Introduction: Intelligent Network (IN), also known as Advanced Intelligent Network (AIN), is an evolving service-independent architecture introduced into the Public Switched Telephone Network (PSTN) over a decade ago. The term Intelligent Network is used in the international standards under the Q.12xx Series Recommendations published by ITUT. The term Advanced Intelligent Network is used by Telcordia (Bellcore) in its publications (AIN Release 0.1, 0.2 and 0.X) that are in accord with ITU-T. In the scope of this document we will use the term IN to cover both terms.

The essence of IN architecture is to separate the service logic and intelligence from the switching environment and move them onto much more powerful and flexible general-purpose computers called Service Control Points (SCP). Such an architecture provides many advantages not only for the telecommunication service providers but also for their end-users. It allows service providers to create, deploy and maintain a wide variety of services quickly and economically by using a broad array of hardware, software and other development resources, without the limitation of trying to develop these capabilities within a proprietary switching environment. At the same time, it gives end-users the control over service definition and customization without requiring telecommunication service provider intervention. In today’s highly competitive telecommunication service marketplace, marked by globalization and deregulation, these advantages make IN a necessity for more and more network operating companies.

Advanced communication services developed using the IN architectures are information intensive. They require realtime access to vast amounts of data about each service and each subscriber with 24x7 availability and high data integrity. Moreover, these services often require complex data models and types. Today, the object-oriented paradigm has become the norm of the software industry and it has been successfully and widely used in the telecommunication domain for many years. Since an object database management system (ODBMS) provides a natural way for any object-oriented system to persistently store its data, an ODBMS-based IN application will have many advantages over its RDBMS counterpart. These advantages include reduced time to market, simplifying the task of handling complex data models and types and preserving the object-oriented characteristics for reusability and reduced maintenance cost.
This white paper will first provide a more detailed description on the IN background and architecture, then follow with a technical discussion on using an object database for IN applications.

**Background:** Before the concept of IN services was introduced, most digital telephony switches contained basic call processing (BCP) and database functions, in addition to circuitswitching capabilities. Since switch vendors used different ways to control the services and data in their switches, the operating company required extensive coordination to configure a network across a heterogeneous environment. In addition, all the data had to be duplicated for all the switches and had to be kept in sync. This required complex management systems. Such monolithic approaches made switching systems very difficult and expensive to change and maintain, which resulted in slow response to customers' requirements and delays in introducing new services.

To overcome these drawbacks, operating companies decided to introduce a common channel signaling (CCS) architecture and to distribute functions among specialized modules and machines. A CCS switching system separates the signaling part from the user traffic in the network. SS7 is a prominent CCS system defined by the International Telecommunications Union (ITU) in 1980. In the mid-1980s, the work on so-called intelligent networks was started by placing basic services in the SCP. This led to reduced cost and less maintenance complexity. In the early IN development, the interfaces between SCP and switches were tailored to each specific service and there was a lack of common standards defining them. At this early stage of IN, the SCP still only provided basic services.

In March 1993, the ITU-T approved a set of standardized IN capabilities, called Capability Set 1, for intelligent networks. Telcordia (Bellcore) also published its AIN specification Release 0.1 and 0.2. These standards introduced a range of services that support rapid customization of the service implementations. The ITU-T and Bellcore standards contain an extensive set of rules and procedures to govern how information is exchanged between IN components. This led IN into a new era with clearly defined roles for the SCP and other components for IN services. A common set of standardized messages is used for a variety of services in IN networks.
instead of custom messages. Wireless network services are also supported by IN networks. Finally, the IN architecture has become service-independent.

In the last few years, the Internet has become the dominant driving force in the telecommunication industry. Packet-based networks not only carry data but are beginning to carry voice and other traffic also. IN services are evolving quickly to meet these new demands. The IN functions supported by the SCP are being extended to support emerging technologies such as voice-over-IP (VoIP), voice-over-ATM (VoATM) and even the data network services directly.

