary aspects call for special attention. There are reasons for that. Amongst them, noteworthy, the demands from the design of an on-line environment for future IBC networks do show clear similarities with the issues of Open Distributed Processing (ODP). ODP has already become a keyword in the world of information technology. This event may thus lead to remarkable opportunities of technology interchange between the two communities. (As to the subject of ODP, the interested reader may refer to [3] and [5].)

There are technical and practical reasons which enforce future on-line environments to be constructed around operating systems that are more and more distributed and real-time. The need for distribution is a matter of fact: Given the state of technology, in fact, the architecture of the switching systems very much benefits from physical distribution. Especially to best achieve the degree of performance and robustness which characterises the nature of a modern telecommunications environment. The need for real-time, instead, originates from obvious reasons of performance.

The possibility to dramatically increase transmission bandwidth and speed of future communications networks, tends to break the confines between loosely and tightly coupled systems. This causes the nature of control systems for telecommunications networks to be reconsidered in the light of this event. First of all, it should be avoided
that the actual bottle-neck move from the physical transmission network to the control processing (as it is likely to be the case if for example fibre optics media are operated in conjunction with the current generation of control elements). Secondly, it should (or, at least, could) be redefined the notion of the domain of allocation of control and service software, from the closed confines of one given Control and Service Element (CSE) to any suitably wider area., Thus node's boundaries can eventually be crossed. They were often motivated by presumed reasons of performance, in addition to obvious questions of privacy and confidentiality (which will still hold, anyway).

The former argument, also in view of the increasing demand from the IBC user, motivates the following requirement: telecommunications systems must ensure continuity of service and offer overload control capabilities. The system must be permanently operational, whatever the load conditions, and the services provided within fixed time intervals. Moreover, large part of the operations of software maintenance, including service upgrade, shall be performed on-line, without disturbing the activities which carry user traffic. (This results in the on-line environment to include tools for on-line software replacement and extension.)

The latter point, instead, combined with the requirement for on-line software replacement and extension, favours aspects of design openness to play a significant role in the conception of future IBC networks.

The issues presented in this paper cover key aspects of the "Integrated Broadband Communications On-Line Environment" (IOLE) Project. The IOLE Project formed part of the multi-annual "R&D in Advanced Communications Technology in Europe" (RACE) Programme, launched by the Commission of the European Communities in 1988. The aim was at favouring the introduction of a communications infrastructure based on broadband means, for use between the European countries.

The work performed by the IOLE Project just places in the above scenario.

This paper is structured as follows: In section 2 the requirements that the IOLE has to meet, are discussed. In section 3 the design principles adopted in the definition of the IOLE architecture are presented, together with the main technical solutions. Section 4 surveys the key features of the IOLE operating system, while section 5 discusses how the tools for on-line replacement and extension have been integrated with the operating system. In section 6, the most innovative achievements are presented. Section 7, instead, describes the results obtained with the implementation of the prototype. Section 8, finally, provides some concluding remarks.

2. Requirements capturing

To best focus the design of the on-line environment, a comprehensive set of requirements was captured at the project start. Some of the captured requirements directly originate from the state of the art ISDNs: Real-time processing, distributed control, continuity of service, and overload control capabilities in fact represent the key features that modern ISDNs call for. Other requirements result from the increasing expectations on IBC as they are currently being anticipated by potential customers and providers. These requirements include: Scalability, since the geographical area interested in the IBCN services is expected to rapidly and continuously grow; adaptability, because the type, the power, and the scattered location of the network nodes and their working conditions are bound to be fairly diverse over the IBC network; on-line extendability, for progress in hardware and software technology as well as changes in the user demand, will call for replacement and extension of hardware and software modules to be performed on-line.

Due to the former set of requirements, many telecom providers have often searched for custom-made operating systems. Thus their products are tailored around ad-hoc solutions to reach for the required degree of distribution, real-time and fault-tolerance. This results in fixed, pre-determined, "closed" systems. Just in this scenario, the IOLE Project placed its aim to design, in the spirit of RACE, an open, non-proprietary on-line environment capable of meeting the above requirements with a coherent and comprehensive approach. The IOLE aim at being an open system is particularly important since the actual processing requirements on the on-line environment will not be established once and for all, and met by use of fixed, pre-determined measures either.

