Rewrite Based Software Requirement Engineering for Signaling Systems Safety

[1]CHANDRASEKARAN SUBRAMANIAM, [2]VELAYUTHAM PAVANASAM
[1]Computer Science and Engineering, [2]Computer Applications
Sathyabama University
Chennai, Tamil Nadu
INDIA
[1]chandrasekaran_s@msn.com, [2]velayuthamp@gmail.com

Abstract: - The objective of the paper is to propose a rewrite based software requirement engineering model focusing the safety specifications of a distributed signaling system. In safety critical applications, the fault tolerance features can be utilized in the requirement engineering to enhance the software safety through terms, rules and equation rewriting. A formal specification based approach is used to concentrate on the operations that govern the safety of software system from its initial state to the point it reaches the final safe and desired states. The requirements for railway signaling safety is elaborated by the notion of controlled failures that should not have any catastrophic consequences and the system should be fail-safe. The system is modelled to control the behaviour of distributed software components through safe operations to improve the error containment. The assertions for safety and correctness requirements are made by rewriting the needed terms and rules within the concurrent system thereby reducing the total system safety failure rate.

Key-Words: - Software safety, fail-safe, safety requirement engineering, rewrite rules, controlled behavior.

1 Introduction

The software safety must be considered in the analysis, design and development phase of software life cycle and continues during the maintenance stage of the software. The safety requirement analysis should specify the safety constraints that are implicitly time sensitive and process sensitive especially in the case of distributed systems. The earlier software safety models addresses the fault tolerance features which imply that the software system can recover from or in some way tolerates the errors and continues normal permitted operation. But the safety implies that the system either continues its correct operation or fails in a safe manner. The design diversity, check pointing and exception handling have been utilized in the safety critical application. The design diversity technique is used to tolerate solid software faults while check pointing tolerates soft software faults for task continuity. The exception handling avoids system failure at the expense of current task loss. The safety is concerned with failures of any nature that result in improper control functions that could directly or indirectly cause a hazardous condition to exist [1].The fault tolerance is primarily concerned with runtime prevention of failures in any shape or form (including prevention of safety critical failures). A fault-tolerant and safe system will be of minimize form (including prevention of safety critical failures). A fault-tolerant and safe system will minimize overall failures and ensure that when a failure occurs, it is a safe failure [2]. The main focus of this work is a fine grained approach to software safety rather than system safety. Hence in any software system safety model, the software hazards must be identified during the analysis phase. The probability of occurrence of hazards and their severity should be analyzed to incorporate a set of enhanced safety specifications for safe software. Based on the specifications, the design criteria must be followed to ensure safety that may have many constraints which creates negative impact on safety performance. So the safety rules and functions should be rewritten in each and every software component as demanded by the system in order to pass the parameters in a controlled manner and at the same time the hazard and risk management policies are to be referred.The goal of the software safety analysis is to ensure that the application software with its safety related requirements specifications is sufficiently satisfied that assume the safe behavior in all circumstances in which the system will operate [3]. The software safety analysis should provide the necessary input to software design and development stage such as safety design requirements, implementation

recommendations, or design changes that can be incorporated into the existing software with minimal impact with the help of a rewrite system. The safety must be checked mainly for synchronization in communication problems rather than design complexity problems. The use of Specification and Description Language (SDL) allows exploring the behavior of the different processes in a distributed system components and their communication problems. The unified modeling language (UML) doesn't have the necessary level of formality to provide the necessary verification facilities, which can make it a candidate formal specification notation suitable for high Safety Integrity Level (SIL) systems [4]. To achieve system safety through hardware systems, the redundancy and diversity are the most common ways to reduce system hazards. For software intensive safety-critical systems, the software design must satisfy safety constraints . It is not necessary that only the active code may create hazards but the dead code can also lead to a safety failure. Based on DO178B, the European Software Standard ECSS-Q-80-01 requires that dead code can be removed and an analysis needs to be performed to assess the effect and need for verification [5].The different fault tolerant mechanisms are incorporated to tolerate the safety faults in different levels like the device level as well as the network level in the case of pervasive applications. As in the case of a distributed traffic control software, the safety model by the system components have to be analyzed so that the same model can be extended to other vehicular networks [6]. Today, reliability and safety is realistically achieved through fault tolerance which is a collection of highly specialized measures such as redundancies, recovery and protection mechanism etc. to assure its attainment. These redundancies come in different forms. They may include redundancies in hardware, redundancies in software and employment of diverse tools and methodologies during the development process [7]. Hence the form of needed redundancy for safety can be decided through a process flow in the requirement engineering phase. The main focus of the work is to propose a requirement engineering process model for safety through the use of rewrite rules and equations available in the rewrite engine. The various inter process communications are established while managing the risks and hazards due to improper operations on the software.

