S38.3115 Signaling Protocols – Lecture Notes

Lecture 2 – Modeling of signaling systems and historical examples in PSTN

Modeling of signaling 1
Signaling Flow Chart 2
Finite state machine (FSM) and EFSA 4
Representing FSMs graphically 5
Telephony signaling specifications are written in SDL 5
From specifications to implementation 7
Other representations of FSM and EFSA 8
Generations of exchanges 8
Hardware and software of digital switching systems 9
Classification of signaling systems 10
Examples 11
Analogue subscriber signaling 11
R2 – an example analogue trunk signaling system 13
Limitations of analogue signaling systems 14

Modeling of signaling

Signaling and call control are implemented in software. Two exchanges implemented by different vendors need to be able to accurately talk to each other and understand exactly what the other is sending. The end result needs to meet high availability performance requirements. If two vendors understand a written specification for a signaling system in even a rarely occurring situation differently, the result is less than satisfactory. It is said that if we need one unit of design work to implement a program for non-real time data processing with 1000 lines of code, a real-time program of the same size may easily require 10 units of design work, a real-time program of the same size that must talk to a system programmed by someone else possibly using another programming language may require 100 units of design work. If that program needs to be adaptable to slightly different environments, e.g. in different countries, things become even more complex – even more design efforts are spent. The last applies to for example signaling in telephone exchanges.

To achieve a high level of accuracy in specification of signaling functionality, more than text documents are needed. One can claim that actually for the purpose of designing a large system or even a subsystem or a non-trivial

©Raimo Kantola (2008)

program a model is always needed. The model will depict the grand idea the programmer has in mind while structuring the system. Without such an idea, all that results is spaghetti that no one else besides the author will understand and will be unable to maintain. One can generalize this by the claim that a large system like an exchange can be seen as a set of models organized hierarchically on program, on subsystem and on network element levels.

Maybe the most traditional method that is used to describe signaling is called a *flow chart* or signaling chart. Figure 2.1 shows an example. It reflects an analogue signaling environment for PSTN subscribers.

Signaling Flow Chart

A signaling flow chart will have several participating *elements*, systems or subsystems that *send and receive signals between each other for a particular purpose or a situation*. The purpose, the context or situation may be shown in the title of the chart. The chart will show the *sequence of signals* between the elements. Signals usually have names. Since the chart shows a sequence, time passes from top to bottom of page on the chart. The chart does not have a time scale, it just depicts the sequence.

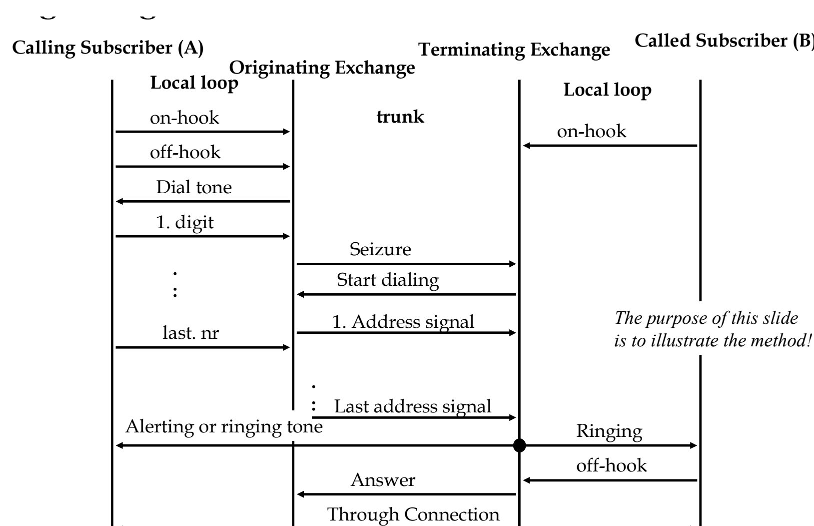

Figure 2.1: A signaling flow example.

