Intelligent Networks and the HP OpenCall Technology

The HP OpenCall product family is a portfolio of computer-based telecommunications platforms designed to offer a foundation for advanced network services based on intelligent network concepts. This article concentrates on the HP OpenCall *service execution platform*, *service management platform*, and *service creation environment.*

by Tarek Dehni, John O'Connell, and Nicolas Raguideau

Intelligent networks are an expanding area within the telecommunications industry. The adoption of intelligent network technology has been driven by its ability to allow telecommunications network operators to install and provision new, revenue-generating communication services in their networks. With these services installed within the network, the extra functionality they provide can easily and instantaneously be made available to the whole customer base. Examples of such services are the freephone services (the cost of the telephone call is paid by the called party), credit card calling, and the CLASS services (custom local area signaling services) in North America.

At the same time, the standardization of some key interfaces within the telecommunications network has allowed greater competition between network equipment providers, offering the possibility of genuinely multivendor networks. The opening up of previously proprietary switch interfaces has made it easier for network operators to add new functionality to their networks, since this functionality can now be implemented outside the switch, often on industry-standard computer platforms. Today, with the emergence of new fixed and mobile network operators in many areas of the world, two new drivers for intelligent networks have emerged. Firstly, there is the need for interoperability between these networks. Secondly, operators seek to differentiate themselves on their service offerings. Both imply an even stronger requirement to support extra intelligence in the network. This will ensure the continued demand for more open and flexible intelligent network solutions.

Hewlett-Packard's product strategy for the intelligent network market is based on the HP OpenCall product family, a portfolio of computer-based telecommunications platforms designed to offer a solid foundation for competitive, revenue-generating services based on intelligent network architectures. This article concentrates on the HP OpenCall *service execution platform*, *service management platform*, and *service creation environment*, with particular emphasis on the architecture and design of the service execution platform. The HP OpenCall *SS7 platform* is described in ***Article 7***.

In this paper, we introduce the key concepts in intelligent networks including the role of standardization, we explore the system requirements for a class of intelligent network elements (those elements targeted by the HP OpenCall platforms), and we highlight the key aspects of the design of the HP OpenCall platforms.

Intelligent Networks

The telephony service is very simple. Its objective is the transport of speech information over a distance in real time. Telephony networks were originally designed with the assumption that the same service would be offered to all users, and this held true for a long time. Users could select one of a range of destinations and be called by other users. Over the years the focus of telephony service providers has been to improve the technology to offer these basic services to a larger number of customers, and over longer and longer distances. At the same time, terminals have become mobile, with mobile phone users demanding the same levels of services.

As a consequence of this evolution, today's telephony networks consist of a mix of more or less integrated technologies and networks that have been deployed over more than 30 years, forming a very complex and large-scale global infrastructure.

In this context, the task of provisioning a new service in an operator's network is extremely complex and may vary considerably depending on the network infrastructure. The conventional method of continually integrating these new functions into public exchanges is costly and lacks flexibility, making it difficult for network operators to compete effectively in an increasingly competitive environment.

This situation has led network operators and their suppliers to look for a better approach, in which control functions and data management linked to the creation, deployment, and modification of services can evolve separately from the basic

switching or existing functions of an exchange. They assigned standards organizations (ITU-T, ETSI, BellCore) the responsibility of defining an architectural framework for the creation, execution, and management of network services.

Intelligent Network Conceptual Model

The ITU-T (International Telecommunications Union—Telecommunications Standardization Sector) developed the *Intelligent Network Conceptual Model* to provide the framework for the description of intelligent network concepts and their relations. The Intelligent Network Conceptual Model consists of four planes, each of which is a different abstraction of the telecommunications network. The ITU-T also planned the specification of the target intelligent network architecture through several study periods, thereby enabling incremental implementations. These successive standardized functions are referred to as *intelligent network capability sets* (see Fig. 1).

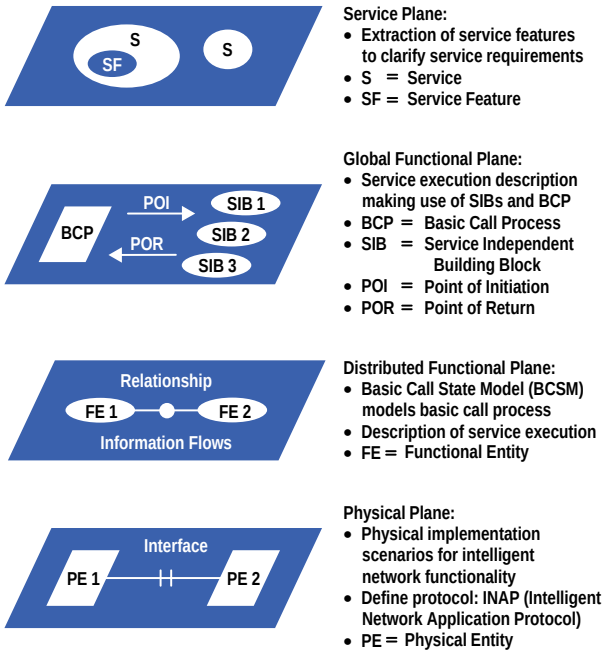


Fig. 1. *The Intelligent Network Conceptual Model of the ITU-T is a framework for describing and specifying intelligent network systems.*

Service Plane. The *service plane* describes services and the service features as seen from a user perspective. A service feature is the smallest part of a service that can be perceived by a user. The service plane does not consider how the service is implemented or provisioned in the network.

Global Functional Plane. The *global functional plane* describes the design of services as a combination of *service independent building blocks*. Service independent building blocks give a model of the network as a single entity, that is, there is no consideration of how the functionality is distributed over the network.

A specific service independent building block is the *basic call process*, which corresponds to the basic call service. It has *points of initiation* and *points of return*. An instance of a service logic can be called from a point of initiation, and after execution of the service logic, the basic call process is recalled in a point of return. Service logic corresponds to services or service features in the service plane.

Distributed Functional Plane. The *distributed functional plane* (Fig. 2) gives a distributed functional view of the network. *Functional entities* are groupings of functionality that are entirely contained in a physical entity. In other words, they cannot be split among several physical entities. The distributed functional plane describes the functional entities together with their relationships.

