Patterns for Distributed Machine Control System Modes

Marko Leppänen
{firstname.lastname}@tut.fi

Department of Software Systems
Tampere University of Technology
Finland

1 Introduction

In this paper we will present four patterns for distributed machine control systems. A distributed machine control system is a software system that is specifically designed to control a certain hardware system. This special hardware in turn operates some work machine, which can be a forest harvester, a drilling machine, elevator system etc. or some process automation system. Some of the key attributes of such software systems are the close relation to the hardware, real time requirements, safety issues, fault tolerance, high availability and long life cycle. Distribution plays a major part in the control systems as the different parts of the machine are physically far from each other and they must communicate with each other in order to perform their functionalities.

The patterns in this paper are part of a larger collection which was collected during years 2008-2011 in collaboration with industrial companies. Some real products by these companies were inspected during architectural evaluations and whenever a pattern idea was met, the initial pattern drafts were written down. These draft patterns were then reviewed by industrial experts, who had design experience from such systems. After these additional insights, the current patterns were written.

The published patterns are a part of a larger body of literature, which is not yet publicly available. All these patterns together form a pattern language, which consists of more than 70 patterns at the moment. The current pattern language is partially presented in a pattern graph to give reader an idea of how these selected patterns fit in the language. These four patterns are closely related in the pattern language and therefore are ideal to be submitted together as a whole. All the pattern names in the following sections are written in SMALL CAPS.

2 Patterns

In this section, a set of four patterns is presented. The patterns together form a branch in the pattern graph in Fig. 1. The pattern graph is read so, that a pattern is a box in the graph and a arrow presents a certain connection between the patterns. The arrow means that the pattern from which the arrow emerges is refined by the pattern that the arrow points to. In other words, if the designed system still has some unresolved problems even after some pattern is applied, the designer can look to the refining patterns for yet another solution.

The main focus of these patterns is in the mode-based behavior of a control system. The patterns refine each other making the original design more elaborate. The CONTROL SYSTEM pattern which is the root of this branch and is referenced in the following patterns. The CONTROL SYSTEM is the central pattern in designing these systems. It presents the first design problem the system architect will face: Is a control system needed in this context? The pattern is introduced in patlet form in Table 1.

Table 1: Patlets

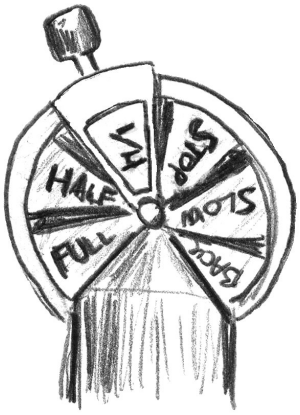
Pattern Name	**Description**
CONTROL SYSTEM	**How to implement a work machine that offers interoperability between systems and is highly operable with good performance? Therefore: Implement control system software that controls the machine and can communicate with other machines and systems.**

Fig. 1. A part of the pattern language for distributed machine control systems in a graph form.

2.1 Operating Mode

...there is a distributed CONTROL SYSTEM, that is normally used for a productive work. However, there are multiple situations when the system is not in the active use, but is rather under some maintenance operations or has encountered a fault. These special situations must be handled somewhat differently than the normal production operation. For example, during an update, all actuators should be turned off for safety reasons and only the bus and nodes should be active and the main user interface shows information about the update progress.

How to make sure that only the functionalities which are required can be used in current operating context?



As the control system is a complex system which has multiple different usages and functions. These rise from the fact that the system has a plethora of different, often even contracting requirements. The system may even have many different user roles. Therefore, the system may have drastically different use cases which should be handled without possibly dangerous consequences. For example, a harvester is used to fell trees in the forest when the harvester head malfunctions. When a maintenance person arrives to the site, it should be easy to inspect the work machine so that all available information is at hand. If the maintenance person wishes to test the system, it should not cause risky situations to the operator or the environment.

For a system developer, it is cumbersome to build functions that depend on many different flags or conditions. For example, if all the system functions must ask if the parking brake is engaged, the bus load becomes too big and real-time requirements are more difficult to fulfill.

Therefore:
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.

* * *

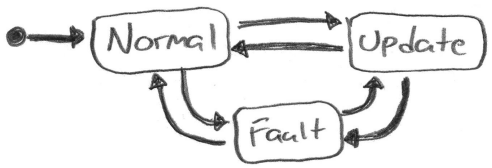
There may be a mode for normal operation, maintenance mode, fault handling mode, update mode, parking mode and so on. These modes may be nested in a hierarchical way. Every mode may have some submodes, but it is important to keep the hierarchy simple enough. A good rule of thumb would be to use only three levels of mode nesting. One of the highest level modes should the default mode when the system starts up and all other modes need some special action to be entered.

