An Input and Output Pattern Language

Lessons from Telecommunications

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 Communication 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 ("IO") 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 topical pattern groupings. The patterns are then introduced. Supplemental material includes thumbnail sketches of the patterns.

Background

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

A key difference between the input/output of a telecommunications system and a general computer is that the IO 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 IO channel. This is not the case in a switching system. These systems manage specialized hardware to connect (switch) different telephone lines. This function does not require CRTs, paper printers, keyboards, mice or magnetic tape.

Another significant difference is the multitude of workers that are responsible for system maintenance and administration. This workforce 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).

In many computing environments a console terminal is the primary channel for the operator to communicate with the system. Most users are not authorized to send and receive administrative commands. In telecommunications systems there are many terminals within each workcenter that are each as powerful as a traditional console.

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. As a result, the workforce is a long physical distance from the primary input/output devices when they are working inside the system. It also means that the workforce has many pieces of equipment to operate and maintain.

Reliability is a very important system capability. Reliability engineers calculated the expected availability of the system based upon some predicted Mean Time To Repair (MTTR). This is the time it takes for maintenance personnel to respond to and repair a component. It is therefore very important that the maintenance people be informed quickly of any problems. Reliability, usually expressed in terms of a few minutes of downtime 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 availability, 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) In some cases these primitive input/output devices are used because they are well known to the system designers and are more reliable.

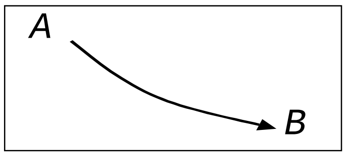
Even as telephone networks get more complicated, the desire to remotely monitor and maintain them has increased. The ability to remotely monitor system operations adds an additional level of complexity. Handling this additional complexity wasn't always provided for in the initial designs.