The organization of the paper is as follows: Section 2 proposes the safety requirement engineering processes through periodical rewriting of rules, conditions and equations pertaining to the safe operation of the software. Section 3 discusses the various types of safety faults within the different software modules that may cause improper functioning of the system and section 4 elaborates the specification based computational model for software safety requirements. Section 5 explains the above safety model for the distributed railway signaling system using rewrite logic represented in Maude specification language and section 6 focuses the various hazards and the corresponding design criteria and action to avoid or minimize the safety risks with some sample scenarios. The above work is again modeled and simulated for its timely response towards any safety incidents in the case of train passage between stations using Timed Petri Net.

2 Safety Requirement Engineering Processes

The system safety can be viewed as one of the important qualities resulting from the collective behavior of its heterogeneous components and its run time architecture in response to the faulty operational environment. The safe design of a distributed system is possible only if the expected risks due to the malfunctioning of various hardware and system software components are considered and specified during the initial stages of the system development cycle. At the same time, the improper usage or non conformance to the requirement specifications may lead to software or the system crashes. The safety and correctness properties of individual components are to be checked during the analysis phase after the requirements have been transformed into formal specifications. The steps involved in the proposed safety requirement engineering model for distributed system are given below:

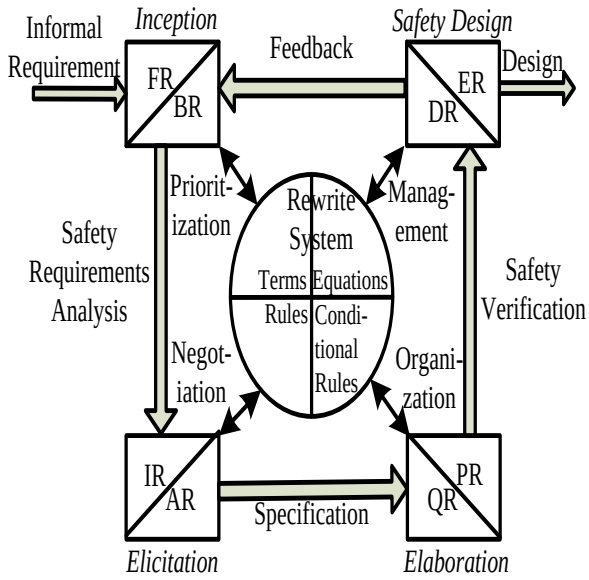
Step 1: *Inception for Safety*: In this phase, identification and categorization of user safety requirements in terms of its system functional and behavioral safety are carried out.

Step 2: *Elicitation on Safety*: In this phase, the identification of system risks is elicited and the risk management activities are performed through the interaction with the rewrite engine in the form of negotiation rules and terms that are to be repeated in safety critical modules.

Step 3: *Elaboration at Functional Safety*: The hazards due to various hardware and application software components are identified and elaborated through safety conditions and equations with the help of hazard management segment and its interaction with the centralized rewrite engine.

ISBN: 978-1-61804-019-0

Step 4: *Elaboration with Behavior Safety*: This phase in the safety requirement engineering process focuses on the safety constraints and consequences of the system through the activities in rewrite system through conditional equations.
Step 5: *Feedback for Safety*: After the formal verification of the filtered specification, the model or algebraic based representation of the system that is a mishap model will be drawn during this phase. The mishap model is sent and discussed with the customer to finally arrive at a design model. This phase comprises of safety assessment and management techniques available in the rewrite system as shown in Fig.1.
The user safety is encompassed in the functional and behavioral requirements of the application along with its performance and quality requirements. The application safety is also hidden in each of the interfaces through which a safety design model can be arrived. The rewrite system is represented in the form of rules and equations to manage the possible risks in the system. The final output is delivered as a verified requirement document to develop a safe design for the software application.



