Operations and Maintenance 2

Robert Hanmer
Lucent Technologies
2000 Lucent Lane
Naperville, IL 60566-7033 USA
hanmer@lucent.com
+1 630 979 4786

Abstract

Large computer systems consist of many parts, only some of which contribute directly to the application for which the system was built. This collection of patterns describes several capabilities of a system that perform this background, supporting role. The roles of these patterns is to aid in application specific data, performance and health measurements, managing faults, and also managing the entire system complex remotely. The patterns here, as well as others previously workshopped at other conferences, combine to form a pattern language describing a small telecommunications switch. Previous patterns have discussed the call processing application and other portions of the system's supporting capabilities.

There is a partitioning of functionality within any complex computer system of two main parts: one that performs the application for which the system was built, and the other one that is not directly involved with that application but that support the application functions. The patterns here address some of the important things that are not directly involved with the application functions. These patterns are part of a larger collection of patterns that describe the architecture of a telephone switching system.

Call Processing [6] introduced the main application component of a telephone switching system, the HALF CALL. This is the pattern that handles the details of the telephone or data connection to exchange information between two parties to the call. To conserve resources are shared and *switched* in real-time to provide the needed connection. This switching function makes a logical electronic path in a circuit switching system, or directs a packet between specific ports.

Defining the application architecture is important to guarantee that the system will meet the needs of its users. Equally important is the design of the supporting infrastructure. In many systems while the application is the glamorous part of the system, the supporting functions actually make up a majority of the system. The patterns in [7] describe several functions that are necessary to switch telephone calls, but which are not actually involved in seeing that a particular telephone connection or call is made. The patterns documented here continue and expand these supporting functions.

The following figure sketches out the relationship between the patterns presented here and those found in Call Processing [6] and OAM-1 [7].

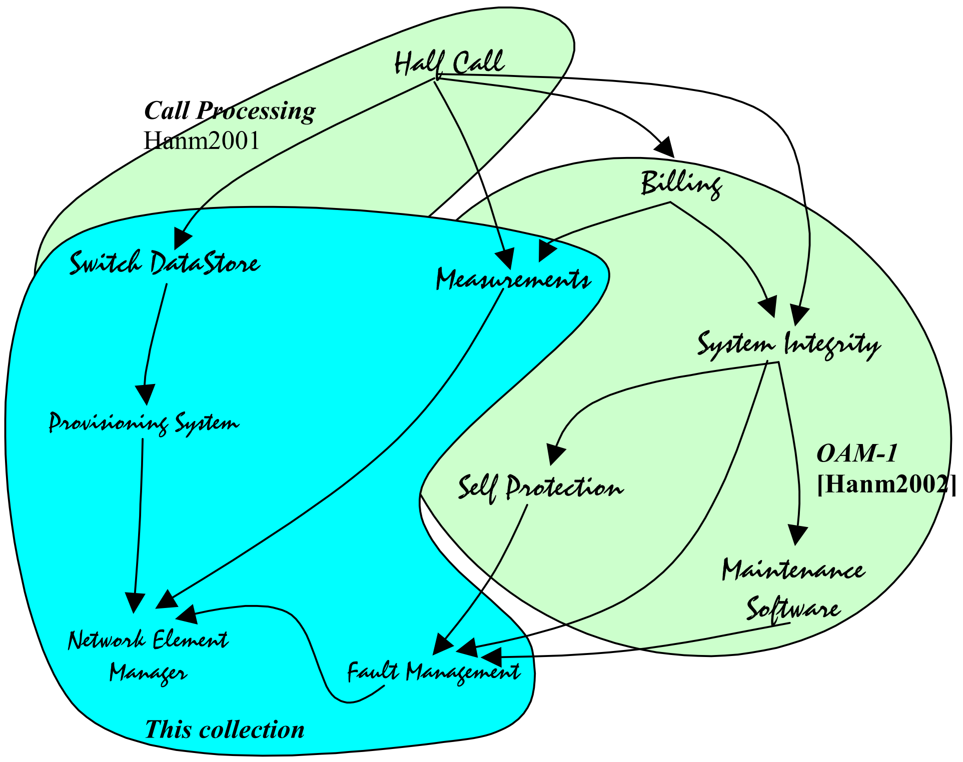


Figure 1

The patterns are presented in the Alexandrian form. The structure of each pattern is as follows: Each pattern begins with a numbered title. The number is used to reference to the other patterns in this collection. A photograph is then used to illustrate some essential aspect of the pattern. For example, the pattern MEASUREMENTS (3) shows a scientist taking a measurement at a field site. Following the photograph is a description of the context where the problem exists. Three diamonds follow this.

