Telecommunications Input and Output Pattern Language

Robert Hanmer
Lucent Technologies
hanmer@lucent.com
2000 North Naperville Road
PO Box 3033
Naperville, IL 60566-7033
voice: 630 979-4786

Greg Stymfal
AG Communications Systems
stymfalg@agcs.com
2500 Utopia Drive
PO Box 52179
Phoenix, AZ 85072-2179
voice: 602 582-7138

Introduction

A specialized set of patterns for defining the human-machine interface has come into use within the world of telecommunications switching products. The patterns presented here provide for the essential interaction between a system and its human masters. Several of the patterns discuss concepts specific to a telecommunications system, but most are general enough to provide insight for anyone designing the Input/Output ("I/O") interface for a large system.

This paper begins by discussing the environment within which these patterns are applicable. Two different methods of visualizing the overall structure of this language are then presented, one graphical and the other in the form of pattern groupings by related topics. The patterns are then introduced. Supplemental material includes thumbnail sketches of several patterns referenced frequently.

Background

The environment within which telecommunications switching equipment is installed is different than that of a traditional computing center. This section describes some of these differences. These differences constitute much of the common context within which these patterns are useful.

A key difference between the input output of a telecommunications system and a general computer is that the I/O of a telephone switching system is purely secondary to its main purpose. A general computer might be specialized to a special function, such as processing mathematical equations quickly, but the results still must proceed through the Input/Output channel. This is not the case in a switching system. These systems are designed to manage specialized hardware to connect (switch) different telephone lines. This function does not require CRTs, paper printers, keyboards, mice or magnetic tape.

Another key difference is the multitude of workers that are responsible for system maintenance and administration. This "On-Site Workforce" (OSWF) can be grouped into several different communities of interest ([HJJS+], [GHHJ], [GHRS]):

- Those concerned with the maintenance of the machine. This community has a workcenter called the "Maintenance Operations Center" (MOC).
- Those concerned with administrating the machine. Their workcenter is called the "Maintenance Administration Center" (MAC).
- Those concerned with the telephone lines and trunks connected to the switching machine. They have two workcenters in addition to the MAC, called the "Trunk Operations Center" (TOC) and the "Terminal Equipment Center" (TEC).

Telephone switching systems are large. A switching office serving a city center might occupy several thousand square feet, and have tens of thousands of telephone lines and trunks connected to it. This leads to the workforce being a long physical distance from the primary input/output devices when they are working inside the system. It also means that to the workforce having many pieces of equipment that they are responsible to operate and maintain.

Reliability is a very important system capability. Reliability engineers calculated the expected reliability based upon some predicted “Mean Time To Repair” (MTTR). The time it takes for the maintenance personnel to respond to a failing component is a key contributor to this metric. It is therefore very important that the maintenance people be informed quickly of any problems. This reliability, which is usually a few minutes per office per year, also leads the designers towards using custom hardware and software especially designed to facilitate reliable systems. In order to maintain this reliability, telephone switching systems are typically monitored around the clock by personnel trained and ready to respond to any emergency.

