TelORB—The distributed communications operating system

Lars Hennert and Alexander Larruy

Future telecommunications platforms must fulfill both traditional requirements for availability and performance and increasingly stringent requirements for open-endedness and scalability.

TelORB is a distributed operating system for large-scale, embedded, real-time applications that require non-stop operation. It is composed of a modern OS kernel, a real-time database, software-configuration control, and an associated development environment for writing task-specific application code. A CORBA-compliant object request broker and a Java virtual machine run on top of TelORB.

The authors describe the TelORB operating system platform, its unique characteristics, and processing entities. These include device processors that directly control hardware with stringent real-time requirements; the TelORB operating system, which controls traffic availability and soft, real-time performance; and UNIX or Windows NT, which provide standard programming environments for less critical, real-time platform functionality and applications.

TRADEMARKS

Java™ is a trademark owned by Sun Microsystems Inc. in the United States and other countries.
Windows NT is a registered trademark of Microsoft Corporation.

Introduction

Future platforms for use in telecommunications must offer extremely high availability, their systems must operate non-stop, regardless of hardware or software errors, and they must allow operators to upgrade hardware and software during full operation without disturbing the applications that run on them. These rigorous requirements for robustness must not affect system performance.

In the field of telecommunications, performance-related requirements are often specified in terms of statistics. For example, it must be possible, 90% of the time, to perform a certain operation within a specified period. During the remaining 10% of the time the stipulated threshold may be exceeded. Real-time performance of this kind, sometimes referred to as soft, real-time performance, is generally sufficient for telecommunications operating systems.

The platforms must also be scalable in terms of capacity: operators should be able to increase system capacity simply by plugging in new processing equipment, as opposed to having to replace the processors in a plant with more powerful ones.

Finally, there is a strong trend nowadays toward open systems. Openness, however, comes in many varieties:

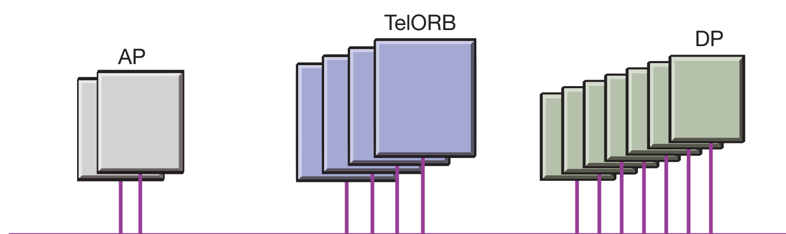
- Open hardware platform. Operators must be able to use commercially available off-the-shelf hardware. This requirement guarantees that the systems can keep up with, and take advantage of, the latest advances in hardware design.
- Programming languages. Operators should be able to hire skilled developers who can become productive quickly without first having to attend lengthy courses.
- Interoperability. Open systems must support standard protocols so as to communicate with external systems from a variety of vendors.
- Compatibility with third-party software. The application program interfaces (API) in open systems should run on standard operating systems.

TelORB-based systems support each of these requirements. However, the processors that run the TelORB operating system do not meet the last requirement; it is fulfilled by the system's adjunct processors, which use UNIX or Windows NT.

General architecture

In short, TelORB provides an environment for applications that control traffic and require soft, real-time responsiveness, high throughput, high availability (minute-per-year downtime), and scalability (in the sense that capacity can be increased by adding processors). Applications that run on TelORB are expected to serve numerous system end-users. These applications should also be permitted to evolve or to be developed continuously, thereby offering users new services. TelORB does not provide an environment for controlling small, low-cost hardware devices that require stringent, real-time responsiveness (for example, bounded, worst-case behavior). Nor does it provide a standard programming environ-

Figure 1
TelORB-based systems may consist of processors running the TelORB operating system, adjunct processors running UNIX or Windows NT, and device processors that run commercial, embedded, real-time operating systems.



ment into which third-party software can be integrated and custom adaptations quickly introduced—these environments are accommodated in the system architecture by device processors (DP) and adjunct processors (AP).

DPs are typically low-cost processors that control a handful of hardware devices. While the memory footprint of the DP operating system may be an issue, DPs require few (if any) middleware components, and the application software is fairly stable and well defined. To simplify DP software further, the DPs are owned and managed by applications running on TelORB.

The APs run standard operating systems, such as UNIX or Windows NT. They host operation and maintenance-related (O&M) platform components, such as off-line databases for complex queries, logging facilities, and management models, but can also host application software, such as purchased protocol stacks or specialized end-user services. A system may be composed of up to several hundred DPs, two or more TelORB processors, and one or more APs (Figure 1).

Within the system, one or more networks interconnect the various processors. Present-day TelORB systems use dual Ethernet, but ATM or practically any other network solution can be used as long as adequate bandwidth is provided. Different networks with different media may even coexist in the same system: the TelORB processors and the adjunct processors could, for example, be interconnected with Ethernet, whereas the device processors could be accessed through an ATM network.

The TelORB inter-process communication (IPC) protocol is used for transporting data between TelORB processors. A variation of that protocol is used between TelORB and the device processors. The standard user datagram protocol/Internet protocol (UDP/IP) and the transmission control protocol/Internet protocol (TCP/IP) are used for transporting data between TelORB and the adjunct processors. Applications can use CORBA or dialogs (when within TelORB) on top of these transport protocols. TelORB communicates directly with the device processors via the transportation layer.