The next section of the pattern is the problem. It is printed in a bold font. A description of the problem, how it relates to the context and some possible solutions are discussed. The keyword *therefore* follows to introduce the solution section, which is also printed in a bold font and includes a sketch of the solution. Three diamonds provide a separation between the solution and the resulting context. The resulting context section introduces the situation after the solution has been applied. It frequently will point the reader to other patterns that can be used to resolve new forces that this solution has introduced.

1. Switch Data Store[1]

[2]

Some applications in telecommunications or data communications construct a permanent connection and mapping between endpoints. This is also true at lower levels, e.g. the physical layer, in the overall protocol hierarchies. If a call arrives on one endpoint, it is always routed to the same other endpoint as shown in figure 2.

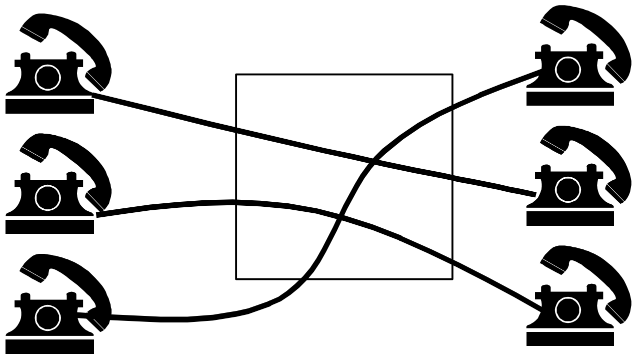


Figure 2

As networks grow and become more pervasive, direct connections between endpoints are no longer flexible enough. They also require dedicated resources because the connections are usually idle for a large percentage of the time. This also requires that a customer have multiple devices – one for each connection that they have. Figure 3 shows direct connections between different pairs of customer endpoints.

[1] An example implementation of this pattern is discussed briefly in [4].

[2] The photos that begin each of the patterns are provide a real-life example of an aspect of the pattern. The photos were selected from the USA National Archives and Records Administration, www.nara.gov

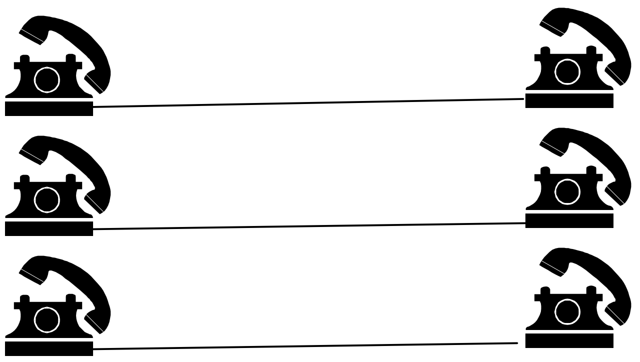


Figure 3

It is more efficient to reconfigure the connection for each call. Then when a call arrives on one endpoint, the destination endpoint it is routed to may vary from call to call. Variable connections, see figure 4, require some information about the connection is required. The endpoints might have different capabilities. Data about these differences can also be saved to prevent mismatching capabilities. This data describes the endpoints and also contains information about the types of calls that can be handled.

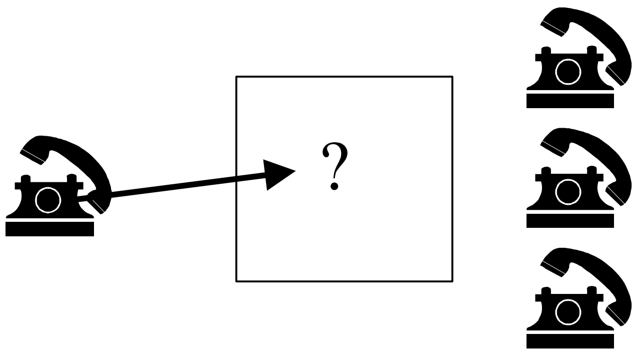


Figure 4

HALF CALL [6] discusses how we can arrange the call processing components that will participate in a connection between endpoints within our system. It describes the statics of the system's components, but it doesn't discuss the dynamics of an actual connection request. System architecture must address both the static and dynamic behaviours and attributes.

❖ ❖ ❖

How can the system dynamically handle connections that aren't predefined?

