Implementation of Real Time Design Patterns in Road Departure Mitigation System

Sai Nishanth Adingi [1], Utkarsha Fegade[2], Mariam Jamal[3], Sai Krupa Peraka[4]

Abstract: This exhibit contains the implementation of various Real-time design pattern to Road Departure Mitigation System The paper covers six different views of the design patterns, namely, Subsystem And Component Architecture Patterns, Concurrency Patterns, Memory Patterns, Resource Patterns, Distribution Patterns , adn Safety and Reliability Patterns.

1 Introduction to RDMS:

[Sai Nishanth Adingi]

The Road Departure Mitigation System (RDM) employs the windshield-mounted camera conjointly utilized by LDW (Lane departure warning system) to spot the facet of the road, as well as painted lane lines, Botts' Dots and cat's-eye markers. When the system detects that the vehicle is getting ready to leave the road, it alerts the driving force with a middle warning message. The system is intended to then use the electrical power-assisted steering system (EPS) to guide the vehicle into its detected lane or to use the brakes to stay the Accord from departure the route altogether. Characteristics:

- Real time system:The system is a hard-embedded real time system with the sensors being installed around the automobile acting upon various parameters to achieve the performance of the system.
- Reactive system:The framework must be prepared to do continuously interacting with the earth, react to the changes, occasions and the intrudes, with no end.
- Dependable system:The system is more interactive with the surrounding environment, thus makes the system highly dependable and minor error can cause a damage easily.
- Continuous system:The data on the onboard chip should be continuous Even a minor bug in the system might lead to fatality.

[1] FH-Dortmund sai.adingi002@stud.fh-dortmund.de
[2] FH-Dortmund utkarsha.fegade001@stud.fh-dortmund.de
[3] FH-Dortmund mariam.jamal001@stud.fh-dortmund.de
[4] FH-Dortmund Sai.peraka003@stud.fh-dortmund.de

2 Subsystem and Component Architecture Pattern

[Sai Nishanth Adingi]

A complex system or a very large system can be decomposed into subsystems by using different patterns where each subsystem can be verifiable at each and it is easy to code as well and it provides opaque interface to its peer and it collaborates for the system behavior. The various types of architectural design patterns:

1. Layered Pattern: In this layered pattern the association between the layers of a complex system are one-way. The data can be sent from higher levels to lower levels only.
2. Five-Layered pattern: This pattern is similar to layered pattern.
3. Microkernel Pattern: Optional sets of services may be added to the system during system builds.
4. Channel Architecture Pattern: This approach works well for non-real-time processing of data.
5. Recursive Containment pattern: In this pattern the complex system can be decomposed to subsystems but it cannot be configured while system is running.
6. Virtual Machine Pattern: The virtual machine itself is often implemented using the layered or microkernel pattern.
7. Component-Based Architecture: This pattern is similar to microkernel pattern.
8. Room Pattern: It is similar to the recursive containment pattern but it is more generalized than room pattern.

Design Criteria / **Architectural Design Patterns**	Abstraction	Configurable	Real-Time	Score
	9	8	7	
Layered Pattern	6	5	8	150
Five-layered Pattern	6	5	8	125
Microkernel Pattern	4	5	9	121
Channel Architecture Pattern	5	4	7	103
Recursive Containment Patterr	9	4	8	117
Hierarchical Control Pattern	9	8	9	185
Virtual Machine Pattern	4	6	7	147
Component Based architecture	6	4	7	97
Room Pattern	7	5	6	104

Fig. 1: Selection of pattern which suits the best for the system based on the highest score.

2.1 Hierarchical Control Pattern

The Hierarchical Control Pattern could be a specialized sort of the Recursive Containment Pattern that distributes complex control algorithms among its various pieces.

The Hierarchical Control Pattern uses two varieties of interfaces: control interfaces that track and govern how behaviors are performed and functional interfaces that provide the opposite the set of interfaces. The control interfaces determine how the execution is allotted. This incorporates the subordinate parts also. The functional interfaces execute the required behavior using the standard of service and policies set via the control interface.

The hierarchical system of control is structured into hierarchies supported relationships of composition, as shown in Figure. The controller orchestrates the distribution of its services by delegation and control of its subordinate sections. Underlying pieces could also be leaf elements or controllers. Leaf elements split the hierarchy that are without subordinate parts. Controllers, on the opposite hand has a subordinate part, and supply two different sorts of interfaces control and functional

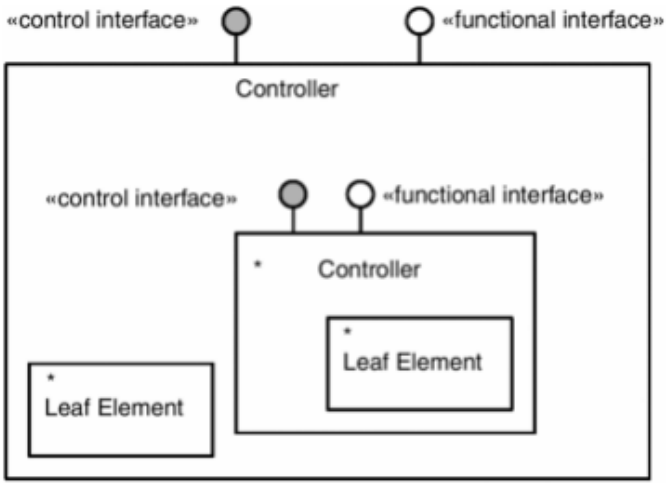

Fig. 2: Hierarchical Control Pattern Structure

Controller-The position of the controller provides its clients with two types of opaque interfaces: device management and functional interfaces. Control Interface- The Control Interface offers tools to monitor the performance of practical services, such as switching to a new algorithm or delivering a specific level of service.