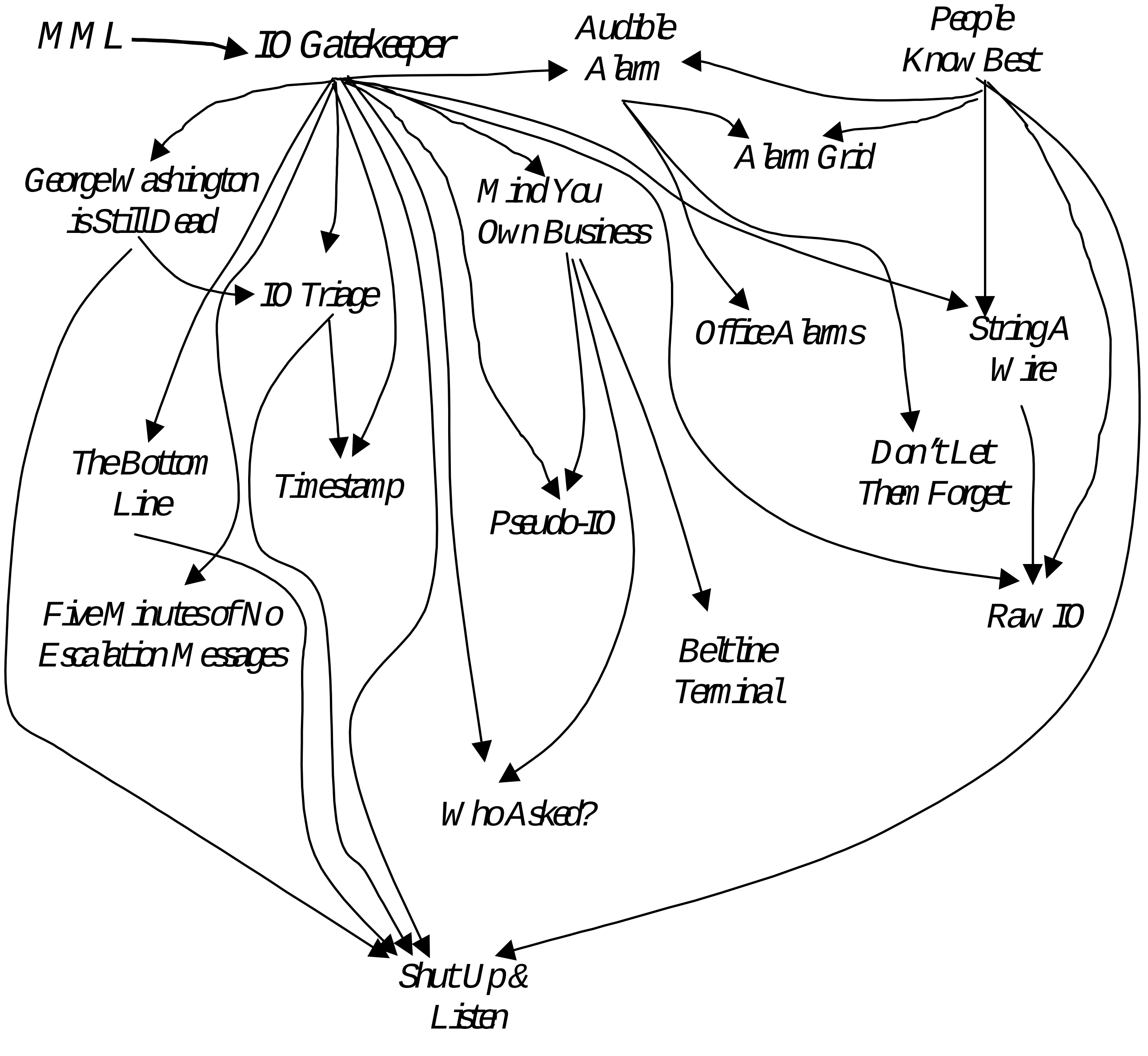
Known Uses

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, the AG Communication Systems GTD-5® local switch, the Siemens EWSD switch and the Alcatel S-11 switch.

Language Map

The following diagram shows the relationships between the patterns in this language. The diagram shows patterns that enhance the solutions other patterns, resolve previously unresolved forces in a pattern, or takes advantage of an earlier pattern to provide some new system capability. In the example to the right, pattern *B* refines pattern *A*, helping to solve unresolved forces or new problems that *A* introduced.





Categories of Patterns

Four functional categories are used later as keywords on each pattern. Both *MML* and *IO Gatekeeper* deal with all four of these topics. The categories are physical location, the relative importance of input, the quantity of messages, and periods of emergency interaction.

1. Physical **Location**– Switching offices are large in physical terms. These patterns present measures to be taken to mitigate the effects of large size.
    - *Mind Your Own Business*
    - *Beltline Terminal*
    - *Pseudo IO*
    - *Audible Alarms*
    - *Alarm Grid*
    - *Office Alarms*
    - *Who Asked?*
2. Relative **Importance** of Input – Some interactions with the system are more important than others. These patterns present the measures that need to be taken to ensure that the important interactions 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 numbers of 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.
    - *Audible Alarms*
    - *Alarm Grid*
    - *Office Alarms*
    - *String A Wire*
    - *Raw IO*
    - *Don't Let Them Forget*

The Patterns:

MML

Keywords: Location, Importance, Quantity, Emergency

also known as **huMan Machine Language**

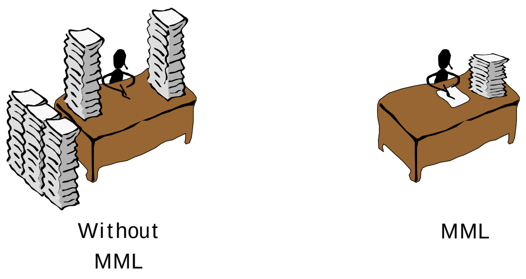
Problem: How can a communications with a large and complex machine be made easier for humans and more reliable?

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

Forces:

- Switching system software development will be easier if each software subsystem defined and implemented its own input output specifications. It is easier to develop alone than to coordinate with others.
- Development expense and system architecture will be more streamlined if the different subsystems define the system languages to a common specification.
- At least one third of errors during system operation are due to mistakes made by the human operator. (The other two thirds are hardware and software faults.) Any steps that can be done to reduce the human-machine IO system's contribution to this statistic will be beneficial.

Solution: Use a standard messaging format. 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. This reduces the amount of documentation that is required because there can be commonality between subsystems.



Resulting Context: A large volume of IO 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*, *Who Asked?* 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 computer to computer communications. Sometimes this is too much overhead and a pattern such as *Pseudo-IO* helps by having the monitoring computer use the *MML* messages intended for humans.

Examples: MML (huMan-Machine Language, an international standard) and PDS (Program Documentation Standard) are two examples.

Rationale: Imagine if all application/user programs on a general computing system used the same standard message language. That is the scope of this pattern. This is accomplished in the world of telecommunications as described in the background section. Doing so lowers the rate of procedural (human caused) errors and hence increases the system availability.

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

IO Gatekeeper

Keywords: Location, Importance, Quantity, Emergency

Problem: How can a large system support the existence of a single *MML* for a large community of authorized users?

Context: Input output is secondary to the system's primary purpose.

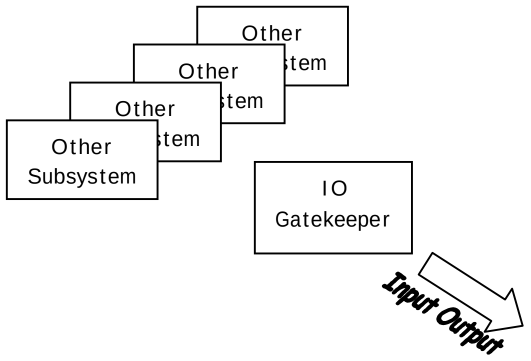
There may be many different subsystems that need some sort of interface with the human world.

Forces:

♦ Allowing each subsystem to send messages to the terminals independantly produces anarchy. Obtaining a cohesive view of the system's status is impossible due to the disordered presentation.

♦ Creating a centralized subsystem introduces a bottleneck into the input output processing.