Modes have a strong connection to state chart states, so the system designer may draw some state diagrams when documenting and designing the system modality. There are even some pragmatic approaches that rely fully on the state paradigm when designing and implementing software systems [1]. These approaches suit well systems which have strong modality. Distributed machine control systems often exhibit these attributes.

Modes can be designed for completely different use cases (maintenance mode vs. normal mode) or then they can implement some functionality. For example, if the parking brake is engaged, it may trigger a mode change that affects many other functionalities. It is easier to design the subsystems so that they

detect the global mode and use this information to deduce if some action is allowed.

Fault handling must be thought out in early stages of the system design. It is very hard to add fault handling afterwards, when the business logic is already designed and implemented. This can be remedied by using a separate fault handling mode, that is entered when a fault is encountered. In this way, the normal operation is halted until the failure is cleared out.



* * *

The system's main functionality is easier to understand as the modes isolate the different usages in a compact way and it is easy to validate if the system is prepared for exceptional use cases.

The modes encourage piece-meal growth of the system architecture as the main functionality can be implemented separate from the more elaborate use cases.

However, there are some functionalities which are needed to be implemented in multiple modes. This may clutter the design as the module may have to check in which mode it is used in.

* * *

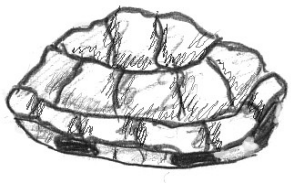
A harvester has a parking brake that prevents it from accidentally moving. When the parking brake is engaged, the harvester can not move, but on the other hand, opening the cabin door does not cause an alarm as it would normally do. The parking brake functionality is implemented as a submode to the normal operating mode. Thus all the modules may just check the mode and perform their operations accordingly.

2.2 Safe State

...there is a distributed machine CONTROL SYSTEM, which consists of several semi-independent parts. These parts may be for example sensors and actuators, buses, controllers and computers. All hardware may break and software systems may contain bugs that cause the system malfunction. Malfunctioning control system acts erratically and may cause severe harm to itself, surrounding environment and/or operator or other people nearby. The system has employed OPERATING MODES pattern to enable state-based behavior.

How to minimize the possibility that operator, machine or surroundings are harmed when some part of the machine malfunctions?

The system should never cause harm to environment or people, even during a malfunction. This might be difficult to ensure as the malfunctions can cause unpredictable behaviour.



Therefore:

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.

* * *

There is only one safe state which is defined according to the system's requirements in its operating environment. When a predefined fault occurs, the malfunctioning system performs some last preparations and enters the safe state. The system will not re-enter the normal operating mode until the failure is corrected and some resetting action is made. This mode is therefore reserved for the most severe failures as the system can not anymore give any more added value to the user and it is not possible to use it anymore for productive work. In a complicated multi-node system it is however possible to just for some of the nodes to enter the safe state and some other subsystems can still function. For example, when a low level machine control system fails, the higher level monitoring systems may still function and give additional information about the failure.

It would be best to design the safe state so, that the system becomes automatically more safe if some parts are incapacitated. For example, in a modern nuclear reactor, the magnetically attached control rods are automatically dropped to the reactor core when the control system fails to keep them up. In this way, the nuclear fission stops in the case the control system is not working properly.

LIMP HOME is an alternative approach for failures that are not so severe. It allows some simple functionality to be still carried on regardless of some failure in the system.

* * *

The system is entering a predetermined safe state in the case of a malfunction. This safe state usually prevents any further harm to the environment, the operator or the system itself. However, the safe state might not be easy to determine as the machine may face some unforeseen situations.

* * *

An industrial bus connects process monitoring and controlling I/O cards together with an higher level services that are located in the control room. The control room PC polls the card in a fixed interval and the I/O card keeps track if it is read from a higher level device. Thus, the reading of the card acts as a Heartbeat. If the polling ceases for a certain time, the I/O card decides that the upper level has crashed and enters a safe state to prevent further harm. This safe state is device-dependent. For example, if the card controls pumps that move fluids in the pipes,

2.3 Limp Home

...there is a distributed machine CONTROL SYSTEM which consists of several semi-independent parts. These parts may be for example sensors and actuators, buses, controllers and computers. Some parts are just hardware and some parts include software systems. The full functionality mode depends on the services provided by all these separate parts. However, many of these parts exist only for some highly specialized functions. All hardware are prone to malfunction, but it would be useful if the whole machine is not incapacitated if something less crucial breaks. The system has employed OPERATING MODE pattern to enable state-based behavior.

When a part of the machine is malfunctioning, how to still operate the machine to some extent? For example, to drive the machine from forest to the nearest road.

It is not sensible to incapacitate the whole system if some optional system malfunctions. If a part of the systems exists only to make the working environment more pleasant or to make little enhancements to the productivity, the operator might want to keep up working even if this kind of functionality is lost. For example, the maintenance facilities may be far away from the machine. So, even if some parts of the machine are malfunctioning it would be useful to use the machine to some degree until a prescheduled maintenance break or until the machine would in any case leave the work site.