Functional Interface-These services are delivered using the policies, algorithms, and qualities of service specified in using the Control Interface.

The hierarchical pattern is applied to our system road departure mitigation system because we can configure the system by switching to a different algorithm. It can be changed by the client or driver of the car by turning off the lane departure warning through settings or through a switch which is available on the steering. The Road departure mitigation system

will fail whenever there is a curvy road and single marking lane so it can be configured while the system is running without effecting any parts of the system.

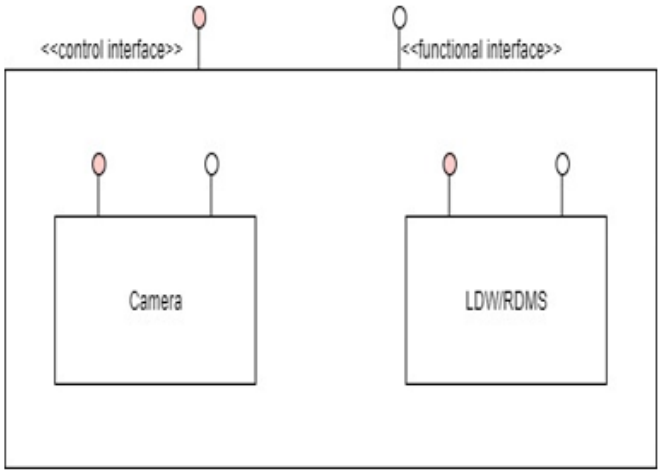

Fig. 3: Hierarchical Control Pattern Structure is applied to RDMS

The primary advantage of this approach is that it allows for the visualization of complex structures at several abstract levels. In this way it could be easily replace and rectify whenever there is error in the system. In this the data can be transferred to its peer and collaborates and govern the execution proceeds.

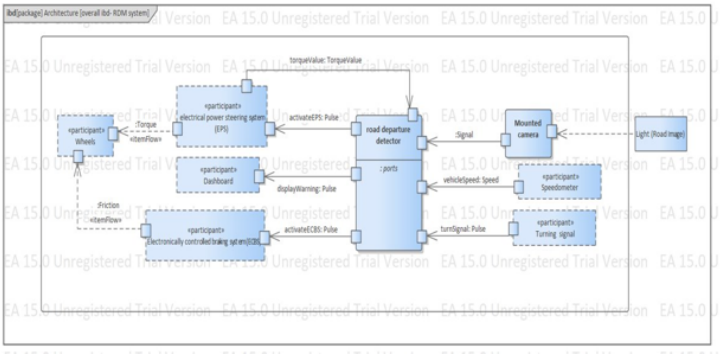

Fig. 4: Detailed abstraction of RDMS system

3 Concurrency Patterns

[Utkarsha Fegade]

The concurrency architecture includes the control and scheduling of architectural elements. It also includes the management of shared resources. In real-time systems, if the executing threads are completely independent, concurrency management could be very easy. But in reality, threads are usually not independent. Hence the threads must synchronize, coordinate, share information, and shared resources must be carefully managed. If not properly managed, they could lead to the occurrence of problems like corruption and erroneous computation. The concurrency patterns provide a solution to these problems.

The various concurrency patterns compared during this project are:

1. Message Queuing Pattern: Robust asynchronous task rendezvous
2. Interrupt Pattern: Fast short event handling
3. Guarded Call Pattern: Robust synchronous task rendezvous
4. Rendezvous Pattern: Generalized task rendezvous
5. Cyclic Executive Pattern: Simple task scheduling
6. Round Robin Pattern: Fairness in task scheduling
7. Static Priority Pattern: Preemptive multitasking for schedulable systems
8. Dynamic Priority Pattern: Preemptive multitasking for complex systems

	Design Criteria			Weighted Score
	Schedulability	Shared Resource Protection	Adherance to deadlines	
Weight	6	7	9	
Design solution	Score			
Message Queuing Pattern	4	8	3	107
Interrupt Pattern	4	5	5	104
Guarded Call Pattern	2	8	6	122
Rendezvous Pattern	5	0	5	75
Cyclic Executive Pattern	5	2	3	71
Round Robin Pattern	5	2	4	80
Static Priority Pattern	7	8	6	152
Dynamic Priority Pattern	8	8	9	185

Fig. 5: Concurrency Patterns Rank Analysis

Fig.5 shows the Rank-Design table for various concurrency patterns. As can be concluded from the table, the pattern with highest weighted score is the Dynamic Priority Pattern, hence it is the most beneficial pattern for Road Departure Mitigation System. This pattern emphasizes on urgency over criticality. One of the most common algorithms used to apply this pattern is the Earliest Deadline First. In EDF, the priority is decided on the basis of the upcoming deadline of the thread.

Dynamic priority Pattern has a dynamic allocation of the priorities i.e. the priorities are assigned at the run time. Hence, priorities cannot be fixed at the design time. This pattern is

best suited for the system with a set of tasks that have approximately equal criticality so that focus could be laid on urgency.

The reasons this pattern is the best fit for Road Departure Mitigation System are as follows:

1. As discussed earlier, one of the major characteristics of the RDMS is that it is a hard-real time system. That means adherence to the deadline is very critical. And this pattern prioritizes urgency.
2. Along with adherence to deadlines, this pattern also has a provision for the protection of shared resources.
3. All the tasks in the RDMS have approximately equal criticality.
4. The system is quite complex with a large number of sensors and controls involved.

