Developing a Multi-Platform Z39.50 Service

Terry Sullivan and Mark Hinnebusch
The Florida Center for Library Automation
Gainesville, Florida

Abstract

It is possible to develop a Z39.50 service that is independent of the underlying computer hardware and operating system and that will interoperate using many different transport mechanisms, such as TCP/IP, OSI, SNA, and Named Pipes. This article discusses the design and implementation of one such service, developed at the Florida Center for Library Automation, with special regard to independence issues.

A System Model for Independence

Z39.50 has been crafted with a single overriding goal, which is to provide a "lingua franca" for search and retrieval between disparate systems, usually geographically dispersed and provided by different vendors. The primary function of the Z39.50 Implementors' Group (ZIG) has been the crafting, through intense negotiation among implementors, of mechanisms that generalize the services provided by, or envisioned by, various system vendors, with interoperability between these various systems as the principal goal.

It has been clear to a number of implementors that Z39.50 not only provides a mechanism for interoperability with others' systems but also offers a powerful model for the distribution of services within a single system or system complex, or between various products offered by the same vendor.

The Florida Center for Library Automation was an early implementor of Z39.50. At the time we began work on Z39.50, we had little reason to believe that we understood how the protocol and the software that we were developing would be used. This meant, in turn, that we could make no assumptions about the underlying hardware and operating system services upon which we would need to build our Z39.50 software.

We posited a distributed model in which Z39.50 agents could communicate with corresponding agents in remote systems as well as with those located in the same system. This is best shown by a diagram. In Figure 1, the relationship of the Z39.50 service providers and service users is diagrammed. Service providers are programs that implement the Z39.50 protocol. Service users are system components that utilize the services of the service providers. In this figure, all of the four elements reside within a single system, but there is no requirement that this be so. Each could be in a separate system, or any two or more could be co-resident.

A traditional database server/requester system, implemented monolithically, can be viewed as the origin and target service users, as in Figure 2.

But suppose you want to serve multiple service users. You can replicate the Z39.50 origin and/or target service providers and tightly couple them to the various service users. Alternatively, you can have a single Z39.50 service provider with the intelligence to support multiple service users. Or, you can interconnect multiple service providers and service users in a distributed network, as shown in Figure 3.

In such a configuration it is imperative that the Z39.50 software be available on a variety of platforms so that the best machine can be used for each particular situation.