The latter set of requirements, instead, is yet quite unexplored in the current scene of public domain telecommunications. No existing telecom-oriented operating system, as far as known at least, appears to grant sufficient coverage of those features. Furthermore, such requirements as scalability, adaptability and extendability are very well suited to exercise ODP techniques. These techniques have been employed in the design of the IOLE operating system and encouraging results have been achieved.

3. Key design aspects

The on-line environment shall support the whole spectrum of telecommunications software, ranging from basic call handling and maintenance to value added
services. The very core of the on-line environment is an advanced distributed operating system with real-time features.

This operating system is characterised by:

i. an on-line interface general enough to allow the user to view and access the network as a unique distributed platform;
ii. basic fault-tolerance mechanisms to support the implementation of both transaction-oriented and replication-oriented algorithms and policies;
iii. advanced tools, completely integrated with the IOLE operating system, for on-line function replacement and extension;
iv. support for selective monitoring of events in the system.

In the following, the IOLE operating system is presented from two different perspectives: One, which fits in the abstract view of the telecommunications software designer. The other, more implementation-oriented, which offers a view on the key design principles behind the IOLE operating system.

3.1. The architecture model

With the foreseen IBC network, the difficulties for the IBC programmer to get a comprehensive view of the system are bound to grow dramatically. This is primarily due to the factor of increasing distribution that will characterise the IBC network. In particular, the amount, the nature and the scale of the network control elements.

For a fruitful transition to IBC therefore the future user of the on-line environment needs help to master its complexity as well as to exploit its enormous potential. To this end, an overall architecture that can offer a suitable view of the on-line environment and ease the design of telecommunications software, is required. The architecture of the on-line environment must be powerful enough to face the increasing demand for modelling flexibility of the future telecommunications software. Nevertheless, this architecture must not be stringent as the IBC network shall be open to progress in technology and provision of new services. Thus the on-line environment must be an open system. (A similar argument is sustained in [1].) It is anticipated that the introduction of IBC will lead to a high degree of user control, which is a significant aspect of openness itself. Moreover, it should be possible for the software designer to benefit from the distributed nature of the IBC network. The IOLE aim was at doing by use of ODP techniques. In particular, to feature access, location, failure and replication transparency.

The architecture proposed for IOLE seems to meet the above requirements. The principles behind the IOLE architecture are centred around one key assumption: By use of the ATM, it should be possible to separate the control network from the user network, thus having two separate planes, one for the control information, one for the user traffic. The possibility to achieve this separation provides for more flexible criteria to model the control network. This modelling flexibility allows one to define logical networks largely independent from the topology of the underlying physical network. Logical networks could be shaped so as to meet specific needs of the control and service software. Moreover, since the control and service software may be subjected to on-line extension and replacement, shape and feature of a logical network might even change over time upon such circumstances.

The physical network consists of sparse, interconnected network nodes. Each of these nodes includes one or more CSE. Nowadays, the nature of such CSEs already favours some degree of distribution of control. The CSEs, in fact, are composed of a variable number of interconnected processing units. The interconnection between these units may be either the transport network itself or an ad-hoc high performance intra-node interconnection network.

Based on the above understanding, the IOLE architecture recognises three levels of abstraction in the construction of the logical network:

a. The first level is that of the individual processing units that compose the CSEs. These are termed the Computing Elements (CEs). They are self-contained computers including processing, storage and communications resources. Part of these resources may be made available for use by selected remote CEs. Other resources instead may well be for private use in the local processing of the CE.

b. The second level originates from the functional interconnection of selected CEs to form an entity called the Logical Processor (LP). By functional interconnection, we mean here a partnership dictated by functional requirements and supported by information exchange thru the control plane.

c. The third and last level of abstraction is determined by the composition of LPs to form the logical network. The software layer running on top of the IOLE operating system shall provide for complete network transparency. Once a telecommunications service is made available at the network level, it becomes a network object. In concrete terms, a network object is an aggregate of IOLE objects which may run on any number of LPs, and cooperate by means of appropriate user-defined protocols.

The network is seen by the IBC user as a collection of services; the network configuration management, instead, views it as an organised collection of LPs supporting a
number of services. Hierarchies of services may be easily achieved. In the user view, any such hierarchy would result in network objects calling one another's services in a predetermined order. In implementation terms, instead, this would correspond to object(s) on one LP to cooperate peer objects of other LPs in order to carry out a given service protocol. Such cooperation takes place via exchange of request-reply messages.