In our example (Figure 2.1) we take a look at a hypothetical analogue PSTN network. In the beginning, *off-hook* is the event when the caller picks up the phone. Prior to that the phone was in the *on-hook state* or the previous event of the phone was when the phone was put on hook. The response of the originating exchange to the off-hook signal is the *dial-tone*. When the caller hears the dial tone, he or she will start pushing buttons on the phone or rotate the disk on the phone. When the originating exchange has received enough digits, it will be able to determine the circuit (a timeslot in the PCM system) or

trunk that it will use on the outgoing side and it will send the *seizure signal* onto that trunk. This signal is used to indicate the reservation of the trunk for a new call.

Our example shows only two involved exchanges. On reception of the seizure signal, the terminating exchange responds with a *start-dialing signal*. The terminating exchange will need some dialed digits to determine which of its subscribers it should alert. These are sent in *Address signals* one by one. When all of the dialed digits have been received by the terminating exchange, it will send a Ringing current or voltage to B-subscribers local loop. This will make the phone ring. In the backward direction the terminating exchange will send an *Alerting or Ringing tone* on the backward voice path. At this moment, the backward voice path needs to be through-connected in the originating exchange.

Off-hook at the terminating side indicates that B-subscriber has answered. On reception of the off-hook signal, the terminating exchange will send an *Answer signal* in the backwards direction. When the Answer signal is received by the originating exchange, it will through-connect also the forward direction of the voice path and the involved parties can talk. At the same time, as the through-connection is made by the originating exchange, it will start collecting charging information for the call.

In PSTN pulse charging was used, pulses were collected in counters for all calls, for trunk calls etc. The pace of pulsing depended on the call tariff. In GSM and in many other modern networks, time based call charging is based on call seconds. In such a case, a CDR – Call Data Record (possibly ca 1 kbyte of data) is created for each call. The CDR will store the caller's and callee's telephone numbers, start time of the call and end time of the call and possibly other relevant information.

The advantages of signaling flow charts are

1) flow charts are visually clear and easy to understand
2) the whole series of situations that may take place related to a call, can be broken down into several charts and the whole set of charts will clearly show the sequence of main events in all important situations

The disadvantages of the flow chart method are

1) the difficulty of showing all exceptional sequences: if one tries, the result is a large set of very similar charts that may easily confuse the reader and have a lot of redundant information.
2) A typical situation is that an element is signaling in two directions at the same time, at least release should be possible from both parties at almost any time. This is difficult to show or leads to a lot of very similar charts and thus easily confuses the reader.

©Raimo Kantola (2008)

It follows from the advantages of the flow chart method that for signaling specification, the method is often the first that is used to lay down the main sequences that constitute the purpose for a protocol or a service. For accurate specification some other method is needed.

Finite state machine (FSM) and EFSA

A finite state machine is a five tuple $< s_0, I, O, S, f >$ where s_0 is the initial state, I - the set of incoming signals, O - the set of outgoing signals, S - the set of states and function f is the mapping: $I \times S \rightarrow S \times O$.

FSMs are widely used in computer science for various purposes. One example is machine language processing. State represents the concept of memory. Consequently, the basic hardware elements that make computer memory, i.e flip-flops are also nicely modeled as FSMs.

It turns out that in telephony and for signaling protocols almost every program in an exchange can nicely be modeled using FSMs. The model that is used is a bit richer than a basic FSM. We call this an Extended Finite State Automaton or EFSA. In addition to state, it has memory in secondary state variables. This extension reflects the need to process several different kinds of data in a signaling or call control program.

Now imagine that an exchange has four functions (programs) each modeled as an EFSA for processing a call: incoming call signaling process, incoming call control process, outgoing call control process and outgoing signaling process. The resulting programming model is called that of *communicating state machines*. This is very different for example from the well known *client-server model*. In the client server model, the server does not have state related to a particular call or service. The server just responds always in the same way when asked irrespective of what happened earlier. Of course it is common that the server has some internal data, it may even have a database. However, from the client perspective, the server responds in the same way irrespective of what the client has asked earlier.

Programmers typically believe that programming in the client server model is much easier than programming communicating state machines. This may be because a system of communicating state machine exhibits some peculiar failure modes. Such a system, because of a programming error, may end up in a dead lock – a state from which it will not recover on its own. Because of this, a lot of computer scientists have spent their whole lives in creating methods that would allow detecting dead-locks or more generally errors in specifications or implementations of this type of programs. As a result, a lot of methods and tools have been created.