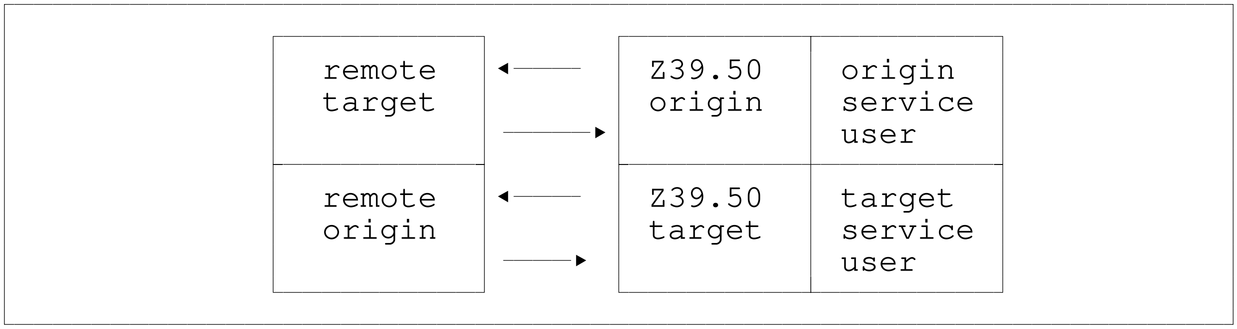

Figure 1

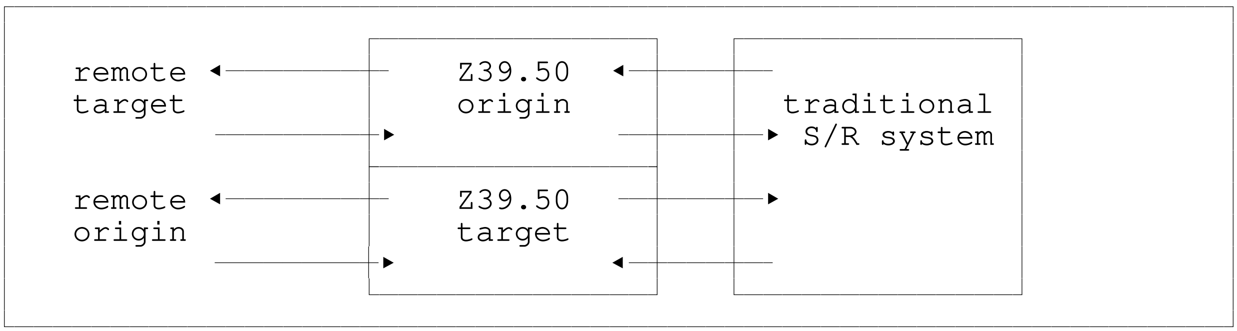

Figure 2

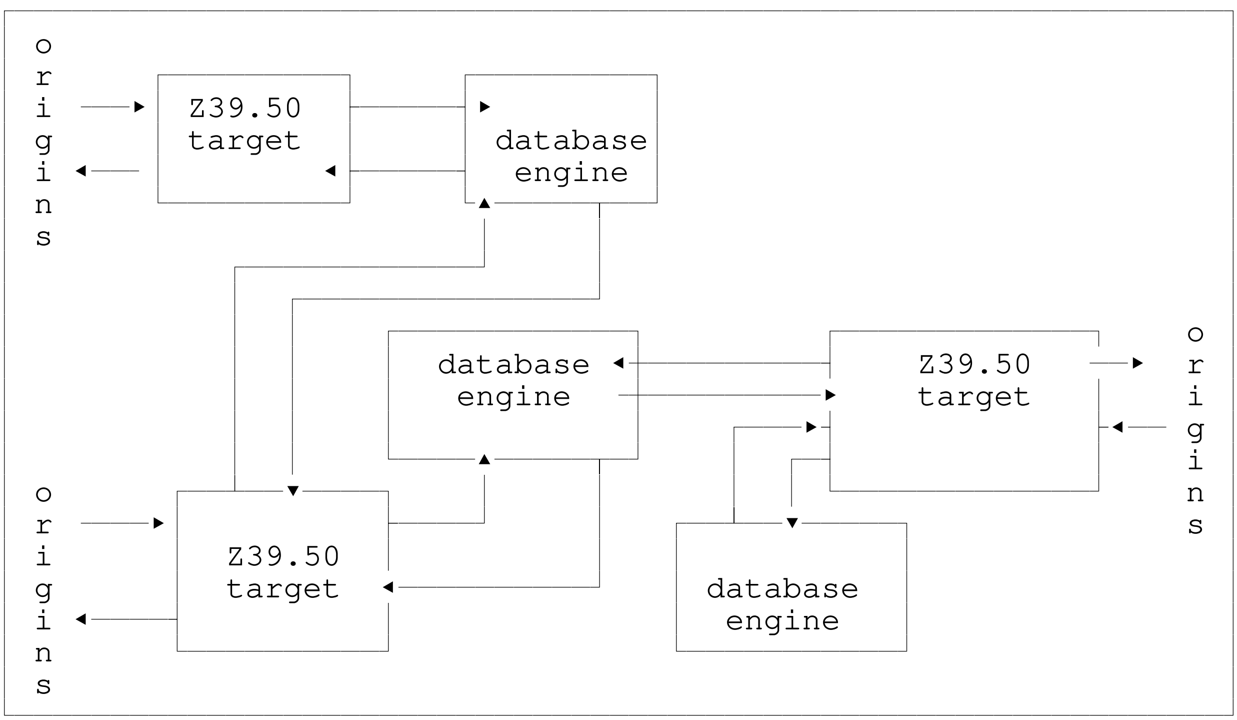

Figure 3

When FCLA first became active with Z39.50, there were strong indications that OSI would be mandated for use in federally acquired systems. The US GOSIP was being promulgated and the State of Florida had followed suit, producing a Florida GOSIP, aligned with the US GOSIP. Because of this, our original plans were based on OSI. However, we discovered, as the ZIG was formed, that other implementors were more interested in offering Z39.50 over the more popular and more widely implemented TCP/IP protocols. FCLA shifted its focus to developing a combined OSI/TCP model. Over time, we have recognized the value of running Z39.50 over LANs, hence the need for NetBios and Named Pipe support.

FCLA developed Z39.50 code capable of supporting the model shown in Figure 3. The code is named AccessFlorida, and is running on an IBM 3090 mainframe as a separate VTAM application under the MVS operating system, on an IBM RS/6000 running AIX 3.2, and on an IBM 486 DX class PC running OS/2. We are in the process of moving the AIX version to a new IBM SP2 PowerParallel multiprocessor and we know of at least one instance of porting AccessFlorida to Microsoft NT.

While AccessFlorida is designed to provide Z39.50 origin and target services, its design allows for its use with many different connection oriented protocols. This feature will be exploited in the future. We are currently providing a prototype Z39.50/HTML gateway using AccessFlorida, and we have long term plans to support X12 (business transactions) messaging using this versatile code base.

The remainder of this paper discusses the design of AccessFlorida as it pertains to independence from hardware and operating systems.

Figure 4 represents the conceptual layering of AccessFlorida onto each system upon which it is implemented. This layering helps to identify and isolate system dependent and network dependent fragments within AccessFlorida. Each piece of the diagram was examined and this analysis affected the design at the very start of the project. The next two sections detail the research that went into the review of the different portions of the diagram, the outcome of this research, and its effect on both the design and implementation of AccessFlorida.

Operating System Independence

One of the primary design goals of AccessFlorida is to isolate the system from all dependencies based on a particular operating system and network interface. This requirement is met in two ways. The first is in choosing a programming language that is supported by all operating systems we plan to use. The second is to isolate all system calls in separate modules and add a thin layer of code to reduce the dependencies on these calls.

