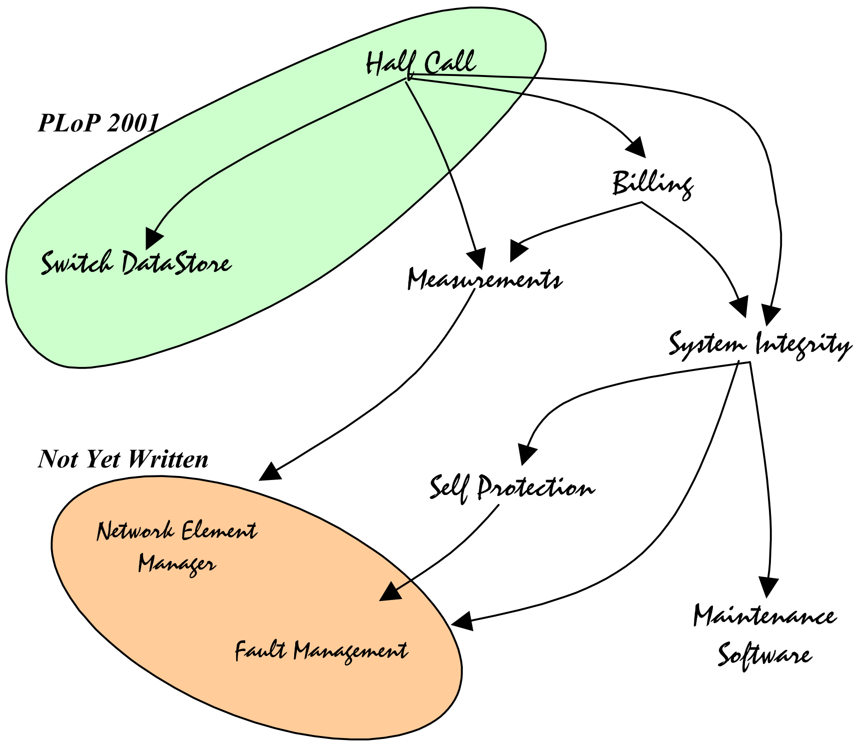
Operations and Maintenance 1

Robert Hanmer
Lucent Technologies
2701 Lucent Lane
Lisle, IL 60532
hanmer@lucent.com
+1 630 979 4786

There is a dichotomy within any complex computer system of two main parts: those that perform the application for which the system was built, and those that are not directly involved with that application but that support the application functions. Call Processing [Hanm2001] introduced the main application component of a telephone switching system, the HALF CALL. The patterns presented here describe a number of functions that are necessary to switch telephone calls profitably, but that are not actually involved in seeing that a particular telephone connection or call is made.

The following figure sketches out the relationship between the patterns presented here and those found in Call Processing [Hanm2001]. Those from Call Processing are shown with a colored background. Several yet-to-be-written patterns are also shown since they are mentioned in several of these patterns.



1. BILLING[1]

… You have great technology, but technology alone can't pay the bills. You want to make a profit. The way you can do that is to charge for usage and services.

By handling many active calls at any time, you can maximize your equipment usage. The call entities are HALF CALLS [Hanm2001]. This means that there are two objects that are responsible for the call creation and termination. Frequently there are governmental regulations (i.e. laws) that impose restrictions on what you can charge and how you can do it.

❖ ❖ ❖

How can we remember the information about calls so that we can later charge the user for their usage?

Software is in control of the call. The HALF CALL() pattern is used to allocate the work between two different call entities on the call. Splitting the call between the two half objects results in a distributed view of the call, and leaves open the question of which one should be in charge of recording this information.

The telecommunications marketplace has evolved to the point where different rates apply in many different situations. For example, you might charge less for usage in the middle of the night, or on weekends; you might have a special arrangement with your largest customers; etc. If that is not enough variability, you must also consider how you are going to charge for the usage: on a per call basis, on a time-used basis, a flat rate for a time period, etc.