Characteristics

Some key features of TelORB are its real-time characteristics and its support of continuous operation.

Real time

TelORB is intended for use with soft, real-time applications; that is, applications with load-dependent, statistically deterministic behavior. Support for stringent, real-time applications (applications with bounded, worst-case behavior) affects performance and application flexibility and is therefore left to the device processors.

Priorities and scheduling

Processes execute on one of four priority levels: high, normal, low, and background. The normal level is intended for ordinary telecom traffic applications, whereas the low

BOX A, ABBREVIATIONS AND TERMS

AP
Adjunct processor
API
Application program interface
ATM
Asynchronous transfer mode
Callback function
Function that is called as a result of an external event (for example, an incoming message on a communication link)
CORBA
Common object request broker architecture
DBMS
Database management system
Delos
One of the interface specification languages in TelORB
delux
The Delos compiler
DOA
Database object agent
DP
Device processor
DU
Distribution unit
Forlopp
Chain of interconnected processes resulting from an external event; several resources may be allocated during this chain of processes in order to handle the originating event
IDL
Interface definition language
IDP
Internal delivery package
IIOP
Internet inter-ORB protocol
IPC
Inter-process communication
LM
Load module
LPC
Linked procedure call
Managed object
Object that can be accessed from the O&M system
MI
Managed item
MIB
Managed information base
NTP
Network time protocol
O&M
Operation and maintenance
OMG
Object management group
ORB
Object request broker
OS
Operating system
OU
Object unit
Persistent object
Database object
SCC
Source code component
Scheduling queue
Queue in the kernel that holds processes that are ready to be executed
Supervisory mode
Execution mode in a microprocessor that contains the complete instruction set; application processes execute in the user mode, which has a slightly limited set of instructions
SWI
Software interface
TCP/IP
Transmission control protocol/Internet protocol
TMN
Telecommunication management network
Trigger (database)
Optional user-defined function that is called when certain events take place (for instance, when a database object is created or deleted)
UDP
User datagram protocol
Zone
Cluster of interconnected TelORB processors

level is intended for maintenance. The high-priority level should be used frugally for processes that involve the servicing of hardware. The background level could be used for audits and hardware diagnostic tests. TelORB uses a simple scheduling policy: the highest priority process that is ready to execute is allowed to execute for at most one time slice (about two milliseconds) until it becomes blocked, idle, or its time slice expires. If its time slice expires, the process is queued last at its priority level. This procedure is then repeated with the subsequent highest priority process that is ready to execute.

Obviously, average scheduling delays depend on the load of the processor. TelORB has a load-regulation mechanism that permits applications to reject parts of the traffic operations, in order to sustain real-time performance.

Interruptible kernel

By allowing operations within the OS kernel to be interrupted, the interrupt response time can be kept within reasonable bounds. However, to keep the kernel from becoming overly complex and error-prone, an interrupt-service routine restricts the number of operations that are allowed to run in the kernel. One such operation is the scheduling of a delayed interrupt-service routine, which (because it is synchronized with respect to other kernel operations) can use additional operations. Applications are advised to do the least amount of work in the real interrupt-service routine, postponing the rest of the work for the delayed interrupt-service routine.

Continuous operation

Today, more and more systems must operate non-stop—they are expected to provide year-round service, 24 hours a day, seven days a week. TelORB was designed specifically for these kinds of system. Its memory-protection hardware protects systems from ordinary application faults, and its fault-tolerant software permits in-service upgrades.

Memory protection

TelORB processes have their own memory space, which cannot be manipulated from other processes. Data in the OS kernel is also protected from manipulation by processes. Thus, errors in any given process cannot affect other processes. This minimizes the impact that a software error might have on the overall system.

Fault-tolerance implemented in software

To handle, and recover from, hardware errors and errors in the OS kernel, TelORB reconfigures the processes of a failing processor to other processors in the system. Consequently, TelORB systems do not require expensive, fault-tolerant hardware (Box B).

Network redundancy

To handle catastrophic situations (earthquakes and fire, for example), TelORB supports the option of having a redundant, geographically separate system that works in a standby fashion. The redundant system is continuously updated during normal operation and can take over immediately without any loss of data.

In-service upgrade

New software can be loaded and taken into operation while the system is running and providing service. Obviously, this requires cooperation between TelORB and the application programs that is facilitated by the framework for implementing processes. Since the same mechanisms are used for in-service upgrades and for reconfiguring the system, operators can easily verify the aspects of an application program to be upgraded.

BOX B: RECOVERY LEVELS IN TELORB

TelORB supports the following recovery levels:

- Database transaction rollback. The application is informed when a transaction is unable to commit successfully.
- Process abort/restart. If a static process encounters an internal error from which it cannot recover (for example, division by zero) it is restarted. Similarly, if a dynamic process encounters an internal error from which it cannot recover, then it is aborted.
- Forlopp recover. TelORB informs processes that are interconnected with communication links when the remote side of the link disappears. This enables the application to clean up any resources allocated to a certain chain of events.
- Processor reload. TelORB attempts to reload the processor after an error has occurred in the hardware or in software in the kernel code.
- Zone reload. If there is a risk of inconsistency in the system state—for example, when two or more processors fail simultaneously—TelORB reloads the entire cluster with the most recent backup.