Consequence of the above is a relatively easy construction of hierarchies of network services. This in turn leads to easy modelling of multi-media services (which is perhaps the most promising feature of future IBC networks).

One might object that the pattern of cooperation among network objects could be acceptable for distributed domains confined over local area networks, hardly suitable for networks covering such wide areas as the IBC networks are meant to. Answer to this point has been given earlier in this paper. The assumption is made that for networks based on broadband means the area to cover is not a factor to influence performance or reliability.

![Configuration of a logical network](image)

The configuration of one LP, that is the decision as to which CEs should be chosen from an existing physical placement, may respond to demands from the IBC software. This assumption should hold both in the case of an already existing configuration of an LP, where the new software may be allocated quite freely on selected CEs (providing of course the required resources are available and they can be actually allocated), and when a new ad-hoc LP is to be created for example to run new services. In the latter case, new CEs, as appropriate, might be added to the existing physical configuration. It will be then the responsibility of the booting procedure of the IOLE operating system, driven by the so-called "configuration plan", to call for and coordinate with all the CEs that form part of the new LP.

The configuration plan is an off-line prepared region of memory, accessible by the boot procedure, where all the configuration information are stored.

The concept of the LP allows services to be designed and implemented as much as possible independently of the physical network configuration. This implies, for example, that instead of accepting the control software provided by an exchange provider, a PTT could purchase it independently (or even develop it independently). This splits for the first time the provision of hardware and software for the exchanges, which could result in
significant savings for a PTT. (This remarkable argument is discussed in detail in [4].)

The LP may be viewed as a domain of rights. For example, charging and routing services are likely to be provided on the basis of single charging areas, while other dependencies could be national, phone company, continent, or provider. One could even conceive a situation where an LP is set "with random characteristics" like for example a LP for a statistic package to be run by some regulator, maybe the CCITT, that might want to poll traffic at several random points across the network. Of course, it still is possible to assume that most services will be provided locally. Under such conservative assumptions, of course, the LP shall be modelled as a single CSE.

The concept of LP allows one to set a clear separation between the concerns of software configuration and those of the actual processing. Future telecommunications software should be specified in terms of two separate parts: one part should drive the search for the best suited configuration, conforming the state of the network. The other part, completely configuration-independent, should consist of a software description modelled in terms of the location-independent abstractions supported by the on-line environment. Ideally, in the presence of a sophisticated software development support environment, the software configuration should be automatically determined in response to explicit configuration requirements from the software itself. Whatever the final configuration is the one authorised by network management authorities, should not interfere with the design of the software to run. The concept of the LP, as supported by the IOLE operating system, offers a real implementation of a distributed uniform platform. This allows the telecommunications software to be designed in exactly the same manner even if differently configured. The factors that can influence software configuration (for example allocation requirements and communications requirements) are seen as special attributes of the IOLE objects. Upon load and configuration, such attributes do not influence the actual processing.

3.2. Openness and distribution

The on-line environment must hide the distributed nature of the IBC network from the user who is no longer bound to mind where in the network his software is stored and executed. The IBC network appears to be an aggregate of interconnected LPs. Each LP virtually behaves like a single, very powerful processor. The on-line environment however provides also primitives to plant storage and execution of the application software onto specified locations and devices.

The overall consistence of the system is ensured by the IOLE operating system running on every network control element. An instance of IOLE has:

i. to construct and maintain one LP,
ii. to support and manage the abstractions which run on the LP, and
iii. to enable cooperation between remote LP's.

The IBC requirements discussed in section 2 above put stringent demands on the IOLE operating system. In order to meet such requirements, a layered architecture has been adopted. At each layer selected requirements are addressed. The match of the features of these layers against the various requirements has been driven by the aim to balance efficiency and flexibility. (In any future design review process, any of the two could be then traded against the other.)