Representing FSMs graphically

May be the most traditional graphical representation of FSM is the one with bubbles and arrows as in Figure 2.2: An example FSM.

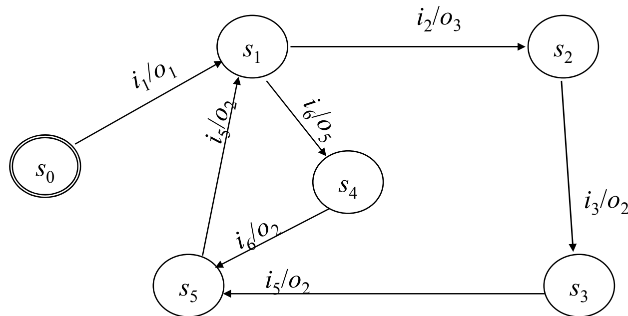

Figure 2.2: An example FSM

A circle or a bubble shows a state and an arrow shows *a state transition*. A state transition is labeled with the signal that caused it to take place and possibly (after the slash) with the signal that was sent out in the transition. This representation is compact and exact. Its limitation is in the difficulty of adding secondary state variable without loosing clarity.

Telephony signaling specifications are written in SDL

ITU-T (or actually its predecessor CCITT) has specified the *Specification and Description Language or SDL* for short for the purpose of writing signaling protocol specifications. Consequently richer and richer variants of the SDL language were standardized in 1980, 1984, 1988 and 1992 by CCITT. Many companies, among them software tool vendors and switching system vendors have created their own tools for writing SDL and gradually adding programming level details so that the tool set can be used first for system independent specification, then for adding system specific (such as DX 200 specific) details and finally for writing signaling and call control programs themselves. It may be that only program skeletons are written in SDL while many subroutines that are needed in the state transitions are written in some other language such as C or C++.

SDL has both a graphical and a textual representation. A SDL toolbox usually has means of producing one from the other. Graphical representation is useful for specification purposes. When details are added and translation to programming languages such as the C-language need to be done, the textual representation becomes more important. A well designed SDL toolset is able to maintain correspondence between the graphical and the textual representations while the details are added and even when the programs are later modified to meet new emerging needs.

©Raimo Kantola (2008)

In any graphical SDL, there are symbols for *state*, *reception of a signal*, *sending of a signal, a decision* and *a task*. These symbols are shown in Figure 2.3.

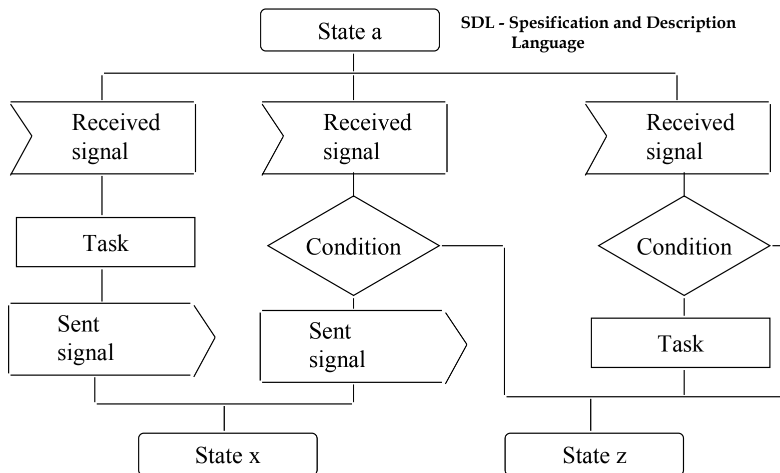

Figure 2.3: An example SDL

Usually, one state with its transitions is depicted in one Figure. It gives all expected signals in the state and possibly shows also how any other signals are processed in that state. A transition always ends in the next state of that transition. Naturally, under different conditions the transition can branch and end in different states. Signal names from the graphical representation can be nicely mapped to the textual representation. The content of a task box in older versions of SDL is just comments in programming language terms. Later versions of SDL allow producing programming level code from the contents of task and decision boxes.