Overload protection

In rare situations, events in the outside world create disproportionate load on the systems, exhausting system resources and simultaneously rendering several processors unusable. TelORB protects the system from situations of this kind by measuring the length of the scheduling queue and rejecting dialog setup attempts when the queue length exceeds set limits. In this case, system response time slows and fewer operations are carried out successfully.

Rapid last resort

Although many precautions have been taken to protect the system, there is always a remote chance that it will fail completely. For instance, several processors might conceivably fail at the same time, making it impossible for TelORB to maintain a consistent system state without reverting to a previously saved state. Thus, as a last resort, TelORB reloads all processors with a backup of the database. To minimize downtime, TelORB uses a multicast loading protocol that enables the parallel loading of all processors in the system without a bottleneck in the communications medium, and without requiring disks to be attached to each processor. If the network redundancy feature is being used, then even "catastrophic" situations can be handled without loss of data.

Program environment

Programs written for TelORB execute as one or more cooperating processes using the database to store configuration parameters and other data that needs to persist. The programs can use several operating system services through the TelORB API. The programs are written in standard C/C++ or Java, with interoperability interfaces specified in the CORBA interface definition language (IDL). Delos, a proprietary specification language, provides the constructs that standard programming languages lack for specifying processes and database objects (Figure 2).

Processes

Specification

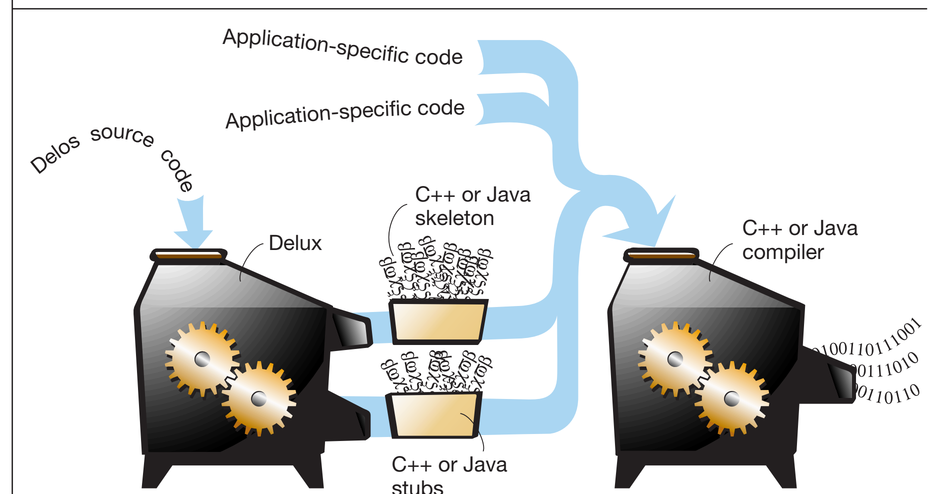
The processes that run on TelORB are specified in Delos, which assigns a name and characteristics to each process type. From the process type, TelORB creates process instances as defined in the specification.

Processes are declared as being static or dynamic. Static process instances, which are created when the system is started or when the process is installed, are recreated after a failure. Dynamic process instances, which are created when addressed by another process, are not recreated after a failure.

The process type specification indicates whether or not a process instance is to be replicated on several processors. When a process instance is replicated, TelORB directs any other process that wants to cooperate with it to its replica (if one exists) on the same processor. If a replica does not exist, the process is directed to an arbitrarily selected replica on some other processor. Note: replicas do not know of one another's existence unless required to do so by the application.

Finally, the process type specification indicates how multiple instances of the same type are to be differentiated and installed. For example, as an alternative to simply being instantiated anywhere whenever addressed, the instances of a dynamic process type can be differentiated by a primary key (that is consistent with corresponding database objects). TelORB can thus direct several calls with the same key to the same instance until it is terminated. For static process types, instances can be created automatically when the system starts, or the application software can install them through the TelORB API. This option is particularly well suited to process instances

Figure 2
The Delos source code is fed into *delux* (the Delos compiler), which generates C++/Java stub and skeleton files. The application-specific code is then manually added to the skeleton files before final compilation (C++ or Java compiler) together with other application code. The IDL source code is handled in a similar way.