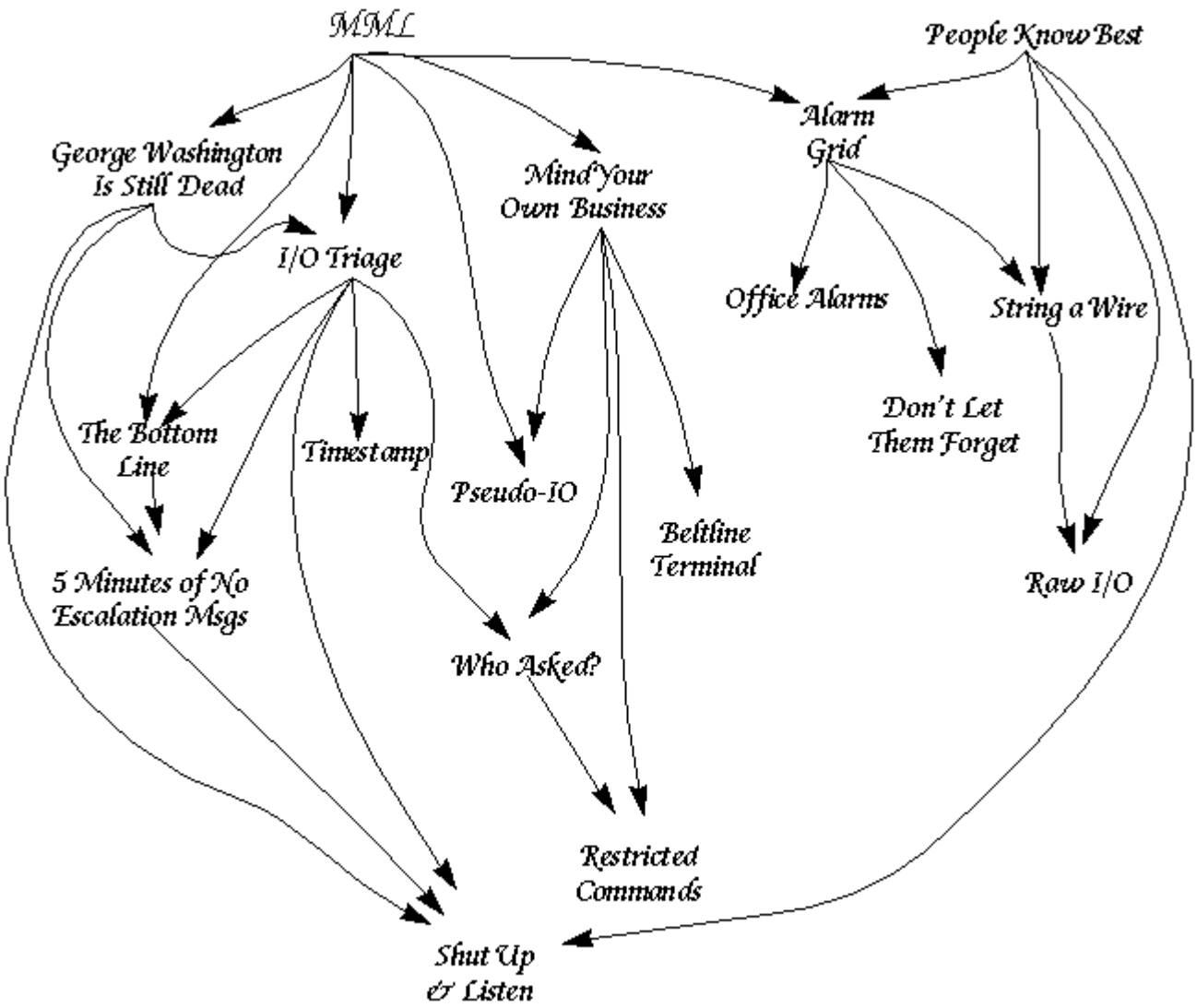
Using custom hardware and software usually results in the system being behind the leading edge of computer technology. When the systems from which these patterns were mined were being created in the 1970's their input/output channels were capable of being operated at either 110 or 1200 baud. ([BDNN+], pp. 169-170)

As telephone networks get more complicated, the desire to remotely monitor and maintain them has increased. The ability to remotely monitor system operations adds an additional level of complexity. Handling this additional complexity wasn't always provided for in the initial designs.

Known Uses

The patterns contained here are the proven practices of a number of telephone switching systems such as the Lucent Technologies 1A ESS™ local switch, the 4ESS™ toll and tandem switch also from Lucent Technologies and the AG Communications Systems GTD-5™ local switch.

Language Map



Groupings of Patterns

These lists divide the patterns up by four different functional concepts that will be used later as keywords on each pattern to help the reader understand how they relate. One pattern, *MML*, deals with all four of these topics. The topics are: physical location, the relative importance of input, the quantity of messages and periods of emergency interaction.

1. Physical **Locations**– Switching offices are large in physical terms. These patterns address the measures need to be taken to mitigate the effects of large size.

- *Mind Your Own Business*
- *Beltline Terminal*
- *Pseudo IO*
- *Alarm Grid*
- *Office Alarms*
- *Restricted Commands*
- *Who Asked?*

2. Relative **Importance** of Input – Some interactions with the system is more important than others. These patterns describe the measures need to be taken to ensure that the important interactions can happen in a timely fashion.

   - *IO Triage*
   - *Timestamp*
   - *Shut Up and Listen*
   - *Don't Let Them Forget*

3. **Quantity** of Messages – Switching systems are large in terms of lines of code and subsystems, many will have things to say at the same time. These patterns help to manage the sheer quantity of messages.

   - *George Washington Is Still Dead*
   - *The Bottom Line*
   - *IO Triage*
   - *5 Minutes of No Escalation Messages*
   - *Shut Up and Listen*

4. Periods of **Emergency** Interaction – During crisis times special rules should apply. These patterns discuss some of these special rules as they impact Input/Output.