The architecture of the IOLE operating system is organised around the following layers.

a. The CE-Kernel Layer, which resides directly on the CE hardware. It may be any suitable off-the-shelf operating system kernel. In any case, it must be a real-time kernel offering location-transparent message passing within the confines of its domain. The CE-Kernel must offer the possibility to control all logical and physical resources which are bound to the CE. For the sake of openness to multi-vendor, it needn't offer a standardised interface. Explicit recommendations have been made in IOLE, however, to determine the feasibility of candidate kernels. Low-level scheduling is performed at this level upon directives coming from the levels above. High-level scheduling, instead, that is allocation of application software to selected CEs is entirely on policies actuated at the level of network management.

b. The so-called Adaption Layer, which resides on top of the CE-Kernel. This is aimed to offer a uniform interface to the layer above; this is done by smoothing the possible heterogeneity of the various CE-Kernels. The motivation for this Adaption Layer is basically to give one the freedom to use any suitable off-the-shelf kernel as the CE-Kernel; the uniform interface required by the level above, puts special demands which can be hardly met by generic kernels; in particular, the availability of features to group and maintain aggregate of related abstractions (for example groups of communications ports) is strictly needed: Should not those features be supplied by the native kernel, they shall be implemented in the Adaption Layer.

c. The Operating System Layer, which is meant to offer the ultimate interface of the on-line environment to the user. Thanks to the interface supplied by the Adaption Layer, the Operating System Layer is portable. This

433
layer must support and maintain the concept of LP, together with all the abstractions offered to the on-line environment user. As already mentioned, the LP is an implementation of a distributed platform which offers access, location and replication transparency; granting these features is the special responsibility of the Operating System Layer. This requires that properties of scalability, reliability and LP-wide distributed resource management are shown. Moreover, mechanisms for use in the implementation of fault tolerance policies must be supplied at this layer. Separation between mechanisms and policies is maintained in IOLE: IOLE provides for mechanisms, whereas policies - which very much depend on the service - shall be implemented within the service. In particular, at the Operating System Layer, IOLE offers transparent management of groups of replicated objects as well as protocols for atomic broadcast. Monitoring schemes and related policies instead are assumed to be user-provided. Transaction management is also offered by IOLE which features the possibility to define atomic transactions, to preserve the system timeliness, transactions may be associated with time deadlines and be seen as schedulable objects themselves.

3.3. The programming model

Before entering into the implementation details, a few words should be spent to briefly outline the programming model supported by the IOLE operating system. The main object offered by the IOLE operating system for the application structuring is the component. Strictly speaking, the component is a protected, reliable distributed, processing environment which must be completely allocated to one LP. The component is meant to incorporate part of the processing resources, protocols and policies which are needed to run an IBC service; one service may be implemented out of one or many cooperating components, where this decision is left to the service designer. The presence of a set of components aiming to offer a telecommunications service must be declared to and authorised by the network management authority; upon authorisation, that set of components become a network object, which, in the IOLE terminology, is called the subsystem.

Components may communicate with peer entities only, by means of explicit message passing. One component is allowed to communicate with another by declaring to export a communications interface which matches the interface imported by the partner component(s). This is checked for at configuration time; upon a positive result, the two communicating components are given the capabilities ("tokens") to initiate their communications. The on-line replacement and extension tools will operate on such communications interfaces in a manner completely transparent to the communicating components. Both the exported and the imported interface are completely expressed in location-independent terms.

The actual processing activities in charge of the component are performed by the subcomponents. One component may be implemented out of any number of subcomponents. Subcomponents are processing objects allocated to the CEs. On the CE, the subcomponent is the unit of protection, the unit of replication for replication-based fault tolerance measures, and the finest-grained unit subject of on-line extension or replacement.

The communications scheme for subcomponents is two-fold: (i) For communications which are confined within one component, subcomponents may exchange asynchronous messages with one another or invoke remote procedure calls. (ii) Being the actual processing entities, the subcomponents also perform the actual communications between components, no matter whether remote or not; in this case, every such message is treated, routed and delivered according to the information stored in the communications interfaces belonging to the originating and receiving components.

The scheduling of subcomponents on one CE is performed by the CE-Kernel. If needed, subcomponents can be associated with deadlines (whether hard or soft); the CE-Kernel and its scheduling policies are assumed to ensure predictable behaviour, on a "best effort" base. This implies that the various activities to be performed on a CE are prioritised according to some allocation planning established as much as possible off-line; the allocation of priorities is aimed to ensure that critical deadlines are always met, perhaps for the price of missing others which are not critical ones. It is the responsibility of the CE-Kernel to schedule the activities according to their current priority and, if needed, change dynamically certain priorities to ensure fulfilment of critical deadlines especially aimed to overcome the priority inversion problem. (As to the subject of predictability, see for example [10] and [11].)

