Event Notification Patterns for Distributed Machine Control Systems

Veli-Pekka Eloranta *
{firstname.lastname}@tut.fi

Department of Software Systems
Tampere University of Technology
Finland

1 Introduction

In this paper we will present three design patterns for distributed machine control system. By distributed machine control system we mean a system consisting of multiple embedded controllers which communicate with each other. These actuators control, monitor and assist the operating of a work machine or an automation process. Typically these kinds of systems have strict real-time requirements which set certain limitations for their architecture. Other key drivers of these systems are distribution, fault tolerance, safety and long life cycle.

Control systems have become large and complex systems where the software architecture plays a central role in the overall quality of the machines. However, there is not so much literature on the specific aspects of these systems. Therefore, we feel that there is a need for a pattern language to ease the burden of designing such systems.

Three patterns for this paper are a part of larger body of literature and these were selected as they are quite thightly coupled and form a whole that is part of even a bigger whole. In this paper, we will provide patlets of the referenced patterns in the pattern language to help the reader to understand these three patterns better. Other patterns in the language are not currently publicly available.

Patterns in this paper are mined during 2008-2011 in an industrial context. Initial drafts of the patterns were found during architecture evaluations in the Finnish machine industry. Then the initial versions of the patterns were given to domain experts for review. After they had reviewed the patterns, we interviewed them to gain more insights to the domain and the patterns. Finally, the current version of the patterns were written.

2 The context of the patterns

The three patterns presented in this paper are a part of larger body of literature. These patterns belong to a pattern language for building software architecture of distributed machine control systems. The whole language consists of 70 patterns and it is illustrated in Fig. 1.

The semantics of an arrow pointing from pattern A to pattern B in our language is "pattern B refines pattern A". This means that if the architect has solved some design problems with pattern A, the design context is now compatible with the required context of pattern B. The designer might look at all refining patterns if there are still some unsolved problems in the context.

The root pattern of the language is CONTROL SYSTEM which describes why to have control system in the work machine. Patterns presented in this paper refine CONTROL SYSTEM pattern by providing means to handle events occurring in the system. However, in a typical situation ISOLATE FUNCTIONALITIES is also applied and then notifications are provided over bus.

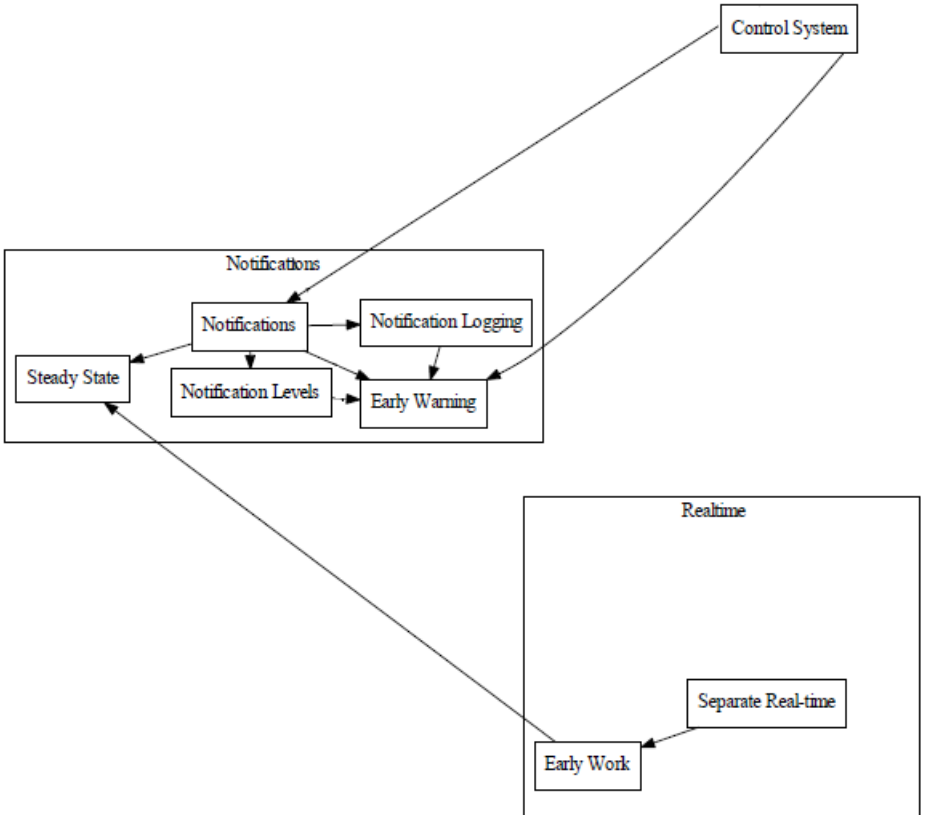


Fig. 1. Pattern language for machine control systems.