Having to keep track of all the details can be so tedious that you might even consider giving up and providing service for free. But if you do, how will you generate the revenue necessary to operate the service, or even to pay for the equipment?

[1] An example implementation of this pattern is discussed briefly in [CCRSS]. **[This is primarily a local switch function, so check 5E BSTJ articles.]**

Ultimately you need to report to the customer what they owe you. Who should be responsible for getting you this information? Which of the HALF CALLS ()? Certain features such as call redirection or call transfer can be applied to the call causing you to lose the half call entity that is supplying the usage information. Many aspects of recording usage occur autonomously with respect to the HALF CALLS (). The information needs to get sent to another system for review, and backups of the data should be made.

Therefore,

Create a billing subsystem that will receive data from the half call objects. This subsystem relieves the active call handling objects from the many decisions about how and when to charge the customer. All callse share the billing subsystem which handles the background tasks required to actually charge for usage.

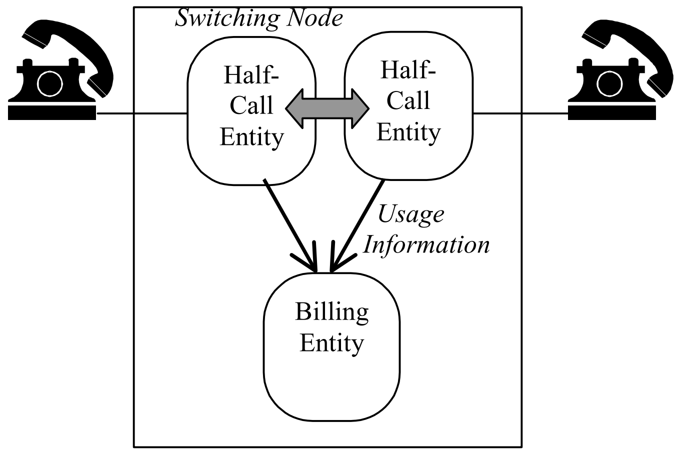


Figure 1 A Billing Entity is added

❖ ❖ ❖

Generally you want to maximize revenue from the switching system. To ensure this, you should have a SYSTEM INTEGRITY (3) system that will monitor the situation and trigger some maintenance actions if they are necessary. The MEASUREMENT (2) system supplies data that can cross-check the statistics that the BILLING system produces.

After implementing this pattern, the system will be able to keep track of its use so that users can be assessed for the resources that they use.

2. MEASUREMENTS[2]

… 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, missing revenue opportunities — you need information about the activities. 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 can 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.

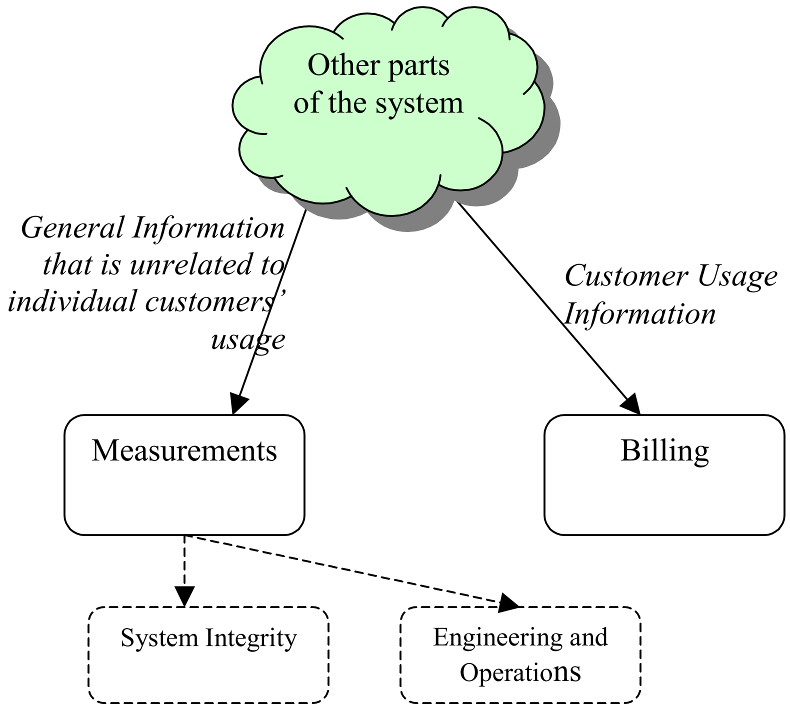
BILLING (1) collects key information from the HALF CALLS () to be able to charge customers for their usage. This information is extremely important, but does not paint a complete picture of the system.