Telephone systems once solved this problem with an intelligent agent -- a human operator. Calls would arrive from the originating party and ask the human operator to speak with a specific party. The operator would either know from prior experience or look up which connection to make to connect the parties.

This prior knowledge might be in the form of a lookup such as *to connect Frank to Jane, connect ports 3 and 14 together*, as shown in Table 1. As the number of possible connections grew, the amount of knowledge that is required grows.

Frank	3
Joe	17
Jane	14
Billy	10

Table 1

Eventually the amount of information becomes more than the operator can easily handle. A better solution is needed.

An important thing to note is that the queries to the system are very consistent. The same information is sought each time and the request is almost of the same form.

How do I connect calling party a to called party b?

The calling party might not even be described by name; the name is not important. And sometimes it can be distracting. Frank may be calling Jane, but he might be using Ralph's telephone. What is important is the port through which the calling party accesses the system as shown in figure 5.

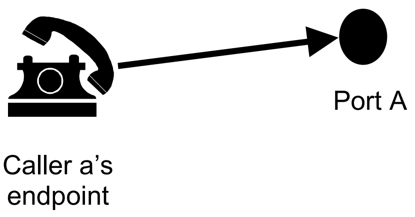


Figure 5

This changes the query to:

How do I connect the calling party on port A to called party b?

The modern telephone network identifies particular parties with a number. Usually a telephone number, such as 123 4567, or an IP address 123.12.34.56. The actual party at the end of the number may or may not be the desired party B, but the number points to a place that B has been and is expected to answer regularly.

So the query is now:

How do I connect the calling party on port A to some called party on port B?

A small database can store the data needed for such a regular query. Originally human operators performed these lookups. Early automatic systems implemented the translation and switching in hardware. If a call desired destination **b** it was switched to port **B** all in hardware. Eventually software replaced this hardware function.

The database must support the simple queries like those just described, and also be able to support a number of administrative actions.

Sometimes customers receive new, additional telephone numbers or network addresses. For example, customer **b** obtains an additional number, and is now on both ports **B1** and **B**? Some way of updating the database is required.

Transactions that are required include:

- the initial population of number to port mappings into the switch database (customer c is assigned to port C),
- changing entries (customer b is moved to port D), and

- deleting an entry altogether (customer d leaves the system).

Few other types of transactions will be required. The data needs are quite simple.

Since this database will be interrogated on every connection within the system it must return its answer quickly.

If the database is unavailable due to a failure, the system cannot connect calls, so the database system must have few faults.

A general-purpose database could be used to store and retrieve the information needed. However most database systems require more overhead than can be afforded, both in terms of retrieval time and memory overhead. A better solution is to use a small custom configured database system that is tailored to the situation. Therefore,

Install a small, custom data storage system that will be able to quickly and reliably decide how to connect two parties to the call. Figure 6 shows the connection of a database containing customer data with one of the Half Call entities associated with the call. Table 2 outlines the responsibilities of the customer data database.

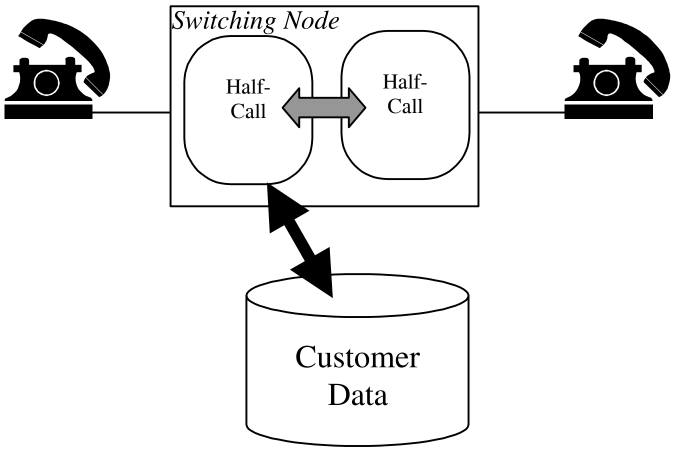


Figure 6

Switch Data Store

Responsibilities:
- Reply to queries about port/customer location
- Update port/customer location mapping
- Delete port/customer location mapping
- Add new port/customer location mapping

Table 2

❖ ❖ ❖

In many systems, such as at the lower level protocol routers and switches this database might be implemented in hardware. And some protocols might include all the addressing information within the contents of the message, eliminating the need for a custom database.