The identified functional entities are as follows:

- The *call control access function* models the interface with the end-user terminal.
- The *call control function* provides call and connection control, that is, basic call processing.
- The *service switching function* models the call control function as seen from the service control function.
- The *service control function* provides the logic and processing capabilities for intelligent network-provided services. It interacts with the service switching function to modify the behavior of the basic call, and with the two entities below to access additional logic or obtain service or user data.

- The *service data function* contains customer and network data for real-time access from the service control function.
- The *specialized resource function* provides additional specialized resources required for intelligent network-provided services, such as dual-tone multifrequency (DTMF) receivers, announcements, conference bridges, and so on.

Finally, on the management side, three functional entities are defined:

- The *service management function* allows for deployment and provisioning of intelligent network services and for support of ongoing operations. Its management domain can cover billing and statistics as well as service data.
- The *service creation environment function* allows new services to be safely and rapidly defined, implemented, and tested before deployment.
- The *service management access function* provides the interface between service managers and the service management function.

It is envisioned that the service independent building blocks specified in the global functional plane will be realized in the distributed functional plane by a sequence of coordinated actions to be performed by various functional entities.

Physical Plane. The physical plane describes the physical alternatives for the implementation of an intelligent network. The identified possible physical nodes include *service control points*, *switches*, and *intelligent peripherals*.

The information flows between functional entities contained in separate physical entities imply a need to specify and to standardize the interfaces and protocols between these separate physical entities.

The following protocols have been defined by the ITU-T:

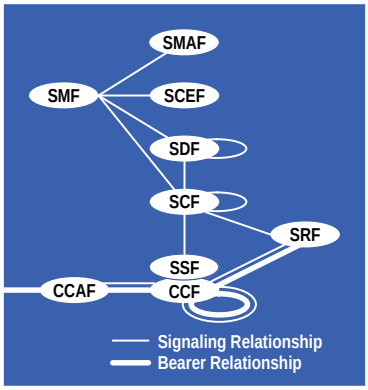
- The ISDN User Part (ISUP) and the Telephony User Part (TUP) are instantiations of the information flows between call control functions.
- The Intelligent Network Application Protocol (INAP) covers the information flows between service switching functions with a series of message sets.
- The information flow between the service control function and the service data function is based on the X.500 specifications.

Except for the TUP, these protocols are network services on top of the telephone companies' Signaling System #7 (SS7) signaling networks.

Intelligent Network Rollout

The general architectural concepts of intelligent networks are applicable to a wide range of telecommunications networks including plain old telephony services (POTS) networks, mobile communication networks (GSM, PCN, DECT), ISDN, and future broadband networks. Furthermore, these well-defined architectural principles can also be found in standards from other organizations that have defined equivalent or domain-specific architectures. BellCore's AIN (Advanced Intelligent Network) architecture shares many features of the ITU-T approach, while ETSI, for example, has identified various physical nodes (HLR, VLR, MSC) communicating via the MAP protocol in mobile networks.

Although it didn't deliver all of its original promises, the intelligent network concept is considered a success. Today freephone remains the major revenue-earning service for intelligent networks, and it is continuing to grow. Freephone, split-charge, and premium rate services still generate almost 50% of intelligent network revenue. Another service that is providing significant revenue to network operators is virtual private network service, which is currently experiencing the most rapid growth.



Functional Entities:
SMAF: Service Management Access Function
SCEF: Service Creation Environment Function
SMF: Service Management Function
SDF: Service Data Function
SCF: Service Creation Function
SRF: Specialized Resource Function
SSF: Service Switching Function
CCF: Call Control Function
CCAF: Call Control Access Function

Fig. 2. *Intelligent network distributed functional plane, showing functional entities.*

It is interesting that all of these services share the characteristic that they require data to be available throughout a network. This class of service clearly promotes a centralized intelligent network solution. In fact, the fundamental success of the intelligent network concept is that it has simplified data management of those services for which it has succeeded (although service management remains a relatively minor consideration in standards organizations). The full potential of other types of intelligent network services still needs to be realized.

Intelligent Network Element Requirements

Hewlett-Packard has developed the HP OpenCall service execution platform as an open, programmable, scalable, highly available, and easily manageable platform that can be used as a basis for implementing a range of different elements of an intelligent network. The platform provides a set of basic functionalities that are common to many intelligent network elements. By installing suitable user-defined applications or services on it, the HP OpenCall platform can be extended to provide the service control function, service data function, specialized resource function, and other functionality to other nodes of the SS7 network, such as switches. Thus, the aim of the HP OpenCall service execution platform is to provide a platform that can be easily extended to meet the requirements of different intelligent network elements. The common requirements of these different network elements are summarized in the following paragraphs.

Openness and Flexibility. One of the key goals of the intelligent network is to promote multivendor solutions to allow technology from different equipment providers to interwork. In contrast, many of the early intelligent network solutions were implemented on proprietary solutions. These applications are often not portable across platforms, so customers are often tied to a single equipment provider and cannot always benefit from the latest advances in hardware.

Furthermore, intelligent networks are seen as evolutions of existing networks, that is, new network elements, implementing intelligent network functionality, are expected to interwork with existing equipment. This implies that the new elements must support multiple entry points to ensure easy integration with other products, both at the SS7 network interface and at the interface with the operations support systems that manage the telephone network.

This need for multivendor solutions also drives the standardization activity in intelligent networks (see ***Subarticle 6a, "Standardization—A Phased Approach"***). In theory, if the interfaces between network elements are standardized and clearly defined, interworking should be easy. However, despite the existence of multiple standards, local differences make it necessary for a platform to adapt to many different environments in terms of connectivity, protocol support, and management. Furthermore, there is no standard environment in the telecommunications central office. Each central office often has its own collection of legacy systems. This implies a need to be able to add network-specific, protocol-specific, and environment-specific intelligence to meet customer requirements.

Rapid Service Deployment. In the increasingly competitive telecommunications market, network operators see the need to differentiate themselves on their service offerings. Operators want to be able to define, deploy, and customize services quickly, while still guaranteeing the high levels of quality and reliability that have traditionally been associated with telecommunications networks. There is a need to support all aspects of the service life cycle, including definition, validation, installation, and management.

Performance and Determinism. The round-trip real-time budget from a switch to a network element such as a service control point is typically quoted as 250 milliseconds (mean) with about 95% of responses within 500 ms. This includes the switch processing time, the transmission and queuing times of both the request and response messages, and the service control point processing time (encoding and decoding of messages, service activation and execution, query to database). Clearly, the faster the transmission links and the smaller the buffering in the system, the more time is available for service control point processing. For a simple freephone service, once the SS7 transmission times are subtracted, we obtain a requirement for a mean service control point processing time of 50 ms with 95% completing within 62 ms. The behavior must be controllable so that the system is deterministic. Transaction rates of up to 10,000 transactions per second for network elements have been requested.