Today, the number of new services supported by IN applications is rapidly growing. For example, a caller soon will be able to click on a pizza ad on a Web page and automatically be connected to the pizza company's closest local branch for ordering and delivery. Other services — for example, allowing end-users to turn on/off service features such as call-forwarding, callwaiting and call-screening directly from PCs at their homes or offices — already have started field trials.

**Architecture:** The IN architecture specifies a number of key elements in the network topology. Figure 1 provides a graphical view of the IN architecture and depicts which elements have associated databases. These components are:
• Service Switching Point (SSP). The SSP is a telecommunications switch that identifies and suspends calls requiring advanced service processing and forwards them to an appropriate SCP via the Signaling Transfer Point (STP) or a local adjunct to the SSP. The SCP/adjunct will interact with its database to process the incoming service request and send a message back to the SSP. Upon receipt of this message, the SSP will complete the call.

• Service Control Point (SCP). The SCP is at the heart of the IN services. It handles calls requiring advanced service processing. Normally, they are deployed in mated pairs that duplicate the same capabilities to ensure high availability and allow for load balancing. Each SCP may have several processors. These processors may access a common database of services and subscribers or each processor may have its own database, depending on the implementation. When the SCP receives a request from an SSP, it may simply query its database for the necessary information and then process the call. For instance, some customers want calls to their 800 numbers to be routed to different physical locations, depending on the area code of the caller. In this case, the SCP would simply look up which physical location should receive the call and inform the SCP about the destination. In other cases, the SCP may require further information from the user. To handle these calls, the SCP will route the call to an IP that can interface with the customer. For example, with credit card calls, the SCP would route the call to an IP that could accept Touch-Tone® input. Once the IP had received the customer’s credit card number, the SCP would verify its validity and complete the call.

• Adjunct. This performs the same function as the SCP, except that is it used for one or a few services on a single switch. Typically this arrangement is for supporting very fast response time, since the connection between an SSP and an adjunct is local at high speed without the overhead of going through the STP.

• Signaling Transfer Point (STP). The STPs are part of the SS7 backbone signaling-network. They are simply intelligent routers that route messages between SSPs and appropriate SCPs.

• Intelligent Peripheral (IP). The intelligent peripheral is a device that can connect to an IN call to provide services like tone genera-
evolving to become the dominant means of transporting not only data, but voice and video as well. The IN concept also is evolving to expand its services into packet networks. By abstracting the IN services functions as a pure software platform and making them accessible from the Internet (or even an intranet) in addition to the SS7 backbone signaling network, the IN services can be expanded to support data networks, as well as voice networks. The new generation of SCP should be able to provide independent services to switches and other types of network elements. Figure 2 is a view of such a new architecture.

Internet Protocol (IP)/IN is starting to take shape and there already are new products for IP/IN on the market. New services such as Web-based conference calling, click-to-dial (from a Web page) and click-to-fax (from a Web page) are a few examples of the results of the marriage between the Internet and traditional telephony.
Implementation Environment: Regardless of the services an IN application supports — whether for a telephony or data network, wireless or terrestrial — their implementation environment can be viewed as an object-oriented software platform. There are some common requirements that need to be placed on such an environment to implement IN applications successfully.

Often, an object-oriented and distributed software platform (such as CORBA) is required for IN implementation in order to satisfy the following critical needs:

- Time-to-market
- Performance
- Availability
- Reliability
- Flexibility
- Scalability
- Transaction management

Equally important, such an IN platform requires extensive database support for data persistence. The employed database must be able to match the capabilities of the distributed OO software platform described above.

From a database perspective, creating a new service requires creating a generic data model, or schema, that the SCP database will use to process service logic on incoming calls that use this new service. A simple example is used here to make things more clear.