The database can get much more complicated as additional features such as address translations are required. An example of this is when a telephone call with one of the toll-free area codes such as 800, 877, or 888 is made. In these cases the systems processing the call request must translate the number that was entered (888 123 4567) into the number of a real telephone. [10] This complication evolved into telephone systems. Initially toll-free service in the USA was provided to serve the needs of traveling salesmen to call the home office.

In all probability the system's database will not be populated through entirely manual actions, nor will it remain constant. The types of transactions are simple, yet the system can benefit through having a PROVISIONING SYSTEM (2) that will administer the changes.

2. Provisioning System

There is a lot of data in a telephone switch that must be maintained. This is the data that the SWITCH DATA STORE (1) contains. Getting it into the system is to "provision" it.

❖ ❖ ❖

Somehow we need to get the data into the SWITCH DATA STORE **(1).**

We could type it in each time we need it. Or each time we turn the system on. Since the system tries to run non-stop (SYSTEM INTEGRITY [7]), we wouldn't even need to do this very often. But we might get something wrong. We might forget something or get something wrong (e.g. a typographical error).

The switch is designed to process telephone calls, not to interface to a provisioning person. Its primary purpose is to switch telephone calls, not to look to its provisioning staff to be putting in data updates.

Keyboards or really any human controlled interface to the system is slow. With 1000 bytes of data to be entered over a 9600 bps serial link (Does this make sense today? What's the speed of the keyboard connection of a PC?) This takes { 1000 bytes / (8 bits/byte) * 1 sec/9600 bits} seconds to get the data into the system. There are higher speed links nowadays, such as high speed Ethernet. High-speed interfaces allow the data to be transferred much more quickly, which minimizes out of service time.

Humans can't communicate with the system at these high speeds but another computer can.

The data in the system might not be in a human readable form. So someone must translate it, or the system must translate it. This translation takes time from the primary purpose of the system.

Several different switches might need the same data. If a common source were to provide the data to each of them then there is less chance that the data will differ erroneously.

Therefore,

Build a computer system that will interface with the switch to put in the data it needs in its Switch Data Store. This system will have a model of the Switch Data Store

and will interface with the switching node to access and update the node's database, as shown in figure 7.

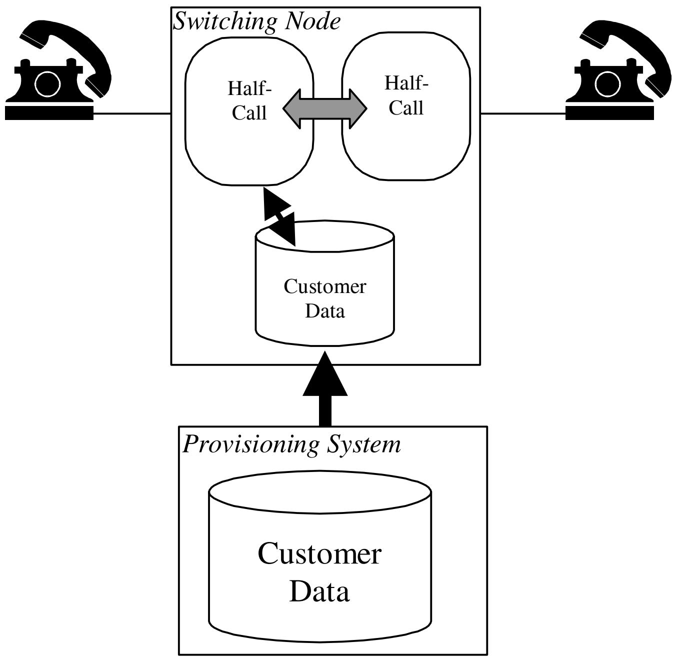


Figure 7

❖ ❖ ❖

Initial and ongoing provisioning will be done much faster than if the only interfaces on the system are the human interfaces or a serial link.

Interactions with this other system must be restricted somehow (PROTECT THYSELF [7]) to prevent it from using too much of the switch's resources. Remember, that Provisioning is necessary for the switch's primary function, but the act of provisioning is not the switch's primary function.

3. Measurements[3]

… The telecom system consists of many parts, some hardware and some software. You hope to make money by using this system to serve people who will pay you for services. You might need to add additional capabilities or additional systems to support additional customers in the future. In order to supply the appropriate amount of service to a region — not too much, wasting your capital, nor too little, and missing revenue opportunities — you need information about the activities. If a particular geographic region supported by a switching system grows very rapidly the system might not be able to handle the load. By knowing how much the system is being used, your engineering staff can make good engineering decisions about deployment of new capacity, or moving capacity that currently exists. You can't afford to guess at how much the system's components are being used.