Solution: Design a centralized point to conduct all communication between humans and the system. Design and use an internal interface language that supports the use of *MML* towards the human interfaces. This will function as a gatekeeper or doorway through which all input output must pass. Use a *Singleton* [GOF] to enforce a single occurrence of the Gatekeeper.



Resulting Context: By defining the interfacing language for the gatekeeper development and use of the system is made easier.

The gatekeeper function allows many useful features to be created. These include *timestamping*, identifying important messages (*IO Triage*), sorting messages to internal or external recipients (*Pseudo-IO* and *Who Asked?*), throttling the output of messages (*The Bottom Line* and *George Washington is Still Dead*). The Gatekeeper can also tie the input output system together with the alarm system (*Audible Alarm)*, using some tag associated with the message (such as its priority tag, see *IO Triage*) to indicate if an alarm should be sounded. The Gatekeeper can also be made responsible for privileges and only allow priviledged people or terminal groups to execute the most potentially destructive commands.

The gatekeeper by definition will consolidate IO together into a single stream. The resulting output stream can be torrential. The patterns *Mind Your Own Business* and *Who Asked?* are employed to sort it out.

Rationale: Since input output is not essential to the primary system processing of telephone calls, the impacts of this bottleneck are mitigated.

If the *IO Gatekeeper* is not a *Singleton* [GOF] few benefits arise from having a gatekeeper at all since the result is the same as if every subsystem communicated directly with humans.

Mind Your Own Business

Keywords: Location

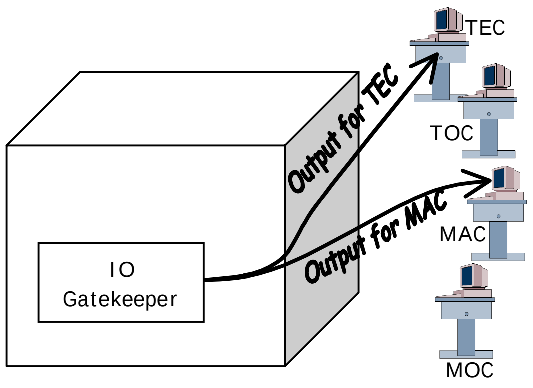
Problem: If the output is all derived from just the *IO Gatekeeper*, who should see it?

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 that supports all IO, consolidating messages from every part of the system into a single unified output.

Forces:

- Only a few 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 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 impossible for workers to find the information they need.
- Access control is enhanced if the type of information a particular terminal receives is changed through simple wiring configurations rather than complicated programming changes.
- 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 output stream specialization is still useful.
- Much of the information the system is routine and not related to any particular user.
- The primary output devices in a maintenance center should always receive output, even if no one is logged on.

Solution: Define different output classifications. The *IO Gatekeeper* should 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 interested in them. Some example logical channels are Maintenance, Secondary Record, Recent Change, and Network Management. A read-only printer is used within maintenance communities to provide for a continual record of activity even if no one is logged on.



Resulting Context: The large volume of IO is reduced on any one channel when it is distributed by function. This results in labor savings -- workers don't need to wade through the output to find the messages that they are interested in. And the reduction in IO helps computer systems monitoring this system by reducing the volume of output they must digest.

Some users move around within an office 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?*.

Example: In a general purpose computing system the operator console receives some messages which user terminals do not receive. In telecommunications systems there are many different operator consoles for different communities of interest.

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

IO Triage

Keywords: Quantity, Importance

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

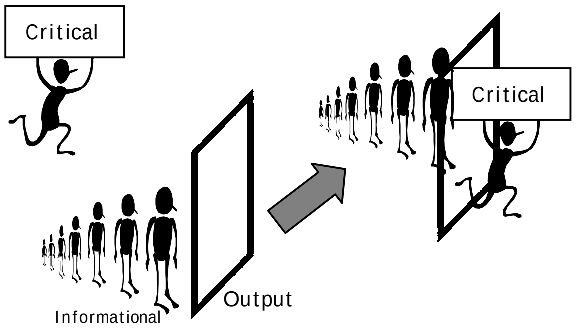
Context: The *IO Gatekeeper* coordinates the IO message stream. A large volume of IO is still presented to the workers even after applying the message reduction patterns *George Washington Is Still Dead* and *The Bottom Line*.

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.
- ♦ Workers might not be looking at the screen when there is something important happening.

Solution: Tag messages with a priority classification. Design input/output software to display a higher priority message on the output device before a message of a lower priority. Use a consistent definition of the different priorities. One such definition is:

Priority Tag	*Meaning*
CRITICAL	The office cannot perform its required actions to process telephone calls. More than one telephone call is affected.
MAJOR	The system is in a state that one more failure will result in a CRITICAL loss of service. One telephone call is affected.
MINOR	A small portion of the system is in trouble. The system is more than one failure away from a total lack of service. No telephone calls are impacted by this problem.
MANUAL	The system is responding to a human input message.
Informational	The system is behaving normally.



Resulting Context: The most important information will be displayed first, followed by less important possibly quite old information. This necessitates the *IO Gatekeeper* to prepare messages for display in the desired order. Use a timestamp or message sequence tag 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.