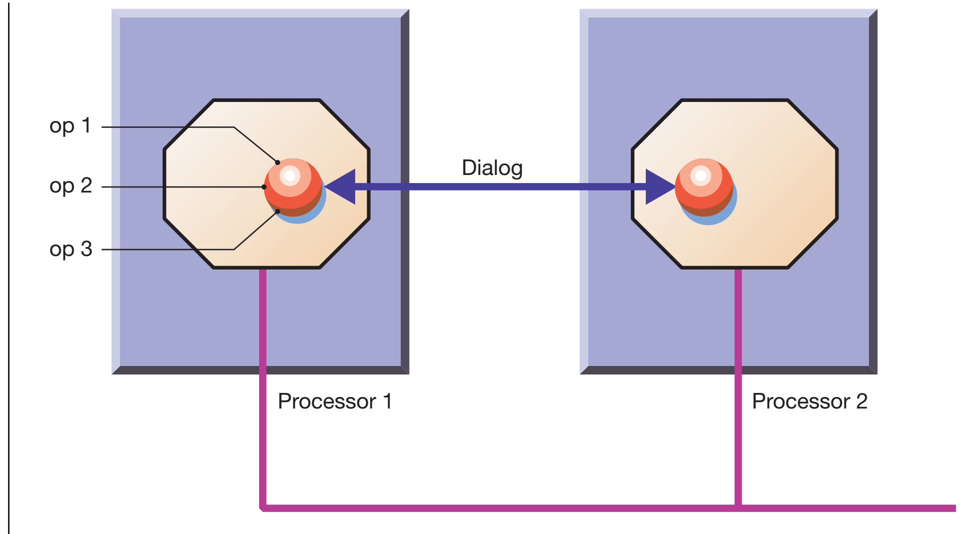


Figure 3
Processes cooperate by means of dialogs, which are specified as a pair of object types that has operations on it (in this example, op1, op2 and op3 can be called from the process in processor 1).

that are directly associated with the hardware configuration, which might vary from site to site and as plants are extended.

Implementation framework

The specified processes are implemented within a framework given by C++ or Java code that was generated from the Delos specification. The framework gives entry points for handling
- startup (initially, and after system crashes);
- the reconfiguration of instances between processors;
- in-service software upgrades; and
- termination.

When TelORB initiates a static process instance, it also indicates (in the instance) one of three reasons for starting it. Either the instance was created for the first time, or it has been recreated after a system crash. Alternatively, the instance was recreated after system reload. In this case, a serious system error occurred making it impossible to preserve database consistency. Consequently, the system was reloaded with a consistent backup of the database, which by necessity did not contain the most recent updates. The hardware state could thus be inconsistent with the state in the database. Notwithstanding, applications that can differentiate between the two usually accept the hardware state.

To facilitate non-disturbing reconfigurations and in-service upgrades, TelORB allows dynamic processes running on the original processor (with the original software) to terminate naturally over a period of time. New processes (running the new software) are created on the new processor. For static processes, TelORB creates another instance on the new processor (with the new software) that is allowed to run in parallel with the old instance. The new instance is also given a reference to the old one, so that the two can communicate—by means of an application-specific state-transfer protocol. Other processes in the system perceive the pair as a single instance. Nonetheless, if operation is not to be disturbed, certain special provisions may be needed for application-specific cooperation between the processes.

The execution paradigm

All execution within a TelORB process takes place as the execution of callback functions (most often as member functions of C++ or Java class instances) associated with events that are external to the process. Ordinarily, the program cannot choose to disable the handling of events—they are executed in the order in which they take place. Programmers must thus adapt to an inherently asynchronous outside world. The handling of all events, however, is serialized.

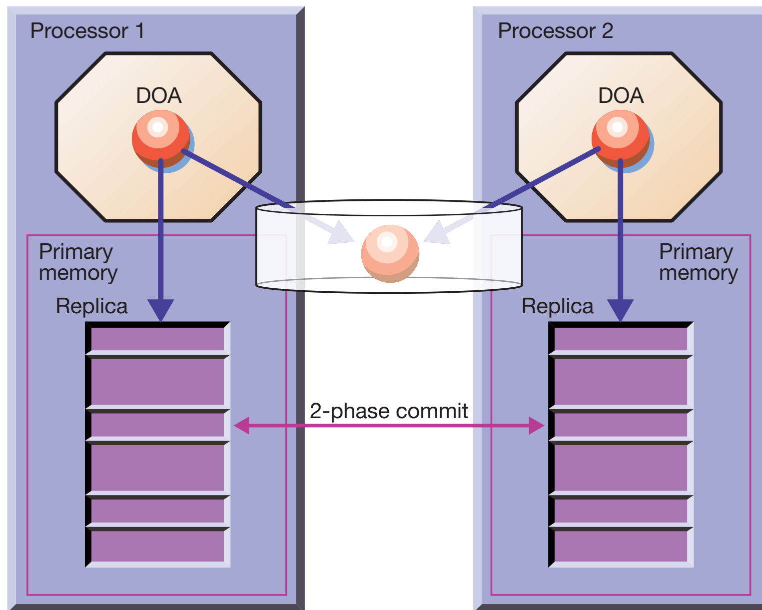


Figure 4
The database object agent (DOA) causes database objects to appear as ordinary objects belonging to a global database. In reality, however, each object is stored as a local replica in the primary memory of the corresponding processor. The database management system (DBMS) manages data access and keeps the replicated data in different processors consistent. Data replicas are updated automatically using a two-phase commit protocol.

Thus, programmers need not bother with the problems of concurrent programming (one callback function is finished before the next is called, which means there is effectively only one thread of execution in a TelORB process). On top of the execution paradigm, programmers are free to make use of lightweight threads. In Java, however, multithreading is supported as described by the language specification.

As an exception to this scheme, TelORB actually provides mechanisms for blocking the process-only execution thread. In this state, it cannot be unblocked except by a single corresponding event or time out.