Finally, assuming the CE-Kernel allows it, multiple threads may also run within the subcomponent; the presence of threads helps when a finer-grained processing capability is required to perform telecommunications functions; the control of multiple lines or trunks attached to the same control element, for example, may benefit from many threads of execution, which run in parallel, perform the same function, and share to the same context (for example tariffs and configuration parameters).

A schematic mapping of the above objects against the IOLE architecture is shown in figure 2.
4. Meeting the requirements

A quick look into the most important aspects that one should consider in the design of an operating system for IBC network is presented in the following; this helps better capture the most advanced aspects of the IOLE design; moreover, it allows one to understand to what extent off-the-shelf products can be used in the generic implementation of an on-line environment for IBC.

Real-Time

The IOLE operating system has to be real-time (more appropriately, it must ensure predictability in the presence of deadlines). To this end, the subcomponents are specified in association with "attributes" which include the applicable deadlines, if any. The scheduling supported by the CE-Kernel is therefore, priority-based and deadline-driven; the CE-Kernel should then work on a best-effort base. The allocation of priorities on the local base is dynamic and regulated by use of the Basic Priority

Fault-Tolerance

The IOLE operating system has to be fault-tolerant; more accurately, it must ensure service continuity in the face of hardware failures. (It is noteworthy that software faults are also a big source of problems; adequate techniques for software validation and maintenance are assumed in this case). To this end, the objects defined in the IOLE programming model are associated with special

Inheritance (BPI) protocol [9]. A variety of for algorithms based priority inheritance allow to manage situations where contention for shared resources may cause task blocking. The BPI protocol bounds the period of time a higher priority activity can be blocked by low priority ones (as in the case of priority inversion).

Distribution

The IOLE operating system has to be distributed (more precisely, it must offer a distributed platform). It is naturally distributed for it must control a set of variously interconnected CEs. The foundation for the support of distribution comes with the use of CE-Kernels which support message passing across remote CEs. On top of this, the modules of the IOLE operating system implement distributed resource management for all the LP-bound resources. To this end, no centralisation schemes are adopted; the system modules exchange messages with one another according to a protocol which allows them to achieve distributed consensus and to maintain the control information updated.

Figure 2: The IOLE processing objects.
The IOLE operating system has to be scalable. This requirement, indeed, properly applies to the Operating System Layer: The performance and effectiveness of its functioning must not suffer from growth in the system scale, for example in terms of interconnected CEs; moreover, it must be possible to add functions to the system and to implement policies on top of it. To this end, the Operating System Layer is structured in terms of self-contained modules which co-operate out of simple protocols based on location-transparent message passing. Every such module is replicated on every CE. All system data are completely encapsulated in such modules; their ownership is determined on the base of usage; a given module encapsulates the data it is expected to access more frequently. All the interactions between the system modules are kept as much as possible location-independent; to achieve this, it plays a crucial role the configuration procedure: At the booting time, a "configuration plan" is passed to the module which is the responsible for the system configuration; the configuration plan determines the allocation of the system modules and gives them the minimum configuration information to enable their co-operation; eventually, the various modules are also passed the control data they are made responsible for. (Cf figure 3.) In this scenario, the addition of a new CE in the LP only requires that instances of a few modules are started on that CE. At their booting-time then, they are passed the configuration information.

Figure 3: Initial configuration of the Operating System Layer.

5. Replacing software on-line

The issues related with on-line replacement and extension are complex and manifold; most of them, however, fall beyond the scope of this paper; what briefly outlined here aims at giving the flavour of the IOLE potential.

The IOLE operating system must enable the replacement and extension of software to be performed on-line; this means without incurring service interruption and with the minimum service degradation. The provision of tools for on-line software replacement, extension and test is one of the most important features shown by the IOLE Project. The unit subject of on-line extension is the component, for it is defined to be the building block in the design of a telecommunications service. The process of replacement or extension will be then performed in a stepwise manner on the various subcomponents present in the component.