The force of people not watching their terminal when something important is being reported will be addressed by the *Audible Alarms* pattern. The priority tag provided by *IO Triage* should be used as an input into the alarm system to sound the appropriate alarm.

Timestamp

Keywords: Importance

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

Context: *IO Triage* has been applied. Messages may not come out in the strict order requested by subsystems.

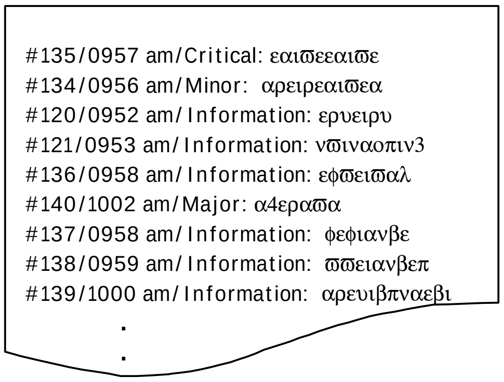
Forces:

♦ Some problems reported in the IO stream are difficult to analyze. Little clues might be important, even the order in which things happen, especially when trying to decipher near coincident failures. The informational message printed fifteen minutes ago might contain the essential clue to help with the critical message printed five minutes ago.

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

♦ The IO Gatekeeper can perform a little extra processing and tag messages with some helpful information.

Solution: Apply a sequence number to all messages when the *IO Gatekeeper* receives them. That will help identify message ordering. Since humans like to work with a time base, the current time can be used as the sequence number. "Current" means when the message was received by the IO subsystem and should contain details resolving to a meaningful time quantum such as seconds or milliseconds.



```
#135/0957 am/Critical: εαιϖεεαιϖε
#134/0956 am/Minor:  αρειρεαιϖεα
#120/0952 am/ Information: ερυειρυ
#121/0953 am/ Information: νϖιναοπιν3
#136/0958 am/ Information: εφϖειϖαλ
#140/1002 am/Major: α4εραϖα
#137/0958 am/ Information:  φεφιανβε
#138/0959 am/ Information:  ϖϖειανβεπ
#139/1000 am/ Information:  αρευιβπναεβι
          .
          .
```

Resulting Context: If the message printing time is also displayed, the difference in times can provide a valuable clue to the analyst.

Who Asked?

Keywords: Location

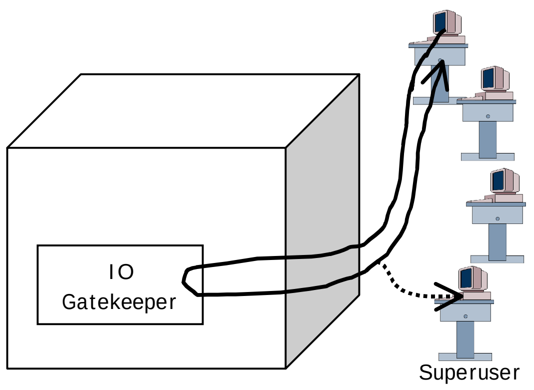
Problem: What logical channel receives the results of a specific manual input request?

Context: A worker usually only uses one terminal at a time. *Mind Your Own Business* has resulted in the IO stream being divided into different logical channels. The *IO Gatekeeper* centralizes output with a prioritization scheme.

Forces:

- Lots of terminals, lots of terminal classifications, many users, do they all see my messages?
- Broadcasting the answer to everyone would confuse the workers that have no idea about that function (i.e. maintenance workers seeing database change output). It also pollutes the output channels with extraneous information, resulting in desired information being lost in reams of information.
- 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: The *IO Gatekeeper* should display the output related to a specific input request to the logical channel that made the request. Send it to all the terminals monitoring a particular logical channel. You might want to put it on additional channels to allow for logging or oversight by a superuser. Give the message a Manual tag to indicate it is a direct response to an input message (see *IO Triage*), so the message won't be misinterpreted as a spontaneous system output.



Resulting Context: Sometimes important information is displayed at the primary maintenance terminal, so people responsible for the correct functioning of the system see it.

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

George Washington is Still Dead

Keywords: Quantity

also known as **Show Changes**

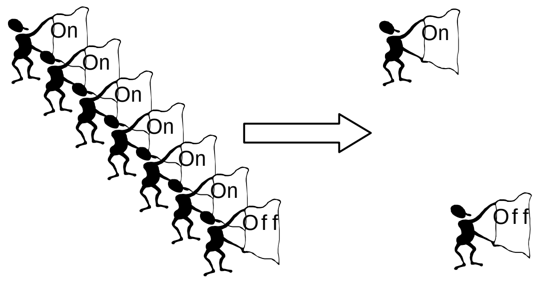
Problem: How can the output be kept free from too many messages saying the same thing?

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 enough to require a new report.
- Too much information deadens the workers.
- The fact that the office state or condition has changed is the real information that the workers need to know.
- Just display messages that report a change in state. Don't display a message each time the alarmed state is detected.

Solution: Have the *IO Gatekeeper* keep track of messages and send only changed state announcements that represent a real change of state.