High Availability. Network elements such as service control points and home location registers are critical components in an intelligent network. Very high system availability is required: no more than three minutes total downtime per year, including both scheduled and unscheduled downtime. This necessitates highly reliable hardware with no single point of failure and software that allows mated applications to back each other up in the event of a natural disaster disabling one particular site.

Furthermore, in the event of a failure at a site, during the failure recovery phase, the network element must be responsive to other network elements, taking less than six seconds to resume service. If not, the network considers that a total failure has occurred at the site.

The availability requirements also pervade the scalability and functional evolution aspects. The system must be capable of expansion and addition of new functionality without disruption of service.

Similar kinds of requirements apply to service management systems, albeit not as severe as for network elements. Service management systems are typically allowed 30 minutes of total downtime per year.

Scalability. A network element must be scalable in terms of processing power, memory, persistent data storage, and communications without compromising system availability. On the hardware side, this means support for online upgrades of processor boards, memory boards, disk subsystems, communication controllers, and links. It must be possible to perform such operations without service interruption. On the software side, it means bringing additional resources into service smoothly and safely. If anything goes wrong, the software must automatically fall back to the last operational configuration.

In general, network elements such as service control points and home location registers are classified in terms of transactions per second (TPS) and customer database size. Scalability then translates into the ability to add new hardware and/or software elements to increase either the maximum supported TPS rate or the maximum customer database size.

Functional Evolution. The ability to add new applications or platform capabilities or simply upgrade existing ones without impacting the availability of the system is vital. This means that such things as updating the operating system, upgrading the application, adding a new communication protocol stack, or changing the firmware on a communication controller must be achieved without disrupting real-time performance or system availability. This ability should not impose too many constraints on software development and installation. In all upgrade situations, a fallback must be possible.

Manageability and Operation Requirements. There are detailed and rigorous requirements concerning installability, physical characteristics, safety, electromagnetic and electrical environments, maintenance, reliability, and so on. Remote management interfaces to large and complex network systems with demanding performance and scalability requirements are needed.

To allow easy management, complex distributed or replicated network elements must be capable of providing a single-system view to operations centers. At the same time, it is also very important to provide per-system views, to allow management of specific systems and to act upon them (typically for fault management or performance management). Newly installed network elements often need to be integrated into existing management systems.

HP OpenCall Platforms

HP OpenCall Service Execution Platform

The HP OpenCall service execution platform is an open, scalable, programmable, highly available, and easily manageable platform. It is implemented as a layer of functionality on top of the HP OpenCall SS7 platform (see *Article 7*). Given the general network element requirements listed above, the following architectural principles and high-level design decisions were adopted when developing the HP OpenCall service execution platform.

The software runs on the HP-UX* operating system, allowing it to benefit immediately from advances in hardware speed and CPU processing power.

All critical hardware and software components are replicated, giving the platform the ability to tolerate any single failure. An active/standby replication model was chosen at the software level, with an instance of the platform software running on two independent systems. Besides providing a high degree of fault tolerance, such an approach also provides the basis for most online upgradability tasks, such as upgrading hardware and the operating system.

Fig. 3 shows a typical site configuration, with two instances of the HP OpenCall service execution platform software executing on two independent HP 9000 machines, but with a single connection to the SS7 network. This site will appear as a single network node to other elements of the SS7 network. The configuration in Fig. 3 is a standard duplex configuration of the platform. Other configurations are possible, such as the mated-pair configuration, described later. In the standard duplex configuration, the two machines are connected via a dual LAN and both hosts are connected to a set of signaling interface units. The HP OpenCall service execution platform software runs on both hosts in active and standby modes. The active host controls the signaling interface units and responds to all incoming requests from the SS7 network. Both hosts are capable of being active (i.e., there is no default active host).

The platform is network independent. It makes no assumption about the structure of the SS7 network. It has the ability to support multiple message sets, but all message set dependent decisions are made at the application level.

Some customization is necessary before an instance of the HP OpenCall service execution platform software can be installed in or connected to an SS7 network. By default, the platform supports no message set, and it offers no service to other network elements. Minimally, users of the platform must install one or more message sets and provide one or more applications to respond to requests coming from other network elements or to send requests to other network elements. Furthermore, to ensure that the resulting solution can be monitored and managed, it should be integrated into an existing management system or a set of management tools should be implemented.

A set of APIs (application programming interfaces) are provided to access platform functionality, to be used either locally or remotely. This provides flexibility with respect to integration with operations support systems and legacy management systems. A set of management tools using these APIs are also provided, and can be used to provide a first level of platform management. Further levels of management (for managing the platform, installed services, customer data, etc.) can be provided by integrating the platform with other external management systems via the provided APIs.

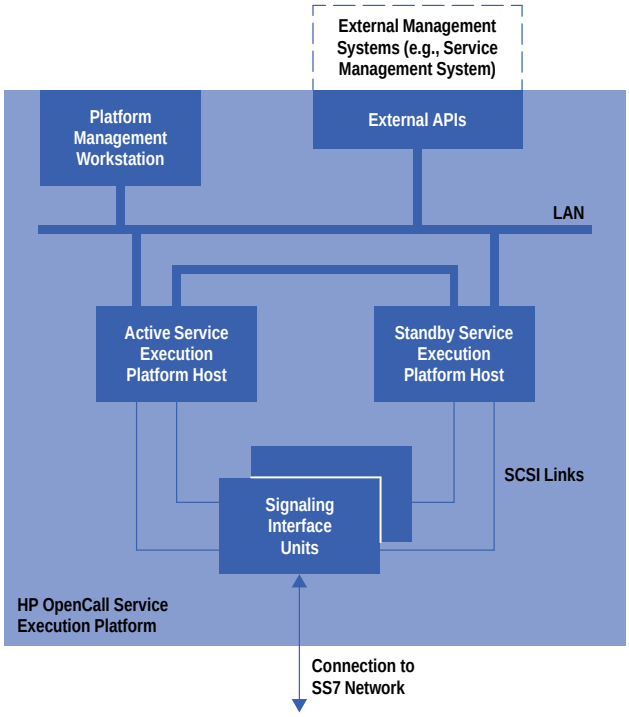


Fig. 3. *Duplex configuration of the HP OpenCall service execution platform. Two instances of the platform execute on two independent HP 9000 host machines with a single connection to the SS7 network.*

Applications, or *services*, executing on the platform are interpreted. New services or new versions of existing services can be installed at run time without interrupting processing of TCAP (Transaction Capabilities Application Part) traffic and without affecting other running services. Because services execute in a virtual machine with no direct access to the operating system, services cannot affect the availability of the platform. Furthermore, the service execution environment can monitor service instances, ensuring that instances do not interfere and that resources are not permanently consumed.