![Diagram of containment relationship for a sample credit card IN application]
Figure 3 presents the object containment relationship for an IN-based credit card service. In this figure, boxes indicate classes, which define the attributes and methods associated with the specific objects. Lines indicate the containment relationship between objects. The numbers in brackets indicate the minimum and maximum number of instances allowed in each containment relationship. For example, a Basic Subscriber object can contain 0 or 1 “Auto Call List” objects and an Auto Call List object can contain 0 to 16 “Auto Call” objects.

The fundamental class in the service is the Basic Subscriber. A Basic Subscriber object can contain 0 or 1 instances of three other classes: Auto Call List, PIN List and Qualified Number List. These classes correspond to three different features of the credit card service. A Basic Subscriber object only contains instances of these classes if the subscriber has purchased the corresponding feature. The auto call feature allows the subscriber to save time by specifying a short code that will automatically dial a frequently called number. From the containment tree, we can easily find out that there can be up to 16 such codes defined for each subscriber. The PIN feature is simply basic credit card calling service. The qualified number feature protects the subscriber against unauthorized use by only allowing calls to these qualified numbers.

When the SCP processes an incoming call, it has to search all PIN objects for a match to the sequence of numbers input by the customer. Then, it has to find the PIN List connected to that PIN object and the Basic Subscriber connected to that PIN List. Finally, it has to find the Qualified Number List connected to that Basic Subscriber’s and then search among all Qualified Numbers contained in that list to make sure one of them matches the number dialed. It has to complete this same sequence of searches and navigation for many calls concurrently before any of the callers get frustrated and hang up.

From the description of this example, we can see two important things from a database point of view. First, we need a very high-performance database to satisfy the fast searching among objects and fast navigation from one object to another. The second interesting thing about this example is that it requires manipulating variable length data. The number of Auto
Calls, PINs and Qualified Numbers can vary. We don’t want to allocate disk space to storing the maximum number if we don’t have to. Given the fact that there could be millions of subscribers for such service, issues like performance and scalability become critical factors for the success of such services. These requirements can be generalized to all IN services. The next section will discuss many of the practical issues of using a database for IN services.

**ODBMS Support For IN Applications**

In the previous section, we took an initial look at IN database requirements. Now we’ll take a more detailed look at these requirements and discuss how an ODBMS in general and Objectivity/DB in particular meet these requirements.

**Time-to-market:** The greatest advantage offered by ODBMs for IN applications comes when building the data object model. In our example, its data model required many associations among different objects. As the services become more sophisticated, more complex object models are required to represent the services. Using a relational database will require an extra step to map these objects and their relationships into relational tables and there is no easy way of doing such mapping. Given the real gap between objects and relational tables, it is very difficult to preserve the object design characteristics such as inheritance and polymorphism during the mapping process. In addition, an RDBMS has many restrictions on application-specific data types since its table does not support such data types directly. On the other hand, an object database such as Objectivity/DB provides natural and direct persistence support for the OO design. Much time and effort can be saved during the design and development phases.

Another important benefit of the object-oriented paradigm comes from its facilitation of reuse. An ODBMS allows us to build many classes once and reuse them in every appropriate service. Having many reusable classes greatly decreases the time required for new service deployment.

Since one of the key goals for IN applications is to bring new services to the market quickly, an ODBMS like Objectivity/DB is an easy choice.
Flexibility: Like any other software product, continued improvements and fast problem resolution are essential, particularly because the market condition is changing at such a fast pace. If the object model for an IN application changes to reflect new requirements or to fix a problem, the ODBMS can handle such changes easily by auto-generating a new set of schemas.

The opposite is true if an RDBMS is employed for the same application. It would require things to be re-mapped between the new object model and the new relational tables. As we have mentioned before, this can be a time-consuming and tedious task.

Reliability and integrity: As mentioned above, IN applications need to operate in a 24x7 environment, necessitating very high reliability. Meantime-between-failures (MTBF) and mean-time-to-repair therefore become very important factors when selecting a database platform. The evaluation of different products should include a careful study of these statistics at various customer sites. The availability of Fault Tolerance, Data Replication and Hot Failover capabilities make Objectivity/DB a perfect choice for this environment.