   - *Alarm Grid*
   - *Office Alarms*
   - *String A Wire*
   - *Raw I/O*
   - *Don't Let Them Forget*

The Patterns:

MML

Keywords: Location, Importance, Quantity, Emergency

Problem: How does a person communicate with a large and complex machine?

Context: There are many different subsystems that will need to communicate with human operators and administrators. A large volume of I/O with many varied user needs and functions is expected.

Forces: Switching system software development will be easier (fewer people to coordinate with) if each software subsystem defined and implemented its own input output specifications.

Development expense and system architecture will be more streamlined if the different subsystems define input output languages to a common specification.

At least one third of errors are due to human operator error. (The other two thirds being hardware

and software.) Any simple steps that can be done to reduce the man-machine input output system's contribution to this statistic will be beneficial.

Solution: Use a standard messaging format. MML (Man-Machine Language, which is a CCITT standard) and the PDS (Program Documentation Standard) are two examples. This allows the system to be consistent when reporting problems in different parts of the system. It also allows humans to become familiar with the format of messages and thus simplifies learning.

Resulting Context: A large volume of I/O with many varied users needs and functions using a standard language/format will be given a consistency to help users. Mechanisms to deal with the large volume of input and output are needed, such as *Bottom Line* and *George Washington is Still Dead*.

Also since the system is quite large, there might be many different users each trying to interact with the system. To keep things straight, a pattern such as *Mind Your Own Business* and *IO Triage* are useful.

Sometimes, especially to save human resources a computer system might be created to monitor the system. In many cases a custom interface can be created to facilitate the computer to computer communications. Sometimes this extra overhead is too much and a pattern such as *Pseudo-IO* helps by having the monitoring computer use the MML messages intended for humans.

There might not be people watching the system all the time, so some alternative method of alerting them to problems is required. *Alarm Grid* describes a method.

Reference: [CFGLR], pp. 245-246.

Mind Your Own Business

Keywords: Location

Context: There is a standard input and output messaging language (*MML*) defined for the whole system. This implies that there will be some framework within the system to that supports all I/O, consolidating messages from every part of the system into a single unified output.

Problem: But if the output is all derived from a single subsystem, who should see it?

Forces: Only a small number of the workers need to see any particular output. For example the people in the Maintenance Operations Center (MOC) don't need to see messages about the progress of a Recent Change (database update) transaction.

The system shouldn't confuse the workers by presenting information that they don't know how to use.

The volume of messages that are possible from a large system is enormous. If it were all to be dumped to one terminal it would be next to impossible for the workers to find the information that they need.

Access control is enhanced if the type of information a particular terminal receives is changed through simple wiring configurations (i.e. which terminal plugs in where).

Frequently the entities watching the outputs are not human, but are other computer systems. But these systems are still only interested in a small part of the possible output, so specialization is still useful.

Much of the information that the system presents to the workers is routine and not related to any particular user that may or may not be logged in at any one time.

The primary output devices in a maintenance center should always receive output, even if no one is logged on.

Solution: Define different output classifications. Mark different terminal/console connections to only receive output for some classifications. These are called "logical channels". The system sends messages only to the community of users that is interested in them. This results in labor savings -- workers don't need to wade through the output to find the messages that they are interested in.

Resulting Context: A large volume of I/O is reduced on any one channel by distributed it by function. Some users move around and might still need to see the output. *Beltline Terminals* addresses this by providing the ability to redirect output to a different output device.

By centralizing output processing (see *IO Triage*), the categories and logical channels can be used to send output to the right place as in *Who Asked?*.

Restricted Commands are useful because some of the workers at some logical channels might be familiar only with their job function and not with the entire range of system capabilities. This could lead to inadvertently entering a message that results in system degradation.

Reference: [CFGLR], pp. 245-246.

IO Triage

Keywords: Quantity, Importance

Problem: Important information is hidden or delayed by less important information.

Context: A large volume of I/O is still presented to the workers (even after applying the message reduction patterns *George Washington Is Still Dead* and *The Bottom Line*). The I/O message stream is standardized (*MML*).

Forces: Too much information deadens the workers.

Information about a critical situation (for example all the telephone lines going out of service, or a critical piece of the system hardware failing) needs to be issued as soon as it is detected to allow the workers to repair it.

Information that reports trivial things can be deferred to display after the important information

is presented.