Resulting Context: Redundancies will be removed from an otherwise large volume of IO.

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.

Related Patterns: 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.

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 is similar to this pattern but deals with multiples of the same report. It summarizes a given report with the number of occurrences. This leads to delay in reporting the event however because of the need for aggregation.

The Bottom Line

Keywords: Quantity

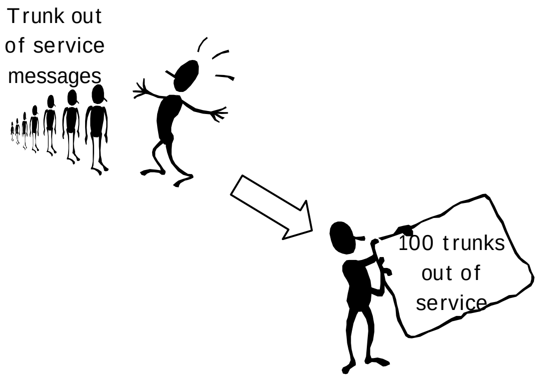
Problem: Many messages are about the same type of event (such as a trunk going out of service) and they flood the output message stream.

Context: Some situations produce many, many reports. For example, if a part of the system interfacing to many 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 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. Reports are not of critical importance and can be delayed slightly during the aggregation period.

Solution: The *IO Gatekeeper* should group messages about a common event and only display a summary message that includes a tally of the number of occurrences. The system should also supply a way to provide the entire output at human request, as it may be essential to identify problems.



Resulting Context: 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.

Related Patterns: George *Washington is Still Dead* deals with state change reports while *the Bottom Line* deals with aggregation of nearly identical messages.

Five Minutes of No Escalation Messages helps for those cases where aggregation delays can only partly be tolerated. It causes messages to be printed out periodically as well as in summary.

Five Minutes of No Escalation Messages

Keywords: Quantity

Problem: The human-machine interface is saturated with error reports for which humans can't do much except worry.

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*).

Some system reports describe events that humans cannot help resolve.

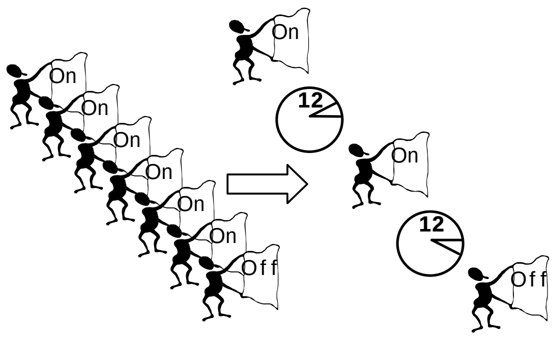
Forces:

- ♦ Many problems work themselves out, given time. There is no sense in wasting time or reducing level of service trying to solve a problem that will go away by itself (*Riding Over Transients [PLOPD-2]*).
- ♦ The switch can use all of its resources displaying messages.
- ♦ Humans panic when they think the switch is out of control (*Minimize Human Intervention* [PLOPD-2]).
- ♦ If the only human action might be detrimental to the system, it can be beneficial to not display information promptly.

Solution: When taking the first action in 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. (see *String A Wire*)

If something is so important that it can't wait five minutes, an *Audible Alarm* 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. There are other computer systems monitoring the actions taken. These systems can deal with a great volume of messages.

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.

Of course the five minute time period alluded to in the title is only a suggestion. It is based upon the situations from which this pattern is mined.

Related Patterns: Note the conflict *People Know Best* [PLOPD-2] in which humans should be kept thoroughly informed of system progress. *Five Minutes of No Escalation Messages* applies when no helpful human actions are possible. This can occur when the system is reporting traffic induced system overloads.

This pattern also applies to those cases where the delay to aggregate messages (see *The Bottom Line*) is too long and some intermediate report is necessary.

Shut Up and Listen

Keywords: Quantity, Importance

Problem: Humans need to be heard by the system. Users need a way to shut the output stream off.

Context: The *IO Gatekeeper* is filling the system output stream with both priority messages reporting abnormal events and informational messages. 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. *People Know Best* [PLOPD-2] states 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.

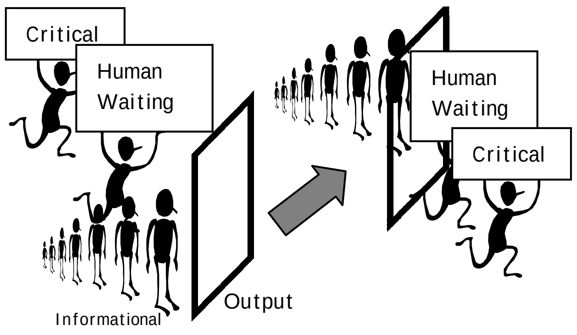
Forces:

- ♦ If the output is shut off, important information might be missed.
- ♦ If users 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 makes message interpretation more difficult. If half duplex is chosen, some way of interrupting the output is necessary.

Solution: The *IO Gatekeeper* should give human input a higher priority than displaying output information. Make sure that input messages are processed even when the output system is operating at full capacity.