FR-Functional Requirements
QR-Quality Requirements
BR-Behavioral Requirements
DR-Design Requirements
PR-Performance Requirements
IR-Interface Requirements
ER-Environment Requirements
AR-Application Requirements

Fig.1 Requirement Engineering with Rewrite Engine

3 Rewrite Based Software Safety

Software safety can be achieved through rewriting of the safety terms and equations in each of the safety critical functions. Especially the safety rules are to be specified and rewritten while verifying the safety design of the application software. The different phases of a rewrite based safety requirement engineering process are shown in Table1. Rewrite logic extends algebraic specification techniques to handle non-deterministic, reactive, and concurrent systems. It is based on the following two observations:
(i) Term rewriting is intrinsically concurrent.
(ii) Rewriting is not necessarily just the operational counterpart of equational deduction, but can be used to specify change in a system.
Therefore non-confluent and non-terminating rewrite systems are meaningful as far as safety is considered. Rewriting logic is logic of change where the dynamic parts of the system are specified by labeled conditional rewrite rules and the static and structural parts are specified by equations [8]. The safety depends not only on the software modules whose functions and operations are performed in a fail-safe mode but also on the well defined data types. That is the software safety can be assured if the operational and axiomatic notations of the safety program construct is understood and safeguarded. The operations that have been exercised as per the safety specifications must establish the correct sequence of state transfer including exceptions since the software modules may contain system modules and functional modules. The system modules contain many rewrite rules which are not equations whereas the functional modules are equations of both conditional and non conditional types and operations. Apart from the local declaration of data types, some modules may import other modules to determine the system functionality. The functional safety can be verified whether the imported functional modules are protected or extended type of the base module. The operators must be checked for safe construction of the functions within a module.

3.1 Type Errors

In this category of software safety errors, the error may occur due to incorrect assignment of the ‘type’ and ‘value’ of the data that leads to fault which in turn creates safety failures. For example in a railway application, the required hardware safety elements are to be mapped under different data category based on their deployment. The physical objects like red lamp signal at the station will be represented as

'*sort*' type whereas the electronic signal between two stations may be treated as '*kind*' type since it may contain some errors due to transmission. The required software safety related data may be a well structured one or it may contain some error related elements. Hence two different data types are to be defined and declared to distinguish them for further processing towards software safety. The path and the direction of the travel are considered as the resultant of many conditional equations to assert the journey to required destination from the given source. The data required for the safety of the users is in an unstructured format and available at the outside and inside of the system. Specific operations are to be performed on the data so that the system safety will be ensured to the maximum. For example, the display board and warning sign board availability may be ensured by the system but not the utility. The needed data types are represented as '*sort*' and '*kind*' types used in Maude language.

3.2 Communication Errors

The control messages between the various software components are safe guarded through a set of local transition rules declared as per the functional modules within the system. The rules that can be applied concurrently to different fragments in the distributed system are declared and rewritten as per the demand or query from the remote object. The ordering of the messages and the attributes passed between one or more components is immaterial since the operations are performed concurrently satisfying the local rules. The multi-set rewriting is performed regardless of the location of the objects and the nature of messages in the distributed system.

3.3 Updation Errors

The objects will be updated and rewritten if a rule applies forcing a corresponding change in the states of the individual objects thereby affecting the system state specified in the equation declaration part of the system module. By a multi-set union operator, the associativity and commutativity properties of the equation can be realized. The system configuration includes the multi-set of objects, messages with a multi-set matching algorithm to select the correct instance of the declared classes. The concurrency and non determinism is achieved through the equations in functional modules and that of the data in system modules. But there is no guarantee that all the rewrite sequences will lead to the same output state.