The interfaces of programs that are programmed at least initially in SDL, can totally be produced from the SDL. From the SDL, a tool can produce *header files* that can be used for example in the C –program that will give the details of what needs to be done in each transition.

One can take this idea of specifying the program interfaces in SDL even further. It may be convenient that the whole program and subsystem structure or model for a software release is written in SDL. This is possible for example in DX 200.

From specifications to implementation

The main level program that runs an FSM or an extended FSM is like this:

```
Initialisation
Do Forever
        Receive Message
        A <- Branch (State, (Secondary state,) Message)
        Execute Transition (A)
Od
```

So, all the process that is created does is wait for a signal (or a message) to arrive and executes a transition when one does arrive.

Processes that are run-time objects in a real-time operating system can be executed in at least two different modes. In non-pre-emptive scheduling, a transition, like the one above, is always executed till the end before any other program can run. In pre-emptive scheduling, a transition can be stopped by a process or a signal that has *higher priority*.

One can say that processes that allow pre-emptive scheduling are real processes and the ones that do not allow it are some sort of light weight processes. Pre-emptive scheduling may require taking care of some contention situations that can never take place if non-pre-emptive scheduling is assumed. So, designing and programming “real processes” is a bit more demanding than programming “light weight processes”.

In switching systems programming (or real time operating systems) both signal priorities (e.g. AXE by LME) and process priorities (e.g. DX 200 by Nokia) have been successfully used. Priorities are needed to establish the processing order that takes into account different importance or emergency levels of different events in a real time system.

A real time operating systems that is used to run programs created from SDL, must typically support *message passing* efficiently and allow easy mapping of SDL signals to message passing in the run time environment. For synchronization of processes, such a real-time operating system typically also supports semaphores that are used to control the entry of a program to a critical piece of code where two programs may compete for access to a section of memory. Obviously, if two programs may write to the same memory simultaneously, nothing good will take place – the result is a bug that is difficult to spot. In my experience, programming with message passing can be quite simple while using semaphores may be messy. One should also note that passing information between two processes in a real time system using memory resident files may sometimes be useful. Then a good design is such that one program or process writes and another reads. If the second program never writes to the file, it may be that no semaphores are necessary. Rather, the

synchronization either may be left to fate or handled by priorities of processes. The delicate situation is when several items need to written into a file and the writing program may be interrupted while only some of the items have been written while others are still pending.

Operating systems have different properties in terms of how they handle parallel processing of several programs. If context switching between processes is slow, it makes sense to map many FSMs to a process in such a way that context switches between processes are not needed too often. It may also be that the amount of context information needed for a process is large. In such a case, an operating system on a machine can support only a limited number of processes. This is a further argument to map many FSMs into a single process.

Natively, real-time operating systems that have been designed for example for telephone exchanges are quite efficient in context switching and they can support a large number of processes in a single computer (e.g. > 100 000). In such an environment, it is feasible and most straightforward to map each FSM or EFSA to its own process. The result is that the real time operating system will take care of switching between telephone calls. The programmer of the signaling and call control application needs to worry just about a single call.

Other representations of FSM and EFSA

Both can be represented in a Table format.

Generations of exchanges

Historically, automatic telephone exchanges were implemented using *cross-bar and other electro-mechanical* hardware. For example, in Finland during the 1970s and as late as early 1980's, Televa, a government owned company manufactured KMK cross-bar exchanges. (KMK exchanges were even exported in significant volumes). All signals in such systems were analogue.

Between electro-mechanical and fully digital exchanges, some hybrid systems with an electro-mechanical switching fabric, an analogue voice path but digital processor control were manufactured and taken into use.

Digital trunk exchanges started to appear in 1970's. The microprocessor was invented by Intel in 1972. The first *fully digital local exchange* was manufactured by Nokia and taken into use in Korppoo Finland in 1981[1]. It was an early version of DX 200 with 8-bit micro-processors (Intel 8080).

---

[1] Probably, some other manufacturer will deny this and say that they were the first.