❖ ❖ ❖

How do you know how much your system is being used?

The system is engaged in many activities, some of which are not externally visible. Each of these activities can report its usage in whatever manner their developers decide. Having many different mechanisms for reporting usage can easily lead to chaos, and chaos makes it harder to keep the system operational.

Some of the data that you are interested in to support and administer the system consists of raw counts from the hardware or software subsystems. Some of the data needs to be aggregated or somehow processed to be useful.

BILLING [7] collects key information from the HALF CALLS [6] to be able to charge customers for their usage. This information is extremely important, but does not paint a complete picture of the system.

The system generates much information that can help the network engineers put the appropriate amount of switching capacity in an area. The people associated with the engineering functions like to have real data, rather than just working from their hunches. The

[3] An example implementation of this pattern is discussed in [3].

information will also have internal uses, such as SYSTEM INTEGRITY [7] who can use it to assess system state.

Collecting data about the system's usage and activities is not the primary application of the switching system. The amount of time that is required to do this should be minimized.

Therefore,

Create a subsystem that will keep track of counts and measurements from the parts of the system and provide a framework to produce meaningful reports with meaningful data, refer to figure 8. The measurement subsystem will be most useful if it creates information reports for the maintenance staff at regular intervals.

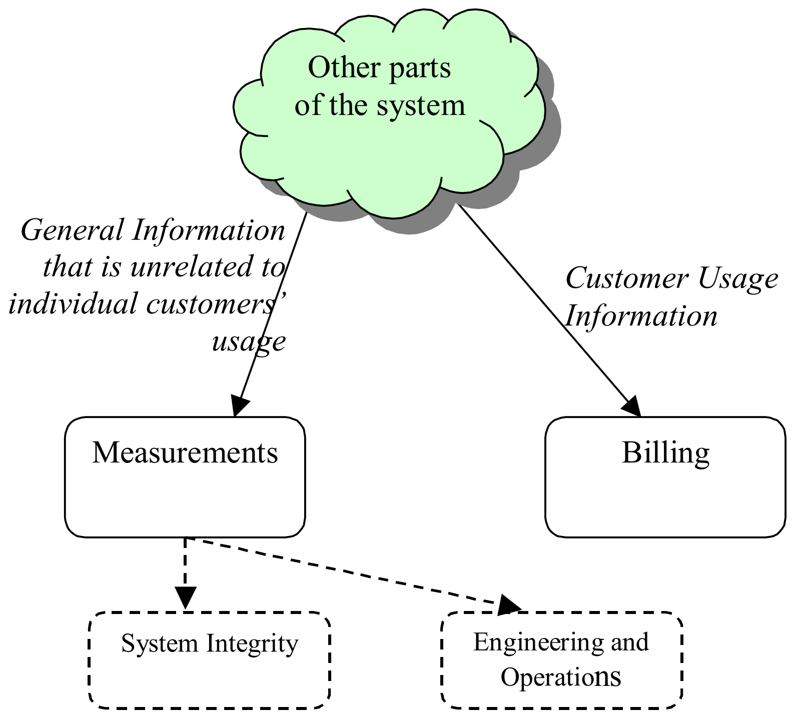

Other parts of the system
General Information that is unrelated to individual customers' usage
Customer Usage Information
Measurements
Billing
System Integrity
Engineering and Operations

Figure 8

❖ ❖ ❖

MEASUREMENTS PROVIDE a way for SYSTEM INTEGRITY [7] to check the system activity levels.

Not all of the people charged with system maintenance are, or will be, local. Some way of getting the data "upstream" is required. The NETWORK ELEMENT MANAGER (4) subsystem supports this.

MEASUREMENTS will provide a way to cross check and validate the results of the BILLING [7] system.

TELECOM DATA HANDLING [2] provides additional details useful in building a MEASUREMENTS system.

4. Network Element Manager

SWITCH DATA STORE (1) benefits from PROVISIONING SYSTEM (2). Making sure that the switch data store is working at peek efficiency is something that can be monitored by MEASUREMENTS (3). MEASUREMENTS need to be sent somewhere to be looked at and maintained otherwise their generation was a waste of time.

❖ ❖ ❖

The administrative and operational interfaces of the switch should go somewhere other than /dev/null, the "bit bucket".