Other patterns are referenced in this paper using SMALL CAPS. Patlets of the patterns that are referenced in this paper are presented in Table 1.

Table 1. Patlets of the patterns that are referenced but not presented in this paper

Pattern Name	**Description**
COMMON LOOK'N'FEEL	How to improve learnability and effectiveness of the user interface? Make all user interface screens and notifications to have unified layout and colouring. Each UI element is presented in the same way and place independent of the view.
CONTROL SYSTEM	How to implement a work machine that offers interoperability between systems and is highly operable with good performance? Implement control system software that controls the machine and can communicate with other machines and systems.
CONTROL SYSTEM VARIANCE	How to build a product consisting of specific software components from a product platform component library? Create a unified way to describe system configurations in a configuration file. This file is used to select the required software components and their configuration parameters for the desired system setup.
GLOBAL TIME	How to prevent different nodes on the system from getting out of sync? Use external clock, e.g GPS or atom clock, to synchronize all the nodes.
ISOLATE FUNCTIONALITIES	What is a reasonable way to create an embedded control system for a large machine? Distribute the system into subsystems according to their functionalities. Interconnect these subsystems with the bus. Use multiple interconnections between subsystems if necessary.
MESSAGE QUEUE	How do you give time for both ends of a message channel to process all messages? Add a queue for receiving and sending messages to each node. Implement a mechanism to put messages in the queue and that will send messages from the queue. The same mechanism can read messages from the bus and add them to the received messages queue.
OPERATING MODES	In order to make sure that only the functionalities which are required can be used in current operating context, design system so, that it consists of multiple functional modes. These modes correspond to certain operating contexts. The mode only allows usage of those operations that are sensible for its operating context.
PRIORITIZED MESSAGES	How to ensure that important messages get handled before other less important messages? The message types are prioritized according to their urgency and separate MESSAGE QUEUEs are implemented for each priority.
SAFE STATE	How to minimize the possibility that operator, machine or surroundings are harmed when some part of the machine malfunctions? Design a safe state that can be entered in case of a malfunction in order to prevent the harm that machine can cause. The safe state is device and functionality dependent and it is not necessarily the same as unpowered state.
STEADY STATE	How to handle transient faults that might be generated during the system start-up? Define a time interval in which the system must reach Steady State. Being in Steady State means that the system is ready for normal operations. Before the Steady State is reached, the system can generate errorneous alarms that can be neglected.
VARIABLE MANAGER	How can you efficiently share system wide information in the distributed embedded system? For each node, add a component, which contains all the information that is relevant to operation of the corresponding node. This information is presented as state variables. The value of a variable is updated every time when a message containing the information is received.
VECTOR CLOCK FOR MESSAGES	How to find out the order of events in distributed system? Give every event a vector clock timestamp. The timestamp consist of separate message counter values for every node. The message counter of a node is updated when a message containing vector clock timestamp with larger value is received.

3 Patterns

In this section, three patterns for delivering and logging events in the distributed machine control system is presented. Patterns are presented in Alexandrian form and they contain also the known usage of the pattern.

3.1 Notifications

.. you have distributed CONTROL SYSTEM and SEPARATE REAL-TIME has been applied to divide the system into machine control level and operator level. HMI has been used to create graphical user interface for the system. There are a lot of messages sent through the bus. However, only subset of the messages are interesting from the operator point of view. You need a way to inform the machine operator of events or state changes taking place in the system. For example, events can be faults occurring in the system or state change such as engaging parking brake. There should be a way to distinguish these event related messages from other messages transffered in the bus as some of the event messages might need urgent attention of the machine operator.

How to inform operator or communicate to subsystems that something worth of noticing has occured in the control system?

Event messages should be easily identifiable amongst the other traffic on the bus. This makes the testing and debugging of the system easier.

It should be relatively easy to add new events, modify or remove events in the system. Especially, if CONTROL SYSTEM VARIANCE is used to create a support for accessory devices , new events are likely to be added when the accessory devices are taken in use.