On-line replacement and extension become feasible when two basic assumptions hold:

i. There must exist a state where the component is extendable; when the component reaches that state, an appropriate controller (run by the on-line environment) is informed. It is assumed that the software designer is aware that his/her software will be subject of extension/replacement.

ii. The new component must offer its own state transfer function which maps the state and corresponding state variables into the form needed by the new component. Of course, depending on the definition of the extendable state the state transfer function may well be an empty one.

To better capture the notion of state as applicable in the above scenario, a few more words are needed. Here it is made the assumption that a significant part of the component's state may be stored in data-bases; the rest, instead, typically consists of temporary information related to the communications activities being performed.
by the component (for example call-handling information). The part of the state which is stored in databases is gathered by the new component by means of its own state transfer function. Furthermore, if it is considered that some temporary information forms part of the state, too, it must be transferred from the old to the new component; this transfer, however, should be performed as transparent as possible, to avoid the danger of inaccuracy.

The requirement of transparent migration of communications-related information put by on-line replacement and extension on the IOLE operating system is met by exploiting the features of the IOLE communications model. The IOLE model, in fact, allows the operating system to transparently exercise special control over the communications activities. Components are specified with a well defined interface. This interface (termed the component communications interface) specifies the communications requirements of the component. Messages are sent to destinations whose unique identifier may be known by querying the IOLE operating system, being supplied with the required "token"; messages can be read from special places called the communications ports.

Each such communications port is associated with a queue which stores the received messages still to be read; the size of the queue is determined at the configuration time; upon reading, the message is removed from the queue. These communications ports show two separate aspects: The physical communications ports are not owned by the component; though being accounted among the component resources, they are completely kept under the control of the IOLE operating system. The logical communications ports, instead, represent capabilities that designate one component as the potential destination of certain messages. There exists a tightly coupled association between the component and its logical communications port(s); strictly speaking, this means that the communications interface can only change with the change of the component (extension). There exists, instead, a loosely coupled association between the logical and the physical ports; this implies that such an association may change transparently to the component, as planned and operated during the process of extension or replacement; in practice, the physical communications port can be migrated, with the option to keep or clear the queued messages, from the space of one component to the space of another. This process is completely transparent to the involved components. Of course, if no physical port corresponds to a logical port, a reading attempt would result in a failure; this case, however, is not expected to happen because the change of association and the migration of physical ports are only permitted during the process of replacement or extension. (In that case, the old component is forced to terminate, thus impeding any further attempt of reading from the migrated ports.)

6. The key to IOLE: The agents

Two aspects of the IOLE implementation deserve special attention since they incorporate most of the novelty which comes with the IOLE design. Both aspects are warranted by the notion of "remote agent"; the resulting IOLE concepts are termed the control thread, for use in the application modelling, and the configuration agent, for use in the system modelling.

By construction, the IOLE operating system may be seen as an enhancement of a distributed kernel which is assumed to provide transparent message passing across remote sites. This sort of dependence is the result of a precise design decision: In order for designer of the IOLE operating system to build up an open, distributed and scalable system, in fact, such features as data encapsulation, message passing, and location transparency are of prime importance. Nowadays, these features are often incorporated in the kernel of some state of the art operating systems (cf for example [2]; for further readings on the subject, the reader is also referenced to [6]). The IOLE architecture has been designed just taking this point into account: The Operating System Layer, in fact, can be constructed by employing abstractions supplied by most commercial kernels like multi-threaded processes (actors) communicating via explicit, location-transparent message passing. It is fundamentally thanks to the control thread and the configuration agent that IOLE can warrant its major features, that is scalability and openness.

