Architectural Design Pattern Representation For Communications-Based Train Control System (CBTCS)

Sahith Rampelli*,Sri Datta Virivinti**
*Assistant. Professor, Department of CSE, CVR College of Engineering, Hyderabad, AP, India-501 510

**Department of Information Technology, CVR College of Engineering, Hyderabad, AP, India- 501 510

Abstract

Communications-Based Train Control System (CBTCS)[1] is a railway signaling system that makes use of the telecommunications between the train and track equipment for the traffic management. By using the CBTC system, the exact position of a train is known more accurately than with the traditional signaling systems. This results in a more efficient and safe way to manage the railway traffic. This paper attempts to identify the design problems with respect to reliability and safety factors in the CBTC system and suggests suitable Architectural design patterns [2] to solve the problems.

1. Introduction

Communications-Based Train Control System (CBTC) is a type of rail signaling system that makes use of telecommunications between the train and track equipment for traffic management and infrastructure control.

The notion of design of patterns evolved from the approach to designing buildings developed by architect Christopher Alexander. Architectural design is concerned with large-scale strategic decisions that have broad and widely-ranging effects. Architectural decisions include the placement of software modules on different processors, real-time scheduling policies, identification of concurrency models, and inter-process and inter-thread communication

2. Motivation

Now a days, Due increase of population in Metropolitan cities, Metro-Rail-ways are introduced in-order to manage the traffic congestion. A reliable real time design pattern [3] should be implemented in order to design "Communication Based Train Control System".

3. CBTCS DESIGN PROBLEMS

Problem 1) If the communications link between any of the trains is disrupted.

Description: The primary risk of a CBTC system is that if the communications link between any of the trains is disrupted then all or part of the system might have to enter a fail-safe state until the problem is remedied. Depending on the severity of the communication loss, this state can range from vehicles temporarily reducing speed, coming to a halt or operating in a degraded mode until communications are re-established. If communication outage is permanent some sort of contingency operation must be implemented which may consist of manual operation using absolute block or, in the worst case, the substitution of an alternative form of transportation. As a result, high availability of CBTC systems is crucial for proper operation, especially if we consider that such systems are used to increase transport capacity and reduce headway. System redundancy and recovery mechanisms must then be thoroughly checked to achieve a high robustness in operation. With the increased availability of the CBTC system, it must also be considered the need for an extensive training and periodical refresh of system operators on the recovery procedures. In fact, one of the major system hazards in CBTC systems is the probability of human error and improper application of recovery procedures if the system becomes unavailable.

Problem 2) Communications failures can result from equipment malfunction, electromagnetic interference, weak signal strength or saturation of the communications medium.

Description: In this case, an interruption can result in a service brake or emergency brake application as real time situational awareness is a critical safety

requirement for CBTC and if these interruptions are frequent enough it could seriously impact service. This is the reason why, historically, CBTC systems first implemented radio communication systems in 2003, when the required technology was mature enough for critical applications.

Problem 3) Intrusion of the communications network and tampering with safety critical messages.

Description: CBTC systems that make use of open standards for wireless digital communications link have a much larger attack surface and can be subject to various types of hacking network and tampering messages. In the worst case, could result a safety hazard.

4.ARCHITECTURAL DESIGN PATTERNS FOR CBTCS

Architectural design is the identification and definition of large-scale design strategies. These strategic decisions determine how major architectural pieces of the system will be structured, mapped to physical devices, and interact with each other. The effect of such design decisions is widespread and affects most or all components.

Hardware Patterns

- Triple Modular Redundancy Pattern.
- Homogeneous Redundancy Pattern.
- Heterogeneous Redundancy Pattern.
- M-Out-Of-N Pattern.
- Monitor-Actuator Pattern.
- Sanity Check Pattern.
- Watchdog Pattern.
- Safety Executive Pattern.

Software Patterns [4]

- N-Version Programming Pattern.
- Recovery Block Pattern.
- Acceptance Voting Pattern.
- N-Self Checking Programming Pattern.
- Recovery Block with Backup Voting Pattern.

Hybrid Patterns

- The Secure Reliability (SecRel)
- Reliable Security (RelSec)

4.1. Problem 1: In CBTC system, if the communications link between any of the trains is disrupted.

Suggested Pattern(s): Triple Modular Redundancy Pattern (TMR), Homogeneous Redundancy Pattern, Heterogeneous Redundancy Pattern.