Process cooperation

Processes cooperate by invoking remote operations that are collected in the Delos notion of dialogs. A dialog is specified as a pair of object types that has operations, and sometimes results, on it. C++ classes are generated from the specification. Some classes marshal and unmarshal the operations with their arguments; others provide the framework for implementing the actions of invocations within the processes (Figure 3).

To set up a dialog, one process addresses the other process by its type and, where needed, data that differentiates a particular instance. After the dialog has been set up, a connection is established between each member, to enable members to signal in the event that one of them dies. When the processes terminate, a dialog-shutdown procedure safely closes the connection.

The database

While data inside a process is volatile (it is lost if the process or the processor it runs on crashes), data stored in the TelORB database persists even after a process or processor crashes. Besides storing data, the database shares data between processes. The TelORB database is an object-oriented, real-time database that stores data in primary memory on the processors (Figure 4).

Specification

Data in the database consists of instances of persistent object types (specified in Delos) that have attributes of the persistently stored data and an associated set of methods that the application can use to manipulate the data.

The attributes include ordinary data types—integer, enumeration, record, and so on—as well as a data type which is specific to the TelORB database, and which holds references to other database objects, much like pointers in programming languages. These reference attributes may have either a single value or a multiple value. Object types may be derived from other types, in which case the behavior of the methods be-

comes polymorphic; that is, an application can open an object of a basic type and find an instance of a specialized type. When methods are invoked, the implementation of the specialization is executed. Delos also allows certain properties to be specified for attributes; for example:

- an array type attribute can have an element access property, which means that instead of retrieving the entire attribute, individual array elements can be retrieved from the database;
- a reference type attribute can have an inverse property (in this case, a reference type attribute in the referenced object must refer back to the referrer); and
- attributes can be optional, making it possible to lower the expense of storing default values for large attributes.

C++ and Java classes, which provide necessary interfaces to the database and the framework for implementing methods and triggers, are generated from the Delos specifications.

Operations

With the object types thus specified, applications can

- create new objects;
- open existing objects;
- fetch the values of, and assign new values to, attributes;
- invoke methods; and
- close and delete objects.

Database consistency is maintained by grouping operations into atomic transactions: either all operations are performed within a transaction or no operations are performed. TelORB maintains locks on objects to prevent several processes from changing the same data simultaneously. Applications are also allowed to introduce their own integrity checks by implementing trigger functions called during certain operations.

Replication

Data in the database is stored entirely in the primary memory of the processors, which makes operation very fast. To survive crashes, data is replicated on at least two processors. An extended, optimized, two-phase commit protocol is employed to replicate changes quickly and safely.

Network redundancy

The mechanisms for network redundancy are closely related to the database, since they always synchronize the database contents of the geographically separate standby side with data in the active system.

Lookup

For an object to be found in the database, it must either have one attribute designated as a primary key, which uniquely identifies an instance of a particular type, or it must be referenced from a reference type attribute of some other database object.

Objects in the database may also be found by means of iterators, which retrieve every object of a particular type (which matches the selection criteria defined for the iterator). Iterators can be applied to the entire database or to a multivalue reference type attribute. When an iterator is applied to the

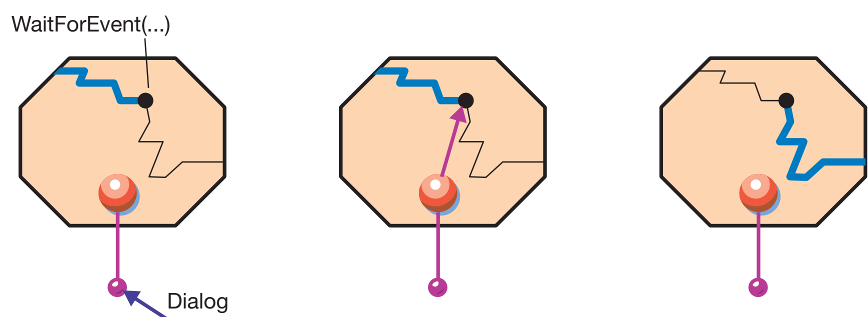


Figure 5
An example of the use of events. A method that receives a certain message in a dialog can post an event to a thread that is monitoring the event. When the thread calls the "WaitForEvent" method, it picks up any event that has already been posted or, if none, waits for an event to be posted. This renders a synchronous behavior to TelORB's otherwise inherently asynchronous programming model.

entire database, it can only retrieve types that have a primary key.

The O&M model

For O&M, TelORB requires that the specified object model be separate from the actual application implementation. This object model consists of CORBA objects, which have attributes, actions, and relations to other objects.

The managed objects of a system form a management information base (MIB), which contains information on object contents and on object types. This information can be displayed by a graphical application.

The O&M model also includes notifications, which the system sends to the operator, to inform him of particular events. A notification is always associated with a particular CORBA object. A special kind of notification is the alarm, which is sent when the system requires operator intervention (a typical example is when a processor board has failed and must be repaired).

Services

TelORB provides applications with several services, through

- the TelORB API—by means of C++ and Java classes (where standard Java classes do not suffice);
- Delos object types;
- Delos and the code generated from the specifications; and
- Corba IDL and the code generated from the specifications.