In considering the programming language, we examined each targeted platform to determine which languages were supported. On each platform, the network interface was also taken into consideration, since the API presented by these interfaces would also have an influence on the choice of language. The outcome of this research was the selection of C, using the IBM supplied compilers for each platform. On MVS the compiler is C/370, on OS2 the compiler was originally IBM C/2 and later the C Set++ compiler, and on AIX the native C compiler is used.

Even in choosing a common language and using compilers supplied by the same company, care is needed during design and implementation. Each platform has specific requirements that bind both compiler and linker. For example, on MVS, all external names are limited to eight characters; on MVS and OS2 (using the FAT file system), file names are limited to eight characters; and on AIX, file names are case sensitive. Each implementation of the compiler has extensions that are not supported on all platforms. Furthermore, MVS and the other platforms have different file systems.

These restrictions resulted in four design decisions. The first is to limit all filename references to eight lower case characters.

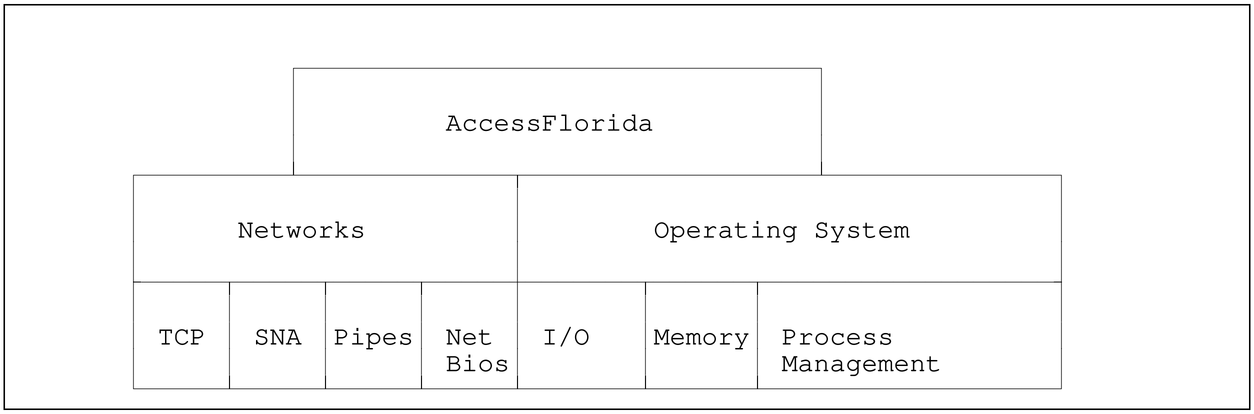

Figure 4

The second is to include a header file in each software module that isolates all system dependent requirements. This header file is tailored to each operating system. It initially redefined every external function name to eight characters for the MVS system; later it was used to easily switch the system dependent header files, again by redefining these names; and finally, it is used to redefine system wide parameters, such as buffer sizes and default parameters. The third design decision is that AccessFlorida will not make any assumptions concerning the location and names of the files it uses for processing. We decided that this will be controlled by setting up the appropriate environment on each system. The fourth decision is to completely base the code on ANSI C and to not use any extensions supplied by the various compilers.

Another area of concern is how each operating system handles process management. On AIX, a process can only spawn duplicate images of itself; on OS2, a process can be multi-threaded or spawn copies of itself, and on MVS one process can create a new process, using ATTACH, which can be an independent process or a thread. Each of these methods, although similar in nature, produces different effects on a software product and adds operating system overhead in the management of these newly created resources. Since the main resources that AccessFlorida manages are the Z39.50 connections between remote clients and remote servers, we decided to design a synchronous system that maintains it own resources in an identical fashion on all platforms. This leads to another design decision to not multithread or spawn, but rather to enhance the management of resources in AccessFlorida in such a way as to provide an efficient scheduler where all resources are given appropriate time to complete processing.

In effect, AccessFlorida becomes a dispatcher of its own threads and resources. The management of these resources and their execution path is implemented in the concept of the Connection Description Block (CDB), and the execution of a state machine. The CDB contains all of the information that must be maintained during a context switch, and it contains only the necessary information. Because of this, internal context switching is far more efficient than operating system context switching as implemented through a process or a thread switch, where the process or thread contexts must be maintained by the operating system.

The last major area of concern lay in the management of memory on the various platforms. The C language enables AccessFlorida to use memory in a consistent manner regardless of operating system. However, since the design required that AccessFlorida handle its resources in a highly efficient manner, we decided that to improve performance, once memory is obtained by AccessFlorida it remains under its control. In short, the software manages its own heap and only calls the operating system when it needs to expand its heap size. This management of memory is further refined into two categories, buffer management and dynamic management.