Let's consider TMR Pattern for the above said problem

4.1.1 Triple Modular Redundancy Pattern (TMR)

Other Name: 2-oo-3 Redundancy Pattern, Homogeneous Triplex Pattern.

Type: Hardware

Abstract:

The TMR pattern operates three channels in parallel rather than operating a single channel and switching over to an alternative when a fault is detected. By operating the channels in parallel, the TMR pattern detects random faults.

The TMR pattern runs the channels in parallel and at the end compares the results of the computational channels together. As long as two channels agree on the output, then any deviating computation of the third channel is discarded. This allows the system to operate in the presence of a fault and continue to provide functionality.

Context:

The Triple Modular Redundancy Pattern is a pattern used to enhance reliability and safety in situations where there is no fail-safe state. The TMR pattern offers an odd number of channels (three) operating in parallel.

Problem: To provide protection against random faults (failures) with the additional constraint that when a fault is detected, neither the input data should be lost, nor should additional time be required to provide a correct output response.

Implication: To enhance reliability and safety.

Implementation: The development of the TMR pattern is common to replicate the hardware and software to avoid common mode faults so that each channel uses its own memory, CPU and so on.

Consequences: The TMR Pattern can only detect random faults. It involves high recurring cost because the hardware and software in the channels must be replicated. The TMR pattern is a common one in applications where reliability needs are very high and worth the additional cost to replicate the channels.

Related Patterns: Heterogeneous redundancy (protection against systematic faults is desired). A Homogeneous Redundancy Pattern (can be used if the data can be lost when a failure occurs or when it is okay to re-execute the failed computational step).

Example: With reference to the use of TMR pattern in CBTC System, The TMR Pattern will be operated on three channels, if the communication link between any of the trains is disrupted then very immediately one of active channel will take over the substitution of an alternative form of transportation. Hence, the communication link is active until the error is repaired or required component is substituted. As a result, high availability of CBTC system is achieved.

4.1.2 Homogeneous Redundancy Pattern

Other Names: Switch to Backup Pattern, Homogeneous Redundancy Pattern, Standby-Spare Pattern, Dynamic Redundancy Pattern, Two-Channel Redundancy Pattern
Type: Hardware Pattern
Abstract:
An obvious approach to solving the problem of things breaking is to provide multiple copies of that thing. In safety and reliability architectures, the fundamental unit is called a *channel.* A channel is a kind of subsystem, or run-time organizational unit, which is end-to-end in its scope, from the monitoring of real world signals to the control of actuators that do the work of the system. The Homogeneous Redundancy Pattern replicates channels with a switch-to-backup policy in the case of an error.
Context:
Homogeneous Redundancy Pattern is primarily a pattern to improve reliability by offering multiple channels. These channels can operate in sequence, as in the Switch To Backup Pattern (another name for this pattern), or in parallel, as in the Triple Modular Redundancy Pattern, described later. The pattern improves reliability by addressing random faults (failures). Since the redundancy is homogeneous, by definition any systematic fault in one copy of the system is replicated in its clones, so it provides no protection against systematic faults (errors).
Problem:
The problem addressed by the Homogenous Redundancy Pattern is to provide protection against random faults that is, failures in the system execution and to be able to continue to provide functionality in the presence of a failure. The primary channel should continue to run as long as there are no problems. In the case of failure within the channel, the system must be able to detect the fault and switch to the backup channel.
Structure:
The checking components implement a switch-to-backup policy by invoking the other channel when an error is detected in the currently operating channel.
Implication: safety and reliability in the presence of either systematic or random faults
Implementation:
The implementation of this pattern is only a bit more work than the implementation of a non-redundant system. To remove common fault modes, the computing hardware (CPU, memory, etc.) as well as mechanical systems should be replicated. The only special work is the logic to identify the faults and switch to the alternative channel when a fault is detected.
Consequences:
The Homogenous Redundancy Pattern has a number of advantages. It is conceptually simple and easy to design. It provides good coverage for random (that is, hardware and transient) faults, although only if the hardware is itself replicated. It is usually a simple matter to get good isolation of faults and to eliminate common mode faults. The pattern applies when random faults occur at a significantly higher rate than systematic faults, such as in rough or arduous physical environments. It also is useful for safety-critical or high-reliability systems that must continue to operate in the presence of faults.
An advantage of this pattern is the low R&D cost – since there is only a single channel to design. Its primary disadvantage is the lack of coverage for systematic faults and increased deployment costs over non-redundant systems.
The disadvantages of the pattern are primarily the higher recurring cost and a lack of coverage for systematic faults. Because the electronic and mechanical hardware must be duplicated for maximal coverage, each shipping system must bear the cost of additional hardware components. Furthermore, since the channels are clones, any systematic fault in one channel must, by definition, appear in the other. The pattern runs a single channel and switches over to a backup channel only when a fault is detected. This means that the computation step is lost when a fault is detected and either the data is lost or recovery time to redo the computation must be taken into account in time-critical situations.