If we don't need these subsystems that generate information that will help manage the system effectively then we shouldn't build them. But as their descriptions suggest there are reasons to collect measurements. To make the best decisions about engineering of the network it's elements (the switching systems) should be studied for a period of time.

Once collected by the system they should go somewhere.

We can print them out…but that isn't always useful. A network of several switches will all be collecting measurements. Individual paper measurements from several different switching systems would be difficult to analyze. We could send them to another computer system that will store them and allow later retrieval.

If another computer is involved in collecting the data then why not have it do some other functions, such as providing a convenient interface for a human manager to monitor the switch. And why not adapt it to support several switches at the same time?

Our switch is like an embedded system that has certain interfaces that the primary application sees the telephoning public and others that the primary users don't. For example, the primary throttle interface from a driver to a car's primary computer as well as the interface to the mechanic's diagnostic machine.

Therefore,

Build a computer system, external to the switch that can be used to oversee and administer the operations of the switch as shown in figure 9. This system should support several "target" switches at a time through collecting MEASUREMENT **and provisioning**

related data from these switches. Switch control functions, such as those helpful to provide human oversight of SYSTEM INTEGRITY **[7] should be included in this system.**

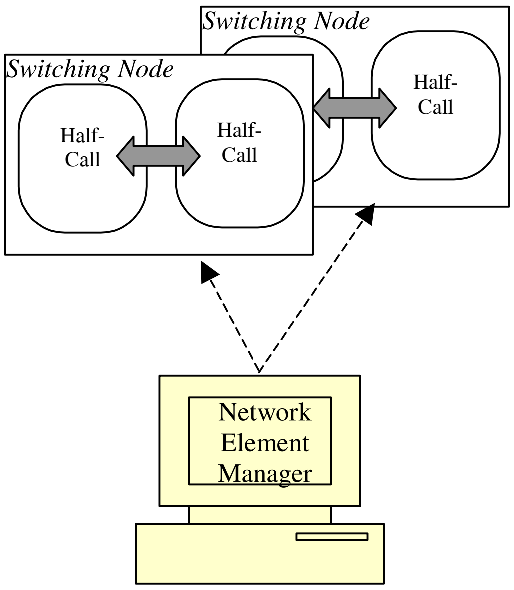


Figure 9

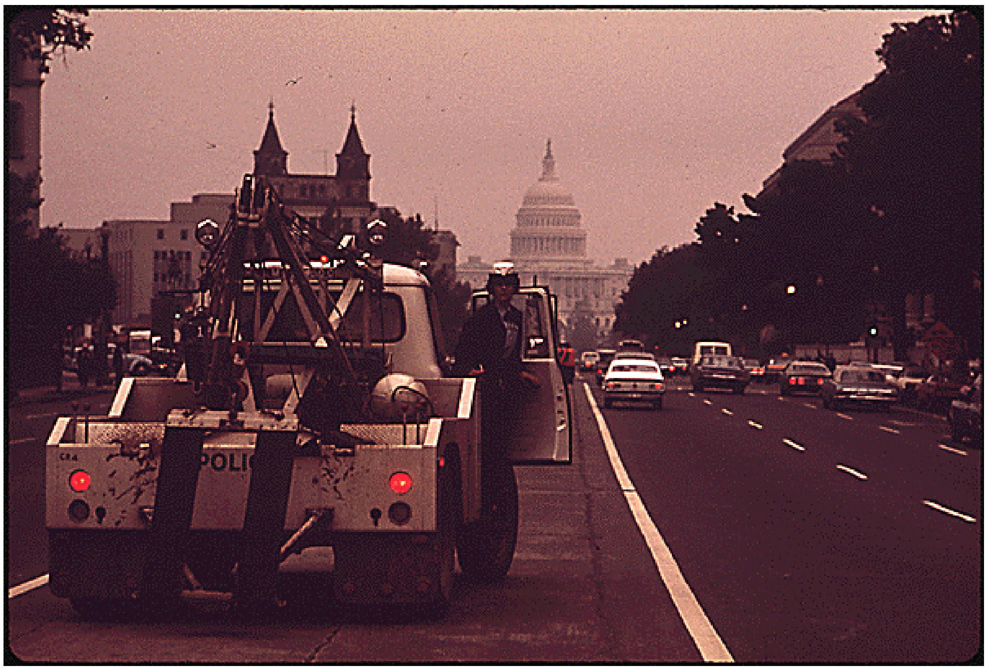
❖ ❖ ❖

Unlike the switching system network elements being monitored and controlled, the primary function of the NETWORK ELEMENT MANAGER will be to monitor the other systems and to interface with human network managers and engineers. Remember that the primary function of the switching system is to process calls or packets, not to deal with MEASUREMENTS. The designers must manage this difference in concerns or else inappropriate responsibilities will be created.