3.1 Example model

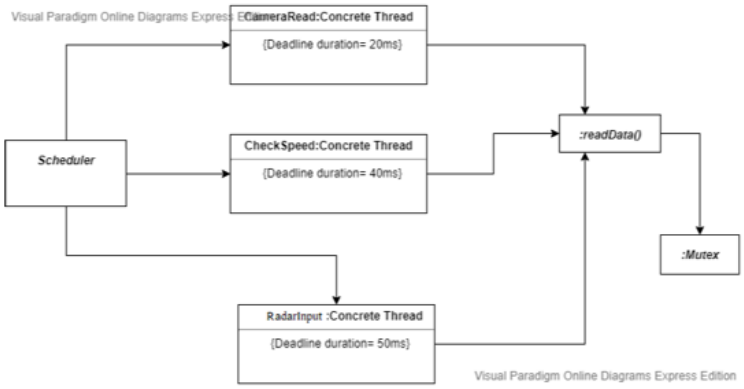

Fig. 6: Block Diagram

Fig.6 shows the basic block diagram of how the Dynamic Priority Pattern could be applied to RDMS. The scheduler block is responsible for applying the pre-decided scheduling algorithm for the threads. RadarInput, CheckSpeed, and CameraRead are the 3 tasks that act as concrete threads i.e. they would be performing the actual task. All these threads have different deadlines and priorities and share a common resource readData function, which is protected using Mutex.

Fig.7 shows the sequence diagram of RDMS after applying the Dynamic Priority Pattern. As shown in the figure, first the scheduler will run the CameraRead which is the high priority task. It will read data and once it has finished the execution it will send a reply to the Scheduler that task is finished. Now the scheduler will run the medium priority

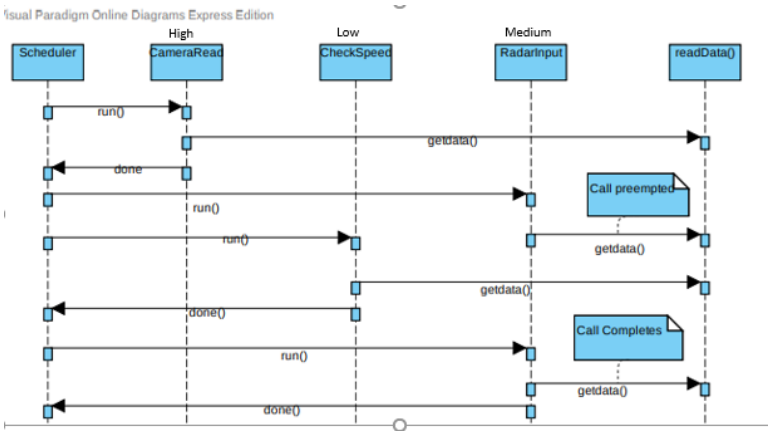

Fig. 7: Sequence Diagram(*all signals in the diagram are asynchronous*)

task i.e. RadarInput. When this task is executing, the scheduler runs the low priority task CheckSpeed. As the deadline for CheckSpeed is less than RadarInput, the CheckSpeed task would be assigned a higher priority, and RadarInput will be pre-empted. Once the CheckSpeed task finishes its execution, the RadarInput task will continue from where it was pre-empted and complete its execution. Hence, it can be seen that the priority assignment happens during run-time with this pattern based on the deadline and not criticality.

4 Memory Patterns

[Mariam Jamal]

Memory Patterns provide a generalized solution for the efficient management of memory as a resource. Given the ever-increasing complexity of embedded systems and the drive towards more connected devices, one of the major challenges during design time is how to allocate and manage the shared resource viz. data and algorithms in an efficient and robust manner, thus ensuring the overall schedulability of the system. One definitive cause is due to the memory wall [1], which in simple terms is "..the growing gap between the computational performance and the memory access latencies and bandwidth".

As seen in Fig.8, there is a rapid growth in the performance of the processor (60percent per year) while the memory performance lags substantially behind with an increase rate of just 7 percent per year. This gap grows at a rate of 50percent each year, hence, unravelling a deep issue that needs to be attended. Therefore, in conclusion, we can say that memory is a significant bottleneck of performance which is crucial criteria for real time embedded systems. It also has a deep impact on the main cost metrics such as power consumption, area and throughput.

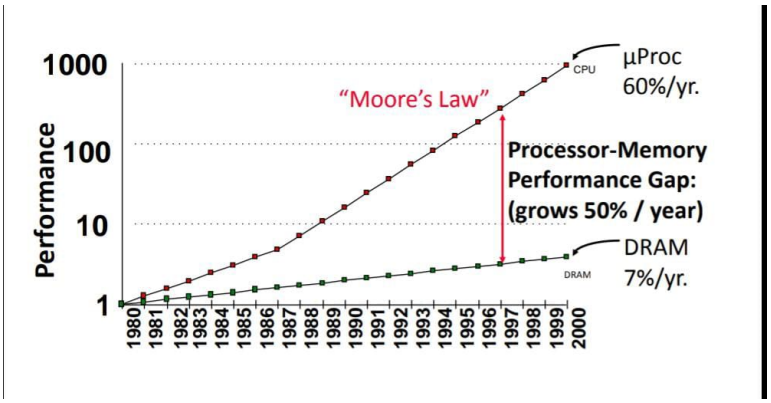

Fig. 8: Processor-Memory Performance Gap[2]

Although we are provided with a multitude of memory design patterns, but for the purpose of this project, we focused on the following patterns:

1. Static Allocation Pattern: Allocating memory upfront during design time.
2. Pool Allocation Pattern: Pre-Allocates pool of objects which are frequently needed.
3. Fixed Size Buffer: Allocating memory in same sized blocks.
4. Smart Pointers: Making pointers reliable by using them as objects.
5. Garbage Collection: Automatically reclaiming lost memory by identifying unreferenced objects and freeing up memory space.
6. Garbage Collector: Works in conjunction with Garbage Collector to compact the freed memory space.