6.1. Achieving scalability

The property of system scalability can be achieved by modelling the Operating System Layer in terms of virtually identical modules which operate in virtually the same environment regardless of their actual physical placement (cf figure 5); it is clear that such a construction is not expected to suffer from modifications to the system scale, since the addition of a new site in the system would just result in a standard set of system modules to be started in the new site, and supplied with a "system map" to instantiate the necessary links and routes for the communications with the rest of the system. The most innovative IOLE feature stays in the possibility to construct the logical network independently of the physical network; it therefore appears that the above design becomes feasible when complete location
independence and access transparence are granted for the functioning of as many as possible of the system modules. This is achieved by defining special entities (very few, indeed) aimed at maintaining and controlling the required system configuration on every site of the LP. The Configuration Manager is in charge of setting up and maintaining the system configuration. The Configuration Manager is also responsible for the maintenance of the application software. A single instance of it is started on one (CE) per Logical Processor. This is normally the site from which the system is booted (the so-called master-CE). Then, on every CE of the LP, an "agent" of the Configuration Manager is created. This is called the configuration agent. It is in charge of ensuring the correct boot of the Operating System Layer on all sites of the LP. Having done that, the configuration agent starts to do its actual job: It traps the messages that the other modules of the operating system send to the Configuration Manager; these are normally permission requests to operate changes on the configuration of their site. The configuration agent however maintains only a subset of the complete system map. It therefore tries to serve the request locally first. If it is unable, then forwards it to the Configuration Manager. In the course of this process, the configuration agent can update the configuration information that were found to be out of date or even missing. In this way, the set of configuration agent maintain a sort of distributed system cache. The presence of the configuration agent is intended for a less heavy management of the system configuration. It also reduces the number of processes, the amount of resources, and the amount of data to be maintained on each CE. Furthermore, the configuration agent can also perform a sort of 'are you still alive?' polling towards the Configuration Manager. Based on that, it is possible to implement recovery algorithms to face failures at the master site. Since the configuration agents maintain the configuration information being used more recently, a sort of partitioned cache is also achieved which may be used by the recovery procedure to restore the new master.

6.2. Achieving openness

By its very nature, the property of system openness applies to both the design of the Operating System Layer and, more important, the design of the control and service software that the IOLE operating system has to run. Openness at the level of the programming model for the control and service software is achieved via the concept of the control thread. The IOLE operating system is in practice an enhancement of a distributed kernel; this implies that the low level programming model is fixed in the kernel. IOLE therefore has to set a mapping between its own programming model (cf section 3 above) and that of the underlying kernel. This mapping is basically established, operated and maintained in the Operating System Layer. What is very fundamental to IOLE is that a number of modifications to the system configuration can be performed transparently; think for example of on-line replacement or extension.

A very significant case of modification in the system configuration is when changes are made to the communications interfaces of given components. These changes might reflect for example the change of version, the addition or the cancellation of a network service. Remarkably, such modifications should neither be perceived, nor performed nor controlled by the application itself. These actions in fact are performed by the IOLE operating system in response to requests coming from authoritative sources, say the network operator. The concept of the control thread is the key means to achieve this. The control thread may be seen as an "agent" of the Configuration Manager for all aspects of process management. It, in fact, forms part of the application component. One control thread per component, running in the component's space, transparently to the application itself. The control thread is in charge of performing, from inside the component, all the actions which can lead to modifications of the application status and configuration. As a result of this, the application components can be transparently mapped on the low level programming model supported by the CE-kernel. It is up to the Operating System Layer then to ensure that the resulting objects can be managed in accordance with the IOLE programming model. On the one hand this allows one to maintain the level of protection and data encapsulation as they are granted by the off-the-shelf CE-kernel in use. On the other hand, it gives external (authorised) entities the possibility to operate on the configuration of the application component, transparently to the application itself. This is the key to on-line extension and replacement.

IOLE is a system based on two different levels of communications. One level is that of the IOLE programming model as it is offered to the application. The other level is for restricted use by the Operating System Layer, and relies on the support of the native CE-kernel. The former is fundamentally a virtualisation of the latter.

Thus it can be understood why the calls to the IOLE operating system primitives result in a transparent message exchange between the invoking application component and the kernel process involved in the primitive service. Because of the presence of the control thread, this message exchange may be a triangular one, also involving the control thread in addition to the two parties mentioned above. The intriguing point is that this

438
can be kept completely transparent to the application itself, and, in the presence of an efficient kernel, introduces a very limited overhead. An example of resolution of calls of IOLE primitives is shown in figure 4 below.

![Component diagram](image)

Figure 4: Transparent resolution of IOLE primitive calls

7. Project status

The IOLE Project started in January 1988 and finished in March 1991. It was carried out by a consortium of seven industrial partners from Germany, France, Italy, Spain and Austria. The overall project goals by far exceeded the design and implementation of the on-line environment described in this paper; substantial effort was devoted in particular to the definition of certifiable methodologies for on-line replacement, extension and testing.