In addition to reliability, IN applications require both data integrity and referential integrity. Data integrity measures a database's resistance to data corruption. Objectivity/DB provides high data integrity through the use of safe object identifiers (OID) or object references. OIDs provide a level of indirection between the application and an object's location on data pages. In contrast, some other products allow the use of direct pointers to data. Such pointers could become invalid after a transaction has been committed and reusing them can lead to data corruption. Data corruption could lead to dropping calls at the very least and possibly the total failure of an SCP.

Objectivity/DB provides referential integrity through bi-directional associations. The database automatically deletes an association when either of the objects in the association is deleted. In our earlier IN service example, if a subscriber discontinued service, we would delete the corresponding Basic Subscriber object. Objectivity/DB would automatically delete the associated lists and list member objects. By propagating delete operations, Objectivity/
DB prevents PIN objects with dangling references to Basic Subscribers from remaining in the database. If we did not propagate the delete operation in this case, a former subscriber who tried to use his/her old PIN would cause an application crash as the database tried to locate the nonexistent Basic Subscriber object on the other end of the connection.

**Availability:** Closely related to reliability and data integrity is the availability of the database. IN applications require 24x7 operation and cannot tolerate the database being unavailable. Objectivity/DB offers unmatched availability through its support of schema evolution, fault-tolerant configuration and data replication.

Objectivity/DB schema evolution provides a mechanism for changing the physical layout of objects in an application while the database continues to operate. For example, suppose that the example above was altered to handle voice or data communications. The new service would allow a Basic Subscriber to indicate whether each Qualified Number was for normal use or data transmission. This is likely to involve a change to the Auto Call object, checking for the additional data member to see if it is for voice or data transmission. Objectivity/DB’s schema evolution supports such changes without ever taking the database offline.

Availability also is a requirement in the event of underlying system or network errors. Objectivity/DB provides the ability to divide a federation of databases into autonomous partitions, each of which will continue to operate despite the failure of servers or networks in other partitions. Further, replicas of a database may be created in each partition, providing not only better availability to the replicated data, but improved read performance as well. If a network problem interrupts service to one of the database replicas, another replica will transparently provide the requested data to the application.

**Transaction:** Objectivity/DB provides full support for atomicity, consistency, isolation and durability (ACID) properties in transaction management. In addition, Objectivity/DB provides different levels of granularity for transaction concurrency control.
Transaction characteristics vary significantly among IN applications. For example, two of the most important transaction characteristics are the number of simultaneous readers and the number of simultaneous writers that access the database. These characteristics can vary greatly depending on implementation choices. One Objectivity/DB customer building an IN application tried two different ways of configuring an SCP database. The first used eight server processes accessing eight separate databases with no overlapping data. In this case, there would be only one reader and one writer for each database. The second approach used four server processes accessing a single database. In this case, there would be multiple readers and writers for the database. Service providers may want to make this choice on a service-by-service basis to optimize for the unique calling pattern associated with the services. To meet these varying needs, Objectivity/DB allows IN developers to tailor the concurrency control characteristics of different applications.

Objectivity/DB allows flexible concurrency control through a locking mode called Multiple Reader, One Writer — or MROW — which allows multiple readers and a single writer to access the same data at the same time. The readers view the object as it existed at the point of the last successfully committed transaction. The writer can update the object without interfering with existing readers. If the writer chooses to commit the transaction, Objectivity/DB makes the new updates visible to everyone at the start of the next new transaction. This mode allows the maximum level of concurrent access to the objects for applications with many readers, significantly increasing performance. Service providers that chose database architecture with a single database would find this feature extremely useful.