4.1.3 Heterogeneous Redundancy Pattern

Other Names: Diverse Redundancy and N-way Programming.
Type: Hardware
Abstract:
For high-safety and reliability systems, it is common to provide redundant channels to enable the system to identify faults and to continue safe and reliable operation in the presence of faults. Similar to its homogeneous cousin, the Heterogeneous Redundancy Pattern provides redundant channels as an architectural means to improve safety and reliability. What sets the Heterogeneous Redundancy Pattern apart is that the channels are not mere replicas but are constructed from independent designs. This means that identical design errors are unlikely to appear in multiple channels. The primary downside of this pattern is its high design development cost that comes on top of the high recurring cost typical of heavyweight redundant channels.
There are a number of useful variants of the Heterogeneous Redundancy Pattern that provide the

detection of both kinds of faults but are lower cost and may not provide continued operation in the presence of faults. See, for example, the Monitor-Actuator and Sanity Check Patterns.

Context: Protection against random and systematic faults without a fail-safe state

Problem:

The Heterogeneous Redundancy Pattern provides protection against both kinds of faults—systematic errors as well as random failures. Assuming that the design includes independence of faults, the pattern provides single fault safety in the same way as the Homogeneous Redundancy Pattern—that is, when the primary channel detects a fault, the secondary channel takes over.

Structure:

Indeed, the pattern is almost identical, with the primary difference being that the components of the two channels are the result of independent design efforts. The independent design effort may use the same algorithm with different teams or— even better— different algorithms with different teams.

Implementation:

The implementation of this pattern requires fault independence. That means that the hardware components must be replicated in both channels (CPU, memory, and so on). It is common to replicate the computing hardware rather than use different CPUs, but different computing hardware does give a slightly increased level of safety. The sensors and actuators are, however, usually different hardware implementations, often using different technologies. It is best if the software is not only designed by different teams but also uses different algorithms. Simply using independent teams doesn't provide total independence of systematic faults, since the teams will tend to make mistakes in the same portions of the application (such as the hard parts).

Consequences:

This pattern has two "heavyweight" channels. This means both are relatively expensive to design and construct, and either can perform the actuation processing with similar levels of fidelity. Similar to the Homogeneous Redundancy Pattern, this pattern has a high recurring cost due to the inclusion of additional hardware support for the redundancy. However, in addition to this, the Heterogeneous Redundancy Pattern also has a high development cost because multiple independent designs must be performed, usually with different teams to provide independence of systematic faults. This is generally considered the safest architectural pattern and the most expensive as well. With only two channels, however, it may have lower availability than with the Triple Modular Redundancy Pattern. To enhance availability, a Triple Modular Redundancy Pattern may be used with heterogeneous channels to get the best (and the worst) of both worlds.

Related Patterns:

As mentioned earlier, this is a very expensive pattern to implement. Reduced cost can be had at the expense of reducing safety coverage as well. A Homogeneous Redundancy Pattern can be used with the effect of lowering the ability to detect systematic faults and lowering development cost. A Triple Modular Redundancy Pattern implemented with heterogeneous channels improves availability over the Heterogeneous Redundancy Pattern but at the cost of increasing both the development and recurring cost by one third. When protection should be provided but the system does not need to continue operation in the presence of a fault, then a lower-weight solution, such as the Monitor-Actuator or Sanity Check Pattern may be used.

4.2. Problem 2: Communications failures can result from equipment malfunction, electromagnetic interference, weak signal strength or saturation of the communications medium.