Provide a way to interrupt the output stream long enough for a human to get a request into the system. Design this so that no output messages are lost.

Mark the output related to human input messages at priority level comparable with CRITICAL output messages (see *IO Triage*) so that responses to human input 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 [BDNN+]) that bypasses the IO subsystem can be provided for high priority input.

Pseudo-IO

Keywords: Location

Problem: Some subsystem provides a way for humans to access information. Now another subsystem needs this same information also.

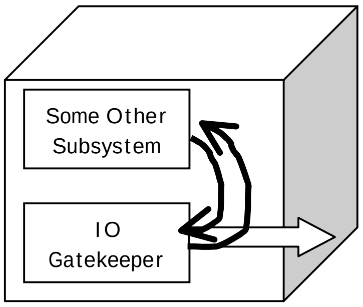
Context: The system is large and has several subsystems. The *IO Gatekeeper* processes individually buffered messages and distributes them to the appropriate subsystem to handle them. IO that previously went only from machine to human now must also go between subsystems inside the system.

The system has many logical IO channels. (see *Mind Your Own Business*).

Forces:

- Adding a machine to machine interface increases the complexity of the system.
- Adding in a powerful new interface or interface protocol is difficult once the system architecture is set.
- Perturbing a working subsystem to add a new interface to a new subsystem risks introducing new faults and breaking the working subsystem.

Solution: Allow one subsystem to insert a message into the input message stream. In other words, allow the system itself to create a message that is processed just like any other input message received from outside the system.



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

Beltline Terminal

Keywords: Location

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

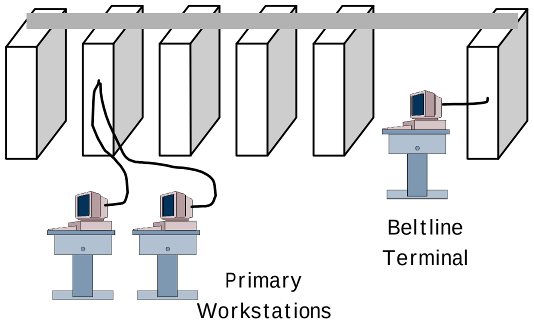
Context: A large volume of IO is distributed by function (see *Mind Your Own Business*). Certain terminals are dedicated for specific workers. Some workers move around in the office. Because of the size of the system, they might be a long way from their maintenance terminal when they want to enter or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

For many years within a telephone office there have been jacks to plug in audio headsets to allow 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 availability if the worker can attend to their work quickly. Long walks to the terminal area increase the Mean Time To Repair (MTTR).

Solution: Provide remote terminal connections so workers can plug in a terminal input/output device anywhere they want. This is called a "beltline terminal". Allow the worker to redirect



(copy) the output of the desired logical channel to the beltline terminal.

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

Rationale: 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.

Reference: [HJJS+], p. 1050

Audible Alarm

Keywords: Location, Emergency

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

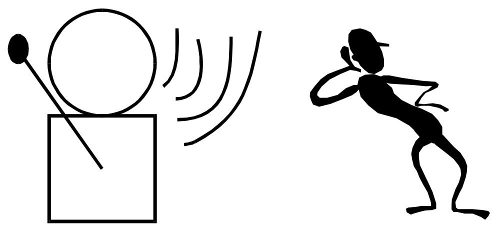
Context: Some things are too important to wait for a potentially slow *IO Gatekeeper* to process and to wait for some human to notice and read. Even with messages receiving a priority tag (*IO Triage*) the *IO Gatekeeper* must still queue the message for output. *People Know Best* [PLOPD-2] requires that support systems and humans that monitor the system have full information about its state. Due to the complexity of the input/output system the overhead to perform the standard IO cannot be afforded in all circumstances, so *MML* does not apply.

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 IO system has a lower processing priority than the systems that ensure that the system is sane and functioning normally. We cannot count on the IO 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 may be looking elsewhere and they will not see a report on a terminal.

Solution: Provide a method of reporting alarms audibly in the office. This will reach out and alert humans that there is some problem and they should find out what it is to reduce telephone service impacts. Remote visual indicator should also be provided, such as colored lights spread throughout the office.

The audible alarm system should be driven by the IO protocol defined within the system. Messages being passed to the *IO Gatekeeper* should have a special tag to indicate that an alarm should be sounded and which alarm. The priority tag defined by *IO Triage* can be used for this. There are a multitude of sounds that can be used to indicate different alarm severities, including sirens, bells, gongs, etc.



Resulting Context: 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 availability.

Priority messages will be presented to personnel in a way that stimulates several 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 alarm system will typically sound alarms both locally at the switching office and to some remote monitoring location.

A method of silencing the alarms by a manual action needs to be implemented. This will allow the craft to have some piece and quiet while they try to isolate the problem. This method needs to rely upon as little software as possible, since they might want to retire the alarm while the software system is insane. A pushbutton driving a hardware circuit would be the best solution. Now that they have a method of silencing the alarms, *Don't Let Them Forget* that the alarm happened and there is something needing their attention.