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,

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

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. 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

❖ ❖ ❖

MEASUREMENTS provides a way for SYSTEM INTEGRITY (3) 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 (not yet written) subsystem supports this.

MEASUREMENTS will provide a way to cross check and validate the results of the BILLING (1) system.

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

3. System Integrity[3]

… You want to maximize the revenue that the system generates. The system consists of many hardware and software components. They will not all be fault-free.

There has to be a way to minimize the time that the system cannot do the routine application work that it was designed for. Things that get in the way include fault handling, long duration routine tasks and tasks that are outside the normal scope of activity.

The purchaser of your telecom system wants to have a reliable network with which to offer services to their customers. A more reliable network will increase system availablility, thereby allowing the purchaser to handle more of their customer's requests and increase their revenue. A more reliablie network will also have lower maintenance costs.

Telecommunications systems generally do not endanger human safety if they are not working correctly. However, certain applications such as Enhanced-911[4] do have bearing on safety, which results in stringent external requirements being placed on availability.

❖ ❖ ❖

How can we ensure that the system is performing in a way that will maximize availability?

You could throw the system away when it breaks (has a failure), but you have a large investment in it.

The system needs to monitor itself to achieve results but that takes time and effort.

Many symptoms look like other symptoms — so some part of the system or an operator is required to discern between them.

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

[4] Todo: Add a citation for E911 service.

MEASUREMENTS (2) and BILLING (1) subsystems collect data, but don't analyze the data. MEASUREMENTS(2) displays the information in the form of reports, and BILLING(1) collects the information required to charge for usage. The system-wide counts and measurements collected and processed by these two subsystems give them access to many system health indicators. But they aren't charged with responsibility for analyzing the data and taking corrective action. No part of the system other than MEASUREMENTS has the complete view that it does.

There are problems using MEASUREMENTS (2) to provide the global error detection because it is computing data at regular intervals, which might mean that the system has to wait until the next interval to detect an error. Delays in reaching these decisions can result in lost revenue.

Some faults and potential errors can be detected and stimulated by creating stimuli within the system. Someone needs to be in charge for every maintenance and fault recover action.

Therefore,

Empower part of the system to monitor system health and invoke appropriate corrective action. This subsystem doesn't need to and probably should not actually take the action because this will distract it from its primary functions of monitoring and decision making. The SYSTEM INTEGRITY subsystem should invoke the appropriate FAULT MANAGEMENT subsystem when errors are detected.

❖ ❖ ❖

The system that results is one that can watch over itself and discern between a myriad of possible impediments to system availability. It can also direct parts of the system (with the help of MAINTENANCE SOFTWARE (4)) to take corrective action. It invokes FAULT MANAGEMENT (not yet written) when appropriate. If the problems are determined to be from too much traffic then it will invoke SELF PROTECTION (5).

Patterns such as those found in Fault Tolerant Telecommunications Patterns [ACGHKN], especially SICO FIRST AND ALWAYS [ACGHKN] provide additional details useful to implement SYSTEM INTEGRITY.

4. MAINTENANCE SOFTWARE[5]

… SYSTEM INTEGRITY (3) helps the system to take care of itself by reporting barriers to system availability. The goal is to maximize productive (and revenue-generating) availability. The system is made up of a large variety of hardware and software components.