Events occurring in the system should be traceable. Therefore, it should be relatively easy to detect from where the event has originated. Furthermore, it should be possible to find the cause of the event easily.

Communication of events should be fast in the system. Other traffic on the bus should not delay the delivery of event information even when the load of the bus is high.

Event information should be human readable. However, transferring a human readable text on the bus, increases the bus load.

In order to make the system safe and robust and at the same time easy to operate and understand, even handling should be consistent through the system.

Therefore:

Communicate noteworthy or alarming events and state changes in the system using notifications. Additionally, implement notification service on operator level. This service is used to create, handle and deliver notifications.

⋆ ⋆ ⋆

All the notifications created on operator level should be triggered using the notification service. The notification service is implemented on operator level and it can use VARIABLE MANAGER to retrieve notifications from the nodes of the system. If VARIABLE MANAGER is not used, nodes must communicate using notifications messages. Notification service forwards notifications to components which have been configured to be interested in certain notification. Furthermore, notification service can be used to forward notifications to other machine control level nodes. For example, if node A detects a fault and creates a notification. This event is delivered to notification service which then notifies other machine control level nodes interested in this event. Fig. 3.3 illustrates typical structure of the system with notification service in place.

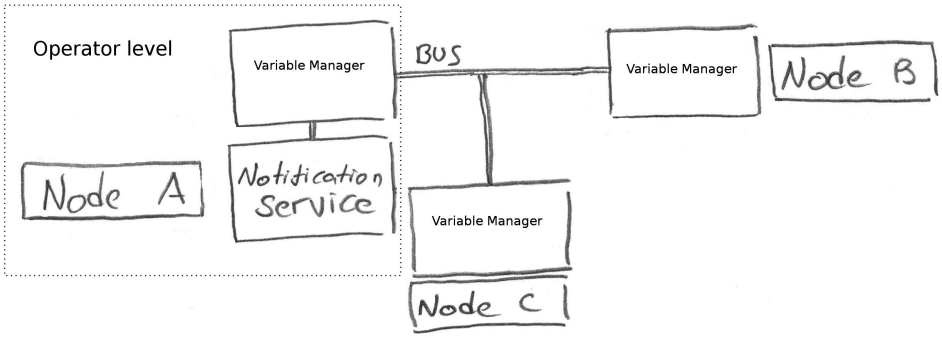


Fig. 2. Typical system structure when the notification service is used on operator level.

Using notification service eases decoupling of the application code from the notification processing. Notification service should have an interface for propagating notification, receiving an notification and setting the state of the notifcation. In addition, there should be a system-wide configuration file, e.g in XML that notification service reads . The configuration file contains notification IDs for each event and list of nodes which are interested in each notification. The node itself should take care of the logic when the notification is triggered and informed to notification service.

User interface should have a mechanism to show notifications on screen. When the notification service delivers notification to user interface component, the user interface should show it to the machine operator. Typically, this is shown as a pop-up window. The machine operator can then set off this notification by click OK on the pop-up window. This should be informed back to the notification service, so that the notification service can change the state of the notification.

Notifications can be delivered as certain kinds of messages that are sent on the bus (if VARIABLE MANAGER is not used). If VARIABLE MANAGER is in use, the same message type can be used on operator level when notification service delivers events to applications. Notifications contain information consisting of notification identifier (ID), notification state, notification data and optional description text. The message structure is illustrated in Fig. 3.

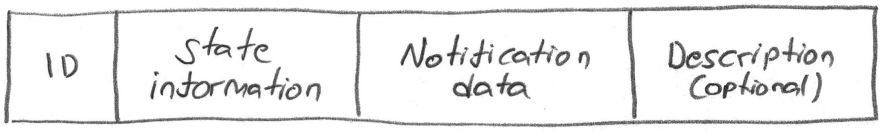


Fig. 3. Typical structure of the notification message.

Each notification has its own notification type identifier (ID). By this ID, the notification can be identified. For example, ID 501 could mean that oil pressure is low and ID 502 that oil pressure has reached critical level. Typically this is a number which has some semantics. For example, the first two (or three) numbers may pin-point the origin of the notification, i.e. which sensor or actuator caused the nofitication. Notification service can also use this ID to retrieve the human readable message to be shown with the notification data. Take into account, that notification messages should be internationalizable. Furthermore, in some cases, it may make sense to put unique notification ID to the notification message as well. If this is the case, the unique ID is a running number. For example, first message low oil pressure notification sent has ID 1, the second ID 2, etc.