Buffer management is responsible for obtaining memory for the buffers that are used to send and receive data over the communications networks. The management of these buffers consists of obtaining memory in relatively large chunks and ensuring that this memory remains persistent for as long as the buffer remains active, that is, contains valid data. Each buffer is indirectly connected to a CDB and has enough side information to identify the amount of valid data that it contains, the network that it is associated with, and the connection within that network that is using it. Two identical types of buffers are used: receive buffers and send buffers. These buffers are implemented in a separate module where they are managed independently of the rest of the software. This module's API presents functions necessary to retrieve individual buffers, to clear the memory associated with the buffers, and to provide the ability to reuse every buffer.

Dynamic memory management is used during the encoding and decoding process. As with the use of buffers, this processing has to proceed in an efficient manner. Unlike buffer management, where large chunks of memory are used all at once, this dynamic memory manager is responsible for retrieving relatively small chunks of memory from a larger memory pool. Again, the goal is to reuse memory once allocated, thereby reducing calls to the operating system. The management of this memory resides in a separate software module and is independent of the rest of AccessFlorida. This module allocates a large chunk of memory once, during initialization, and reuses this memory throughout the execution of AccessFlorida. Dynamic memory management is highly efficient and aids greatly in the decoding and encoding of data from and to the network buffers.

Network Independence

Another major area of concern in the design of AccessFlorida deals with the various APIs presented by the targeted networks, and the implementation of these APIs on the various platforms. From the beginning, the design provided for a layered approach that helped to reduce the cost of supporting multiple networks. This layering is conceptually represented in Figure 4, where each segment of the network layer is translated into software modules.

AccessFlorida accesses each network through one set of functions. Access to these functions depends on whether information is being received or sent. The internal API to connect and send data over the networks is implemented in the connection manager, while the functions to receive and accept connections are implemented as part of the main processing of the control program. All of these calls are used at a layer above the network layer to isolate the network specific dependencies from the main processing in AccessFlorida. The overall operation of this interface is depicted in Figure 5.

The actual network interface is implemented in separate software modules, all exposing the same external API to the rest of AccessFlorida. There are currently three such modules in AccessFlorida, one for the TCP/IP network, one for the SNA network, and another for transmission of data via Named Pipes. The OSI support in AccessFlorida is now obsolete and would require some modification to be re-enabled. The network interface modules are used to present the same API regardless of operating system. The implementation of the API is accomplished through these software modules, plus one additional module that actually performs the specific call to the network API. This isolation has greatly helped in the porting of AccessFlorida from the original MVS implementation to both the AIX and OS2 operating systems by isolating the specific implementations in lower layered software modules. The original networks used in AccessFlorida were a pure OSI stack, as provided by the IBM OSI/CS product, and IBM's SNA network. Both of these networks are message driven, delivering information in well defined packets or buffers.
These message based protocols fit neatly with the bit stream generated by the BER encoding of