Services are independent of the operating system, protecting them from changes and upgrades to operating system. No knowledge of the operating system is required to write the service. Services are written in SLEL (Service Logic Execution Language). Most of the basic concepts in SLEL have been adapted from SDL (Specification and Description Language), enhancing it with some features specific to intelligent networks. This has the advantage that SDL is well-known in the telecommunications industry, and many telecommunications standards are specified in SDL.

A replicated in-memory relational database is provided as part of the service execution environment. The structure and contents of this database are under the user's control. By holding all customer-related data in RAM, services can respect the real-time response time requirements imposed by switches, since there is no disk access to retrieve call-related data. To achieve data persistency, a copy of the database is maintained on a standby host.

A Management Information Base (MIB) collects information on the state of the platform, making this information available both via an API and via a set of management tools, and allows external applications to manage and configure the platform. All management operations are directed to the active system—the standby system replays all management commands—thus presenting a single-system view to external applications.

The platform is implemented as a set of UNIX® operating system processes, allowing it to profit from multiprocessor hardware.

HP OpenCall Service Creation Environment

The HP OpenCall service creation environment allows easy definition, validation, and testing of services. Services are defined as finite-state machines, using a graphical language. The service creation environment provides a set of tools to allow the validation and simulation of services before they are deployed on the HP OpenCall service execution platform. The same service execution environment as described above exists in the service creation environment to ensure that the same behavior is observed when the service is installed in the HP OpenCall service execution platform.

HP OpenCall Service Management Platform

The HP OpenCall service management platform is capable of managing multiple HP OpenCall service execution platform sites. Such a distributed configuration introduces an extra degree of scalability, both in terms of transaction throughput capacity and database capacity.

Platform Design

This section discusses five key aspects of the HP OpenCall service execution platform solution: the service execution environment, platform flexibility, high availability, database replication, and scalability.

Service Execution Environment

The HP OpenCall service execution platform provides an execution environment for telecommunications services. These services are usually developed in response to new telecommunications requirements, and typically provide additional functionality to other elements in the SS7 network.

Programs defined in the Service Logic Execution Language (SLEL) are interpreted by the platform at run time, and can use language primitives to access the functionality of the underlying platform. The primitives enable them to send and receive TCAP messages, read and write message attributes, access and update the in-memory database, communicate over other external connections, send and receive events, set and reset timers, manage shared resources, log data to disk, and perform other functions.

Programs written in SLEL define finite-state machines, that is, a service waits in any given state until a recognizable input stimulates a transition to the next state. During the transition the service can carry out processing and output messages. Fig. 4 summarizes the functionality of the SLEL virtual machine.

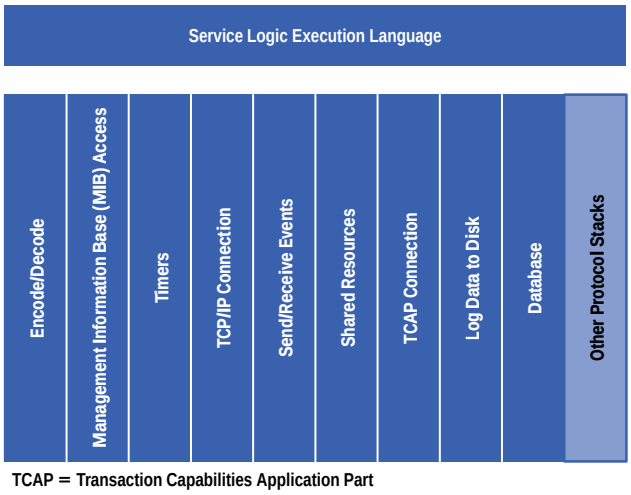


Fig. 4. *Services are written as finite-state machines in the Service Logic Execution Language (SLEL) and run on the SLEL virtual machine, which provides the functionality shown here.*

The following principles have been adopted in the development of the HP OpenCall service execution environment:

- Services are not aware of the operating system. They are isolated from the HP-UX interface. This allows easy migration between HP-UX versions. Furthermore, the application developer only needs to provide service-specific logic and need not be concerned with the startup, switchover, and failure recovery aspects of the platform, since these are transparent to the service logic.
- Services are not aware of replication. Replication of services and restart of services after a failure are handled automatically by the platform. No failure recovery code needs to be written by the service developer. The service programmer can assume a single-system view. To maintain this illusion, services can only access the local MIB and can only receive locally generated events. Furthermore, the service execution environment on the standby node is an exact replica of the active node's, providing the same services, same resources, same MIB structure, and so on.
- Real-time response to switches. The service execution environment gives the highest priority to the processing of TCAP messages and other service-related events (e.g., popped timers, received events). All other activities such as database or MIB accesses by external applications are run as background tasks. Service execution cannot be interrupted. A single state transition runs to completion. All local access by a service is synchronous (even to the database). A service only blocks if it explicitly requests blocking, for example to wait for the next TCAP message, wait for a timer, or wait for an event.
- Services cannot crash the platform. The service execution environment presents a virtual machine as its upper interface. Services can only access the platform functionality from SLEL. No pointer manipulation is available. Resource allocation and deallocation are done automatically by the platform. There is no possibility for core dumps or memory leaks.

- Online upgradability of services is possible. Because services are interpreted, services can be enabled, disabled, installed, or removed at run time without stopping the platform. Multiple versions of a service can be installed simultaneously, although only one can be enabled at any time. Instances of the previously enabled version are allowed to run to completion, so that TCAP traffic is not interrupted.
- The platform manages and monitors service instances. A single service cannot block other services from executing. There is a limit on the number of instructions that can be executed before a service is terminated. This prevents infinite loops and prevents one service from blocking out all other services. Resources are automatically reclaimed once the service instance has completed. There is also a limit on the total lifetime of a service, as a way of garbage collecting unwanted service instances. Both limits can be set on a per-service basis, and can be altered at run time.