As mentioned earlier, notifications have states. Typically these states are "new (active)", "Set off (active)", "Set off (inactive)", "Normal". This means that when a notification is triggered, it is new and active. In this case, the notification service delivers the message also to the user interface of the machine and it is shown to the user. When operator sets alarm off by clicking OK from the user interface, the state of the alarm is changed to "Set off (active)" or "Set off (inactive). Difference between these two set off states is that if the cause of the notification is not removed, i.e. fault is still on, the notification is still active. For example, if oil pressure is too low and machine operator clicks OK the notification is set off but it is still active as the oil pressure is still too low. If this is the case, the notification might be given again after a certain time period has elapsed. Now if operator adds oil, the alarm state is "Set off (inactive)" as the alarm does not require attention anymore. Probably the alarm state is then during the next reboot set to "Normal" which is the default value. In addition, notifications may have time to live, so their state may change over time, even without the operator intervention. When the notification state is changed, the notification service informs all nodes interested in the notification, that the state has changed. Furthermore, the source of the notification is also informed about the state change and it can do necessary actions.

Notification data contains additional information about the event that has happened. The notification sender attachs this to the notification message, so it can be shown to the operator of the machine, or written in the logs. For example, notification informing that the oil pressure is too low, could contain current oil pressure value in notification data field of the message.

The description text is not typically included in the message as it would be transferred over the bus and it would increase the bus load. Therefore, description texts are mapped to notification IDs in notification service. IDs are then used by the notification service to show the corresponding description text.

Notifications can also have priorities. If multiple notifications are active simultaneously, they should be shown in the order of importance to the machine operator. One approach to this problem is described in NOTIFICATION LEVELS also PRIORITIZED MESSAGES can be used

to define priorities between notifications. There are also times when notification occurred can be ignored, for example during start up as described in STEADY STATE pattern.

⋆ ⋆ ⋆

Exceptional and noteworthy events or state changes occuring in the system can be informed system wide in a unified way. It is rather easy to add, modify or remove events using the configuration file that the notification service reads.

Notifications contain the notification ID which identifies the notification type. It delivers information to the notification receiver (cause, origin, etc). Human readable description can be found by using notification ID so the human readable text does not need to be transferred over the bus. Therefore, notifications causes only minimal amount extraneous bus traffic.

Notification states make it possible to have notifications active, but in the background. This makes it possible to show the same notification again if the problem has not been removed. This makes the machine more reliable and easier to operate.

Notification priorities may increase the amount of configuring required. It can make the system more complex and harder to maintain.

Notifications in distributed system may cause some additional bus traffic as the number of sent messages increases. Therefore notifications may somewhat decrease the system performance if throughput of the bus is already a problem. However, it might be the case that the information would have to be transferred anyway.

It requires manual work to configure notifications for each node. Notification IDs also need to be documented, so it causes some extra work.

⋆ ⋆ ⋆

In a forest harvester, the harvester head uses notification service to inform the rest of the system about the exceptional events taking place in the harverster head. Marker colour (that is used to mark the cut trees) runs out. The marker's control application generates puts a fault on in the VARIABLE MANAGER. The notification service notices this and creates a notification message containing the notification ID that corresponds to the occurred event. In addition, it adds the marker colour container current filling level to notification data and sets the notification state to new (active). Then the notification is sent to bus from where the nodes interested in this information reads it. Operator level PC reads the message and informs the UI application that marker colour container empty notification has been received. The user interface application then shows a pop-up on the screen informing the machine operator about the situation. Operator clicks OK on the pop up and it is closed. The user interface application calls the interface of the notification service to set the notification state to set off (active). The operator can continue working but the marker colour can not be used anymore. Once the marker colour container is filled up again, the node controlling the marker colour sends a notification message informing that the state of marker colour notification is not Set off (inactive). This message is again sent using marker colour controller node's notification

service. Once the system is rebooted next time the state of this notification is then reset to Normal unless the marker colour runs out again.

3.2 Notification levels

.. you have distributed CONTROL SYSTEM and NOTIFICATIONS are used to communicate that something worth of noticing has occurred in the system. However, there are different kind of events occurring, e.g. operating mode changes, faults, errors and state changes. It could be useful to have more granularity in notifications as different events have different consequences. Sometimes you need to inform the operator that a process, the machine operator has started is completed, e.g. once the automation sequence of drill placement has completed. Furthermore, there might be a need to inform the machine operator about warnings, e.g. oil pressure low. This may not require immediate actions, but should be brought to machine operator's attention. In addition, there might be some faults, e.g. a node notices that a sensor is broken down and needs to inform this fault. The sensor fault may disable some functionality of the machine as a consequence. In general, different events have consequences of different severity.