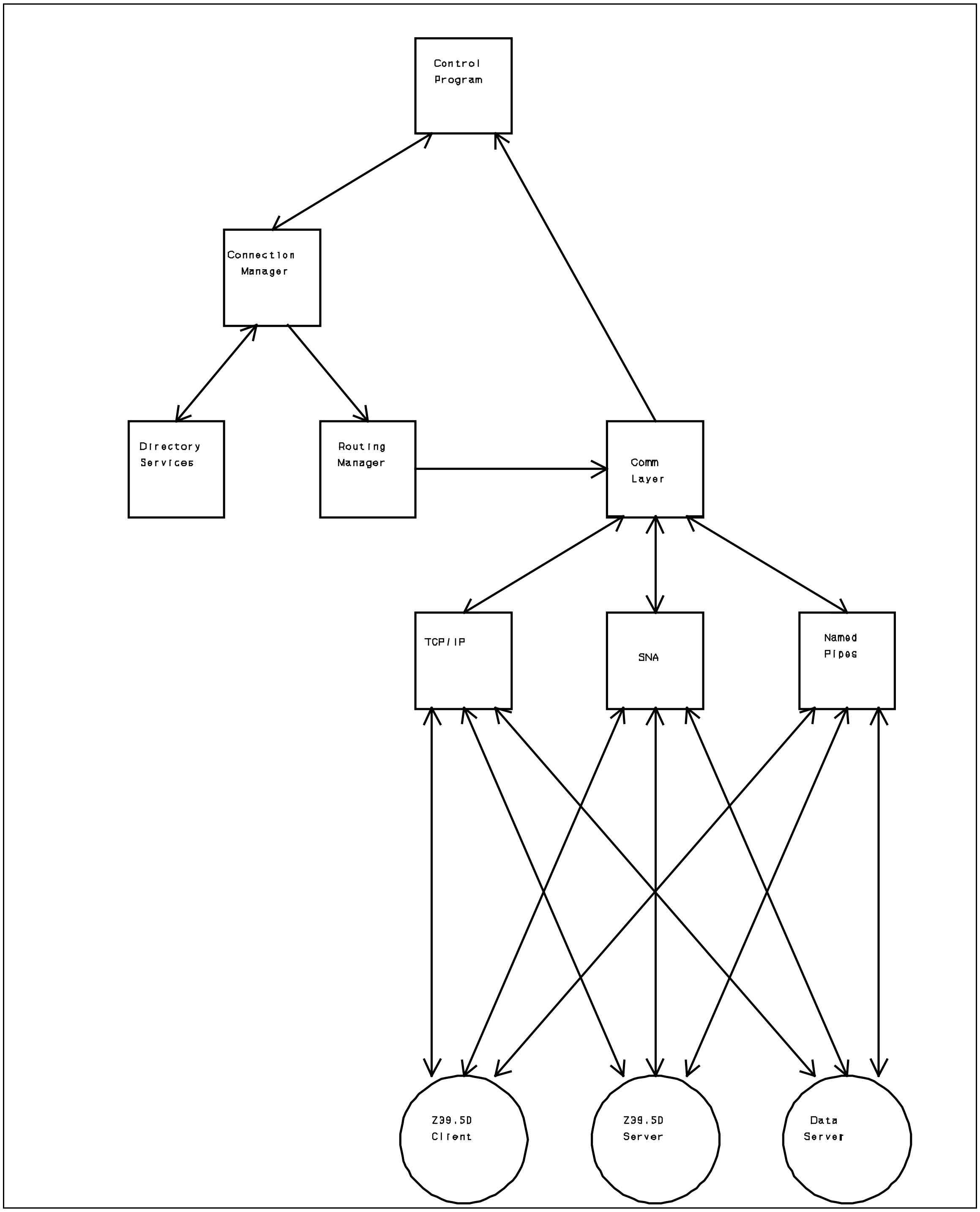

Figure 5

the ASN.1 of Z39.50. This encoding generates well defined delineations within the bit stream sent through the networks. When we dropped OSI/CS, we migrated our BER support to SNA-CC, originally developed by Michael Sample at the University of British Columbia. With the introduction of the SNACC generated code from the ASN.1 of Z39.50 Version 2, and the SNACC runtime libraries, these message driven networks worked fine, since one entire APDU could be received or sent with one call to the network. With the introduction of TCP/IP, which uses a stream based protocol, a layer is required to provide this packetized functionality over the stream-based socket interface of TCP/IP. When remote access was provided by using Named Pipes, a message based scheme was chosen to retain the sending and receiving of complete APDUs.

OSI

The IBM OSI/CS implementation had problems at the Presentation layer. To circumvent these problems, access to this network was accomplished at the session layer. The Presentation layer was then replaced with code generated by the public domain product, SNACC. This was the implementation that AccessFlorida first used when it was released into the public domain as a production system. There was still a connectivity problem with the OSI network. Connectivity is one of the first requirements to interoperate in an open environment. In the United States, OSI support usually requires X.25 as the lowest three layers. While there are several X.25 networks, we were unable to connect with other Z39.50 OSI implementations because there were no adequate bridges between the various X.25 networks and there was no other OSI based Z39.50 service in our X.25 domain. This lack of connectivity led to the introduction of TCP/IP into the network layer used by AccessFlorida.

The OSI network was never ported to the other operating system platforms; its implementation remains solely on the MVS system. However, a by-product of AccessFlorida is the implementation of Tosi, a thin implementation of the upper layers of OSI, that is also transport independent, that has been placed in the public domain.

SNA

AccessFlorida must be able to communicate via SNA since our primary database server is implemented on the IBM CICS/MVS platform, SNA support is excellent in CICS, and other communication protocols were not supported in CICS when we began this project. The implementation of the SNA network is accomplished by designating AccessFlorida as an LU6.2 application, capable of managing its own communication and resources on an SNA network. The specific protocol used for communication is Advanced Peer to Peer Communication, more commonly known as APPC. LU6.2 is an IBM proprietary protocol represented by two publicly available APIs: CPI/C and APPC. APPC is a message based communication protocol, sending data in easily identified packets over a highly delineated and secure communication line. One LU6.2 application can open several sessions, or links, with one or more LU6.2 applications. Sessions may be parallel, i.e., capable of running simultaneous conversations between the applications. In IBM terminology a session is established between the two LU6.2 applications and then conversations are established between specific code fragments, or transactions, of the two applications. Thus the applications are linked at two levels with security provided at each level. Communication is actually at the conversation level and in most instances is two way (i.e., duplex) although one way conversations are also common.

APPC is a highly reliable and secure form of communication, but unfortunately specific implementations of APPC differ greatly. On MVS, access to APPC is accomplished through an assembler interface that is linked into the rest of AccessFlorida. The interface on the other platforms is accomplished through C function calls, but still differ enough that each platform requires its own specific APPC module. These system dependencies are isolated by providing an internal API for this network that is used by the rest of AccessFlorida. All that was required was the rewrite of one module to interface to the specific implementation of APPC on each operating system.

As stated earlier, each pair of LU6.2 applications taking part in APPC communication is required to open one or more sessions between them. AccessFlorida applies a special semantic meaning to establishment of these sessions. Once a session is created between AccessFlorida and another LU6.2 application, the two applications are said to be in a connected state with respect to the network, and a closed state with respect to Z39.50. In OSI terminology, the creation and establishment of these sessions is the same as establishing both a presentation connection and an application association.