Objectivity/DB provides another flexible mechanism, called containers, that allows service developers to increase or decrease the level of locking granularity. Objectivity/DB locks entire containers of objects that are logically related. Containers may hold any number of objects and change sizes dynamically to accommodate new objects. Certain services may have a few objects that get updated very frequently. A company that allocates 800 calls to different customer service centers may want to change this allocation dynamically. For these objects, the developer may want to have finelocking granularity to decrease concurrency conflicts. Putting each such
object in its own container gives the finest-locking granularity. This needs to be balanced with the overall system performance since this approach can decrease locking efficiency by processing a lock for each object. Objectivity/DB allows developers to use increased locking granularity only when they need it. The vast majority of objects will be updated infrequently and will be updated in logical groups of objects. Putting many of these objects in each container yields high locking efficiency. Developers can choose the appropriate place on this spectrum on an object-by-object basis.

Performance: Performance is a crucial issue for IN applications. Objectivity/DB increases performance through page clustering and an efficient client cache. As mentioned before, the data model for most IN applications has many complex relationships among objects. A query to the database often requires the retrieval of multiple objects. The performance of the application is directly related to how fast the database can find objects and retrieve them from disk. Disk access is the single largest factor in determining speed of an application. Since the page is the smallest unit of disk access, whenever the disk needs to retrieve an object, it grabs the entire page on which the object resides. Consequently, the placement of objects on the disk greatly affects the performance of the application. Retrieving two objects on different pages requires two disk accesses. Retrieving a hundred objects from the same page will need only one. Objectivity/DB’s page clustering allows a developer to achieve the latter outcome. When creating an object, the developer specifies another object that the new object should be near. In our example, we would want to put all Qualified Number objects and their associated Qualified Number List objects on the same page. Therefore, when we retrieve a Qualified Number List from disk, we automatically get the needed Qualified Number at the same time. By repeating this process throughout the application, the developer can greatly reduce the number of disk accesses, thus achieving better performance.

Once the system fetches a page from disk, it caches it in memory. The location of the cache can greatly affect performance also. Some object databases and most relational databases rely on a server side cache. When the database server retrieves a disk page, it copies the page into the cache of the server process and delivers the data to the client one object or record.
at a time. The database must go through the following steps to fulfill a client request for data:

1. Fetch page from disk into server process cache.
2. Copy data into application memory.
3. Application accesses the data, possibly modifying it.
4. Copy data back to server in the case of data modification.
5. Copy data back to disk.

Objectivity/DB improves performance by having the client fetch the page directly from disk into its cache, eliminating the intermediate step of copying the data to the server cache. Furthermore, since the objects are clustered, the application will have related objects already in memory for the next transaction, further decreasing disk access to boost performance.

**Scalability:** Objectivity/DB allows applications to handle variable-length data by providing variable-length array data structures. In our example, a PIN List contains a variable length array of PINs. If a subscriber only defines three PINs, the database only uses disk space for the three PINs instead of allocating fixed space for all 32 possible PINs. This feature keeps databases smaller, decreasing disk space needs and increasing performance. Objectivity goes a step further by allowing objects to have multiple variable-length arrays. Such dynamic length data records can be very useful in other cases also. Imagine that the number of PINs is doubled from 32 to 64 for a subscriber; nothing needs to be changed as far as the Objectivity/DB is concerned. In addition to this, you never have to worry about the maximum database size because Objectivity/DB has been used in applications that exceed 25 terabytes of data.

**Conclusion:** The key objective of the IN architecture is to allow rapid service deployment and customization by using general-purpose computing platforms and distributed object-oriented software technology to handle service logic and intelligence. Objectivity/DB provides excellent support in such IN implementation environments. Objectivity/DB can reduce time-to-market and maintenance cost by working directly with objects rather than relational tables. At the same time, Objectivity/DB provides high availability and reliability, superb performance, excellent scalability and full transaction support. It is clear that Objectivity/DB is ideally suited for IN and other
Using An Object Database In Intelligent Network Applications

mission-critical telecommunication applications.

For more information on this subject, send your inquiry to info@objectivity.com or call (800) 767-6259.