Suggested Patterns:

Safety-Executive pattern,

Monitor Actuator and Watchdog pattern.

Let's consider the Safety-Executive pattern for the above said problem,

4.2.1 Safety-Executive Pattern

Let's consider the Safety-Executive pattern for the above said problem,

Other Name: Safety Kernel Pattern

Type: Hardware and Software

Abstract: Systems often cannot merely be shut down in the event of a fault. Sometimes this is because they are in the middle of handling some dangerous materials or a high-energy state of the system (such as high speed or high voltage potential). Simply shutting the system off in such a state is potentially very hazardous. In the presence of a fault, the system must be guided through a potentially complicated series of steps to reach a condition known to be a fail-safe state. The Safety Executive Pattern models exactly this situation in which a Safety Executive component coordinates the activities of potentially many actuation channels and safety measures to reach a fail-safe state.

Problem: The problem addressed by the Safety Executive Pattern is to provide a means to coordinate and control the execution of safety measures when the safety measures are complex. Ex: The shutdown of the channel or system is a complex process, where several safety-related actions have to be controlled simultaneously.

Context: The problem addressed by the Safety Executive Pattern is to provide a means to coordinate and control the execution of safety measures when the safety measures are complex. Ex: The shutdown of the channel or system is a complex process, where several safety-related actions have to be controlled simultaneously.
Implication: Provides Safety and reliability for highly safety-critical real-time systems.
Implementation: Any single component, whether hardware or software, should be allowed to fail without creating a hazard. In the systems for which this pattern is appropriate, this means that the channels will each run on their own CPUs with their own memory; safety-critical information must be protected with CRCs or other means to detect data corruption.
Consequences: It can provide excellent fault protection in highly complex systems and environments.
Related Patterns: The Safety Executive Pattern is used for complex safety-critical applications and it covers a large set of features such as sequence monitoring provided by watchdog and switch-to-backup as in the fail-safe channel.
Example: The Safety Executive pattern is used in highly complex systems, so CBTC system can be implemented with this pattern and also, it has been used in high-speed train control systems.

4.2.2 Monitor Actuator

Pattern Name: Monitor Actuator

Abstract: Many safety-critical systems have what is called a fail-safe state. This is a condition of the system known to be always safe. When this is true, and when the system doesn't have extraordinarily high availability requirements (that is, in the case of a fault detection it is appropriate to enter the fail-safe state), then the safety of the system can be maintained at a lower cost than some of the other patterns. The Monitor-Actuator Pattern is a specialized form of the Heterogeneous Redundancy Pattern because the redundancy provided is different from the primary actuation channel: It provides monitoring, typically of the commanded actuation itself (although it may also monitor the internal operation of the actuation channel as well). Assuming fault independence and a single point fault protection requirement, the basic principle of the Monitor-Actuator Pattern may be summed up this way: If the actuation channel has a fault, the monitoring channel detects it. If the monitoring channel breaks, then the actuation channel continues to operate properly.
Context: Protection against random and systematic faults with a fail-safe state. All safety-critical and reliable architectures have redundancy in some form or another. In some of these patterns, the entire channel, from original data sensing to final output actuation, is replicated in some form or another. In the Monitor-Actuator Pattern, an independent sensor maintains a watch on the actuation channel looking for an indication that the system should be commended into its fail-safe state.
Problem: The Monitor-Actuator Pattern addresses the problem of improving safety in a system with moderate to low availability requirements at a low cost.
Structure:
Both channels run independently and simultaneously.
Consequences:
This pattern is a relatively inexpensive safety solution that is applicable when the system does not have high availability requirements and when there is a fail-safe state. Assuming that its implementation correctly isolates faults, a fault in the *Actuation Channel* will be identified by the *Monitoring Channel.* A fault in the *Monitoring Channel* will not affect the proper execution of the *Actuation Channel.* Because there is minimal redundancy, the system cannot continue to function when a fault is identified.
Implication: Safety
Implementation: The *Monitoring Channel* must take into account lag, measurement jitter, control system jitter, computational accuracies (specifically the propagation and compounding of computational numeric error), and other forms of error in determining whether the actuation channel is acting properly. Another issue with the *Monitoring Channel* is the handling of *transient* faults. In some situations, a single transient fault may not be harmful at all, but *persistent* faults must be identified. In such cases, it may be necessary for the *Monitoring Channel* to maintain a recent history of its monitored values to determine whether an unexpected value indicates a transient or persistent fault.
The system can operate with a fault in the *Monitor Channel*, but if it does so, this is called a *latent fault.* A latent fault is one that by itself does not present a hazard but with the addition of a second fault *does* present a hazard. For this reason, the *Monitor Channel* must be periodically checked. The timeframe for this check must be significantly less than the mean-time between failures (MTBF) of the *Monitor Channel.*
In practice, this check is usually done daily or on every startup, whichever is less. It may, at times, be performed during scheduled maintenance of the system but must be done much more frequently than the MTBF of the channel and any of its components. Often, systems using this pattern use a pair set of *life ticks* sent between the channels to indicate the health of the other system. If one channel does not receive a life