Solution: Tag messages with a priority classification. The input/output software should be designed to expedite the presentation of the higher priority message, putting them on the output device before a message of a lower priority. A consistent definition of the different priorities is required. One such definition is this:

- CRITICAL: The office cannot perform its required actions to process telephone calls.
- MAJOR: The system is in a state that one more failure will result in a CRITICAL loss of service.
- MINOR: A small portion of the system is in trouble. The system is more than one failure away from a lack of service.
- MANUAL: System responses to human input.
- Informational: The system is behaving normally.

Resulting Context: This pattern leads to the most important information coming out first, followed by possibly quite old less important information. This necessitates a gatekeeping functionality in the system to prepare the messages for display in the desired order. Some sort of timestamping or message sequence tagging is needed to facilitate a complete understanding of the system state (see *Timestamp*).

Output messages are sometimes referred to as "Action Messages" because they require some action on the part of the local workforce.

Five levels of output message are used because conventional wisdom is that humans work best with five +/- two things.

Timestamp

Keywords: Importance

Problem: How do workers provide some sense to seemingly random order of output?

Context: *IO Triage* has been applied. Messages may not come out in the order that parts of the system request that they be printed.

Forces: Some problems that are reported in the I/O stream are difficult to analyze. Little clues might be important and the order in which things happen can be important when analyzing output. Especially when trying to decipher near coincident failures. That informational message printed five minutes ago might contain the essential clue to help with the critical message printed fifteen minutes ago.

Time within a computer system is different than time to humans. A timestamp with just the minutes and seconds might hide details from the millisecond time period.

Since there is some gatekeeper (implied by *IO Triage*) it could perform a little extra processing and tag messages with some helpful information.

Solution: Apply a sequence number to all messages when the gatekeeper receives them. That will help identify message ordering.

Since humans like to work with a time base, also display the current time on messages. "Current" means when the message was sent to the gatekeeper. If the message printing time is also displayed, the difference in times can provide a valuable clue to the analyst.

Who Asked?

Keywords: Location

Problem: What logical channel receives the results of a specific manual input request. Lots of terminals, lots of terminal classifications, many users, do they all see my messages?

Context: A worker usually only uses one terminal at a time. *Mind Your Own Business* has resulted in the I/O stream being divided into different logical channels. Due to *IO Triage* having been applied, output is centralized for the prioritization by some gatekeeper function.

Forces: You could broadcast the answer to everyone. But that would confuse the workers that have no idea about that function (i.e. maintenance workers seeing recent change output). It also pollutes the output channels with extraneous information, resulting in desired information being swamped.

There is a class of workers that has primary responsibility for the correct functioning of the system. These workers should be kept informed, to some degree, of what the other communities of workers are doing.

Solution: Display the output related to a specific input request primarily to the logical channel that made the request. You might want to put it on an additional channel to allow for logging or oversight by a "super user".

Resulting Context: Sometimes important information will need to be displayed at the primary maintenance terminal, so that the people most responsible for the correct functioning of the system see it.

By tagging the message as Manual to indicate it is a direct response to an input message (see *IO Triage*), the message won't be misunderstood as being a spontaneous system output.

This pattern is the output converse for *Restricted Commands*.

Reference: [CFGLR], pp. 245-246.

George Washington is Still Dead

Keywords: Quantity

Problem: How can the output be kept free from too many messages saying the same thing?

Context: Sometimes the system needs to output reports about the state of the machine. While a particular problem might be detected hundreds of times (as in *The Bottom Line*), if the state isn't changing frequently the messages might be overkill. For example, if hundreds of trunks go out of service, the office "condition" might change when the first one trunk goes out of service, and then not change again until the 101st trunk goes out of service.

Forces: Displaying the status of the office "condition" for each new trunk out of service report from the second to 100th trunk is too much information. There isn't really that much change to require a new report.

Too much information deadens the workers.

The fact that the office state or "condition" is being changed is the real information that the workers need to know.

Solution: Only display messages that report a change in state. Don't report them each time the alarmed state is detected.

Resulting Context: Redundancies will be removed from an otherwise large volume of I/O.

Sometimes there are no actions that the workers can perform in response to a change in system state. It is in these circumstances that the *Five Minutes of No Escalation Messages* pattern applies.

The reports of state changes are important, so some method of reporting them is required. *IO Triage* is needed to keep the important information coming out even when the output channels are flooded.

Even with the reduction in messages from applying this pattern, a mechanism to ensure that humans are able to input a message against a flood of output is required. *Shut Up and Listen* addresses this problem.