Table 1 Safety Requirement Engineering Phases

Safety Requirement Phases	***Safety Activities***	***Interface to safety Rewrite Engine Components***
Inception for Safety	Identify user Safety FR/BR	Priority Terms & types
Elicitation on Safety	Identify system risks IR/AR	Negotiation Rules for Risk
Elaboration at Functional Safety	Hardware and application software	Organization conditions
Elaboration with Behavior Safety	Safety Constraints	Conditional equations
Feedback for Safety	Mishap Model	System updating invariants

4 Computational Safety Requirement Specification Model

The safety engineering may be considered as a set or domain with the specification for those requirements as its elements. Since these requirements will be a mixture of both hardware and software safety requirements, it may be grouped into hardware safety requirement cluster called 'H' and the software safety requirement cluster called 'S'. These two requirements are to be synchronized to achieve the overall system safety requirements 'Y'. These requirements are specified and applied with the help of rewrite system so as to realize a safe system operation.This may be represented as an algebraic equation involving multi set of objects and rewrite rules as shown in Equations below:

$$Y_{safety} = H_{safety} * S_{safety} * U_{safety} \quad (1)$$

$$\forall h_i \in H\ (\exists s_j \in S) \text{ for } \forall u_k \in U,$$

$$1 \leq i < n, 1 \leq j < m, 1 \leq k < l \quad (2)$$

$$\forall s_i \in S\ (\exists u_j \in U) \text{ for } \forall h_k \in H,$$

$$1 \leq i < n, 1 \leq j < m, 1 \leq k < l \quad (3)$$

For a safety critical application like railway signaling, the passengers or people should be aware of the various warning messages, sign board

displays and passenger instructions. Hence the system safety can be effectively achieved by the specifications mentioned in another proposed set called 'user ware', U. Most of the track accidents or mishaps are occurring due to the behavior of the people without giving importance to the safety instructions or violating these warnings leading to hazards. Hence it is appropriate to include a set of safety specifications to be followed by the user to ensure safety. Assuming there are n, m and l hardware, software and use ware safety requirements respectively, the system safety can be represented as

$$\forall u_i \in U\ (\exists h_j \in H) \text{ for } \forall\ s_k \in S,$$

$$1 \leq i < n, 1 \leq j < m,\ 1 \leq k < l \qquad (4)$$

Assuming equations (3) and (4) are satisfied , meaning that the hardware errors and user errors are fixed and if there is no dead code, then the Software Safety Index (SSI) can be determined in the corresponding requirement engineering phases as given below.

User Safety Index (USI) @ Inception

$$= \frac{\sum_{i=1}^{U_c} M_{HM} * w_i\ [(n_{ey} + n_{ex}) + (n_{sp} + n_{ss})]}{n_{errors}} \qquad (5)$$

Where U_c is the number of concurrent users, M_{HM} is the hazard management factor based on the safety policy. w_i is the weightage factor for user safety operation, n_{ey} is the number of safe entry operation, n_{ex} is the number of exit operation, n_{sp} is the number of safety process, n_{ss} is the number of safety specification and n_{errors} is the total number of errors in safety operations. The number of 'entry' and 'exit' operation are monitored in the rewrite engine and the numbers of specification terms are corrected within the engine.

Component Safety Index (CSI) @ Elicitation

$$= \frac{\sum_{j=1}^{C} M_{RM} * w_j\ [(n_i + n_{sp})]}{n_{errors}} \qquad (6)$$

Where C is the number of components, M_{RM} is the risk management factor based on the risk policy. w_j is the weightage factor for component operation, n_i is the number of safe interfaces, n_{sp} is the number of safety operations and n_{errors} is the total number of errors in safety operations.

Functional Safety Index (FSI) @ Elaboration

$$= \frac{\sum_{k=1}^{D} M_{KM} * w_k\ [(n_{sc} + n_{ca})]}{n_{errors}} \qquad (7)$$

Where D is the number of design criteria, M_{KM} is the knowledge management factor. W_k is the weightage factor for design criteria, n_{sc} is the number of safety constraints and n_{ca} is the number of corrective action. The number of 'design criteria' and 'safety constraints' are monitored in the rewrite engine and the number of corrective actions as specification terms are counted within the engine itself.

5 The Safety and Standards in Railway Signaling System

The Indian Railway that has more than 4,015kilometers of track and an electric length of 20,059 kilometers is facing many safety failures for the past two decades. The failures are mainly due to non compliance and non-standard hardware and software components and their installations. Apart from the technical faults, the people or passengers are not giving importance to the railway department's repeated warnings and instructions. As mentioned earlier, the three components hardware, software and user ware are to designed and developed focusing the safety in the railway signaling systems. For example, sometimes passengers are seen on trains hanging out windows and even on the roof creating safety problems. The interior of many train compartments are poorly maintained from rust, dirt and common wear and tear. Although accidents such as derailment and collisions are less common in recent times, many are run over by trains, especially in crowded areas. Indian Railways have accepted the fact that given the size of operations, eliminating accidents is an unrealistic goal, and at best they can only minimize the accident rate. Human error is the primary cause, leading to 83% of all train accidents in India. While strengthening and modernisation of railway infrastructure is in progress, much of the network still uses old signalling and has antiquated bridges[9]. While accident rates are low - 0.55 accidents per million train kilometre, the absolute number of people killed is high because of the large number of people making use of the network. Lack of funds is a major constraint for speedy modernisation of the network, which is further hampered by diversion of funds meant for