The system tries to take care of itself with mechanisms such as MINIMIZE HUMAN INTERVENTION [ACGHKN]. When faults are identified earlier, the errors and failures that it induces are less severe. Faults in software remain dormant or undetected until the faulty piece of software is executed. Such faults are called *latent faults* because they can remain hidden for a time. When the faulty software is executed, the fault will be discovered. This can happen during either normal or maintenance activity. It is less desirable for it to happen during normal activity.

❖ ❖ ❖

How can we keep latent faults from becoming errors?

Hardware and software components when left to themselves will degrade. Problems can also creep in when many software updates are applied incrementally. If a component (hardware or software) is not exercised occasionally, how can we be sure that it will operate correctly when it is next requested?

The platform that our system is built upon probably has a variety of tasks that perform some maintenance activities on the system. Generic maintenance tasks might not execute through the code with the fault however. For example, if the fault is in a device driver it is unlikely that it will be encountered until the system actually has a need to execute the driver in normal operations. Further, few platforms will have built-in support for telecommunications-specific peripherals.

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

Therefore,

Install a framework of components that can test and diagnose the system to locate and identify faults before they cause errors. By locating faults before they cause errors, the system can isolate them and prevent them from actually causing an error that might result in a customer-impacting failure [LA].

❖ ❖ ❖

The maintenance subsystem should perform Routine Exercises (REX)[6] to continually try components in real service. Because PEOPLE KNOW BEST [ACGHKN] the MAINTENANCE SOFTWARE should allow humans to demand that certain functions occur on different schedules.

In addition to routinely exercising software via routine exercises, software diagnostics are useful. Hardware components are frequently supplied with diagnostic programs that will test that subcomponents and circuits are functioning properly. In much the same way, diagnostic test programs can invoke parts of the operational software to detect if it no longer produces correct results [Hanm2000a].

[6] [MRY], page 1163.

5. SELF PROTECTION[7]

… The obvious way to handle too much traffic is to ask your peers to stop sending so much, that is, move *your* problem to your neighbors. If the peers are willing and cooperative this solution can work quite well. Individual control bits can be included in ordinary messages between systems to let each other know if the workload is too high.

❖ ❖ ❖

What happens when neighbors don't (or can't) restrict the amount of traffic that they are sending to you? What about when the peers are themselves receiving too much traffic? In this situation they have every right to expect that their neighboring systems, including you, will restrict the amount of traffic.

Eventually the problem becomes so great that the real sources of traffic, the telephoning public, must be involved to restrict the traffic on the network. There are generally not mechanisms to support this however.

A few things help, such as delaying dial tone. When a finite number of sources of the dial tone sound exist, a customer might have to wait to be assigned the dial tone. This will slow down the entries of traffic to the system. However advances have moved the dial tone sources from being separate pieces of hardware to data files that many more customers can hear simultaneously.

Another technique is to give the customer audible indications such as the "fast busy" tone or an "all circuits are busy" message. Both of these require system resources to alert the customer. So these mechanisms are not free in terms of reducing the load on an overloaded system.

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

A problem of a different sort occurs across the boundary between different carriers. They might not be interested in reducing the amount of traffic that crosses the corporate boundary. For example, there might be a payment expected for every call delivered to a different company, even if the recipient can't handle the traffic because it is overloaded. In this example, there is incentive to advance the call, even if the recipient system is requesting relief from its neighbors.

In some circumstances, the excess of messages is isolated to a single system. Perhaps it is a local disaster that is not widely known, or is of major importance only locally (such as sporting event tickets). Obviously the neighboring systems cannot aid in relieving this congestion.

Yet another problem that might be encountered is when the system believes that it is receiving too much traffic and is thus overloaded, however the apparent surge in traffic is actually due to a hardware or software fault in the local system. For example, a signaling node erroneously inserts multiple legitimate call messages into the processing stream, possibly due to a counter overflow or other simple programming mistake.