How to group different kinds of notifications according to their severity?

Different kinds of events may have different consequences. It should be easy to distinct between these different event types and to handle all consequences in a uniform way.

Some events taking place in the system, might freeze all operations whereas some events are supposed to be taking place. Events that can make the system stop functioning should be handled first. Genereally, events should be processed in the order of severity of the consequences.

Sometimes if an event is not occurring in the system, it might be a sign of a fault or failure. It should be easy to verity that an event is taking place when it is supposed to.

Operator should be able to easily see what kind of event has occurred. It should be possible to detect the event type just by taking a glance at the user interface, not reading the description of the event.

Therefore:

Attach level information to the notifications. Typical notification levels are notices, warnings and faults. Each notification level has its own way to remedy the situation and each level is presented to machine operator in a certain way.

⋆ ⋆ ⋆

All notification levels have their own severity and ways to react to the occurred event. Notice notification level consists of events that are supposed to happen in the system. For example, when the machine operator starts system self-diagnostics the process will run on its own.

Now, when this process has finished a notice level notification should be triggered. This notification informs the operator that the self-diagnostics process has finished. The user interface will then show this notice notification to the machine operator. When the operator clicks OK in the notification dialog, it may propagate a mode change in the system, e.g from diagnostic mode to normal operating mode (see OPERATING MODES for more details). Warnings are notifications that tell the machine operator that something needing attention has occurred. For example, if oil pressure of the motor runs too low, a warning is given. Faults are used to inform that some part of the system is malfunctioning. For example, if boom positioning sensor has broken down, the fault notification is sent by the boom controller. Faults can make the system to enter SAFE STATE automatically when they occur or if SAFE STATE is not used, faults may make the machine otherwise inoperable. The structure of notification message when using notitication levels is depicted in Fig. 4.

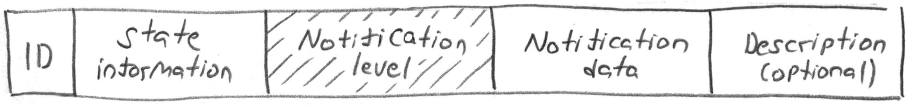


Fig. 4. Typical structure of the notification message including notification level.

Basically, the notification levels are added to the notification message as a new field. Sometimes also the first (or some other) digit of the notification ID can be used to express the notification level, e.g. 1 for notices, 2 for warnings and so on. Notification service can use this level information to determine the priority of messages. Typically faults are most urgent, then warnings and last notifications. However, one might also use different granularity for the notifications, e.g. operating event, notice, warning, fault and in this case the priority may be different. All notifications have states (as desribed in NOTIFICATIONS), when notifications are enhanced with levels, all notification states are not utilized. For example, for notices, states "new (active)" and "normal" are typically enough as there is no need to have other states. It is just enough that the machine operator is informed about the event, no need to remind him or her later.

Usage of notification levels is carried out through the notification service. Notification service is still used to trigger, deliver and receive notifications as it was described in NOTIFICATIONS pattern. The interface might be slightly changed to support different notication levels, i.e. adding a parameter to functions or so.

Notifications should also have priorities and by using levels the priority is easy to define. If multiple notifications are active simultaneously, they should be shown in the order of importance to the machine operator. Within a single notification level, e.g. fault, it might be necessary to prioritize notifications. If this is the case, in addition to the notification level, there should be a urgency field added to the notification message. These urgencies can then be used to determine the priority within a notification level.

Different notification levels should be shown in a distinct way in the user interface. Each notification level should be made visually different, notices can have a different icon than warnings and faults and so on. Additionally, the pop up can be one of different color for each level. Furthermore, the most severe notification types may also trigger a warning sound. This

makes it easy for the machine operator to distinct which kind of notification was triggered. One might want to take a look also at COMMON LOOK-N-FEEL pattern for more advise on the user interface design.

When using notification levels it might be necessary to use PRIORITIZED MESSAGES pattern within the notification service to implement the priority order of different notification levels.

⋆ ⋆ ⋆

Different kinds of events can be distinguished easily using separate notification levels. This makes, for example, UI application development easier as each level has its own way to be shown in the UI. Additionally, user can distinguish different notification types just by taking a glance to the UI.