Platform Flexibility

There is a strong requirement for the HP OpenCall service execution platform to be flexible. Obviously, it should make as few assumptions as possible about the applications that are installed and executing on it, and it should be able to integrate into any central office environment, both in terms of its connection with the SS7 network and its links with management applications.

Multiple Message Sets. The platform by default supports both a TCAP/SS7 connection and a TCP/IP connection. It makes no assumption about the protocols that are used above these well-standardized layers. In practice, any message set defined in ASN.1 (Abstract Syntax Notation One) can be loaded into the platform, and multiple message sets can be supported. The platform can encode and decode messages belonging to one of the installed message sets.

The message set customization tools take as input an annotated ASN.1 definition of a message set. The output is a *message set information base*, which contains a concise definition of the message set. Multiple message set definitions can be stored in a single message set information base. The HP OpenCall service execution platform's service execution environment loads the message set definitions from the message set information base at startup time. Services running in the execution environment can request the platform's encode/decode engine to process any incoming or outgoing message with respect to the installed message sets.

The same message set information base can be loaded into the HP OpenCall service creation environment. The message set definitions are available to service developers as part of the help facility. The message set information base is also available in the validation environment, allowing the user to submit message-set-specific message sequences to test the logic flow of the developed services. The traffic simulation tool uses the message set information base to encode and decode the user-supplied message sequences.

Flexible Service Instantiation. When a new TCAP transaction request is received by the platform, service instances must be created and executed to process the message. To offer a high degree of flexibility, the policy for choosing the correct service instances is programmable, that is, a user-supplied piece of service logic is executed to choose the most appropriate service. The decision can be based on any criterion, such as priority of the request, customer-specific data held in the database, overload status of the platform, and so on.

To allow tracking of resources, only one service instance controls the TCAP transaction, although it can be passed between instances. In this way, if the instance that currently controls a transaction exits unexpectedly, the platform can close the associated transactions and free the associated resources.

Flexible Service Structure. The intelligent network market has traditionally taken a service independent building block-based approach to service implementation. That is, a set of service independent building blocks are provided by the underlying execution platform, allowing applications to merely link these building blocks together to provide services to the SS7 network. The disadvantage of this approach is that the only functionality available to programmers is the set of available service independent building blocks. These are often message-set-specific and difficult to customize.

The HP OpenCall service execution platform does not provide a default set of service independent building blocks. Instead, it provides the means to structure applications as a set of components. Thus, if required, a user can implement the ITU-defined set of standard service independent building blocks and then use those components to implement applications to provide higher-level services. Of course, the user can also decide to implement an entirely different set of service independent building blocks.

Furthermore, the platform does not distinguish between a service independent building block (or component) and an application. Instead, it views both as services. Both can be arbitrarily complex pieces of logic, defined and installed by the user. How they interact and what functionality they provide are entirely under the user's control. To provide a single service to the SS7 network might only involve a single instance of service logic, or it might involve the creation and interworking of multiple such instances.

Platform Management. The platform exports information on its state via a Management Information Base (MIB). The MIB can be used to both monitor and control the platform. For example, installing a new service or a new version of an existing service onto the platform is performed via the MIB. Similarly, adding a new TCP/IP connection or supporting a new subsystem number is also achieved via the MIB. Both cases involve creating a new object in the MIB. Statistics on CPU use, TCAP traffic, service execution, database memory use, and so on are all held in the MIB and can be retrieved by external applications by issuing a request to the appropriate objects.

The information is presented as a hierarchy of objects, with each object representing a part of the platform's functionality. The hierarchical structure provides an intuitive naming scheme, and this also allows easy integration into standard CMIS (Common Management Information Service) or SNMP (Simple Network Management Protocol) management frameworks. Users can create new objects, delete existing objects (obviously changing the functionality of the platform in the process), update existing objects, or just retrieve information from individual objects. As mentioned previously, APIs that can be used remotely are provided, along with a set of management tools that provide a first level of platform management.

Customized Overload Policy. One of the most important responsibilities of any SS7 network element is to respond in a timely manner to incoming requests. The response time on requests must appear to be bounded, and 99% of replies must be generated within a fixed time interval. This implies that the network element must react to overload situations.

With the HP OpenCall service execution platform, the overload policy is programmable. That is, user-defined logic specifies how the network element reacts when the load is high. The platform itself collects statistics on a number of overload indicators such as CPU use, transaction rate, number of unprocessed messages, and average queuing time for such messages. These values are available in the MIB and can be viewed both by the overload service (the logic implementing the overload policy) and by external management applications.

The programmability of the overload policy offers a high level of flexibility. For example, under heavy load, the overload service may decide to:

- Reject new incoming requests but continue to accept messages relating to ongoing transactions.
- Request that other network elements reduce the number of requests. Obviously such a policy is network-specific and message-set-specific, requiring knowledge of both the SS7 network configuration and the message sets supported by remote switches.
- Reject new requests that require complex processing (obviously application-specific), or reject requests for low-revenue-generating services (again, application-specific).

The platform also provides a set of hooks to control the traffic flow. The overload policy can, for example, request the platform to reject new transaction requests (i.e., only accept messages relating to ongoing transactions), to limit the number of instances of a given service, or to reject traffic from a particular remote network element.

High Availability

All critical components of the HP OpenCall service execution platform are replicated (see Fig. 5). The core set of software processes operate in active/standby mode. This forms the basis both of the platform fault tolerance and of its online upgradability policy. It uses the HP OpenCall SS7 high availability platform, with every critical process being a client of the fault tolerance controller (see ***Article 8***). For simplicity, not all of the processes are shown, and not all of the interprocess links are shown.

Besides replication of processes, the platform also uses mirrored disks, duplicate signaling interface units, and dual LAN connections.

The principles that form the basis for the high availability policy are discussed in the following paragraphs.

Active/Standby Model. An instance of the HP OpenCall service execution platform platform runs on each of two independent machines. One instance, the *active*, is responsible for responding to all incoming requests, whether from the SS7 network or from management applications. The other instance, the *standby*, maintains a copy of the active's state, always ready to take over processing of such requests. The decision to adopt an active/standby model brings the following benefits:

- The code is simpler, resulting in fewer run-time errors.
- It is easy to provide a single-system view (that is, the active instance defines the state of the platform).
- It isolates errors because the two hosts are not performing the same tasks or the same types of tasks.
- The standby host is available for online upgrades, configuration changes, and so on.