Threads and events

The lightweight threads in TelORB are designed for use with events specified in Delos. A Delos event is a type of message that is sent from an object to any thread that monitors the event. The intention has been to provide a way of adding or changing threads that execute control flows without having to change the implementation of objects that represent resources with more static behavior.

A new thread is created simply by instantiating a C++ class derived from a thread-base class in the TelORB API. The main program for the new thread consists of an overridden virtual member function. The program executed by the thread typically monitors several events from different objects and, where appropriate, waits for any of a select set of events, taking appropriate actions in response to the event received (Figure 5). These lightweight threads and events are only used for C++ code. The Java language provides its own threads and synchronization primitives.

Timers

A process can use one-time as well as periodic timers that can be set to expire in steps of five milliseconds, from ten milliseconds up to about 20 days. When the timer expires, TelORB executes a callback function. A timer can be cancelled at any time before the callback function has been executed.

Clock and calendar

TelORB provides real-time clock and calendar functions for converting the internal format into a standardized calendar format. Support for local calendars, including adjustments for daylight-saving time, has been prepared but is not fully implemented. The real-time clock is synchronized with different processors by virtue of a TelORB adaptation of the network time protocol (NTP).

Static process installation

As noted earlier, static process instances can be installed by application software through the TelORB API. Since these instances typically relate to the hardware configuration, applications must often check to see on which processor a process is to run. For this reason, the application installs a process instance for creation within a processor group. TelORB provides certain predefined processor groups, such as "all processors," but generally, each application must provide its own appropriate processor groups and register them through the TelORB API.

Handling software errors

Basically, any process in which an error has been detected is considered a bad process that should be terminated immediately—giving a crash dump for fault localization. TelORB automatically terminates processes with errors detected by hardware (dereferences of zero pointers, for example). But for errors detected by the application software, TelORB provides an interface through which information on the error can be attached to the crash dump. After termination, a static process instance is recreated in the usual way.

Processes that cooperate with a bad process are notified by means of dialog abortion indications. TelORB will not automatically terminate a process that receives a dialog abort indication, but leaves it to the

memory management, safe access to process data, the use of other drivers, and the ability to signal users of drivers in processes. Drivers can be opened, closed, and controlled from a process.

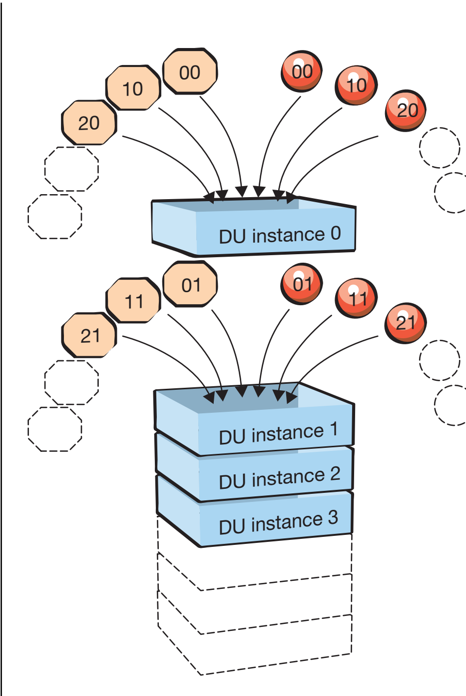


Figure 6
For dynamic processes that are identified by a key, the instances of several processes and corresponding database objects can be grouped into a distribution unit (DU) instance. TelORB guarantees that the contents of the DU are kept intact regardless of reconfigurations, thus ensuring that the processor always has local access from one of these processes to the corresponding database object.

application program to select an appropriate action or response.

Drivers

Drivers provide low-level control of hardware. They are programs that are executed in the processor's supervisor mode, which has the same privilege mode as the kernel. Applications can use their own drivers to control application-specific hardware. The TelORB API provides services for implementing drivers and for accessing them from processes.

TelORB gives drivers the following functionality: timers, installation of interrupt-service routines, interrupt level control,

Distribution

To make the most of the distributed processor platform, processes and data must be distributed in a way that balances processor load and minimizes communication between processors. As relates to TelORB, two simplistic approaches are either to distribute all process and database object instances arbitrarily, or to let the plant engineer specify the distribution of each individual instance. The first approach would probably result in bad performance, whereas the second approach would put unrealistic requirements on plant engineers. Moreover, each would fill memory with tables of the addresses of different instances.

TelORB allows the application developer to group individual instances into a manageable number of distribution units (DU). This grouping is typically done per process type or database object type, by associating a Delos distribution unit type specification with the specification of the process or database object type. The distribution unit type specifies the number of distribution units over which the instances should be spread. When more than one distribution unit has been specified for the type, the application program must supply a function that maps instances onto distribution units. TelORB also allows process instances and database objects to be grouped or co-located into the same distribution unit. This minimizes the inter-processor communication needed for manipulating database objects (Figure 6).

When the plant engineer configures the plant, he should group processors into distribution pools and allocate distribution unit types to pools. TelORB configures (in run-time) individual distribution units from the types to processors within the pool, and reconfigures them as necessary; for example, when a processor crashes, when the system is extended through the addition of a new processor, and when the assignment of a processor to a pool is changed (Figure 7).