Notification levels makes it possible to remedy a certain kind of events in their specific way. For example, when a fault occurs, the machine can always enter SAFE STATE automatically or stop all operations.

When using notification levels, it is easy to create a priority order of notifications. This is especially useful if multiple notifications are active simultaneously. However, within a notification level, more fine-grained approach might be needed.

Notification levels are relatively easy to add, modify and remove. This makes it easier to modify the system.

Notification levels requires additional manual configuring of the system. Each level has to be defined and configured which notification resides on which level. Configuration might become quite complex, when notification levels changes between software versions.

Within a notification level more fine-grained priority order might be required. This needs additional configuring in the system and may require new field in the notification message.

Deducting notification states becomes more complex as in some notification levels all states are not necessarily used.

⋆ ⋆ ⋆

In forest harvester the machine operator activates the tree cutting by pressing a button in the hand panel. Once the machine has finished cutting the tree, the harverster head updates its status variable in VARIABLE MANAGERinforming the notification service that the cutting operation has finished. Notification service creates a notification which level is notice as this event does not require any human intervention. The notification message is sent to the bus and the cabin computer running the user interface receives this message. Once received, the notification triggers a state change in the system as the system is now ready for the next operation. Additionally, the cabin computer lights up a led in the dashboard, informing the machine operator that (s)he can start feeding the log through the delimbing knives. In another case, the machine operator has left the cabin door open and tries to start to cut a tree. However, once the operator presses the cut button, the cabin computer generates a warning level notification

using the notification service. This notification is delivered via bus to harvester head which does not start the cutting operation. Cabin computer shows a warning to the user, that the cabin door is open and it should be closed for safety reasons before the cutting operation can commence. If the user closes the warning pop up and tries again to start the cutting operation, the warning is shown again. When the door is closed, the status is updated to normal and again a notification is generated using the notification service. Now, once the harvester head has received this notification, it changes its mode and can start to cut the tree. In third case, the machine operator has started the sawing process of a log and the saw chain breaks. Harverster head generates a fault notification and sends it to the bus. All nodes go to SAFE STATE which means they stop the current operation. A fault is shown in the user interface in the cabin to the machine operator. Now machine operator need to take the necessary actions to change the saw chain. Once she has done that, she must restart the system to disable the fault notification. At the start-up the machine runs diagnostics that determine if the saw is OK and as it notices that the saw is OK, it resets the notification state to "normal".

3.3 Notification logging

... you have distributed CONTROL SYSTEM where noteworthy or suspicious events are communicated using NOTIFICATIONS. There is a notification service implemented to handle notifications. The fast-paced real-time environment makes it hard to detect the source of a fault or error while the machine is operating. Therefore, machine operator or maintenance person needs to search for the faulty component afterwards once the machine is stopped. Furthermore, the machine manufacturer would like to gather information of typical faults for their own analysis so that the same faults typical to the model can be avoided in the next version. In addition, it would be beneficial during the yearly maintenance of the machine to find out what kind of faults and errors have occurred in the system.

How to find out later on what notifications have occured in the system?

In many cases, when a fault or error occurs, it can not be deducted at the moment what was the root cause of the fault as the environment is real time. It might seem that a controller is faulty, but it might be caused by another controller or broken sensor. That's why it should be possible to find out later on what exactly has happened in the system.

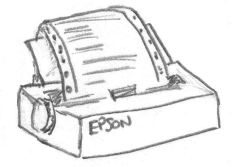
The order of occurred events is important. It should be possible to know reliably the order of the events.

Sometimes it is necessary to know the exact time when an event has occurred.

Once a fault has occurred, it should be easy and fast for maintenance person to find out the root cause of the problem.

Diagnostics during the yearly maintenance of the machine should be able to tell what kind of errors have happened in the system. There should be a way to produce a report of the events that have occurred in the system.

It is valuable for research and development to know the typical faults and errors that are taking place in the system. In this way, the next version of the system can be developed so that these typical faults can be avoided.



Therefore:

Create a logging mechanism that logs selected notifications that occurs in the system. Add timestamps and notification source to all logged notifications. In this way, order of notifications can be deducted.