The alternative, to adopt a load-sharing model (with requests being processed in parallel on two or more hosts), would have required more complex communication protocols between the instances (to ensure synchronization and a single-system view) as well as a greater possibility of an error simultaneously impacting more than one host.

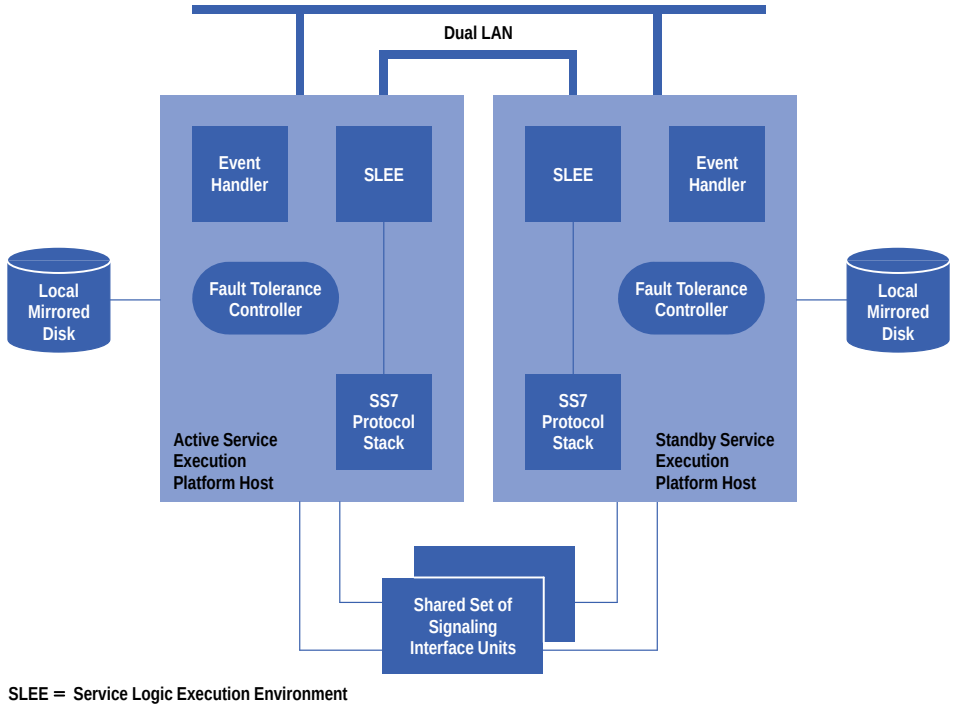


Fig. 5. *Replication of critical components in the HP OpenCall service execution platform.*

High Service Availability. The solution guarantees high service availability to the SS7 network in the event of any single hardware or software failure. To achieve this, the standby host must always be ready to take over in the event of a failure on the active host. For this reason, the standby maintains a copy of the in-memory database and of the service execution environment. All relevant changes to the state of the active (e.g., changes to the active database) are immediately propagated to the standby. In the event of a switchover, the state of the standby instance defines the current state. That is, the standby does not need to retrieve information from either the failed instance or from other external applications to become active. Thus, the transition to the active state is instantaneous, guaranteeing high service availability.

Centralized Fault Recovery Decisions. All switchover and process restart decisions are made by a centralized process, the *fault tolerance controller*, on each node. These two processes continuously exchange information on the state of the two hosts. All other processes obey the centralized decision-maker. This greatly simplifies failure recovery and error handling. In the event that both LANs go down, the signaling interface units are used as a tiebreaker. The node that controls the signaling interface units remains active (see ***Article 8***).

Online Upgradability and Maintenance. Because all of the critical components are replicated, the existence of the standby host is used as a basis for all online upgradability operations, such as changing the operating system, installing new versions of the platform, reconfiguring services or the service execution environment, performing rollback operations on the database, and so on. Because most upgrade operations can effectively be performed online in this way, the platform meets its downtime-per-year requirements.

Replication Does not Impact Services. Service programmers do not need to be aware of the replication of the platform. Furthermore, propagation of data to the standby is performed as a background task, implying a minimal impact on response time. Algorithms for resynchronizing the standby after a failure are also run in the background.

Respawnability. It is important that a failed host or a failed process be restarted quickly. If the standby host is not available, this introduces a window of vulnerability during which a second failure could cause the whole network element to be unavailable, directly impacting service availability. This respawnability feature is possible because of the continuous availability of an active system. The failed instance will restart, rebuilding itself as a copy of its peer. Because the active instance is assumed to be consistent, the respawned instance is also guaranteed to be consistent. However, to avoid rebuilding an instance to a nonconsistent state, instances are not respawned if the peer is not active (which might happen in the rare case of a double failure). In such cases, manual intervention is required to restart the complete platform. This respawn policy ensures rapid failure recovery in the case of a single failure but prevents erroneous failure recovery in the case of a double failure.

Support for Complete Restart. Although the platform is designed to tolerate any single failure, in certain cases it may be necessary to stop and restart both hosts (either because of a double failure or for operational reasons). Disk-based copies of both the MIB and the in-memory database are maintained, and these can be used to rebuild both hosts. However, some data loss may occur. This is considered to be acceptable, since double failures are rare.

Database Replication

At the core of the service execution environment is an in-memory replicated database. The database is in memory to guarantee rapid access and update times. To ensure high availability, copies of the in-memory database are kept on both the active host and the standby host. Critical updates to the active database are propagated to the standby.

The structure and contents of this database are under the user's control. When defining the database structure, the user must also define the replication policy for every field. Of course, propagating updates to the standby will impact performance, but because the user is allowed to specify the replication policy, the user can also control the impact.

Traditionally, database systems achieve persistency by maintaining a copy of the database on disk or on mirrored disks. In the HP OpenCall service execution platform, the primary standby copy is held in memory on a standby host. This offers a number of advantages over the traditional approach:

- Writing to a remote in-memory copy is quicker than logging to disk, and therefore has a smaller impact on SS7 traffic.
- The degree of availability is higher. In the event of a failure on the active host, the standby copy is immediately available. It is not necessary to recreate a copy of the database.
- The standby copy can be taken offline and used to restructure the database or to roll back to a previous version of the database.