The Bottom Line

Keywords: Quantity

Problem: Many messages are about the same type of event (such as a trunk going out of service) and they flood the output message stream.

Context: Certain circumstances result in many, many reports. For example, if a part of the system interfacing to hundreds of trunks has just gone out of service, a message reporting that a trunk is out of service might display hundreds of reports, one for each trunk. This pattern applies to the places where a report should be generated because some condition has just been detected.

Forces: Outputting a report when every event happens pollutes the output channels, diverting attention from other activities. Even with a priority output system in place (see *IO Triage*), the channel can become full and important output or input (see *Shut Up and Listen*) can be missed.

Sometimes reporting of many events gives the workers an idea of a trend, but this information can be given many times, or as a single message reporting many events.

Solution: Group messages about a common event and only display a summary message that includes a tally of the number of occurrences.

Resulting Context: The system should also provide a way to provide the entire output at human request, as it may be essential to identify problems.

Even with the reduction in messages from applying this pattern, a mechanism to ensure that humans are able to input a message against a flood of output is required. *Shut Up and Listen* addresses this problem.

This pattern relies upon the developers of the individual subsystems. It is their responsibility to perform the summarization of messages.

Five Minutes of No Escalation Messages

Keywords: Quantity

Problem: Rolling in console messages: the human-machine interface is saturated with error reports that may be rolling off the screen, or consuming resources just to display the messages.

Context: Any continuous-running, fault-tolerant system with escalation, where transient conditions may be present. The transient nature produces copious messages even with summary and priority filters (*The Bottom Line*, *George Washington is Still Dead*, *IO Triage*).

Forces: There is no sense in wasting time or reducing level of service trying to solve a problem that will go away by itself.

Many problems work themselves out, given time.

You don't want the switch using all of its resources displaying messages.

You don't want to panic the user by making them think the switch is out of control (*Minimize Human Intervention* [PLOPD-2]).

The only user action related to the escalation messages may be inappropriate to the goal of preserving system sanity.

There are other computer systems monitoring the actions taken. These systems can deal with a great volume of messages.

Solution: When taking the first action down a scenario that could lead to an excess number of messages, display a message. Then periodically display an update message. If the abnormal

condition ends, display a message that everything is back to normal. Do not display a message for every change in state.

Continue continuous machine to machine communication of status and actions throughout this period.

If something is so important that it can't wait five minutes, the *Alarm Grid* should report it.

Resulting Context: This solution will keep the system operator from panicking from seeing too many messages. Machine to machine messages and measurements will keep a record for later evaluation as well as keeping the systems actions visible to people who can deal with it.

Other messages that are not related to the escalating situation that is producing too many messages will be displayed as though the system were normal. Thus the volume of escalation messages does not adversely affect the normal functioning of the system.

Note the conflict with *People Know Best* [PLOPD-2] in which humans should be kept thoroughly informed of system progress.

Shut Up and Listen

Keywords: Quantity, Importance

Problem: Humans need to be heard by the system.

Context: A priority scheme for message output is in place (see *IO Triage*). This will get the most important information out to the workers first. It doesn't address humans taking control and altering system behavior. The *People Know Best* [PLOPD-2] pattern discusses that even when the system is designed to correct problems automatically experts will sometimes be able to help the system through "stressful" incidents.

Sometimes, even though *George Washington Is Still Dead* and *The Bottom Line* are applied, the output stream is still flooded.

Forces: The system output stream is busy with both priority messages reporting some abnormal event, and informational messages. Users need a way to shut the output stream off. But if they shut it off, important information might be missed. If they wait until the outputting is completed the system might be in a degraded service mode. Then again, in some circumstances the output stream may be perpetually flooded.

Full Duplex input/output might help, but that has problems of message interpretation being made more difficult. If half duplex is chosen some way of interrupting the output is necessary.

Solution: Give a higher priority to recognizing human input commands than is given to display more output information. Provide the system with a way to make sure that input messages are processed even when the output system is operating at full capacity. Mark the output related to human input messages at priority level comparable with CRITICAL output messages priority (see *IO Triage*) so that their responses are not stuck at the end of the output message queue.

Resulting Context: Human requests (input messages) will be heard even though the output buffer is full. Both Input and Output messages are prioritized.

Human input messages are all treated as being at the same priority level. A separate input mechanism (such as a Master Control Console) that bypasses the I/O subsystem is provided for high priority input.

Restricted Commands

Keywords: Location

Problem: How should the system be protected from malicious or naive users?

