



Platform Independent SS7

Impact of SS7oIP on SS7 Application Development

A White Paper produced by

IntelliNet Technologies, Inc.

1990 W. New Haven Ave., Suite 312

Melbourne, FL 32904 USA

Tel: 1 (321) 726-0686

FAX: 1 (321) 726-0683

Web: www.intellinet-tech.com

Copyright © 2000 IntelliNet Technologies, Inc. All rights reserved.

Executive Summary

Implementing platform independent applications have always been on most developer's wish list. The emergence of a new range of telephony platforms has pushed that to the forefront.

Porting to a new platform is not simply moving to a new operating system. It includes duplicating the entire environment and all application interfaces on the new platform

This paper follows the progress of SS7 applications over the years and reviews a generic architecture of an SS7 application. It identifies a core state machine and five interfaces - SS7, Database, Switch control, Management and 3rd party IPC.

The paper outlines development techniques that can be used to ensure that porting SS7 applications and all the interfaces can be achieved at minimum effort. A number of SS7 stacks are reviewed. Comparisons are made with the platform independent IntelliSS7™ Suite of products including advantages of one approach over another.

Finally, it looks at the emergence of SS7oIP and the requirement for transport independent applications that can seamlessly operate in both the SS7 and IP networks.

Content

1. Introduction	3
2. The Case for Platform Independence	3
3. SS7 in the Telephony Network	4
4. The Architecture and Interfaces	5
5. Platform Independent Development Model	6
5.1 SS7 Application Engine	6
5.2 SS7 Interface	6
5.2.1 Access Method	7
5.2.2 Data Format	7
5.3 Switch Control Interface	8
5.4 Database Interface	8
6. Influence of SS7oIP	9
7. Limitations of Platform Independence	9
8. Conclusion	10

1. Introduction

With the rapid convergence of voice and data networks, the ability to inter-work between the two has become a necessity. The primary impact of this convergence has been the need for traditional telephony services like free-phone and virtual private network to operate seamlessly in data-centric networks. This requires interoperability between the IP and SS7 based network databases (known as SCPs).

To the technical community this has provided further momentum to focus on developing applications that can operate on multiple platforms. As a leader in developing SS7 and IP based telecom applications, IntelliNet has been at the forefront of building platform independent SS7 solutions. The IntelliSS7™ Application Development Environment (ADE) embodies this work, and allows third parties to leverage this technology.

This paper shares some of the techniques that have been developed and refined at IntelliNet to deliver such platform independent solutions.



2. The Case for Platform Independence

Platform independent applications can be ported from one platform to another with “minimal” effort. While there has always been a strong business case for incorporating platform independence, the advent of VoIP has added momentum.

New world applications: Convergence of voice and data networks has opened up the telecom industry to a new range of platforms, and dissolved the myth that telephony applications can only run on a limited number of hardened platforms.

Varying market segments: Applications often need to address opposite ends of the market requiring both a low cost non-redundant platform and a higher cost redundant platform.

Changing performance requirements: Changing market size may require applications initially deployed on low performance platforms to be upgraded to high volume environments.

Leveraging new technology: Technical obsolescence may require applications to be moved to a more appropriate platform.

Better price points: Being locked into a specific platform may put you at a negotiating disadvantage with your current platform supplier for price and features.

Portability of course comes at a cost! Developing portable applications require a longer design cycle translating to more time and money. However, the benefits of portability can far outweigh these costs.

The rest of this article is dedicated to reviewing development techniques that can be used to achieve such portability at minimum cost.



3. SS7 in the Telephony Network

SS7 has been accepted as the standard for signaling in the telecom environment. In the 1970's it started as a cost effective and efficient mechanism to exchange call setup messages among switches in a telephony network (ISUP). While it provided a significant performance improvement over traditional in-band signaling, it required a substantial investment to deploy a high-speed signaling network to overlay the already existing voice network.



Figure 1: Using SS7 for Call Control

The growth of Intelligent Network (IN) applications in the 1980's, leading to the separation of switching and service control functionality, imposed additional signaling requirements between the switches (SSPs) and network databases (SCPs). IN leveraged the SS7 signaling network already in place, fueling further growth in the SS7 network. The SS7 protocol was extended to support network database queries (TCAP/AIN).