Conversations are used by AccessFlorida to send APDUs to the partner application by special convention. Each conversation sends or receives data only one time. Thus a conversation is allocated, a PDU sent (or received) over the conversation, and then the conversation is deallocated. There is never any direct notification that a message is received; only indirectly, by the later arrival of a reply, does the sender know that the message was processed at the destination. In order to associate one message with another within the sessions connecting two Z39.50 applications, the reference id is used. Each Z39.50 application protocol data unit contains a reference id. The Z39.50 standard specifies that this field will be echoed by the target if sent by the origin and supplies no additional meaning to this field. With version 3 of the standard and the introduction of concurrent operations, the reference id is used to give the origin the ability to interleave operations within a connection. This is identical to the mechanism used in AccessFlorida to implement Z39.50 over APPC.

TCP/IP

We added the TCP/IP network to AccessFlorida after the OSI and APPC networks were implemented. TCP/IP gives AccessFlorida the greatest connectivity with other Z39.50 applications. The TCP/IP socket implementation and API are straightforward and fit neatly into the internal network API of AccessFlorida. In connecting AccessFlorida to this network, three aspects of the socket protocol were considered.

The first is the fact that a socket connection within TCP/IP is stream-based, while AccessFlorida expects to send and receive information in chunks, one protocol data unit at a time. To accommodate this difference, two approaches were examined. In the first approach, the PDU is decoded or encoded while the PDU is being received or sent over a connected socket. This approach is more commonly referred to as decoding/encoding directly over the socket. The second approach entails decoding only enough information to guarantee that an entire PDU is received, and encoding only enough information to guarantee that an entire PDU is sent. Each approach has benefits. The first might be faster since data would be moved to and from the "line" quicker. The second might provide better troubleshooting and migration paths since data is received and sent in logical messages specified by the application. Since AccessFlorida expects its data to be delivered in complete PDUs, the second approach was adopted.

In adopting this second approach, the C function to read information from the socket was constructed to determine the amount of data needed to complete the PDU by examining the tag and length specification of the encoding. When BER is applied to an ASN.1 specification, the resulting encoding is said to consist of a Tag, Length, and Value; this is commonly called a TLV encoding. For tags that are defined to be constructed, the length of the constructed value may be specified either directly (the definite form) or indirectly (the indefinite form). When the indefinite form is used the value terminates when two null octets (known as the end-of-content octets) are encountered. Whenever a constructed tag is encountered, the value for this encoding consists of at least one more TLV encoding.

With this information, the socket reading function performs a peek on the socket to look at the first ten bytes. It then examines the tag value to calculate the length of the tag. If two null octets are encountered it decrements a counter for the end-of-contents octets and sets the number of octets to be read to two, otherwise it sets the number of octets to be read to the length of the tag. If no end-of-contents octets are encountered, the function then examines the length value. If a definite length is encountered, it sets the number of octets to read to the number of bytes of the length field plus the number specified by the length field. If an indefinite length is encountered, it increments the number of end-of-contents octets to be read and sets the number of bytes to be read to one since an indirect length takes only one byte. After both the tag and length fields are examined, the function then reads the number of bytes calculated. This logic is repeated until the entire PDU is read.

The second consideration dealt with the mechanism AccessFlorida uses when accessing the TCP/IP network. Recall that AccessFlorida operates as its own dispatcher; no threading or spawning is involved in its processing. Since AccessFlorida needs to accommodate multiple connections, the TCP/IP network is accessed in an asychronous mode. Thus, complete PDUs might not be sent or received entirely in one call to the network. In that case, rather than blocking on the socket, AccessFlorida saves enough information to complete the operation later. When data is being received and a PDU has not been completed, the amount of data received, the number of end-of-contents octets to read, and the number of bytes to read to complete a TLV encoding are saved and the process of receiving data is terminated temporarily. The process later resumes tests to see if there is any data on any active socket ready to be received. If, after processing the other active sockets, this socket has more data, the process of reading the PDU is resumed. A similar mechanism is used when sending data.

Since AccessFlorida may be servicing many connections, care is taken to give every active connection an equal opportunity to be processed. This is accomplished by keeping track of both the active sockets and those sockets having data to be read or sent. AccessFlorida uses two TCP/IP supplied structures and a counter for this purpose. The structures are the TCP/IP fd_set structures, which are used to test the availability of sockets for reading and writing. The first fd_set structure is populated with the appropriate socket once a socket connection is made. The socket is only removed after it has been disconnected. The second structure is populated with active sockets that have data to be read or are ready for writing. A counter is initialized to the number of sockets in this second structure. Once the second structure has been populated, the first available socket is removed, the counter is decremented, and the socket is processed. All sockets in this second structure are processed before it is reset using the first structure. Using this method, all sockets that are ready for processing are guaranteed to be processed before any socket is reprocessed.

The third concern dealt with detection of a closed socket. A closed socket is used in version 2 of Z39.50 to abort or gracefully close a connection. AccessFlorida not only needs to detect when a socket is closed, but also to translate the close into either an abort or simple close. The detection of a closed socket is handled by the return code from any TCP/IP system function. Depending on which function is called, AccessFlorida translates the TCP/IP error into either an ABORT or a RELEASE RECEIVED.