Hardware and software of digital switching systems

The early design philosophy of major manufacturers (sounds silly from today's perspective) was that telephony us such a different task from everything else that special purpose processors are needed and they were designed. These processors required their own assembly language and also such special purpose high level programming languages as CHILL (by CCITT).

What actually happened was that special purpose hardware leads to a long time to market of new innovations and a switching system vendor spending efforts into something where other companies designing *general purpose computers and micro-processors* have greater volumes, leading to lower prices and a faster time to market.

The programming environment in digital switching moved gradually to general purpose micro-processors and general purpose programming language such as the C-language. However, SDL as a means of specification with the possibility of a gradual adding of detail and a gradual move to implementation has retained its place in switching systems through the era of GSM till 3G.

PC industry moved gradually into a horizontal value chain: different companies make the micro-processors, PC boards, hardware assembly of PCs, operating systems plus most common programs and finally applications.

Nothing similar took place in the area of digital switching. A digital switching system is a *vertically integrated system*: the manufacturer such as LME or Nokia buys micro-processors and other chips from chip manufacturers, designs its own circuit boards (some of this may be subcontracted but very rarely bought from "open market"), writes its own real-time operating system and programming tools (some of this may be subcontracted or even come from open market) and finally creates the software for its system. Also, in software development the main vendor may use subcontracting. Only recently, these main vendors have started experimenting with open source software. The advantage, that has weighted a lot, is that if you design it yourself, you are the master of your own destiny: create a winning team and beat your competition.

From the network operators perspective this means that all new software to the main network elements for telephony (the same applies to transmission systems) has to come from the equipment manufacturer. The operator can not go to the open market and buy the pieces of software it desires. This has always been unpleasant to operators because for differentiating themselves on the market, they are very dependent on equipment manufacturers. Creating the software takes a lot of effort. Therefore, the equipment manufacturers are not interested and can not usually afford to tailor make software for a particular operator only. A particular operator may buy some differentiating piece of

©Raimo Kantola (2008)

software from a vendor but if it turns out to be important, one can be sure that every other operator will have it too very soon.

This tension between switching system vendors and operators has been the root for creating new value added system types and new telephony architectures. An example is the Intelligent Network Architecture. We will come back to this later.

Classification of signaling systems

Signaling in telephone networks can be classified to *subscriber signaling* and *trunk signaling*. The former takes place between a phone and a local exchange or between a PBX and the local exchange. The latter is used between two exchanges.

Based on the hardware generation and the type of signals, we talk about *analogue signaling systems* such as *analogue subscriber signaling* or early analogue trunk signaling systems. The opposite is digital or *message based signaling systems*. These can use either *binary encoding* of signals or data in messages or *textual encoding* of information in messages. Binary encoding is significantly more compact like we will later show. Its disadvantage is that binary messages are not easily human readable while textual signaling is easy to read and modify for the purpose of introducing new services.

Digital, message based signaling systems appeared when first computer controlled exchanges had been put into service in real networks. This meant that digital exchanges had to be able to process analogue signaling as well.

The principle in telephony networks for new systems has historically been: *any new system must be able to interwork with systems that are already in place*.

An example analogue trunk signaling system is R2. In a digital network, we talk about digital R2. The signals in abstract terms look like analogue signals in digital R2 but they have a straightforward mapping to bits.

Digital trunk signaling that is most common in modern telephone networks is called SS7 or CCS7, short for Common Channel Signaling System Number 7. It was specified by the predecessor of ITU-T namely CCITT.

Signaling may be *in-band*, meaning that the voice channel is used for both voice and signaling related to the particular call. Signaling may also take place outside the voice band. In such a case we talk about *out-of-band* signaling or common channel signaling. “Common channel” means that a common signaling channel is used for the purpose of carrying telephony signaling related to many calls. In SS7, signaling is out-of-band.

©Raimo Kantola (2008)

When SS7 was created, the grand idea of an Integrated Services Digital Network or ISDN was introduced. The idea was that all traffic in the ISDN network would be digital including the traffic on the subscriber line. Consequently, a digital message based signaling system was needed for subscriber access. This is the Digital Subscriber Signaling System number 1 or DSS1. It was specified by the CCITT in Q.931 (layer 2 in Q.920 and Q.921 and layer 3 in Q.930 and Q.931, supplementary services in Q.94x). DSS1 signaling is out-of-band or common channel signaling. On ISDN subscriber lines, there are separate time-slots or channels for signaling and user's data or voice.