Periodically, the active host generates a disk-based copy of the database. This *checkpoint* of the database serves a number of purposes:

- The checkpoint ensures that the platform can recover from double failures.
- The checkpoint is used to reinitialize a standby host if and when it is restarted after a failure or shutdown.
- The checkpoint can be used for auditing purposes by external database management systems.

Three algorithms are critical to the database replication scheme: the data replication algorithm, the resynchronization algorithm, and the rollback/restore algorithm.

Data Replication Algorithm. As mentioned above, the user specifies the replication policy for every field in the database. For certain data in the database, it may not be necessary to replicate changes to the standby host. Typical examples are counters or flags used by services.

Consider a set of fields that collect statistics on SS7 traffic. These would typically be incremented every time that a new request is received, and would be reset to default values periodically. For many services, it may be acceptable not to propagate these traffic statistics to the standby database. This implies that some data will be lost in the event of a failure on the active host (all traffic statistics since the last reset), but it also implies that less CPU time is consumed replicating this data to the standby. This trade-off is application-specific, so the decision to replicate is left to the user.

For fields that are marked for replication, the active database instance will generate an update record called an *external update notification*, containing both the old value and the new value, for every update. This record is then propagated to the standby database instance. By comparing the old value with the new value, the standby database can verify that it is consistent with the active. In the case of an inconsistency, the standby database shuts itself down. It can then be restarted, rebuilding itself as a copy of the active database.

The flow of an external update notification is shown Fig. 6.

To handle double failures, the external update notification is also written to disk on both the active and standby hosts. This is performed as a background task on the active host to minimize the impact on the transaction processing activity. The active host does not attempt to maintain a replica copy of the database on disk. Instead, a log is maintained of all of the updates that have been performed. To rebuild the in-memory database, the log must be replayed.

To manage the disk space required for these external update notification logs, the active host takes regular checkpoints. This task is treated as a low-priority background activity, and the frequency of the checkpoints can be controlled by the user (obviously as a function of the available disk space). When a checkpoint operation has completed, redundant external update notification log files can be removed. By default, the active host keeps the two most recent checkpoint files and intermediate external update notification log files on disk (following the principle of replicating all critical components).

Because database updates must be allowed during the checkpointing operation, each checkpoint file has an associated sequence of external update notifications that were generated while the checkpoint was being taken. To rebuild a copy of the in-memory database, both the checkpoint file and the associated external update notifications are required. Typically, the contents of the disk containing these database-related files will be as depicted in Fig. 7. The external update notifications are stored in a sequence of files. The rate at which one file is closed and a new file in the sequence is opened can be controlled by the user (either as a function of time or file size, or on demand). Thus, multiple external update notification log files can be created between two checkpoints, or indeed during a checkpoint operation. A copy of the in-memory database can be rebuilt either from a checkpoint file and associated external update notification log files (those files generated during the checkpoint) or from a checkpoint file, associated external update notification log files, and subsequent external update notification log files. To hold a copy of the database on a remote storage device, the user should close the current external

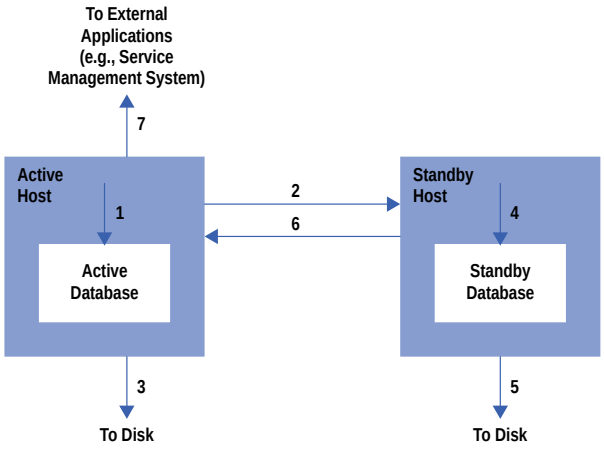


Fig. 6. *External update notification flow. (1) Update performed on active database. (2) Update record (external update notification) sent to standby. (3) External update notification logged to disk on active host. (4) Update performed on standby database. (5) External update notification logged to disk on standby host. (6) Acknowledgment sent to active host. (7) External update notification forwarded to interested external applications.*

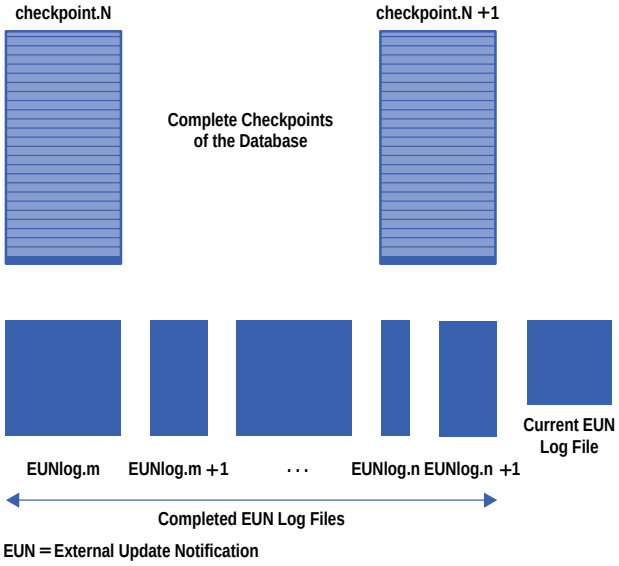


Fig. 7. *Checkpoint and external update notification files on disk.*

update notification log file (the active database will close the current file, assign it a file name in the sequence, and open a new current external update notification log file), and the user should then copy to stable storage the most recent checkpoint file and the external update notification log files generated during and since that operation.

Resynchronization Algorithm. Failures will happen. Therefore, a failure recovery algorithm is required. One critical component is the recovery of the standby copy of the in-memory database. The checkpoint and external update notification log files also play a critical role in this algorithm. The algorithm is complex because updates are permitted on the active database while the standby database is being rebuilt. The resynchronization process can take a long time (in the order of minutes), so it would not be acceptable to disallow database updates during that period.

The sequence of steps in resynchronization is as follows:

1. The standby host copies the most recent database checkpoint file and external update notification log files from the active file system to its local file system.

2. The standby host rebuilds the in-memory copy of the database from this consistent set of files.

3. When this operation is complete, the standby database asks the active database to close the current external update notification log file. It then retrieves that file and replays the external update notification records.