4.1 Road Departure Mitigation System and Memory Patterns

Road Departure Mitigation System is an ADAS component for assisting drivers to stay in lane to prevent accidents and unintentional steering. In road-transport terminology, a RDMS system is a mechanism designed to warn and mitigate accidents caused by the driver when the vehicle begins to move out of its lane (unless a turn signal is on in that direction) on freeways and arterial roads. These systems are designed to minimize accidents by addressing the main causes of collisions: driver error, distractions and drowsiness.

The main characteristics of this system is:

1. Safety-critical System: Any system failure could lead to a significant effect and damage onto the driver/passengers of the system and the environment.

2. Hard Real Time System: Missing of deadlines is just not acceptable as it is a safety critical system.
3. Reactive System: The input data from the mounted camera and other sensors pass through a transformation and is processed to deliver output in the form of user guidance or actuation.
4. Dependable System: The dependability is high, and hence it needs to be efficient, robust and correct.

The design criteria for choosing the right memory allocation pattern is based on the following factors:

1. Ease of Implementation
2. Memory Issues
   a) Memory Leaks
   b) Memory Fragmentation
   c) Memory Exhaustion
3. Efficiency
4. Response Time
5. Additional Overheads (such as memory, time, efforts)

The scores are calculated based on the weights assigned to each of these criteria. For the Memory Issues criteria, the lower the score, more are the issues and similar is the case for additional overheads.

	Design Criteria							Total Score
	Ease of Implementation		Memory Issues		Efficiency	Speed	Overheads	
		Memory Leaks	Memory Fragmentation	Memory Exhaustion				
Weights	W=3	W=5	W=6	W=8	W=7	W=9	W=4	
Design Solutions								
Static Allocation	9	10	10	5	6	8	7	319
Pool Allocation	6	10	10	5	6	8	5	302
Fixed Size Buffer	7	8	6	4	5	7	5	247
Smart Pointers	4	8	7	6	3	4	3	211
Garbage Collector	7	4	2	7	4	5	4	198
Garbage Compactor	5	5	9	9	5	5	3	258

Fig. 9: Decision Analysis Matrix

As seen in the decision analysis matrix in Fig.9, the highest score is achieved by the Static Allocation Pattern. Static Allocation Pattern pre-allocates structure during design time and ensures that dynamic memory allocation is avoided during normal processing. It is simple to understand and implements objects using fixed-sized data structures like arrays or pre-allocated collection class. It also reduces the space overhead for using pointers or global overhead for using the garbage collector, thus the efforts for the programmer are minimized. The memory issues such as leaks and external fragmentation will be diminished as memory

is not deallocated. Although, there will be an increase in system start up time, but the responsiveness and time performance will be better once the operation begins. Fixed size allocation also means that one can predict the system's memory needs exactly at compile time and the time required for the allocation is small and constant, thus making it suitable for real time systems such as RDMS. Thus, for safety-critical systems such as RDMS, it is a good choice and advisable to use this pattern. Infact, the UK's Department of Defence regulations for safety-critical systems permitted only Fixed Allocation, although this has been relaxed recently in some cases [3][Matthews 1989, DEF-STAN 00-55 1997]. Also, as per the MISRA (Motor Industry Software Reliability Association) -C guidelines, rule no. 20.4 "Dynamic Heap Memory Allocation shall not be used."[4]

4.2 Example Model

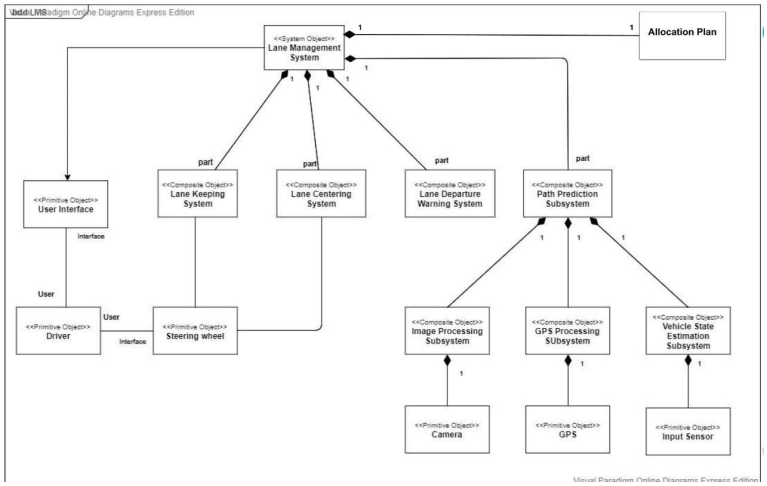

Fig. 10: Structure using Static Allocation Pattern

Fig.10 depicts how the Static Allocation Pattern can be applied to the RDMS. Here, the system is divided 4 composite objects namely: Lane Keeping System, Lane Centering System, Lane Departure Warning System and the Path Prediction Subsystem. The Path Prediction Subsystem further composes three objects, for the Image Processing Subsystem, GPS Processing Subsystem and the Vehicle State Estimation Subsystem. These subsystems are then nested to create the primitive objects such as the Camera, GPS, and the input sensors respectively. The RDMS is also associated with other primitive objects which are not a part of its system such as the User Interface, Driver and Steering Wheel objects.