Figure 2: Using SS7 in Intelligent Networks (IN)

The 1990's heralded an era of explosive growth in the wireless industry with its own set of geographically dispersed network elements and an extensive list of signaling requirements among these elements. Features like seamless roaming, subscriber authentication and short messaging service required subscriber databases to be distributed and accessed throughout the network. This required high speed signaling between mobile switches (MSC) and subscriber databases (HLR). Once again the SS7 network was leveraged and additional protocols like ANSI-41D and GSM MAP were developed on top of the SS7 signaling network to deliver the information.

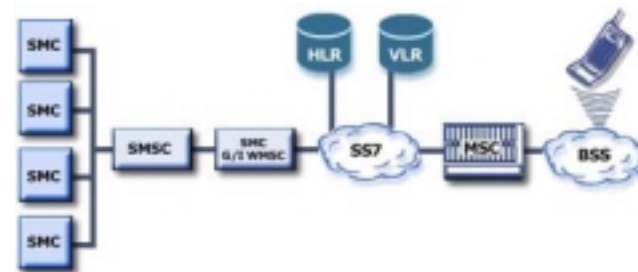


Figure 3: Using SS7 in Wireless Networks

As a new generation of IN and wireless applications (WAP) and location based services proliferate the marketplace, there will be continued demand to leverage and extend the SS7 network. This will continue to influence the way applications are written as highlighted in the next section.



4. The Architecture and Interfaces

Wireless and IN technology have led to an exponential growth in the number of SS7 based applications. Some of these applications are purely signaling based (HLR, SCP, SMS). Others are more complex and require integration between voice and signaling traffic (MSC, SSP, IP).



Figure 4: SS7 Application Interfaces

Irrespective of the end product, all SS7 applications are bound by certain common programming requirements (transaction management, state machine, event driven etc.). The following diagram provides a high-level generic block diagram of a typical SS7 application.

1. **The SS7 Application Engine¹** forms the core of the application. It receives inputs from a number of internal and external interfaces and takes the appropriate actions to coordinate them. The Application Engine is usually implemented as a state machine.
2. **The SS7 Interface** is used by the Application Engine to communicate with the SS7 network.
3. **The Database interface** provides the application to subscriber information used in call processing.
4. **The Switch Control interface** is used to communicate with the switch matrix for applications that require a voice interface.
5. **The Management interface** is used to send events, alarms and statistics and in some cases it is used to receive configuration information.
6. **The Third party IPC** shown in Figure 4 is completely application dependent. Most applications implement proprietary access mechanisms for communicating with the application engine.

¹ This is also often referred to as the call processor



5. Platform Independent Development Model

The primary objective of platform independence is to implement portable applications. Porting to a new platform requires the ability to duplicate all the interfaces including SS7 stack, database interface, switching control interface etc. For example, moving from a Windows NT to a Sun Solaris platform may require selecting a different SS7 vendor and database management system.

In the context of the generic architecture shown in Figure 4, we look at the impact of platform independence on the various interfaces.

5.1 SS7 Application Engine:

Developing the core application engine is the most complex task in the overall implementation and it is likely to use features and capabilities offered by the operating environment. Here are some of the primitives used by the application that are operating system specific.

1. **Threads** are contexts of execution that allow multiple paths of execution within the same process context. They are scheduled and dispatched independently by the operating system and are very efficient. Using threads in the implementation of the SS7 application engine can substantially decrease the complexity of the state machine. Unfortunately, most operating systems have a slightly different implementation of threads.
2. **Inter-process Communication** is required for processes and threads to communicate with each other to share resources and information. The kernel must provide mechanisms that make this communication possible. Traditional operating systems typically provide mechanisms like semaphores, message queues, FIFOs, pipes, and sockets. Other mechanisms may include shared memory, shared files, signals, and process tracing.



Figure 5: Operating System Primitives used in SS7 Applications

3. **Synchronization** is required by threads to serialize access to shared resources. These mechanisms include mutual exclusion locks, task queues, and condition variables. Other mechanisms include spin locks, light semaphores, and reader writer locks. All multi-threaded implementation require such support functionality.