Over the period September 1990 - March 1991 a small scale prototype was designed, implemented and successfully demonstrated. The structure of the prototype was based on a simplified template of a switching system; this included a subset of the on-line environment, a tiny version of the on-line extension module, monitoring tools, and a trial packet-switching application. It run on a network of 386-based personal computers and Sun workstations.

7.1. Results from the prototype

Valuable results have been gathered in the implementation of the IOLE prototype. Focus was mostly put on the Operating System Layer due to two major reasons. First, the most innovative aspects were concentrated in its design, especially features of scalability, openness, and distribution. Second, an off-the-shelf product was used as the CE-Kernel Layer (and the Adaption Layer) for the sake of reduced implementation time. This choice did not leave much room for innovative design of the low levels.

The commercial product selected for this purpose was the Chorus Kernel, which is a distributed operating system kernel. (For information on the Chorus system, see for example [2].)

In the place of a bare kernel however it was adopted the Chorus-based implementation of Unix, called Chorus/MIX. Which is effectively a full-scale operating system. Together with many advantages in the implementation, this decision has impeded to achieve significant performances. In the current implementation, in fact, every IOLE primitive call is serviced going through many levels of software: First the Operating System Layer, then Chorus/MIX, finally the actual Chorus Kernel. (The whole path being quite a long one, indeed).

In spite of this, however, the prototype has been positively evaluated. The main aspects of interest are as follows:
i. The implementation of the Operating System Layer revealed fairly simple, even in the absence of a suited CE-Kernel. It consisted of around 10,000 lines of source code, for less than 20 Kbyte of object code per CE. It was written in the C Programming Language, while C++ is planned for the next release.

ii. The goal to make the interactions between these modules as much as possible location-independent was achieved out of quite light-weight (and simple) protocols.

iii. The idea to diversify the functioning of the modules to replicate on the various CEs, on the base of configuration information passed at the booting time, revealed efficient and flexible.

iv. The aim to enable on-line replacement and extension by making the communicating objects be attached to manageable communications interfaces, showed the expected flexibility and little overhead.

7.2. Prospects

The aspects briefly outlined above still need further refinement and study. One of the aims is especially that of gathering a sufficient amount of quantitative and qualitative results to carry out the ultimate assessment of feasibility of the IOLE design. This should be done in a continuation of the IOLE Project in the phase 2 of the RACE Programme (schedule to start in early 1992).

The relatively poor performance figures achieved with the current prototype are undoubtedly motivated by the hardware and software set-up. Notably, the use of Chorus/MIX in the place of a suitably scaled CE-Kernel, and the use of Ethernet as the interconnection medium instead of more appropriate media and protocols. For most of these problems, however, suitable solutions exist which will be definitely adopted in any future release of the prototype.

8. Concluding remarks

The cost of software development and maintenance is expected to grow with reverse proportion to the hardware costs. This is particularly true in the case of the telecommunications systems. Recent figures forecast the software costs to reach soon around 75% of the cost of maintenance of a modern switching system.

In such a scenario, providers and operators very much look forward to certain well-known "abilities" that would be of great help to hamper the growth of the software costs. Should these dreams might come to reality one would have the opportunity:

- To neatly model the telecommunications software in a manner as independent as possible from non-essential physical constraints.
- To design software by making the assumption that it will be run by an on-line environment of virtually infinite processing capacity.
- To program explicit interactions between software modules only caring about the related communications interfaces, no matter where the modules will be actually located, (bearing in mind that, when needed, it shall be possible to plant software modules onto specified locations and devices).
- To change version of software (with or without changes in the communications interfaces) by smooth use of on-line replacement and extension tools completely integrated with the operating system.

The IOLE operating system which has been briefly presented in this paper, seems to be a promising step towards this goal.

Acknowledgments

Acknowledgements go to Dr Josef Doppelbauer of Alcatel Austria ELIN RC (Vienna, Austria) for his considerable contribution to the development of the IOLE design, and to Sebastien Tyrrell of Alcatel SESA RC (Madrid, Spain) for his generous and often illuminating reasoning around the key IOLE concepts.

Special acknowledgements go to Paola Bonanni and Colm Kenny of Intecs Sistemi (Pisa, Italy). Bearing the most burden of the prototype implementation, they were able to develop and tie to reality the intriguing concept of the control thread.

References