Fig.11 shows the sequence diagram or the scenario of RDMS after applying the Static Allocation Pattern. As displayed in the figure, on calling the Lane Management System constructor first the Lane Keeping System, Lane Centering System, Lane Departure Warning

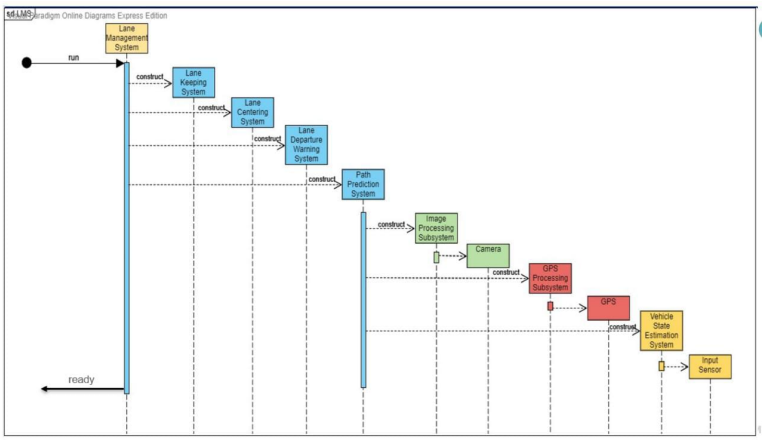

Fig. 11: Scenario for Static Allocation Pattern for RDMS

System and the Path Prediction Subsystem will be constructed and consequently all the remaining composite and primitive objects will be created. Once, all of them are created and initialized, the ready message will be returned ensuring a successful allocation of all the needed objects.

5 Resource Pattern

[Sai Krupa Peraka]

The resource pattern deal with the sharing and management of resources themselves and not the memory they use. The management of resources with potentially many clients is one of the thornier aspects of system design, and several patterns have evolved or been designed over time to deal specifically with just that. The priority of the task determines which tasks will run preferentially when multiple tasks are ready to run—the highest-priority task that is ready. This makes the timing analysis of such systems very easy to compute if certain assumptions are not violated too badly. These assumptions are the following.

- Tasks are periodic with the deadlines coming at the end of the periods
- Infinite preemptibility
- Independence

The various Resources patterns are:

1. Critical Section Pattern: Uses resources run-to-completion

2. Priority Inheritance Pattern: Limits priority inversion
3. Highest Locker Pattern: Limits priority inversion
4. Priority Ceiling Pattern: Limits priority inversion and prevents deadlock
5. Simultaneous Locking Pattern: Prevents deadlock
6. Ordered Locking Pattern: Prevents deadlock

Fig.12 shows the Rank-Design table for various resource patterns. We the weightage score for Ordered locking pattern is highest compared to all, these suits the RMDS most.

Design Patterns	Design Criteria			
	Reliability	Continuity	Schedulability	
Weightage	**3**	**2**	**1**	**Weightage score**
Critical Section Pattern	4	6	6	30
Priority Inheritance Pattern	7	5	7	38
Highest Locker Pattern	6	5	4	32
Priority Ceiling Pattern	5	6	7	34
Simultaneous Locking Pattern	8	6	7	43
Ordered Locking Pattern	8	7	7	45

Fig. 12: Ranking in Resource Patterns

The Ordered Locking Pattern is another way to ensure that deadlock cannot occur—this time by preventing condition 4 (circular waiting) from occurring. It does this by ordering the resources and requiring that they always be accessed by any client in that specified order.

A resource may only be locked if its resource ID is greater than the largest resource ID of any locked resource. The mutex locks the resources, performs the functions, and then unlocks the resources. Use of list to prevent deadlock is possible with ordered locking pattern. Fig 2 shows the flow of control for such a monadic Access Function.

While resource is locked when the system is performing some function, then fear of deadlock arises where resource id and access will be denied. in this situation design timed analysis ensure that all the systems have unique id, so that an opportunity to lock the resource upon request and release after work is done. the list ensures that resource is not hold for longer duration due to deadlock prevention mechanism (like this we minimize the lock rejection).

Why this pattern suits best for Road Departure Mitigation System are as follows:

1. Prevents Deadlock Condition
2. Uses array for holding resource ID’s

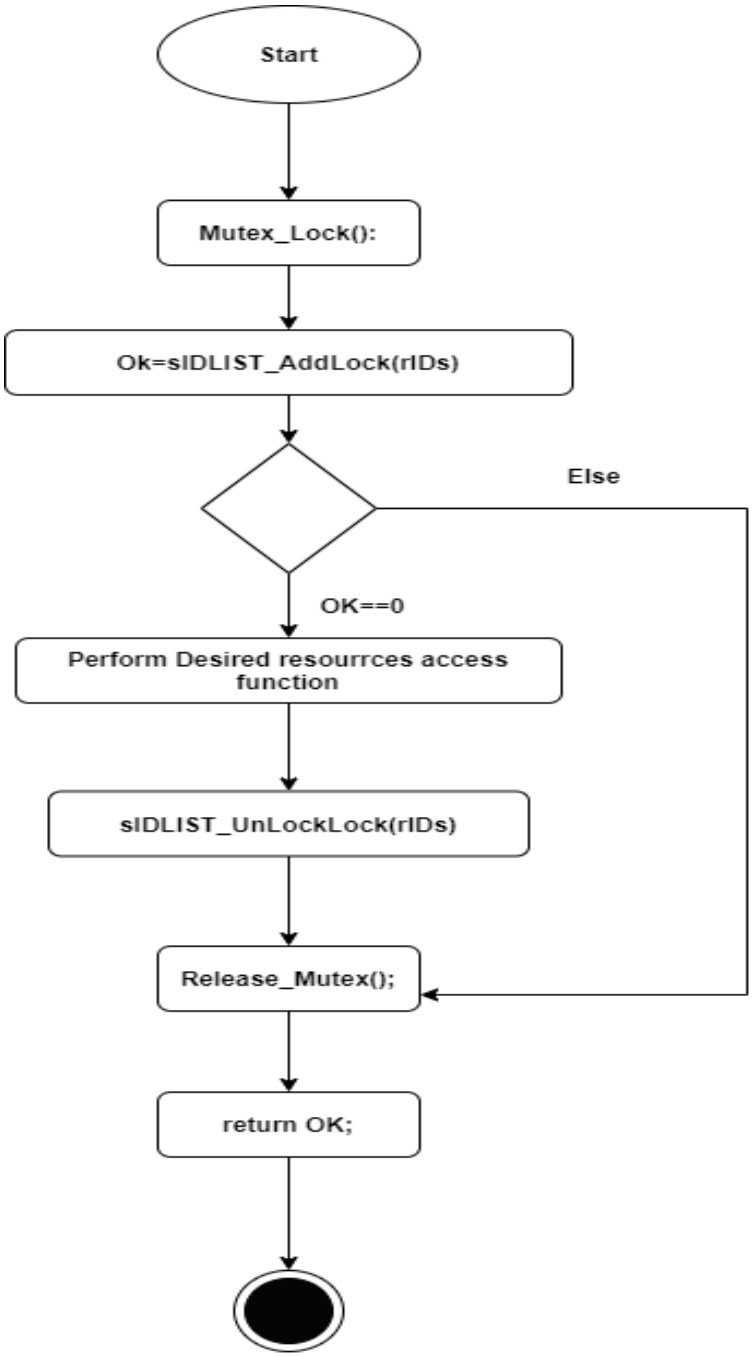

Fig. 13: Flow Chart

5.1 Example Model

The Example model in fig 13 and fig 14 shows the flow chart and respective class diagram. The flow shows how the mutex helps to lock the system using sID's and AddLock method then the system continuous to check if the resource is locked or not. If the system is not occupied, then the resources perform the desired function then it requests to UnlLock ID then Mutex releases the resources. If the system is occupied, then Mutex is requested to release the resources.

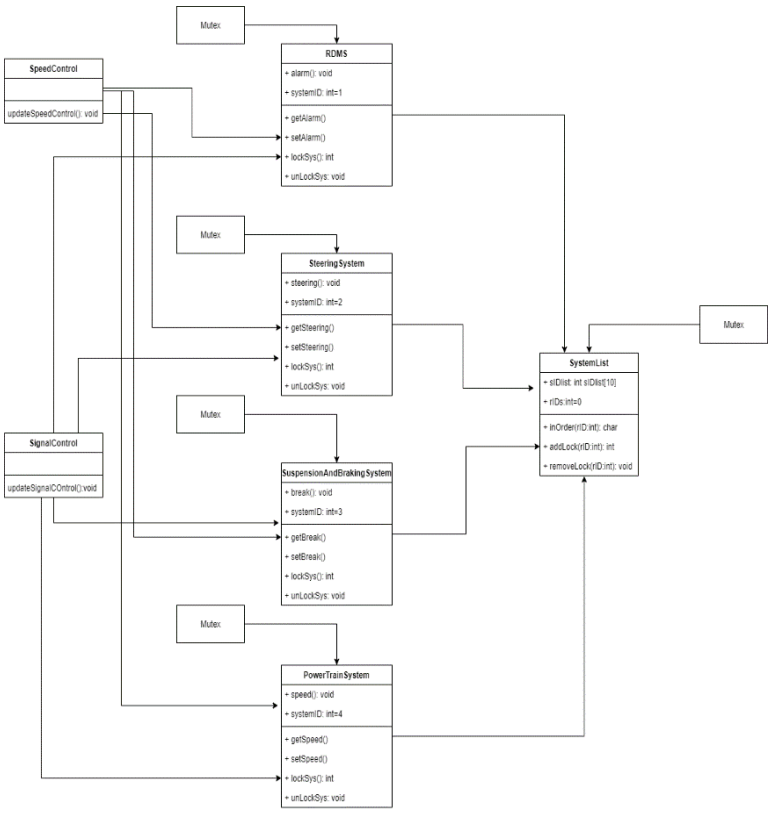

Fig. 14: Ordered Locking Pattern Structure in RDMS

Here in fig 15, the system list holds all the id in an array of the resources in RDMS and then mutex helps to lock the system using system id given to each resource. Each server is hold related, but distinct information. RDMS has information about the alarm. Steering system has the information about the steering and torque to prevent the unnecessary movement. SBS the information about the suspension and braking system. Power train has the information about the speed of the vehicle. The diagram has two different clients, Speed control needs to speed coherent about the state of the vehicle to control speed and safety of the system by

indicating the user. The Signal control needs the same information to prevent the automatic braking and to keep the safety. Each server has its own mutex in the diagram, this is because to avoid the overlaying of number of lines in the system.