4. Step 3 is then repeated. The assumption is that, because the standby host is not performing any other tasks, with each iteration of step 3, the size of the current external update notification log file will be reduced. In effect, the state of the

standby database is moving closer to that of the active database. Eventually the size of the current external update notification log file will be small enough for the algorithm to move to the next step.

5. The standby database again asks the active database to close the current external update notification log file. At this point, a connection is also established between the two database copies. Now, while retrieving and processing the latest external update notification log file, the standby database is also receiving new external update notifications via that connection.

6. Once the latest external update notification log file is processed, the standby database starts to replay the queued external update notifications. At this point, with the establishment of the real-time connection between the two database copies, the two copies are considered to be synchronized, and the standby database is considered to be *hot standby*.

Rollback/Restore Algorithm. As with all database systems, it is also possible to roll the in-memory database back to a previous state. Again, the checkpoint and external update notification log files play a critical role in this algorithm.

This operation can be achieved "online" by using the standby copy of the database, which can be taken offline without impacting the active host. While the latter continues to process TCAP traffic, the rollback algorithm can be applied to the standby database. In effect, it rebuilds itself from a checkpoint file and associated external update notification log files. Once this operation is completed, the user can request a switchover of the two hosts. The previously offline standby host will now begin to receive and to process TCAP traffic. The peer host should then be rebuilt as a copy of this new active host, using the synchronization algorithm described above.

Scalability

The HP OpenCall service execution platform is implemented as a set of processes running on the HP-UX operating system. This gives it a degree of scalability, since it can run on a range of HP machines and can also benefit from multiprocessor hardware. Low-end configurations are executed on a single-processor, 48-MHz machine with 128M bytes of RAM. The high-end configurations currently execute on a dual-processor, 96-MHz machine with 768M bytes of RAM. The migration to HP-UX 10.20 increases the capacity of the platform both in terms of the maximum supported TPS rate and the maximum database size.

An extra degree of scalability is provided with the support of *mated-pair* configurations. Fig. 3 showed the standard single-site duplex configuration, that is, an active/standby configuration running at one geographical location with a single connection to the SS7 network (multiple links are used for redundancy and load sharing, but a single address is shared by these links). A distributed solution, with multiple active/standby configurations running at a set of sites (each with its own SS7 address) provides a number of benefits:

- Extra TPS capacity, since sites can process traffic in parallel
- Increased database capacity
- Tolerance of site failures or outages. This is critical if and when it is necessary to shut down a complete site for maintenance or operational reasons (or in the unlikely case of a double failure hitting both the active and standby hosts).

The HP OpenCall service management platform can be used to manage a distributed configuration, as shown in Fig. 8. Although this is referred to as a mated-pair configuration, it is not limited to two sites; multiple sites can be supported. Furthermore, each site is not required to have the same database, that is, the contents of the databases at different sites can be completely different. However, in general, sites will be paired; hence the name mated-pair. For a given site, a copy of its database will also be maintained at one other, geographically remote site. This remote site then has the ability to take over if the original site is shut down. The role of the HP OpenCall service management platform is to maintain consistency across these multiple sites.

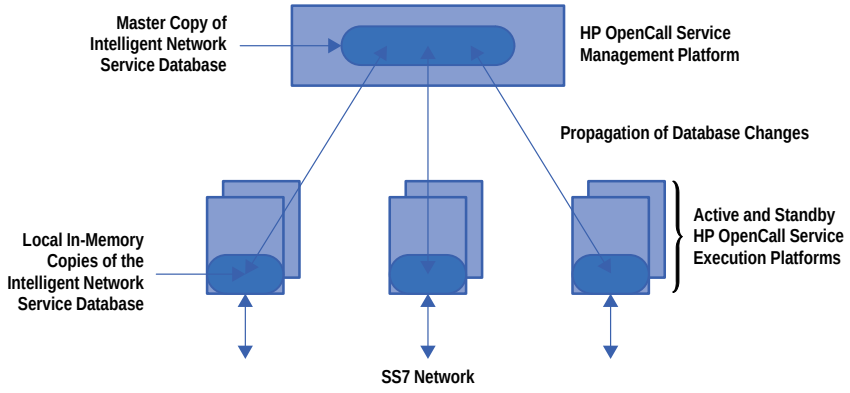


Fig. 8. *Centralized management of multiple HP OpenCall service execution platform nodes.*

The centralized HP OpenCall service management platform maintains a disk-based copy of each in-memory database. It receives notifications (external update notifications) from each site when the in-memory database is changed. Such external update notifications are then propagated to all other sites containing the altered data.

If an operator or system administrator wishes to change the contents of the in-memory database, the request should be sent to the service management platform. It will then forward the request to all sites holding a copy of the altered data.

Sites will typically process TCAP traffic in parallel, so it is possible that the same data may be changed simultaneously at two separate sites. Both will generate external update notifications and both external update notifications will be propagated to the service management platform. The platform is responsible for detecting this conflict and for ensuring the consistency of the replicas. It uses the old value field of the external update notification to detect that simultaneous updates have occurred. Consistency is ensured by rejecting any external update notification in which the old value does not match the current value held by the service management platform, and by applying the current value at the site from which the erroneous external update notification was received. In effect, this establishes the service management platform's database as the master database. In the event of a discrepancy, the service management platform ensures that its view is enforced and that all copies will become consistent.

The HP OpenCall service management platform also supports an auditing function. The service management platform orders a checkpoint of the HP OpenCall service execution platform's in-memory database, and then compares the contents of the resulting checkpoint and external update notification log files with the contents of its own disk-based database. Discrepancies are reported to the system administrator.

Conclusion

The intelligent network architecture allows the telecommunications industry to move to a more open solution, in which the creation, deployment, and modification of services is independent of a particular network equipment supplier and equipment from different providers can interwork to provide new and innovative services. Having independent and neutral intelligent network platform providers can enforce implementation of standards, hence ensuring better interconnectivity.

The HP OpenCall service execution platform is an open and flexible platform. It has been installed in a large number of different networks throughout the world, and has shown its ability to interwork with equipment from a host of other network equipment providers. It has been used to implement mission-critical network elements such as service control points and service data functions using standard, off-the-shelf computer technology. This use of standard hardware and a standard operating system will allow operators to benefit from the evolution of information technology with few additional costs and limited engineering risk.

▶ Go to Subarticle 6a

▶ Go to Next Article

▶ Go to Journal Home Page