Rationale: Audible Alarms provide an interface to the human operators that parallels the interface provided by the IO system. Since it is in parallel, it will provide the operators with information that they might not get through the IO system in a timely manner.

The alarm system should have a low probability of failure. The hardware should have few failure points and 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.

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

Reference: [HJJS+], p. 1051.

Alarm Grid

Keywords: Location, Emergency

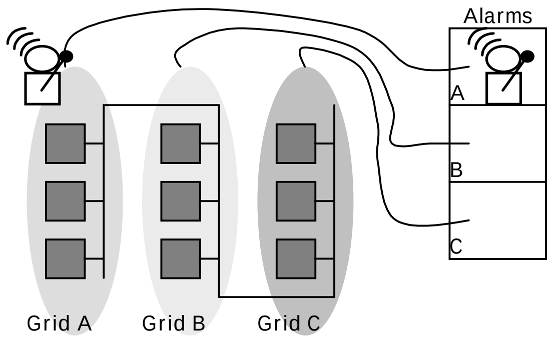
Problem: Workers need to know where to look for the problems, especially hardware problems.

Context: *Audible Alarms* and IO messages report emergency situations. *People Know Best* [PLOPD-2] requires support systems monitoring the system have full information about its state. 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 availability. There might be many alarms asserted simultaneously. For example when a piece of hardware that terminates trunks fails, the system will want to report the hardware failure as well as the failures of all of the trunks that it contains.

Forces:

- Simultaneous alarms confuse workers making them unsure where they should look first.
- The IO system has a lower processing priority than the subsystems that ensure that the system is sane and functioning normally. We cannot count on the IO system to report routine messages quickly during times of crisis. This can delay processing for a long time in a system that is trying to recover from catastrophes.
- *Audible Alarms* are presented by the *IO Gatekeeper* as soon as they are detected.

Solution: Divide the office into smaller "grids" that point to location of the error. Tie alarm circuits together to report to the main office alarm panel. When an alarmable situation occurs within this alarm "grid", alert all concerned personnel via *Audible Alarms* that there is a problem. Also provide visual indicators in the form of colored lights on the ends of frames within a grid to show that abnormal conditions exist somewhere in the grid.



Resulting Context: Alarm grids allow problem reports to selected communities of interest. This is similar to the logical channel concept of *Mind Your Own Business*.

Alarm grids can be arranged hierarchically within an office. This allows refinement of reporting and the potential to tie together several grids to report larger summary events during off-hours, when fewer people are watching the system.

Reference: [HJJS+], p. 1051.

Office Alarms

Keywords: Location, Emergency

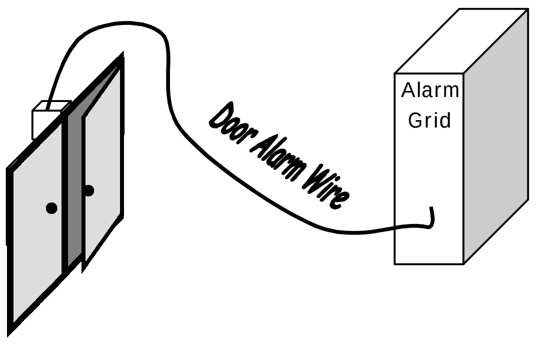
Problem: How can problems that are unique to a particular field site be integrated into the pre-defined classification of messages?

Context: Both visual and audio alarms show emergency information organized by *Alarm Grid*. Different sites might have custom alarming requirements, such as door locks, specialized A/C, coffeepots, related systems, etc.

Forces:

- A separate, parallel interface for site specific alarms would introduce confusion, since its user interface would probably not be identical to the standard.
- 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.
- Humans are confused if some important things are not alarmed while seemingly less important things have pre-defined alarms associated with them.
- The designers and equipment manufacturers can't foresee every unique office configuration containing conditions that need to be alarmed. The switch-owning customers probably don't want to pay for such thoroughness either.
- Good user interface design has similar things behaving in similar manners.

Solution: Design the alarm system to allow easy insertion of new office specific alarms. This is an example of the *Decorator* pattern from [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. Switch owners may tailor this part of the system to their own corporate operating policies.

Don't Let Them Forget

Keywords: Importance, Emergency

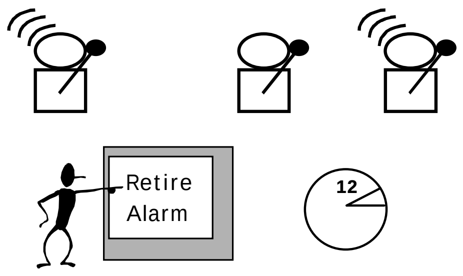
Problem: For how long should the system honor a human's request to silence the alarms?

Context: *Audible Alarms* 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, so a method of silencing, or "retiring", the alarm is provided.

Forces:

- The choices include reasserting alarms immediately, or keeping them quiet for some pre-determined length of time.
- 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 (MTTR).
- 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: Act on all requests to retire an alarm. But 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 the alarms will be quieted.