With a NETWORK ELEMENT MANAGER our system can effectively be monitored and administered by humans. The Measurements that it produces can be analyzed over a period of time and in conjunction with its network peers.

Deciding the capabilities of the system in terms of how many switches it should support should not be made in haste. Monitoring too many or via too much measurement data can make this system unusable.

5. Fault Management[4]



SYSTEM INTEGRITY [7] is responsible for monitoring system health and invoking corrective action. When an error is detected action must be taken so that the effects don't ripple throughout the system. Once isolated they should be remedied, by a software or human induced correction being introduced.

During corrective actions the system might generate other errors. Focus and attention on the overall situation

❖ ❖ ❖

How can SYSTEM INTEGRITY'S focus on monitoring and controlling be preserved when it might run into errors during its corrective action?

The first thing that must be done to isolate a fault is to determine what is faulty. In order to do this the system must be able to "look inside" of the other parts of the system to look for discrepancies. This might mean that a component just drop its barriers of encapsulation to allow examination.

This requires clearly defined interfaces between the potential targets of correction and the entity that is trying to locate and correct the fault. There is the potential for very many interfaces.

Another way of handling this problem is for SYSTEM INTEGRITY to ask each component to resolve issues on its own. The problems with this are that if a component is suspected of causing an error then it can't be trusted to take care of itself, or to keep the interests of the system forefront in its operation. Another difficulty is that this results in very much duplication of code – each object (or family of objects) must have its own fault handling capabilities. In a large system where many developers are writing the code, probably on a functional basis, this will mean many implementations of the same things, and thus potentially very many latent faults.

The part of the system that looks for faults to correct must be pretty fault-free itself. Otherwise a fault in it can spread quickly.

[4] An example implementation of this pattern can be found in [9].

Sometimes the system is configurable with a different set of hardware or software components. The ability to add and subtract specialized fault handlers will make the system more reliable, as only the necessary ones need to be loaded. This in turn becomes a system software maintenance problem as the correct software modules must be loaded and available for whatever configuration the system currently has.

There are many techniques to determine if something is faulty. One simple one is through a LEAKY BUCKET COUNTER [1]. This is mentioned to point out the need for data that describes the system's fault history. The MEASUREMENTS () data will be helpful, but it might not really capture the data that will be most interesting for looking at historical faults. MEASUREMENTS data is probably too high level, or too system specific; fault handling data will need to be lower level, more related to individual components.

Therefore,

Create a FAULT MANAGEMENT **subsystem. This subsystem collects the data it needs by interrogating other parts of the system or through its own historical data. It then isolates and perform corrective repairs to the system as needed. Give the** FAULT MANAGEMENT **subsystem the power it needs to take corrective actions.** SYSTEM INTEGRITY **should be small to monitor and invoke the** FAULT MANAGEMENT **subsystem, refer to figure 10, with little risk to the** SYSTEM INTEGRITY **system.**

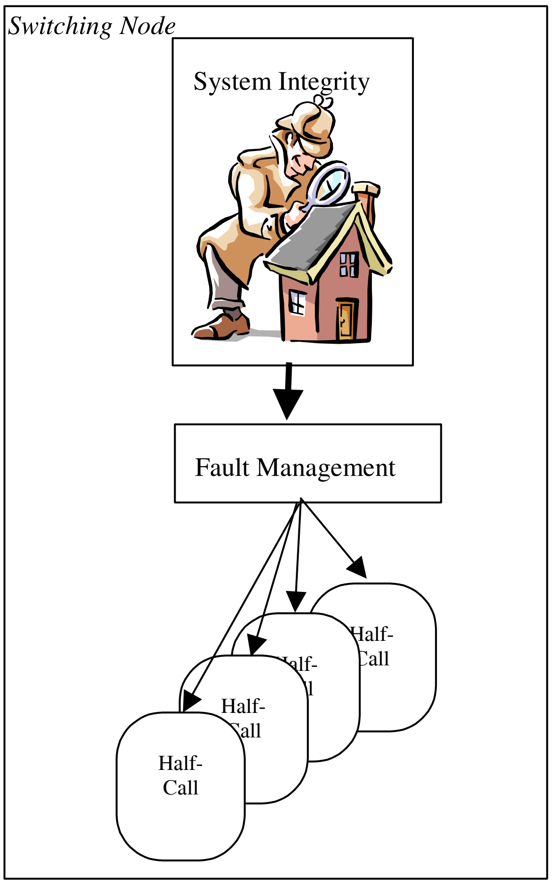


Figure 10

❖ ❖ ❖

The FAULT MANAGEMENT system needs to be simple enough to work even if parts of the system are incapable of working. It must also be able to examine and repair itself.