It is clear from these examples that the application engine needs to use operating system primitives that may vary from system to system making the application platform dependent. The most common strategy is to develop a platform neutral abstraction of most of the concepts and mechanisms described above. IntelliSS7™ provides such capability through the IntelliSS7 Application Toolkit™.

5.2 SS7 Interface:

While the functionality of the SS7 protocol is well defined and implemented by all SS7 vendors, there is no agreement on the application interface to their SS7 stack. Applications developed on a specific vendor SS7 stack will not port to a competing vendors stack.

While reviewing a number of SS7 vendor interfaces, we noted that the interface differed based on (a) the access method to the stack and (b) the data format between the user application and the SS7 stack as discussed below.

5.2.1 Access Method

TCAP based applications interface with the SS7 stack at the TCAP layer. To send data the user needs to build the data block as per encoding rules defined by the application protocol and then use facilities provided by the stack vendor to deliver the message to the SS7 network. The sequence of operations is reversed for receiving data.

There is no standard IPC mechanism between the application and the SS7 stack. Some vendors use TCP sockets, others use message queues, while other implementations totally hide the internal IPC implementation by invoking user provided call back functions.

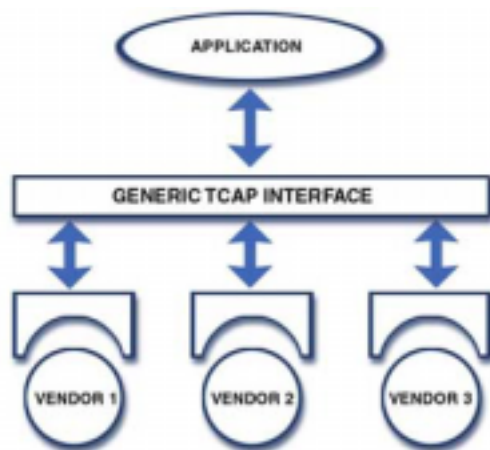


Figure 6: Vendor independent SS7 interface

To prevent the application from being inextricably tied to the vendor stack, it is better to develop an abstraction layer above the vendor interface. First it is necessary to select an access method and then write a thin interface layer to normalize access from the other vendors to your chosen interface.

IntelliSS7™ has adopted a socket based IPC mechanism and normalized other vendor interfaces to map to this interface. This isolation layer abstracts the vendor specific access mechanism from the application layer allowing for easy portability of the application among the various stacks.

In addition to the IPC, vendor stacks can differ in the way data is extracted by the user application from the stack's internal buffers. Some implementations buffer the entire TCAP message as a single unit and deliver the entire packet when the application requests TCAP data. In other implementations the stack passes a single component pre request.

Similarly, when the user sends a TCAP message, some stacks expect the complete message to be delivered while others expect the individual components to be sent to the stack one at a time.

IntelliNet has selected the "multiple request" fetch mechanism. The vendor abstraction layer shown in Figure 6 normalizes the various vendor implementations.

5.2.2 Data Format

The format of the data being exchanged between the user application and the stack is vendor specific. Most vendors provide header files containing data structures to map TCAP messages to the programming construct. However, vendors have their own version of the data structure.

It is necessary to convert the data to a generic format. One way to do this is to define two basic TCAP data primitives - the TCAP dialog and the TCAP component. Each primitive can consist of a union of subtypes. For example the dialog contains ITU_BEGIN, ITU_END etc. and the component subtypes contain ITU_INVOKE, ITU_RETURN_RESULT_NOT_LAST etc.

Also when dealing with both the ANSI and the ITU specifications, where possible, members of the two data structures should have common initializers, allowing the programmer to minimize the number of locations where they must distinguish between the underlying protocol family. This will help with porting between ANSI and ITU.

There are two problems with applying this approach across multiple vendors. Firstly, it is necessary to deal with stack vendors who don't support all variants or all primitives. Secondly, some vendors choose to build TCAP PDUs within user-space (i.e., via a library within the application process) while others choose to build them within the stack itself, which can make normalizing the communication with the TCAP layer fairly challenging.