Thanks to the distribution units, table sizes stay manageable. Experience gained thus far indicates that the application designer must also decide which pools to use and which distribution unit types should be

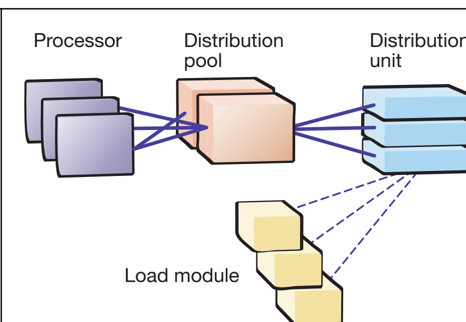


Figure 7
By attaching distribution units to distribution pools and assigning processors to the pools at configuration, TelORB can determine what code is needed to execute the distribution units on a processor in the pool.

allocated to them. Therefore, this information belongs to the internal delivery package (IDP).

Memory model

TelORB divides the processor's logical addressing space into three main parts:
- instruction memory space (holds the code to be executed);
- kernel data memory space (holds data for the TelORB kernel and any drivers); and
- process data memory space (holds data for the process currently executing).

All processes share the code in the instruction memory space and can read (but not write) data in the kernel data memory space. This space is used to accelerate access to the database, and could also be exploited by drivers.

The process data space is mapped to different physical memory when different process instances are dispatched. This way, processes become isolated from each other, so that an error in one process cannot affect another process (Figure 8).

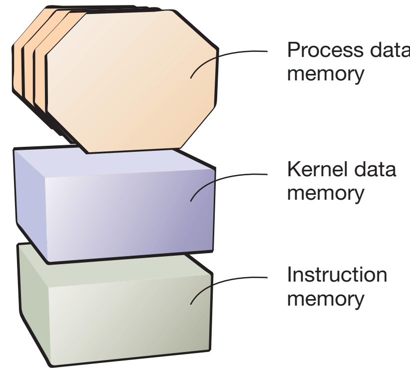


Figure 8
Although each process instance has its own memory space, all process instances share the instruction memory and the kernel data memory. Kernel data can only be written through OS calls.

Separating load modules

Code is loaded into load modules that are not directly related to process types. Although a process is implemented in a load module, it can also execute code in any number of other load modules by means of a mechanism called linked procedure call (LPC). The LPC mechanism is used for implementing database object type methods and triggers (Figure 9).

External communication

For communication with the outside world, TelORB provides common Internet protocols and the means of implementing application-specific protocols, as needed.

Internet protocols

TelORB implements TCP/IP and UDP/IP for communicating with other systems.

CORBA protocols

TelORB supports the Internet interoperability protocol (IIOP), which is transported over TCP/IP.

Development environment

TelORB comes with a development environment that runs on a UNIX workstation and includes a software structure model, build support, and tools for configuring a plant. Application programs, which can be run on the host on simulated processors, use a source-level debugger and other tools for pinpointing faults in the code.

Software structure model

The software structure model is based on managed items (MI), which have attributes (for instance, an identity and version) and relationships to other MIs. A file structure is also associated with each MI type.

Basic managed items

The most basic MI types are
- the load module (LM);
- the internal delivery package (IDP);
- the object unit (OU);
- the source code component (SCC); and
- the software interface (SWI).

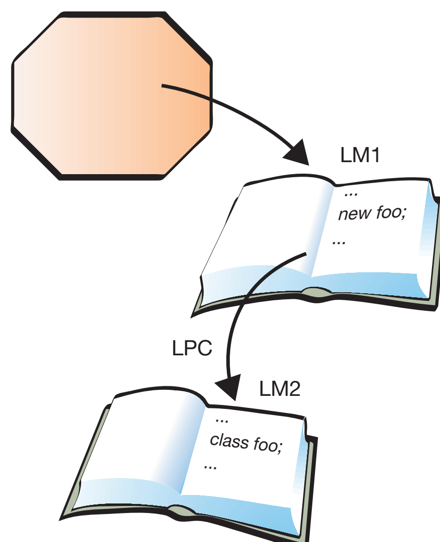


Figure 9
By means of the linked procedure call (LPC) mechanism, a process that executes code in one load module (LM) can create and use objects defined in another LM. If the new LM is updated, the new version will be used without having to relink (offline) the code that uses it.

The load module contains the object code file to be loaded onto a processor, where it is relocated and executed. It has relationships to several object units whose object code is to be included in the load module.

The internal delivery package collects several related load modules which must be used together in the target system but which might be loaded on different processors. It also contains information on the distribution pools to be used when the distribution of the DUs it contains is specified.

An object unit contains source code that is to be included in only one load module. By contrast, a source code component contains library source code whose object code is to be linked to any number of load modules. Object units and source code components provide and use software interfaces.

A software interface contains interface specifications (Delos and IDL specifications and C/C++ header files) that are shared by several managed items. When provided by an object unit, a software interface can be used by any number of object units, source code components, or other software interfaces (Figure 10).