Named Pipes

The last network layer to be added to AccessFlorida was using Named Pipes. This implementation is similar to the SNA network implementation. The addition of this network enables database servers on local area networks to be accessed via Z39.50.

AccessFlorida Components

The various components making up the AccessFlorida system are classified according to the tasks they perform. Each component is briefly described in this section.

The Initialization component consists of the C functions necessary to establish the operational environment and call the communications protocol interfaces to establish the various network environments. This component calls each of the software managers via their respective initialization functions thereby completing the initialization of AccessFlorida.

The Control Program is the primary control process and the heart of the system. It is essentially an infinite loop which calls the network components as needed to perform the main work of the system.

The OSI component is a set of programs that call the necessary OSI/CS subroutines to initialize the OSI environment, accept new origin associations, initiate new target associations, listen for and receive APDUs from end systems, send APDUs to end systems, terminate associations, and terminate the OSI environment.

The TCP/IP component is a set of programs that call the necessary subroutines to initialize the TCP environment, accept new origin associations, initiate new target associations, listen for and receive APDUs from end systems, send APDUs to end systems, terminate associations, and terminate the TCP environment.

The APPC component is a set of programs that call the necessary APPC subroutines to initialize the APPC environment, build new logical associations, listen for and receive messages from end systems, send messages to end systems, terminate logical associations, and terminate the APPC environment. Some APPC services are available from the underlying operating system or there may be basic APPC support subroutines as part of the interface if the underlying system provides only partial APPC support.

The connection manager is implemented in several modules and operates on a connection description block (CDB). It is responsible for keeping track of each connection, the state of the connection, the partners involved in the connection, the messages received and sent on the connection, and the networks involved in the connection.

The buffer manager is responsible for managing all of the receive and send buffer structures in AccessFlorida. It consists of C functions to create and initialize the buffers, to identify and return the buffers, and to release or free any memory associated with the buffers.

The state machine verifies that incoming Z39.50 APDUs and APPC messages are valid for the state of the association, modifies the state to reflect the incoming APDU or message, and defines the actions to be taken in response to the incoming APDU or message given the state of the association. The state machine is implemented in its own module and uses the functions that are made available through the CDB manager.

The Z39.50 protocol manager generates Z39.50 APDUs to send to end systems, using information maintained in memory and associated with the CDB. It also sets information in memory based on the content of APDUs received from end systems. The Z39.50 protocol manager is the only component of the AccessFlorida system which must be aware of the structure of Z39.50 APDUs.

The protocol identifier module identifies the appropriate protocol and is only aware of the structure of the protocols needed to uniquely identify the message and discover which CDB it is associated with. This allows AccessFlorida to operate ultimately as a multiprotocol gateway.

AccessFlorida encodes and decodes all messages through one module that interacts with the SNACC generated decode/encode functions and the SNACC runtime library. The translator converts Z39.50 PDU contents to an internal form and attaches the resulting structures to the CDB. The translator is logically imbedded in the modules used for encoding and decoding messages and is a logical construct. The use of the translator makes integration of new protocols easier and less confusing. By mapping all messages to an internal structure, the problem of supporting more than two protocols is overcome.

The attribute mapper performs Z39.50 attribute mapping to enhance the interoperability of the two connected systems. It operates on the internally defined structure representing a search request.

The routing manager ascertains the target system, communications protocol, and application protocol for a database specified by an origin in a Z39.50 SEARCH APDU. The routing manager logically associates the database with the target and the protocols needed to reach that target.

To identify the target system the routing manager uses a database directory. The routing manager contains the necessary functions to ascertain the target which owns the database being requested and returns both the application protocol and communication protocol used by this target.

The function of identifying the application protocol being received by a remote origin is part of the protocol identifier. The communication protocol is handled by the routing manager which does the actual routing of the messages to the appropriate communication network.

Termination routines perform the housekeeping functions necessary to cleanly terminate processing. They also call the communications protocol interfaces to terminate the OSI, TCP/IP and SNA APPC environments.

Program Logic Flow

The initial entry point of the AccessFlorida system is the main function. This program is started by the operating system. The first function call is to the initialization manager that calls the appropriate functions to initialize all control structures, the CDB chains, and the state machine. It then calls the functions to initialize the OSI, TCP/IP and APPC interfaces to establish the necessary communications environments. Once initialized, AccessFlorida sets a timer and enters a loop that terminates once the timer has expired or AccessFlorida encounters an unresolvable error condition. In this loop AccessFlorida processes all incoming messages on the networks. For each iteration of the loop, the support manager is called to generate statistics and other related information. Eventually the timer expires, and control passes to the termination routine which performs orderly shutdown.

The control program calls the OSI, TCP, APPC or Named Pipe interfaces to accept new associations and to receive Z39.50 APDUs or other messages.

When a message is received by the OSI, TCP/IP, APPC, or pipe interface, control is passed to the connection manager so that it can either initialize a new CDB or obtain the CDB associated with the association upon which the APDU is received. Control then passes to the protocol manager which identifies the application protocol. Using the information returned by the protocol manager, the actual CDB is identified, and the input for this CDB is updated. This CDB is then passed to the state machine which uses an action table to process the CDB and send the appropriate messages to either the remote origin or remote target. This sequence of events is similar for each network.