Context: Some input messages can disrupt the system very dramatically. The system has many input commands; many are quite similar to others. There are also different categories of users that have a need to input different commands (*Mind Your Own Business*). Each user category has its own subset.

Forces: Some commands could result in disastrous results, such as reducing call handling capability or causing some unnecessary and time-consuming recovery action.

There are some users, who have a certain terminal assigned to them (*Who asked?*), that know what is safe and what is not. They can be trusted.

Solution: Restrict certain key input to the primary terminal of a given classification.

This will provide additional tracing for the command and will put the knowledge that it was executed into the hands of the system experts, rather than the person who only knows about a small part of the system.

Pseudo-IO

Keywords: Location

Problem: This is a big system and sometimes one subsystem will provide a way to access certain information for humans. Now another subsystem needs this same information also.

Context: The system is large and has several subsystems. The I/O subsystem is one that processes individually buffered messages and distributes them to the appropriate subsystem to handle them. I/O that used to go only from machine to human now must also go from machine to machine.

The system has many logical I/O channels. (*Mind Your Own Business*) There is some gatekeeper function, introduced by *IO Triage*, which will mark priorities on the messages as well.

Forces: A new machine to machine interface, with all of its inherent complexities, could be defined (in addition to the human interface) or...

Solution: Provide a way for one subsystem to insert a message into the input message stream for

the other subsystem. In other words, allow the system itself to create a message that is processed just like any other input message.

Resulting Context: This provides a system that makes it easy to add new capabilities and to connect them to already present input/output services.

Note the difference between this pattern and *String A Wire* that deals with informing monitoring systems of the system state during emergencies.

Beltline Terminal

Keywords: Location

Problem: The terminal isn't where the worker needs to be to do their work.

Context: A large volume of I/O distributed by function. Some workers move around in the office. With the size of the system measured in floor space, they might be a long way from their maintenance terminal when they want to enter or see some messages related to what they are doing. (e.g. they might be changing circuit packs). Certain terminals are dedicated for these workers. *(*See *Mind Your Own Business.)*

For many years within a telephone office there have been jacks to plug in audio headsets to allow the workers to talk with other workers.

Forces: Reduced labor costs/higher productivity if the worker doesn't need to continually go for long walks to use a terminal.

Reduced possibility of accidentally damaging something by frequent traversals of the office to reach the terminal.

Increased system reliability if the worker can attend to their work quickly. The long walks to the terminal area increase the Mean Time To Repair (MTTR).

The system is already physically large, with many wires. The system is mounted in custom cabinets with custom wiring. Adding a few additional wires around the office to allow the workers to plug in a terminal (in addition to audio headsets) is a minor expense.

Solution: Provide remote terminal connections out in the aisles with the equipment so that the worker can plug in a terminal input/output device anywhere they want. This is called a "beltline terminal". Also provide the ability for the worker to redirect (copy) the output of the desired logical channel to that remote terminal.

Resulting Context: The workers can move around the office and can take their terminal with them. This helps increase productivity and system reliability and reduce MTTR. The typical configuration of a beltline terminal is an ordinary display device, such as a VT100™ terminal, on a cart.

Reference: [HJJS+], p. 1050

Alarm Grid

Keywords: Location, Emergency

Problem: How can we get information to workers immediately when a significant problem has occurred?

Context: Some things are too important to wait for a potentially slow I/O system to process and to wait for some human to notice and read. *People Know Best* [PLOPD-2] requires that humans and support systems monitoring the system have full information about its state. Due to the complexity of the input/output system the overhead to perform the standard I/O cannot be afforded in all circumstances, so *MML* does not apply. The time to begin the correct remedial action has a contribution towards the Mean Time to Repair (MTTR), which in turn has an impact on the system's overall reliability.

Forces: In a large office a simple message on one or several terminals might go unnoticed by the office personnel that are needed to handle the problem. This can lead to service being impacted longer than is absolutely necessary.

The I/O system has a lower processing priority than the systems that ensure that the system is sane and functioning normally. We cannot count on the I/O system being executed in a timely manner during times of crisis. This can delay processing for a long time in a system that is trying to recover from catastrophes.

During crisis situations the humans are probably looking elsewhere and they will not see a report on a terminal.

Solution: Provide a specialized alarming system. On this alarm "grid", alert all concerned personnel via audio and visual mechanisms that there is a problem.