5.3 Switch Control Interface:

While some applications need SS7 signaling capability only, other more complex SS7 network elements require both voice and signaling capability. Such applications not only require establishing voice path, but also need the capability to collect digits, play announcements and switch voice circuits.

Voice resources are provided by programmable switches like the Excel LNX and Summa-4 VCO, or by individual voice cards. Whatever the selection, user applications are likely to encounter proprietary programming interfaces provided by individual vendors. This makes porting the application from one set of voice resources to another very complex.

It is possible to use a similar approach as described in the section 5.2 to achieve switch abstraction. The challenge in implementing such an interface is being able to balance two seemingly contradictory goals - providing flexibility to allow the user's low-level access to as many of the features and parameters as possible while also making the interface easy to use so that it can free up the higher level applications from having to know the details of the switching component.

Most implementations of communication between the Switching Matrix and the application consist of Indications (received from the switch routed to the application) and Requests (received from the application routed to the switch).

IntelliNet has developed a message based API, whose call processing part models a subset of Q.931 - a protocol that has found wide acceptance both in the telco and the data environments. The interface models a subset of Q.931 to provide the application with enough control, but with much simplified message body. This can be further improved by providing another level of abstraction by defining call-processing objects (i.e. Conference object), setting up sequences of actions, better status management.

5.4 Database Interface:

Virtually all SS7 applications require access to a database. This may be a traditional commercial database or homegrown, highly optimized data cache.

Database independence can be addressed to some extent by using a platform independent interface like ODBC, which is available for most commercial databases or platforms. ODBC (Open Database Connectivity) is a widely accepted and standardized API for database access. Today it is supported by all major commercially available databases on all major platforms. ODBC uses Structured Query Language (SQL) as its database access language. There is minimal performance deterioration by using ODBC vs. native access.

Additionally, ease of use can be obtained by developing code generators that can produce user friendly API's that uses ODBC to communicate with the underlying database stack. The data base schema can be the input to the code generator (similar to ASN.1 syntax being the input to the ASN.1 compiler). IntelliSS7 uses such an approach.

.....

6. Influence of SS7oIP

The advent of SS7oIP has added new dimension to platform independence. There is a need for transport independence - developing applications that can span the SS7 and IP networks.

An ideal development environment would make the application writer unaware of the underlying transport. Applications should be able to simultaneously operate in multiple networks where some end points can be in the SS7 domain while others are in the IP domain. Most recommendations of SS7 over IP currently under consideration preserve the upper layers of the SS7 stack (TCAP and SCCP) making this task easier. The lower transport layer (MTP) is replaced by IP.

Ironically, if you have designed your application to be SS7 vendor independent, you will have done most of the work in achieving transport independence.

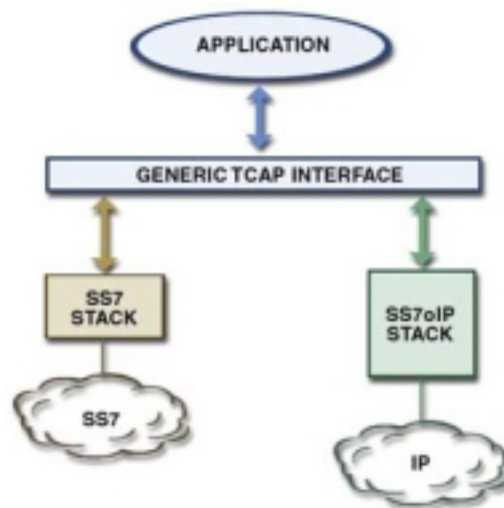


Figure 7: Transport independent SS7 Application interface

There are a number of forum working on implementation of SS7 over IP. The most notable among them is SIGTRAN, TALI and GDI.

.....

7. Limitations of Platform Independence

However, it should be noted that platform independence does have some limitations. These limitations are performance and the quality of the API. They are:

1. Likely to require additional investment both in time and money during the design and the implementation phase.
2. Constrain the design since developers have to accommodate a number of disparate platforms.
3. Require higher level of expertise than may be readily available within the corporate environment.

.....

8. Conclusion

In conclusion, it is safe to say that with the passage of time the need to build applications that can be easily ported to various vendor platforms will become mandatory. This is best achieved by designing in the “platform independent” capability within the applications.



 Content