6. Acknowledgments

All photographs used courtesy of the National Archives and Records Administration of the USA

Thanks to John Letourneau who served as the SugarLoaf PLoP shepherd. Thanks go to my SugarLoaf PLoP Writers' Workshop Group.

7. Pattern Thumbnails

PATTERN	*Reference (Single digits are in this work)*	*Pattern Intent*
BILLING	[7]	Keep records of usage in a centralized object.
FAULT MANAGEMENT	5	A specialized subsystem should quickly handle errors and failures by isolating and treating faults.
HALF CALL	[6]	Use a 2 part (half call) model for call processing.
MAINTENANCE SOFTWARE	[7]	Identify and isolate faults before they are encountered in an operational system.
MEASUREMENTS	3	Provide a single object to collect counts and measurements and then distribute them to concerned parties.
MINIMIZE HUMAN INTERVENTION	[1]	Design the system so that human intervention is not required.
NETWORK ELEMENT MANAGER	4	Provide a single point to consolidate human interaction with switches for both integrity and measurement purposes.
PEOPLE KNOW BEST	[1]	Provide a way for an expert human to override the system's automatic responses.
REASSESS OVERLOAD DECISION	[5]	Conditions change and the system should periodically reassess its decisions to see if they are still valid.
SELF PROTECTION	[7]	In the face of too much incoming traffic, try to push traffic back to neighbors and thus keep your own sanity.
SHED WORK AT THE PERIPHERY	[6],[8]	Try to prevent work from reaching the core of the system; stop it close to the periphery, where fewer system resources will have been expended on it.
SICO FIRST AND ALWAYS	[1]	Give SYSTEM INTEGRITY the power and ability to handle the situations that might arise.
SYSTEM INTEGRITY	[7]	Provide an object that will watch for abnormal system activity and will initiate necessary reactions.
TELECOM DATA HANDLING	[2]	Telecom systems have some unique attributes for their measurements.

8. References

[1] Adams, M. J. Coplien, R. Gamoke, R. Hanmer, F. Keeve and K. Nicodemus. 1996. "A Pattern Language for Improving the Capacity of Reactive Systems" in **Pattern Languages of Program Design — 2**, edited by J. Vlissides, J. Coplien and N. Kerth. Reading, MA: Addison-Wesley Publishing Co.

[2] DeLano, D. 1998. "Telephony Data Handling" in Proceedings of PLoP 1998 Conference.

[3] Greene, T., D. Haenschke, B. Hornbach and C. Johnson, 1977. "No 4 ESS: Network Management and Traffic Administration." **Bell System Technical Journal**, vol. 56, no. 7, Sept., 1977: 1169-1201.

[4] Giunta, J. A., S. F. Heath III, J. T. Raleigh, M. T. Smith, Jr., 1977, "No. 4 ESS: Data/Trunk Administration and Maintenance." **Bell System Technical Journal**, vol. 56, no. 7, Sept., 1977: 1203-1237.

[5] Hanmer, R. 2000. "Real Time and Resource Overload in Proceedings of PLoP 2000 Conference.

[6] Hanmer, R. 2001. "Call Processing" in Proceedings of PLoP 2001 Conference.

[7] Hanmer, R. 2002. "Operations, Administration and Maintenance-1" in Proceedings of PLoP 2002 Conference.

[8] Meszaros, G. 1996 "A Pattern Language for Improving the Capacity of Reactive Systems" in **Pattern Languages of Program Design — 2**, edited by J. Vlissides, J. Coplien and N. Kerth. Reading, MA: Addison-Wesley Publishing Co.

[9] Meyers, M., W. Routt and K. Yoder, 1977. "No 4ESS: Maintenance Software." **Bell System Technical Journal**, vol. 56, no. 7, Sept., 1977: 1139-1167.

[10] Sheinbein, D., and R. P. Weber, 1982. "800 Service Using SPC Network Capability." **Bell System Technical Journal,** vol. 61, No. 7, Part 3, Sept. 1982: 1737-1757.