One opposite to the principle of common channel signaling is the idea to have a dedicated signaling channel for each voice circuit. So, the signaling is out-of-band but it uses a special signaling channel that can be mapped to a particular voice channel. Such a system is called Channel Associated Signaling or CAS. We will come back to this when talking about the supervisory signaling for R2 which is a trunk signaling system.

Examples

Analogue subscriber signaling

In PSTN, subscribers are connected to the local exchange with a pair of copper wires. Power supply for an analogue phone is provided by the exchange on the wires and the same pair of wires, called the local loop, is used to carry signaling and voice. The power supply can be a constant direct current or a DC voltage. When the phone connects the wires together, a current can go through on the local loop, the current that returns to the exchange, can be detected and measured by the exchange. The exchange can change the polarity of the voltage applied to the local loop. The exchange can also send a special voltage or current to the phone making the bell on the phone ring.

Electromechanical dialing can be done for example by rotating a disk with numbers on the perimeter. Disk rotation breaks and reconnects the loop. As a result, in Finland digit 1 corresponds to a pulse of some 100ms in length on the local loop, digit 2 corresponds to two pulses, 9 to nine pulses and finally zero to 10 pulses. The transfer of those 10 pulses takes about 1 second.

Because dialing with a disk takes such a long time, phones with buttons that generate tones were invented. Two analogue frequencies are generated, when a button is pushed, see Figure 2.4.

©Raimo Kantola (2008)

	1209Hz	1336Hz	1447Hz	1633Hz
697Hz	1	2	3	A
770Hz	4	5	6	B
852Hz	7	8	9	C
941Hz	*	0	#	D

Figure 2.4: A Push button phone

The signaling method that emerges with push button phones is called Dual Tone Multi-Frequency dialing or DTMF dialing.

In response to user actions, the local exchange sends tones to the user on the voice band. These include the *dialing tone*, the *ringing tone*, the *busy tone* and for example the *queuing tone*. For example, the dialing tone is a continuous 425 Hz audio signal.

For the purpose of receiving DTMF signals, the local exchange needs a receiver that can be connected to the voice path coming from the dialing user. Because the cost of a local exchange is largely in the subscriber units (e.g. 70% of the LE cost may be in the line cards), DTMF receivers are not necessarily built in such a way that one is readily available for any user. Because dialing is a short period during a call and one subscriber may be engaged in a call on his or her local loop even less than 1/10 of the time, it may be better to place the DTMF receivers behind the switching fabric in the Local Exchange (LE). This way the dialing traffic from the users can be concentrated to the bank of receivers efficiently and the equipment can be kept busy a lot of the time. Obviously, the call control software needs to allocate a DTMF receiver for a user that has taken his or her phone off-hook and release it when the dialing is finished.

Nowadays, a device for DTMF reception is probably implemented using Digital Signal Processors. Early, digital exchanges during the 1980's used special purpose hardware for DTMF signal detection and identification.

One can use a bunch of copper wires to connect a PBX or a key system to a local exchange. Due to limitations of the analogue signaling (there are very few signals the exchange can send to the PBX), in such an arrangement,

©Raimo Kantola (2008)

usually Direct-Dialing-In (DDI) is not possible. This means that users of the PBX are not directly reachable from the public network. In such a case a switchboard operator is needed to manually switch the incoming calls to the PBX extensions.

R2 – an example analogue trunk signaling system

A large number of different analogue trunk signaling systems were developed by switching systems vendors. During the electromechanical era, the used hardware determined what kind of signals could be sent and received. For example in Finland as late as early 1980's, many switching system vendors were present on the market (the market was open to competition!) but the result was that we had more than 70 different analogue signaling system variants in the PSTN.