tick from the other within a specified time frame, then this indicates a fault, and the fail-safe state is entered.

Related Patterns: Sometimes the control signal does not provide the desired end-result to the *Monitoring Channel.* When this is the case, the *Monitoring Channel* must in some sense simulate the processing done in the *Actuation Channel.* When this is done in a lightweight way to get a check on the reasonableness of the resulting actuation output, this is called the Sanity Check Pattern. A *very* lightweight means of providing monitoring is the Watchdog Pattern. This pattern monitors what the Actuator Pattern *thinks* is the right thing to do and not the actual output of the actuation. If it is necessary to continue actuation in the face of a fault, then a heavier-weight pattern, such as the Homogeneous Redundancy or the Heterogeneous Redundancy Pattern, must be used.

4.2.3 Watchdog Pattern

Pattern Name:_Watchdog Pattern

Abstract

A watchdog, used in common computing parlance, is a component that watches out over processing of another component. Its job is to make sure that nothing is obviously wrong, just as a real watchdog protects the entrance to the henhouse without bothering to check if in fact the chickens inside are plotting nefarious deeds. The most common purpose of a watchdog is to check a computation time base or to ensure that computation steps are proceeding in a predefined order. Watchdogs are often used in real-time systems to ensure that time-dependent processing is proceeding appropriately.

Problem

Real-time systems are those that are predictably timely. In the most common (albeit simplified) view, the computations have a deadline by which they must be applied. If the computation occurs after that deadline, the result may either be erroneous or irrelevant—so-called hard real-time systems. Systems implementing PID control loops, for example, are notoriously sensitive to the time lag between the occurrence of the input signal and the output of the control signal. If the output comes too late, then the system cannot be controlled; then the system is said to be in an unstable region.

Context:

The Watchdog Pattern is similar to the Sanity Check Pattern in the sense that it is lightweight and inexpensive. It differs in what it monitors. While the Sanity Check Pattern monitors the actual output of the system using an external environmental sensor, the Watchdog Pattern merely checks that the internal computational processing is proceeding as expected. This means that its coverage is minimal, and a broad set of faults will not be detected. On the other hand, it is a pattern that can add additional safety when combined with other heavier-weight patterns.

Structure

The Actuator Channel operates pretty much independently of the watchdog, sending a liveness message every so often to the watchdog. This is called stroking the watchdog. The watchdog uses the timeliness of the stroking to determine whether a fault has occurred. Most watchdogs check only that a stroke occurs by some elapse of time and don't concern themselves with what happens if the stroke comes too quickly. Some watchdogs check that the stroke comes neither too quickly nor too slowly.

For some systems, protection against a time base fault is safety-critical. In such cases, it is preferable to have an independent time base. This is normally a timing circuit separate and independent from the one used to drive the CPU executing the Actuation Channel. It should be noted that all safety-critical systems are real-time systems because they must respond to a fault by the Fault Tolerance Time

Implication: Safety

Implementation

If the watchdog is to provide protection from time base faults, a separate electronic circuit must supply an independent measure of the flow of time. This means an independent timing circuit used to capture the timing and the watchdog detects a mismatch between the two timing sources but cannot detect whether the fault is the primary actuation channel or the watchdog time base. To prevent a fault where the primary actuation channel gets stuck in a loop (so called live-lock) that strokes the watchdog but doesn't actually perform the appropriate computation and actuation, the watchdog may require data with the strokes that must occur in a specific pattern. To implement a keyed watchdog, the best approach is to not store the keys in memory but to have them dynamically computed as a result of the proper execution of the actuation process. This diminishes the likelihood of a live-lock situation not being detected.