Logging mechanism should be implemented within the notification service. When a notification is triggered, it is also logged. A log entry should contain the notification ID, notification

data and timestamp. Log entries should be written into a file at some point of time. However, if log entries are written to a file as they emerge, it may decrease system performance. Therefore, it is advisable to keep log entries in memory and write them to a file periodically. One should also take into account the case of a sudden power failure. In that case, the log entries in memory might get lost. So it is a trade-off between reliability and traceability. Saving interval could be determined using the criticality of the availability of log entries. Most critical information should probably be written directly to a log file.

Using timestamps for the log entries can be troublesome. It might be the case that the nodes in the system do not share common time. If this is the case, the order of different notifications on separate nodes, might be impossible. In that case one might consider using VECTOR CLOCK FOR MESSAGES or GLOBAL TIME to solve this issue. If real-time is used to determine the order of log entries, the timestamp should use accuracy of 1 ms.

One question is that should the logging be centralized or distributed. Typically, the logging is implemented in distributed fashion as notification service is. This prevents the logging service from producing extra load on the bus. This approach is illustrated in Fig. 5. Often diagnostics tools and analysis requires that all events are gathered to a single place. This is typically carried out so that when a diagnostic service is started, it gathers all logs from nodes and merges them to a single log or handles separate logs as they would be just one log file. This is a good option as diagnostics are usually run when the machine is not operating and therefore the bus load is not high. If multiple log files are merged, the original logs should be left intact and the merged log file should contain the origin of the log entry.

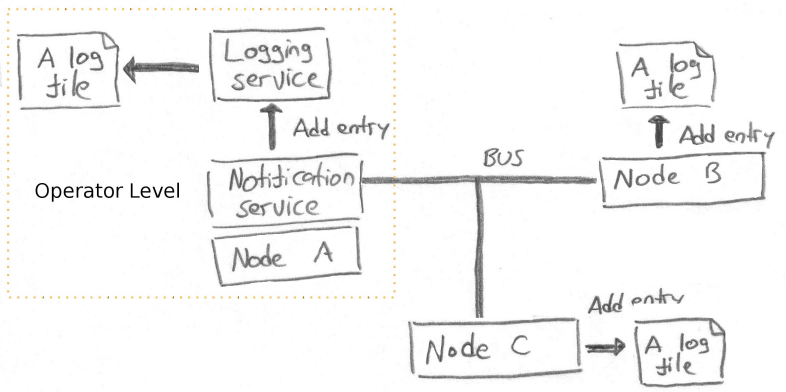


Fig. 5. Typical structure of the notification logging service.

Over time log files can become rather large and it might take a lot of disk space to store them. That's why they should be cleared once in a while. For example, log files can be cleared once the system is updated or during the yearly maintenance. It pretty much depends on the case when this can be done. Sometimes it might be reasonable to clear log files during the reboot.

If necessary, one might consider also filtering notifications that are logged. It might be the case that all notifications are not relevant from the logging point of view. If NOTIFICATION LEVELS are used, it can be easy to determine which levels should be logged and which not. If notifications also have priorities attached to them, this information can be utilized in the log filtering. It might also be a good idea to have own log file for each notification level.

⋆ ⋆ ⋆

Notifications taking place in the system can be recorded in the log files for later examination. This makes it possible to create diagnostic reports later on.

The order of notifications occurred can be determined later on by gathering all log files. This makes locating the fault easier for the maintenance persons.

May decrease system performance as a log entry has to be created for each notification sent.

May increase memory consumption of a node if multiple log entries are kept in memory before writing them to a file.

It might be impossible to implement logging service in low end nodes.

⋆ ⋆ ⋆

In mining drill notification logging is used in each node. Intelligent drill rotation sensor notices that the drill is stuck and sends a notification to the drill controller. This notification is logged to the sensor node. The drill controller stops the drill and sends a notification forward that a fault has occurred. The system is stopped. Now the operator accesses the system diagnostics to see if the system is malfunctioning or if the drill is really stuck in the drilling hole. The diagnostics application gathers log files from all nodes and constitutes a report. From the report the operator can see that the sensor has sent a notification before the drill controller. This indicates that the system is working correctly, because it could be the case that the drilling controller is malfunctioning and would send errorenous notifications.

4 Acknowledgements

I want to thank my colleagues Ville Reijonen, Marko Leppänen and Johannes Koskinen for their help. Especially I want to thank all industrial partners who have made it possible to mine this patterns from their systems: Metso Automation, Kone, Sandvik Mining and Construction, John Deere, Areva T&D. Thanks also to VikingPLoP 2012 participants for the valuable feedback.