During this era signaling systems were really specified by the vendors and international standardization played a weak role. One of the first trunk signaling methods that became an international standard was R2. The method was widely used for example in Finland but on the other hand there are countries that never used it. Many similar signaling methods were used in other countries. For longest, R2 remained useful for the purpose of PABX signaling (although initially designed as a trunk signaling system.) With R2, the vendors could implement DDI for PBXs may be the first time.

Just to give some background on the design assumptions and motivations for early digital signaling systems, we will take a short look at R2.

Signaling in R2 is broken down into the *call establishment signaling or register signaling* (electromechanical switched used to have devices called registers to process signaling) and *supervisory or line signaling*. The signals in these two phases are physically different. For call establishment, multi-frequency signals are used (a different set of frequencies and a different logic than in DTMF, so do not mix R2 with DTMF) while for supervisory signals, digital R2 uses bits in time-slot 16 on the same 2Mbit/s PCM-line that is used to carry the voice for the call. Call establishment signaling in R2 ends when the call is in the *Ringing state*, i.e. when the terminating local loop has been located and the ringing current is applied to the called line. Supervisory signaling is responsible for managing the state of the voice circuits and releasing the call.

In R2, for each signal two frequencies out of six possible are used in both directions for trunk signaling. This gives 15 physically different signals. The physical signals, however, have two logical interpretations depending on the call state. A certain pair of frequencies in the forward direction for example can be an address signal and transfer a dialed digit. In the backward direction, for example, a certain pair of frequencies means: "send next digit".

In R2, each signal is acknowledged: the sender keeps sending until it detects the pair of frequencies that was expected as the response in the backward direction. When the receiver detects a new pair of frequencies, it starts to send an ack for the new signal. When the receiver detects that the sender is no longer sending, it also stops sending the ack. This principle is called *compelled signaling*.

In R2 sending one signal takes a few tens of milliseconds. Because the voice band on the voice circuit is used for signaling, after the call has been established this mode of signaling can not continue.

For line signaling purposes, timeslot 16 on a PCM line is sub-multiplexed between the 30 voice time-slots on the same PCM. A certain bit combination on the sub-multiplexed signaling channel is a telephony signal with a certain meaning. Which bit combinations are used for what purpose identifies a particular CAS system.

Limitations of analogue signaling systems

Signaling is the language that user devices and network nodes talk in order to implement services for end users. *Operators are always looking for ways to differentiate themselves on the market from other operators.*

This is natural, because the alternative business strategies according to Michael Porter include: differentiation, cost competition and niche strategies. Cost competition is almost the same as price competition – an impossible business strategy for most of the players in the market. This is because there can almost certainly be only a single leader in a market. Differentiation is much nicer because if the market can be segmented in a reasonable manner, there can be several winners with this strategy. Niche strategy on the other hand is usually only for small players.

In the light of operator differentiation, CAS and analogue signaling have serious limitations: because of the limited set of signals and slow speed of signaling only a limited set of subscriber services can be implemented using these signaling systems. Typically, in these systems once the call has been established, the set of possible signals is even further limited. This means that the capability to do much of anything with an established call except tear the call down is next to nothing (polarity reversal in analogue subscriber signaling was used to wake up the register in electromechanical exchanges in special cases).

Perhaps, the most significant, architectural limitation in CAS signaling is that signaling, although we have defined it to be the transfer of control information, is not possible without a voice circuit first being reserved. There are, however,

situations where reserving a voice circuit is inconvenient. Two examples of this are: (1) location updates for mobile users and (2) a subscriber service called “Call Back When Free” or “Call Completion to Busy Subscriber”. The latter is activated by the caller A when the callee B is busy. After the activation, A goes on-hook. The idea is that the callee B’s exchange will store this request and when B becomes free, it sends a signal to caller A’s exchange – hopefully without reserving a voice circuit for this purpose. Then caller A’s exchange can first alert A and without A having to redial, the exchange will try to establish the call again to B. The result is that the call will be established in the same direction as originally intended from A to B and consequently charged to A without any cost to B.

When one looks at the architecture of a signaling system, the first thing to understand is how signaling and voice circuit are tied together. The most flexible signaling systems treat voice circuits as resources and the identity of the FSM for signaling is completely independent of the identity of the possible voice circuit.

©Raimo Kantola (2008)