6 Distribution Patterns

[Sai Krupa Peraka]

It defines the policies, procedures, and structure for systems that may, at least potentially, exist on multiple address spaces simultaneously. Distribution comes in two primary forms: asymmetric and symmetric. Asymmetric distribution architectures are those in which the binding of the objects to the address spaces is known at design time. Most real-time and embedded systems are of this kind because it is simpler and requires less overhead. Symmetric distribution architectures are dynamic in the sense that which address space in which an object will execute isn't known until run-time. While this is more complex, it is also a great deal more flexible and allows dynamic load balancing.

We deal with distribution architectures at the application level, which we refer to as collaboration architecture. At the application level, the collaboration architecture focuses on how the objects find and communicate with each other without specific regard to the details of how the communication takes place.

Various types of pattern:

1. Shared Memory Pattern: Using multi ported memory to share global data and send messages
2. Remote Method Call Pattern: Synchronous message passing across processor boundaries
3. Observer Pattern: Efficient invocation of services with multiple clients
4. Data Bus Pattern: Providing a common virtual medium for sharing data
5. Proxy Pattern: Using the Observer Pattern between processors
6. Broker Pattern: Sharing services when object location is unknown at design time

The Ranking of the Distributed pattern explains why he had to go with Shared memory pattern compared to all the patterns available. The main criteria to look in the chapter is memory allocation, cost, and shared resources protection. These design criteria suits best to fit for the RMDS as the character helps to develop robust system. The fig 16 explains in detail about the ranking of the system done in distributed patterns.

This pattern uses a common memory area addressable by multiple processors as a means to send messages and share data.

Design Patterns	Design Criteria			Weightage score
	Memory Allocation	Cost	Shared Resources protection	
Weightage	8	7	5	
Shared Memory Pattern	8	9	8	167
Remote Method Call Pattern	7	5	5	116
Observer Pattern	4	6	7	109
Data Bus Pattern	8	4	8	132
Proxy Pattern	4	6	7	109
Broker Pattern	4	4	4	80

Fig. 15: Ranking in Distributed Patterns

The reasons this pattern is the best fit for Road Departure Mitigation System are as follows:

1. It is a highly safety-critical system
2. Data is Shared among one or more processors
3. Data stored is on global access space

6.1 Example

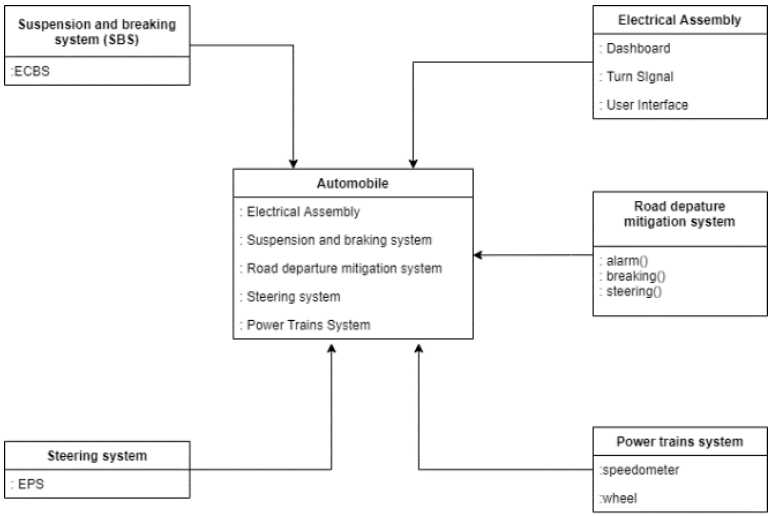

Fig. 16: Block Diagram

The five servers present to the automobile system are SBS, Steering system, Electrical Assembly, Road departure mitigation system and Power train system. These all servers are connected to automobile system where it acts as a shared memory resources for all the other servers. The data in the shared memory is read only access data where the manipulation or lost data may occur, these system helps only read so that no breach is being held during the process. The different data from different servers are been shared among all the other

servers through shared memory also acts as a central shared space. These helps the system to alarm, signal control over speed, torque or vibrate to the steering, if necessary, braking in the emergency cases during accidents. This helps the system to be follow for safety, dependable, reactive, and reliable characteristics.

7 Safety and Reliability Patterns

[Utkarsha Fegade]

Safety could be defined as freedom from accidents or losses. Reliability is defined as the probability that a system will continue to function for a specified period. The safety and reliability architecture is concerned with correct functioning in the presence of faults and errors. Redundancy may be used in many ways to get different degrees and types of safety and reliability.

Various Safety and Reliability Patterns compared during this project are:

1. Protected Single Channel Pattern: Safety without heavyweight redundancy
2. Homogeneous Redundancy: Protection against random faults
3. Triple Modular Redundancy: Protection against random fault with continuation of functionality
4. Heterogeneous Redundancy Pattern: Protection against random and systematic faults without a fail-safe state
5. Monitor-Actuator Pattern: Protection against random and systematic faults with a fail-safe state
6. Sanity Check Pattern: Lightweight protection against random and systematic faults with a fail-safe state
7. Watchdog Pattern: Very lightweight protections and timebase fault and detection of deadlock with a fail-safe state
8. Safety Executive Pattern: Safety for complex systems with complex mechanisms to achieve failsafe states

Fig.17 shows the Rank-Design table for various safety and reliability patterns. As can be concluded from the table, the pattern with the highest weighted score is the Safety Executive Pattern, hence it is the most beneficial pattern for Road Departure Mitigation System.