Related Pattern: This pattern deals with alarmed situations, whereas *George Washington Is Still Dead* is about slightly less critical information.

String A Wire

Keywords: Emergency

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

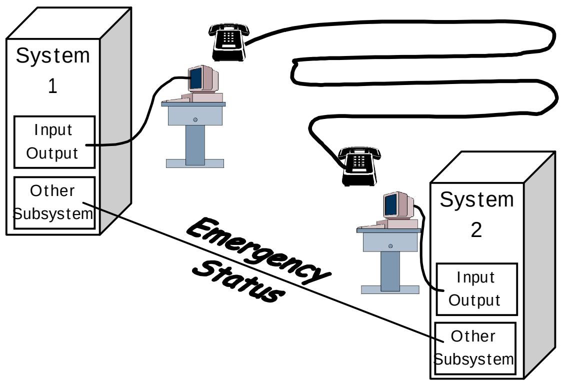
Context: Sometimes the *IO Gatekeeper* is too slow, or is in a partial capability mode, or the system can't afford the resources to send a message to a nearby system.

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.

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 other system's language.

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 IO 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 *Raw IO*.

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

Raw IO

Keywords: Emergency

Problem: How do we provide IO during those times when the IO system is unavailable, for example during system initialization, or times of emergency.

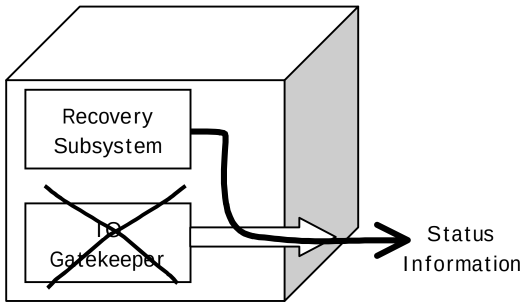
Context: Sometimes the *IO Gatekeeper* is too slow, or is in a partial capability mode, or the system can't afford the resources to do IO. These might be just the situations where people need to be informed about system state. *String A Wire* cannot provide enough information in these circumstances.

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 during times of crisis, there is nothing to help an expert user 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 IO subsystem loose during periods of system recovery.
- ♦ The IO system is large with many more interfaces than the recovery system, so a large amount of code must be available during recovery for IO to work.

Solution: Display the output out via brute force mechanisms such as writing directly to a logical channel and avoiding the IO system. This should be limited to recovery periods only.

The periods during which this raw IO mechanism operates should be limited. The system should try to restore the IO system as soon as possible.



Resulting Context: During critical periods, the safe course is chosen and the recovery programs administer IO rather than relying on the IO system.

Pattern Thumbnails

Input Output Patterns:

Pattern Name	*Pattern Intent*
Alarm Grid	Group alarms into grids to help the workers identify problems.
Audible Alarms	Sound audible alarms to alert office personnel of problems.
Beltline Terminal	Allow workers to take their terminals with them.
Don’t Let Them Forget	Reassert alarms when necessary, purposely forgetting requests to retire the alarms.
Five Minutes of No Escalation Messages	Don't confuse craft with too frequent messages.
George Washington is Still Dead	Issue state change messages only when the state changes, not to remind about the current state..
IO Gatekeeper	Put one process in charge of IO for the system.
IO Triage	Add a priority tag to each output message and sort the output using them.
Mind Your Own Business	Only send output to concerned terminal groups (logical channels).
MML	Use a standardized IO language.
Office Alarms	Allow the alarm system to be customizable with site specific alarms.
Pseudo-IO	Provide for internal subsystems to add IO to the stream.
Raw I/O	Provide a way for recovery systems to bypass the *IO Gatekeeper*.
Shut Up And Listen	Give human input/output messages a high priority.
String A Wire	Provide a system to system emergency information channel.
The Bottom Line	Issue messages to summarize a number of events rather than for each of many events.
Timestamp	Add a timestamp and/or a sequence number to each output message.
Who Asked?	Return output only to the logical channel/terminal that requested it.

Referenced Patterns:

Pattern Name	*Pattern Intent*	*PLOPD-2 Page*
Minimize Human Intervention	Give machine enough smarts to not require human intervention.	551-552
People Know Best	Assume humans know more than the machine.	552-553

Riding Over Transients	Give transients time to clear up on their own.	554-555

Acknowledgments:

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

- Ralph Jones. 1937-1998. Inventor of the *Alarm Grid* concept and also the concept of using message tags as the key to sound an *Audible Alarm*.
- Reviewers within Lucent Technologies: Rick Rockershousen who reviewed 1A ESS™ content; Glen Moore who was one of the original developers of the 4ESS Switch IO system; Chuck Borcher and Deatrice Childs who participated in a review of *Alarm Grid* and *Office Alarms*; and Juel Ulven for reviewing the alarm patterns.
- David DeLano, our PloP-98 shepherd provided invaluable comments to improve these patterns.
- Both the participants in a TelePLoP sponsored workshop and the Zen View writers' workshop group at PloP-98 provided invaluable assistance.

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

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.