There are some functionalities which affect profoundly the overall safety of the system. These functionalities should be kept working as long as possible in any situation.

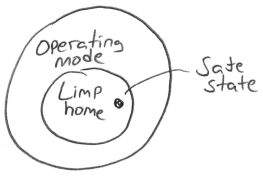
Therefore:

Divide the sensors and actuators into groups according to the high level functionalities, such as drive train, boom operations etc.. The groups may overlap. Malfunctioning device only disables the groups it belongs to and the other groups remain operable.

* * *

The system architect should design the system so, that there is a separate core functionality that is possible to perform under most situations. The core functionality might be for example just the basic movement of the machine. This core functionality is then extended with the more sophisticated services. All additional hardware that is now required is grouped optional. If some hardware in this group malfunctions, only the affected part is disabled and the driver can continue in the degraded mode.A group of devices that are required for a certain service may overlap with other services. If this is the case, if some overlapping device malfunctions, both services are incapacitated. If a device is only in one group, it will incapacitate only this service when a malfunction occurs.

There might be some services where the malfunctioning device only affects some high level functionality and when this kind of device breaks down, the system operator must do more to handicap the situation. If this is the case, the system productivity may be diminished, but the operations may otherwise continue normally. It it crucial for the system designer to identify these kinds of devices and make sure that the manual operation mode is possible.



The system should be fully inoperable only if the operation would severely harm the machine or safety of the people. Even then, some fully manual operating mode should be allowed for unforeseen situations, such as salvaging the machine from a hazardous situation (for example, driving the drill out of a collapsing mine).

The designer should try to make sure that if a subsystem breaks down, it will not lock the rest of the system up. For example, a broken down boom that is attached to some work item should be able to let the object loose in order to ensure that the work machine could still be driven out of the work site.

SAFE STATE is an approach for more severe failures.

* * *

After the devices are grouped according their criticality, the user may still make the decision to operate the system even if some non-critical system is failing. Thus, the system can still produce some value even when some parts of the machine are incapacitated.

In addition, the system is easier to test as the core functionality is clearly separated from the higher level services.

However, in some cases, the breaking down of a subsystem may still incapacitate the whole system.

* * *

A drill machine has core functionality that is implemented using low-end controllers. More elaborate services are provided by a cabin PC. These services include work maps, productivity tracking, fleet control etc.. If the cabin PC breaks down, the machine is still able to drill and drive around the work siten even though the high level supervisor services are down.

2.4 Sensor By-pass

...the distributed machine control system uses some advanced algorithms to control the hardware. These algorithms need some additional sensors that are in essence additional hardware - thus possible malfunctioning parts that are complicating the system design. However, some of these sensors are not crucial for the algorithm, but give only additional precision or some other advantage to the algorithm. Thus the system performance should not be greatly diminished if some of these fine-tuning sensors break.

How to offer operator a way continuing using advanced control mechanism even when some sensor of minor importance is faulty?

Some sensors are incorporated into the system design only to give small advantages in productivity or the user experience. However, they are prone to malfunction, as any hardware is. The system productivity should not decline any more than absolutely necessary, if some fine-tuning sensors break.

Therefore:

Implement a mechanism that the value provided by a sensor can be replaced with a default or simulated value.

* * *

If a sensor is detected to be working improperly, there are two options. The system can cease to use the full functionality provided by this service (/textscDegraded State) or the value it is providing is corrected with an estimate. In some cases, a simpler version of the sensor could also be used, if such sensor is present in the system. For example, a GPS-based leveling sensor might be The estimate could be the last known good value, some extra-/interpolation from it, a constant value or even a simulated value calculated from other inputs. In this way, the full functionality is still retained, but with a cost of a lesser accuracy.

* * *

The system can still use some high level algorithms, although with lesser precision. This means that even if an sensor breaks, the system is still operable to a high degree. If the system has multiple ways of producing a sensor value, an alternative way to deduce the value can be used and thus higher precision may be achieved.

However, it is often difficult or impossible to give reliable estimates of a value if a sensor has broken down. Usually the sensor is such that when it breaks down, no information is available from that part of the machine. Then the control algorithm has basically to guess the values the sensors would be outputting.

* * *

3 Acknowledgements

I wish to especially thank my collagues Veli-Pekka Eloranta, Johannes Koskinen and Ville Reijonen for their valuable help and input during the gathering process of these patterns. I also wish to thank all industrial partners for their willingness to provide the raw data for the pattern mining. These companies include Areva T&D, John Deere, Kone, Metso Automation and Sandvik Mining and Construction. I also wish to thank Nokia Foundation for their scholarship which has aided me in writing these patterns.

References

1. Samek, M.: Practical UML Statecharts in C/C++, Second Edition: Event-Driven Programming for Embedded Systems. 2 edn. Newnes, Newton, MA, USA (2008)