This pattern is a means to coordinate and control the execution of safety measures when the safety measures are complex. Safety Executive Pattern acts as the superset of different Safety and Reliability Patterns. It provides a safety execution channel that could be used in case a fault is detected in the Actuation channel.

	Design Criteria				
	Cost	Safety	Real Time	Fault Coverage	
Weight	8	9	6	7	Weighted Score
Design solution	Score				
Protected Single Channel Pattern	7	5	2	6	155
Homogeneous Redundancy Pattern	4	6	4	4	138
Triple Modular Redundancy Pattern	2	7	4	6	145
Heterogeneous Redundancy Pattern	2	7	4	7	152
Monitor-Actuator Pattern	7	5	5	5	166
Sanity Check Pattern	7	4	5	3	143
Watchdog Pattern	8	4	8	3	169
Safety Executive Pattern	5	9	8	8	225

Fig. 17: Rank Analysis

The reasons this pattern is the best fit for Road Departure Mitigation System are as follows:

1. It is a highly safety-critical system.
2. It is a hard-real time system, hence adherence to the deadline is of utmost importance. This is taken care of by the use of watchdog in this pattern.
3. The immediate shutdown of the system in case of fault can be dangerous and risk to life.

7.1 Example

Fig.18 shows the block diagram of the implementation of the Safety Executive Pattern to RDMS. In the block diagram, the Mounted camera works as the input sensor for the Actuation channel. The actuation channel contains components that perform the end to end actuation required by the system. The camera input processing block is responsible for initial formatting for transformations necessary by the Mounted camera. Camera data processing, process the sensing data in a sequential pattern to compute the ultimate actuation output. Camera output processing is the device driver for the actuator responsible for final formatting. Timebase is the independent timing source used to drive watchdog. Watchdog waits for a stroke event sent by actuation channel components. If it occurs within an appropriate time frame, watchdog may command integrity checks. If it does not, then it shuts down the actuation channel. Safety coordinator controls and coordinates safety processing. There can be only one coordinator per Safety Executive component. Safety Measure controls the detailed behavior of a single safety measure. Safety Executive may contain multiple measures. Safety Policy specifies a policy or strategy for a Safety Coordinator. It may involve a potentially complicated sequence of steps involving multiple Safety Measure objects. If a fault is detected in the Actuation channel, the fail-safe actuation would come into the picture and the Warning System would be actuated with the help of RADAR input instead of Camera.

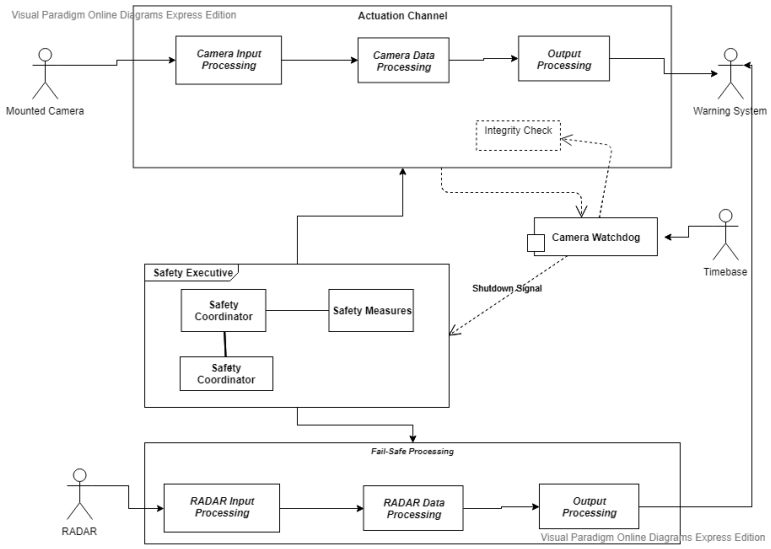

Fig. 18: Block Diagram

8 Conclusion

In this paper, We presented the RDMS as a real-time design approaches the fundamental of the software design patterns. More specifically we related the basic functionality of the RDM system in software patterns. In this way, we contribute to the different engineering solution to the system how different types of ideas can be approached. We have increased consistency of the design and software engineering. However, the rational output has not been presented with the designed patterns but we expecting it performs as expected.

The RDMS model also supports the patterns which are not explained in detail. But, the point here is to select the specific pattern from a different domain to make the system robust, reliable and safety by means of ranking.

9 References

[1] Wulf, Wm and McKee, Sally. (1996). Hitting the Memory Wall: Implications of the Obvious. Computer Architecture News. 23.

[2] Mohammad Zahran, CSCI-GA.3033-012 Multicore Processors: Architecture and Programming, Lecture 3, New York University

[3] Noble, James, and Charles Weir. 2001. Small memory software: patterns for systems with limited memory. Harlow, England: Addison-Wesley.

[4] MIRA Ltd (2004). MISRA-C:2004 Guidelines for the use of the C language in Safety Critical Systems MIRA (Technical report, Motor Industry Software Reliability Association)

[5] Douglass, Bruce. (2003). Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems.

[6] Bruce Powel Douglass (2010) - Design Patterns for Embedded Systems in C- An Embedded Software Engineering Toolkit-Newnes

[7] RDMS: https://www.mattcastruccihonda.com/blog/how-does-hondas-road-departure-mitigation-system-work/

10 Affidavit

We (Mariam Jamal, Sai Nishanth Adingi, Utkarsha Fegade, Sai Krupa Peraka) herewith declare that we have composed the present paper and work our self and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form has not been submitted to any examination body and has not been published. This paper was not yet, even in part, used in another examination or as a course performance.