infrastructure to lower-prioritised purposes due to political compulsions. In order to solve this problem, the Ministry of Railways in 2001 created a non-lapsible safety fund of ₹17,000 crore (US$ 3.9 billion) exclusively for the renewal of overaged tracks, bridges, rolling stock and signalling gear. A black-box testing scheme based on the current status is presented and key techniques to generate dynamic decision table are also introduced. Any kind of computer interlocking control software can be tested effectively by the universal testing platform demonstrated in this paper. And testing results of the software of 10 stations are given. It is verified that the testing platform is useful in discovering software faults, improving the reliability and safety of software control, administrating the quality of development and production of railway computer real-time control systems [10]. The goal of future signaling systems can be stated as to maximize the utilization of the rail transportation infrastructure and to remove the signaling system as a constraint on rail system operations, while still providing for the high level of safety of train movements expected by passengers [11].

The safety requirements of the software must encompass the needs and constraints of the stakeholders of any railway signaling system starting from the signaling officers, network managers, lane operators, railway station masters and passengers.

The CENELEC EN 50128 Standard that is part of a group of related Standards as in EN 50126 and EN 50129. The former addresses system issues while EN 50129 addresses the approval process for individual subsystems which may exist within the overall railway control and protection system, particularly to specify the safety functions allocated to software. The EN 50128 Standard concentrates on the methods which need to be used in order to provide software which meets the demands for software safety integrity levels (SILs): the more dangerous the consequences of a software failure, the higher the software safety integrity level will be [4]. The design and development of safety critical software is to be carried out as per AELC CE-1001-STD REV and EIA SEB6A. For the programmable device safety is addressed in ANSI/IEEE 7-4.3.2 which is the safety standard for application criteria for programmable digital computer systems in safety systems and ANSI/UL 1998 is the software standard for software in programmable components. BS EN 50128:2001 is the safety standard for railway applications. IEC 61508-1 to IEC 61508-6 are the safety standards for functional safety of electrical or electronic or programmable electronic safety-related systems. It is important to note that the software safety depends not only on the operational syntax and semantics of the program construct but also on the executing environmental factors. These factors may be of physical, logical or hardware functioning environment like devices including transmitters and receivers or silicon chips or the information technology equipments like servers. The software safety can be nearly achieved firstly through the basic safety planning and scheduling stage of the software and then in the design and deployment stage in the case of distributed system like railway signaling systems.IEC 60880 is the safety standard for software for computers in the safety systems say, nuclear power stations whereas IEC 60950-1 is the safety standard for safety of IEC 62304 is the safety standard for medical device driver software incorporating the software life cycle processes. IEEE 1228 is the safety standard for software safety plans whereas ISO/IEC TR 15026 is the safety standard for systems and software assurance. The secured communication between segments or stations are to be as per ISO/IEC 27002:2005 is the safety standard for code of practice for information security management. With so many enriching standards, the signaling system is facing safety failure problems. The proposed rewrite based safety requirement model can be applied in all systems at different stations with a minimum change in the rules and constraints. It can be easily networked since it is based on functional programming and assertions can be made within the modules during the run time.