When the watchdog is stroked, it is common to invoke a BIT (Built In Test) of some kind to ensure the proper execution of other aspects of the system. These actions can either return a Boolean value indicating their success or failure, or may directly cause the system to shut down in the case of their failure. For example, the watchdog may execute an action on the evStroke transition that checks for stack overflow and performs CRC checks on the executing application software. If it does a similar check on the application data, it must lock the data resources during this computation, which can adversely affect performance if you're not careful.

Stack overflow may be checked for by writing a known pattern into the stack of each task beyond the expected

stack size. If this pattern is disrupted, then a stack overflow (or something equally bad) has occurred. When watchdog fires because it hasn't been stroked within the specified timeframe, it invokes some safety measure, normally either shutting down the system or causing the system to reset.

Consequences

The Watchdog Pattern is a very lightweight pattern that is rarely used alone in safety-critical systems. It is best at identifying time base faults, particularly when an independent time base drives the Watchdog. It can also be used to detect a deadlock in the actuation channel. To improve deadlock detection, the watchdog may require the strokes to be keyed—that is, to contain data that can be used to identify that strokes from different computational steps occur in the proper sequence. Such a watchdog is called a Keyed Watchdog or a Sequential Watchdog. Because the coverage of the Watchdog Pattern is so minimal, it is rarely used alone. It may be combined with any of the other safety patterns.

Related Patterns

The Watchdog Pattern is about as lightweight (low effort as well as low protection) as a safety and reliability pattern gets. For this reason, it is normally mixed with other patterns as a way to test the time base and to drive periodic BITs.

4.3. Problem 3: Intrusion of the communications network and tampering with safety critical messages.

The above problem deals with the security[6] concerns of the system. These kinds of problems can be solved using "Hybrid Patterns". Hybrid Patterns like "SecRel[5] and RelSec[5]" can provide a variety of reliable mechanisms that can be applied to systems.

Suggested Patterns:

Secure Reliability Pattern and Reliable Security Pattern

4.3.1. Secure Reliability Pattern

A critical system must need a high degree of security and reliability to work properly. CBTCS needs reliable services according to its working, so this pattern is very useful. Misuse of some services may lead to serious damage of the whole system, so the usage of services to users might lead to serious problems. We must have a set of reliability mechanisms and a reference monitor is required to check the rights associated with the user before the service is activated. It should also perform authentication before a user is authorized.

4.3.2. Reliable Security Pattern

This pattern is intended to implement reliable authorization enforcement by applying reliability mechanisms to the system.

5. Conclusion

In this paper we discussed the problems of designing a CBTC system and various design patterns which can be applied in order to design a CBTCS. Architectural Design Patterns solve the design problems in the real-time systems in-order to achieve high reliability and safety. In CBTCS, we can apply design patterns to achieve high reliability and safety.

6. References

[1]http://en.wikipedia.org/wiki/Communications-based_train_control

[2] Design Patterns: Element of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Edition published in 2012.

[3] Douglass, Bruce Powel (2002). *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Addison Wesley.

[4] Real-Time Software Design Patterns, Janusz Zalewski.

[5] Patterns Combining Reliability and Security Ingrid A. Buckley, Eduardo B. Fernandez, and Maria M. Larrondo-Petrie, Published in: IARIA conference, September 25,2011.

[6] N. Yoshioka, H. Washizaki, K. Maruyama, A survey on security patterns, Progress in Informatics, No. 5 pp. 35-47, (2008)

7. Authors

Sahith Rampelli is currently working as Assistant. Professor at CVR College of Engineering, Hyderabad, A.P, INDIA. He has received his MCA. Degree from Kakatiya University and M.Tech. with specialization in Computer Science & Engineering from JNTUH, Hyderabad, INDIA. His main research interest includes Architectural Design Patterns and Real-Time Systems. He has been involved in the organization of a number of conferences and workshops.

Sri Datta Virivinti is a B.tech 3rd year Information Technology student in CVR College of Engineering, Hyderabad-500062, Andhra Pradesh, India.