As these scenarios point out, a system cannot rely upon its neighbors to relieve its own congestion because they might not be able to, they might not be willing to, the congestion might be local or it might even be a fault. If the system can't look outside for assistance, it must turn its focus internally.

Therefore,

Protect thyself. Incorporate features within a system so that it can handle an excess of traffic on its own. Work in conjunction with SYSTEM INTEGRITY (3) to choose whether corrective action is required (FAULT MANAGEMENT (not yet written)) or possibly some preventive maintenance (MAINTENANCE SOFTWARE(4)). You will want to periodically REASSESS OVERLOAD DECISION [Hanm2000].

❖ ❖ ❖

SELF PROTECTION can implement many of the techniques from Real Time and Resource Overload [Hanm2000]. There are a number of techniques that a system can use to reduce its workload. It can put up barriers to accepting more work. Messages or requests for service can be ignored and discarded rather than being queued. It is best to SHED WORK AT THE PERIPHERY [Hanmer2000, Mesz] and discard work as close to the edges of the system as possible. Another technique is to restrict the processing that is done on each request. In periods of congestion, don't process all of the desired extra features that a caller might expect.

If SELF PROTECTION is invoked too frequently your customers will choose a different carrier. Having a MEASUREMENTS (2) system in place will alert you to this possibility. If the system is continually having too much traffic you should also consider whether your SYSTEM INTEGRITY (3) system is failing to identify errors properly and these undetected errors are causing the appearance of overloads.

Acknowledgments

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

Nick Landsberg provided the inspiration for SELF PROTECTION. Thanks to Berna Massingill for shepherding this paper for PLoP 2002.

Pattern Thumbnails

PATTERN	*Reference (Single digits are in this work)*	*Pattern Intent*
BILLING	1	Keep records of usage in a centralized object.
FAULT MANAGEMENT	Not yet written	A specialized subsystem should quickly handle errors and failures by isolating and treating faults.
HALF CALL	Hanm2001	Use a 2 part (half call) model for call processing.
MAINTENANCE SOFTWARE	4	Identify and isolate faults before they are encountered in an operational system.
MEASUREMENTS	2	Provide a single object to collect counts and measurements and then distribute them to concerned parties.
MINIMIZE HUMAN INTERVENTION	ACGHKN	Design the system so that human intervention is not required.
NETWORK ELEMENT MANAGER	Not yet written	Provide a single point to consolidate human interaction with switches for both integrity and measurement purposes.
PEOPLE KNOW BEST	ACGHKN	Provide a way for an expert human to override the system's automatic responses.
REASSESS OVERLOAD DECISION	Hanm2000	Conditions change and the system should periodically reassess its decisions to see if they are still valid.
SELF PROTECTION	5	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	Hanm2000, Mesz	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	ACGHKN	Give SYSTEM INTEGRITY the power and ability to handle the situations that might arise.
SYSTEM INTEGRITY	3	Provide an object that will watch for abnormal system activity and will initiate necessary reactions.
TELECOM DATA HANDLING	DeLa	Telecom systems have some unique attributes for their measurements.

References

[ACGHKN] 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.

[CCRSS] Cieslak, T., L. Croxall, J. Roberts, M. Saad, and J. Scanlon, 1977. "No 4 ESS: Software Organization and Basic Call handling." **Bell System Technical Journal**, vol. 56, no. 7, Sept, 1977: 1113-1138.

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

[GHHJ] 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.

[Hanm2000] Hanmer, R. 2000. "Real Time and Resource Overload" in Proceedings of PLoP 2000 Conference.

[Hanm2000a] Hanmer, R. S. "Software Reliability?" Invited editorial for **ASME Journal of Electronic Packaging,** Volume 122, December, 2000.

[Hanm2001] Hanmer, R. 2001. "Call Processing" in Proceedings of PLoP 2001 Conference.

[LA] Lee, P., and Anderson, T., 1981, **Fault Tolerance: Principles and Practice**, Springer-Verlag, New York.

[Mesz] 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.

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

[SW] 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.