Build support

When the source code is structured according to the model of managed items, it can automatically be built by the build support that ships with TelORB. In particular, the build support

- generates C++ and Java code from Delos and CORBA IDL specifications and files it into the appropriate directories;
- compiles generated code and application source code;
- assembles the load modules (for instance, it determines which LMs of an SCC are to be included); and
- gathers information needed by the plant-configuration tool to assign load modules to the processors.

Languages and compilers

C/C++

Programs in a TelORB system are mostly written in standard C or C++. The current compiler implementation does not support C++ exceptions. The use of templates is discouraged until an acceptable way is found for dealing with them in the build support. C and C++ code is compiled with the GNU C compiler, version 2.7.

Java

TelORB provides a run-time environment for Java programs. TelORB supports a subset of Java with APIs added to enable Java programs to make use of the TelORB ser-

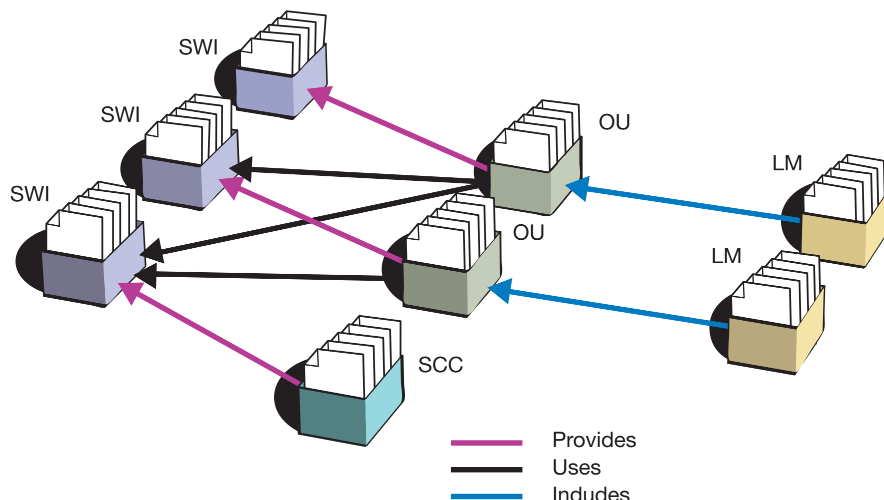


Figure 10
Relationships between the most common types of managed items.

vices. In essence, the subset is the equivalent of standard Java 2 with graphical support removed.

CORBA IDL

The CORBA IDL can be used to specify interfaces within the TelORB system and between TelORB and other systems.

Delos

Delos is a proprietary specification language used for specifying processes and database objects for which C++ has no constructs. Delos was originally a language family used to express interfaces, behavior (coding language), software structure, and distribution. Thus, TelORB is able to express interfaces and distribution. With Delos, developers can express data types, different categories of object types, dialogs, notifications, process types, and distribution unit types.

Delos specifications are compiled with the Delos compiler, *delux*. The compiler is divided into a front end, which parses the specifications and produces an intermediary format, and a back end, which takes the intermediary format and generates C++ and Java code plus some other information required by the target system.

Programming libraries

To simplify application development, TelORB includes the following programming libraries:

- a standard C library (minus a few functions that did not fit into the TelORB environment);
- a C++ class library, which provides lists, queues, collections, and random numbers;
- library classes that contain implementations of Delos data types, such as the string and octet string types and other support for code generated from Delos specifications.

Plant configuration

Engineers configure the plant by running a program called epct, which takes an input file that describes the configuration and outputs a file structure that can be transferred to the TelORB file system. The input file

- lists the internal delivery packages with software to be run in the system;
- defines the distribution pools;
- allocates distribution unit types to the distribution pools;
- creates representations of the processors in the system; and
- allocates the processors to distribution pools.

The output file structure contains the LMs to be loaded, a boot-load table for the first processor to be loaded, and files that specify which managed object operations are needed to take the system into operation.

Debugging

Vega—a simulated processor environment—is the main tool for starting applications. It is a UNIX process with a "hardware" adaptation layer that makes it behave like an ordinary processor. Multiple *Vega* processes can be interconnected to verify distribution aspects.

Several tools are used for debugging, including the following:

- *gdb*. The GNU debugger for high-level debugging of C or C++ code. The tool can be extended with graphics wrappers, such as *xemacs*. At present, a distributed, multithreaded, high-level debugger that supports C++ and Java is being developed for TelORB.
- *sysview*. Another graphics application with which processes can be examined as they run, and from which traces can be initiated and displayed.
- a Telnet-based inspection tool. This tool enables developers to examine the contents of the database from a remote location.

Conclusion

The market for intelligent network solutions is growing rapidly, which means that there will be an increased need for zero-downtime platforms that support a massive amount of transaction-oriented processing. Nodes in intelligent networks often need ultra-fast databases to handle requests from a large number of users. The TelORB platform is well-suited to meet these requirements. Furthermore, its exceptional scalability permits operators to gradually expand their systems as the need arises.

Because the system uses commercially available, "off-the-shelf" processor boards, operators can always take advantage of the latest achievements in hardware design.

Applications are built using well-known languages, such as C++ and Java, and interoperability is provided through the built-in object request broker.

TelORB is a truly open platform whose characteristics, in terms of robustness and flexible configurations, are unparalleled. It is currently deployed as the base for the Jambala platform.