The State Machine, using the information in the CDB, validates the APDU and identifies the appropriate actions to be taken in response to the APDU. The State Machine extracts from its internal state table the identity of a subroutine to be called to handle the input, given the current states of the two sides of the conversation. The subroutine performs an action and sets a return code reflecting the results of that action. The State Machine uses the subroutine identifier, the current conversation states, and the return code from the subroutine to enter the action table to extract a new set of conversation states which it stores in the CDB. If the "continue flag" is on in the action table entry, the State Machine then reenters the state table to ascertain additional actions to be taken. The continue flag is used to force AccessFlorida to switch from one service provider to another, or to continue processing as the current service provider.

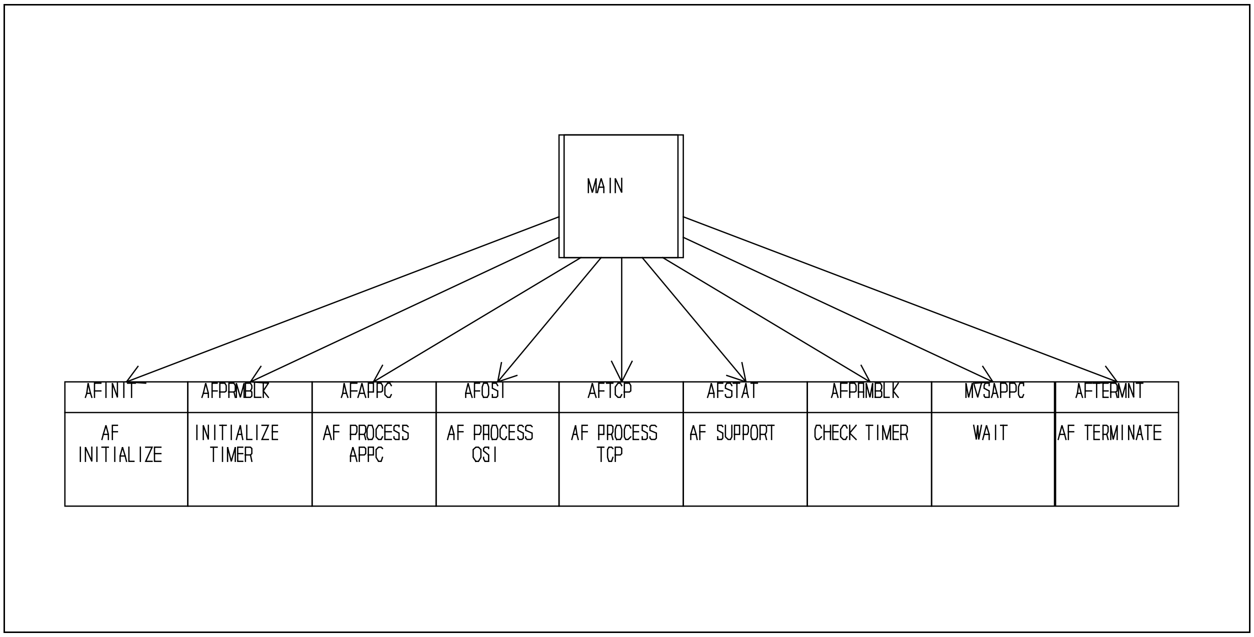

Figure 6

This process continues until the action table entry continue flag is off, at which time the State Machine returns to the calling function in one of the network modules.

If the input is a SEARCH APDU, one of the routines called will be the Query Translator which will perform the necessary translation, calling the attribute mapper to complete that task.

The Z39.50 protocol manager is called to create any Z39.50 APDUs that must be generated and the appropriate communications interface is called to send the generated APDUs.

Conclusion

FCLA expended a significant effort during the design and implementation of its Z39.50 software, AccessFlorida, to ensure future portability across hardware platforms, software bases and communications protocols. This effort has proven worthwhile by the ease with which AccessFlorida has been ported from its original platform, IBM's mainframe workhorse, the 3090 running MVS, to an RS/6000 running the AIX flavor of Unix, and to a PC running OS/2. Work has been done to support communications over TCP/IP, OSI, SNA, and Named Pipes.

The modular approach taken in the design and implementation of AccessFlorida ensures portability to future hardware and operating system options. We know others are interested in porting our software to the OS/400 operating system running on the IBM AS/400 family and we are aware of a working port to Windows NT.

AccessFlorida is designed as a generalized protocol manager and state machine, capable of providing support for, and translation between, multiple protocols, at both the transport and application layers. At the transport layer, AccessFlorida already gateways between TCP/IP, APPC, and Named Pipes, and OSI support could be re-established with minimal effort, should the need arise. At the application level, AccessFlorida currently translates between Z39.50 and internal search engine messages. We have a prototype Z39.50/HTML/HTTP gateway running and have long term plans to perform X12 transactions via AccessFlorida.