This alarm grid will provide an interface to the human operators that parallels the interface provided by the I/O system. Since it is in parallel, it will provide the operators with information that they might not get through the I/O system in a timely manner.

By breaking the equipment into different categories, the alarm grids can be refined to report problems within certain communities only. This is similar to the logical channel concept of *Mind Your Own Business*. The different grids can be grouped together to report larger combined events during off-hours, when fewer people are watching the system.

Resulting Context: Priority messages will be presented to personnel in a way that stimulates three of the five senses.

The specification of the alarmed items is usually done at design time. Since these are big systems that may vary subtly from office to office some method of customizing to a specific customer site is desirable. See *Office Alarms*.

The system should be one that will have a low probability of failure. The hardware should be designed to have few failure points and to use reliable components. Another consideration in the

hardware design is to make the system "toothpick proof". This refers to the ease that office personnel can disable a part of the system by inserting a toothpick into the appropriate relay contact to prevent closure.

In the area of the *Alarm Grid* the usual principle of "failing silently" may not apply. If the alarm grid itself is to fail, having it report its own failure is a desirable event.

The alarm grid will typically be presented both locally at the switching office and to some remote monitoring location.

Reference: [HJJS+], p. 1051.

Office Alarms

Keywords: Location, Emergency

Problem: How can problems that are unique to a particular field site be integrated into the pre-defined classification of messages? Different sites might have custom alarming requirements, such as door locks, specialized A/C, coffeepots, related systems, etc.

Context: Priority messages have visual and audio alarms from the *Alarm Grid*.

Forces: You could define a separate, parallel interface for site specific alarms. That could introduce confusion, as the user interface would probably not be identical to the standard user interface.

The actual interface and system actions when the alarm is "fired" could be transparent and indistinguishable from the base generic supplied alarms. This will help avoid user errors and potential outages based upon failure to act on stimuli.

Alarming some things and not others results in not being given the importance that they should.

The designers and equipment manufacturers can't foresee every unique office configuration containing conditions that need to be alarmed. The customers probably don't want to pay for such thoroughness either.

Good user interface design has similar things behaving in like manners.

Solution: Allow the seamless definition of "office alarms". Site specific alarming should be facilitated through whatever specialized definition mechanism is required.

Adding a new alarm to an existing *Alarm Grid* is an example of the *Decorator* pattern from the [GOF].

Resulting Context: Both generic priority messages and site specific events have visual and audio alarms.

This provides a measure of "customer programmability" to the office. They may tailor at least this small part to their own corporate operating policies.

Don't Let Them Forget

Keywords: Importance, Emergency

Problem: How long should the manual action of silencing the alarms be remembered by the system? Should it reassert alarms immediately, or keep them quieted for some pre-determined length of time?

Context: The *alarm grid* can be very annoying, leading workers to want to silence and ignore them. A system in trouble can have many alarms asserted simultaneously. Sometimes the noise can be overwhelming and distracting to the workers.

Forces: Some mechanism to silence the alarms is desirable to provide some quiet for thought. If the system is in trouble the ability to function properly may be compromised if alarms aren't reported promptly. This can lengthen Mean Time to Repair.

Ignoring the manual action to silence alarms will be annoying and distracting to the worker. They might spend time trying to silence them, instead of resolving the problem that is trying to be reported.

Solution: Don't remember a request to silence alarms. The next time that the system detects an alarmed condition, sound the alarm, regardless of how recently it was retired.

Resulting Context: The system will report alarms whenever they are appropriate. The worker will have to repeatedly silence them, until the problem is corrected and the system no longer reasserts the alarm. The worker will have the grace period of the alarm detection period during which their retire alarm action will be remembered.

This pattern differs from *George Washington Is Still Dead* in the severity of messages. This pattern deals with alarmed situations, whereas the other pattern is about slightly less critical information.

String A Wire

Keywords: Emergency

Problem: Critically important information must get to other computer systems.

Context: Sometimes there is the need for direct Machine to Machine messages. This might be because I/O is sometimes inefficient, or because the system is in a partial capability mode or the system can't afford the resources to do I/O.

Forces: Standard *MML* messages could be used between two systems, but both systems will have to spend resources (memory, time) to encode and decode the message in the foreign language.

An interface specification document has probably been written that outlines how the two systems should communicate. Or at least that the need to communicate exists and what information

should be exchanged.

Solution: Provide a hardwired messaging connection. (E.g. Dynamic Overload Controls, E2A telemetry channels, etc.) Use the interface specification document to describe the interface so that both systems can be developed towards a common interface view.

Resulting context: The system will present information to monitoring systems even when the ordinary I/O methods cannot be afforded due to emergency or resource utilization priorities.

If more information is needed than a simple hardwired messaging connection can provide, consider adding some *Raw I/O* capability.

Reference: [GHHJ], pp. 1174-1175.

Raw I/O

Keywords: Emergency

Problem: How do we get I/O out during those times when the I/O system is unavailable, for example during system initialization, or times of emergency.

Context: Sometimes the system is in a partial capability mode or the system can't afford the resources to do all the I/O for machine to human messages. But these might be just the situations where people need to be informed about system state. *String A Wire* cannot provide enough information in these circumstances.

Forces: Humans watching a system that has no ability to communicate are tempted to do something drastic -- like manually requesting an initialization. This is rarely the right thing to do. (See *Minimize Human Intervention [PLOPD-2]*).

If the system doesn't communicate any information during these times of crisis, there is nothing to help an expert user to help the system (see *People Know Best* [PLOPD-2]).

Recovery and initialization programs are in total control of the system, and typically have the ability to look inside other subsystems and perform their work. In fact this is probably safer than allowing the I/O subsystem loose during periods of system recovery.

Solution: Display the output out via brute force mechanisms such as writing directly to the channel and avoiding the I/O system. This should be restricted to recovery periods only.

The periods during which this raw I/O mechanism should be limited. The system should endeavor to restore the I/O system as early as possible.

Resulting Context: During critical periods the safe course is chosen and the recovery programs administer I/O rather than relying on the I/O system. The I/O system is probably much larger with many more interfaces, so this results in the amount of code that must be available during recovery.

Acknowledgments:

The authors would like to acknowledge the following people for their help preparing this pattern language:

- Reviewers within Lucent Technologies: Rick Rockershousen who reviewed 1A ESS™ content; Glen Moore who was one of the original developers of the 4ESS Switch I/O system; Chuck Borcher and Deatrice Childs who participated in a review of *Alarm Grid* and *Office Alarms*.
- David DeLano, our PloP-98 shepherd provided invaluable comments to improve these patterns.
- Ralph Jones. 1937-1998. Inventor of the *Alarm Grid* concept.

Five Minutes of No Escalation Messages is co-authored by Mike Adams and a previous version was published in [PLOPD-2].

People Know Best was written by Robert Gamoke and is published in [PLOPD-2].

Referenced Patterns:

Pattern Name	*Problem*	*Solution*	*PLOPD-2 Page*
Minimize Human Intervention	People cause many problems	Design system to care for itself	551-552
People Know Best	Balancing human authority with automation	Assume people know best	552-553

References:

[BDNN+]: Budlong, A. H., B. G. DeLugish, S. M. Neville, J. S. Nowak, J. L. Quinn and F. W. Wendland. 1977. "1A Processor: Control System." **Bell System Technical Journal**, vol. 56, no. 2, Feb., 1977: 135-179.

[CFGLR]: Clement, G. F., P. S. Fuss, R. J. Griffith, R. C. Lee and R. D. Royer. 1977. "1A Processor: Control, Admininstrative, and Utility Software." **Bell System Technical Journal**, vol. 56, no. 2, Feb., 1977: 237-254.

[GHHJ]: Green, T. V., D. G. Haenschke, B. H. Hornbach and C. E. Johnson. 1977. "No 4 ESS: Network Management and Traffic Administration." **Bell System Technical Journal**, vol. 56, no. 7, Sept, 1977: 1169-1202.

[GHRS]: Giunta, J. A., S. F. Heath III, J. T. Raleigh, and 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.

[GOF]: Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. **Design Patterns Elements of Reusable Object-Oriented Software**. Reading, MA: Addison-Wesley Publishing Co.

[HJJS+]: Huttenhoff, J. H., J. Janik, Jr., G. D. Johnson, W. R. Schleicher, M. F. Slana, and F. H. Tendick, Jr. 1977. "No 4 ESS: Peripheral System." **Bell System Technical Journal**, vol. 56, no. 7, Sept, 1977: 1029-1055.

[PLOPD-2]: Adams, M., J. Coplien, R. Gamoke, R. Hanmer, F. Keeve and K. Nicodemus. 1996. "Fault-Tolerant Telecommunication System Patterns" in **Pattern Languages of Program Design,** edited by J. M. Vlissides, J. O. Coplien and N. L. Kerth. Reading, MA: Addison-Wesley Publishing Co.