5.1 Safety Specifications in MAUDE

The system is modeled using Maude to control the behavior of the distributed software components towards safe operation so as to improve the error containment. The *sort* and *kind* refers to the keywords in Maude language for the representation of the undefined paths to destination. The conditions that are to be satisfied in order to reach the destination may be represented as shown in the box below:

```
Var S1 S2: Segment
Cmb (S1: S2): Segment
 If target (S1) = Source (S2)
 // follow the train path //
Path = STATION1 STATION2
```

Each train will be represented as a Maude entity to have the following details as shown in below box:

mod RAIL-SIGNALING is
pr time-schedule
incCONFIGURATION
Class identifier, Object identifier, Attribute Set
sorts Station Train Signal Segments .
sorts Station_staattribute StaAttributeSet .
sorts Train_traattribute TraAttribute Set .
sorts Signal_sigattribute SigAttribute Set .
sorts Driver_sigattribute DriAttribute Set .
subsorts Attribute < AttributeSet .
Operation Declaration
op Station : → Cid [ctor] .
op Train : → Cid [ctor] .
op Signal : → Cid [ctor] .
op Segment : → Cid [ctor] .
Multiset Union operation Declaration
op StaAttributeSet SigAttributeSet → TraAttributeSet
[ctor assoc comm id: none] .
ops permitPass signalStop : Boolean → Display[ctor] .
ops permitEnter permitLeave : Boolean → Display[ctor] .
ops reserveSegment releaseSegment : Boolean → Broadcast[ctor]
.
ops listenSignal broadcastSignal → Message[ctor]
ops alertDriver → Message[ctor]
vars St Tr Si Se :Objectid .
vars Arrivaltime Departuretime Waiting time : Time .
<Train1 : Train |
< number>,<direction>,
<departuretime>,<arrivaltime> >
< Station1 : Station |
<code>,<permittedtrains>,
<departuretime>,<arrivaltime> >

The unconditional rule for broadcasting information about the train can be written as shown below as ***r1 [broadcast]:***

r1 [l]: t => t'
where
l = label.
t = initial state.
t' = final state in specific state.
Sort state.
Op idle broadcast : -> State [ctor] .
Op state :_: state -> attribute [ctor] .
Op train :_: string -> attribute [ctor] .
Op station :_: string -> attribute [ctor].
Op stored :_: string -> attribute [ctor] .
Op tid :_ : rat -> attribute[ctor] .
Op departuretime :-> cid
Vars T dtm u s : string
r1 [broadcast] *: < T : Train | tid : c , departuretime : dtm , Direction : u , Station : s, state : idle >*
⇨
< T : Train | tid : c, time : tm, direction : u , station : s , state : broadcast , stored : " ">

The unconditional rule for receiving information about the train can be written as shown below ***r2 [listen]*:**

r2 [listen] *: < T: Train | tid : c , time : tm , direction : d , station : s , state : idle >*
=>
<T : train | tid : c, time : tm , Direction : d , station : s , State : listen , stored : " " > .

The conditional rule for reserving a track can be written as shown below ***cr1 [reserveSegment]:***

cr1 [reservesegment] *: < T : train | code : c , Track : active , tnduration : t1, tkduration : t2 >*
=>
<T : train | code : c , track : idle > if tnstate : active /\ t1 < t2 .
If the train state is active and the duration of train to access the track is less than the track duration to be idle then track is reserved.

The conditional rule for permitting the train can be written as shown below ***cr2 [permitEnter]:***

cr2 [l] : t => t' if cond
l = label.
t = initial state.
t' = final state in specific time if cond.
cr2 [permitenter] *: <T: Train | tid : c, tstate : idle, Requester : r, tnduration : t1 , tkduration : t2 >*
=>
< T : Train | tid : c, tstate : active > If Trackstate : idle /\ t1<t2 .
Tstate: initial waiting for a track to permit.
Requester: requesting train.
Tnduration : t1 : duration of train to pass the specific track.
Tnduration : t2 : duration of track when it is idle.
Trackstate : status of the track (idle or active)
t1 > t2 : duration of train to pass the track is less than the duration of track when idle.

ISBN: 978-1-61804-019-0

The conditional rules for alerting driver can be written as shown below ***cr3 [alertDriver]:***

cr3 [speedalert] *: <T : train | tnspeed : s1 , acceleration : on , alaram : off , highspeed : s2 >*
=>
<T : train | acceleration : off , alaram: on>
If s1>=s2
Tnspeed : s1 : initial train speed .
Acceleration : on : acceleration is increased.
Highspeed : s2 : threshold speed.
s1>=s2 : if train speed is greater than or equal to threshold speed.
cr3 [objectalert] : <T:train | tnspeed: s1, acceleration : on, obstacleobject : false, alarm : off >
=>
<T : train | acceleration : off , alarm : on >
if s1 > 0 , obstacleobject : true .
if the train sense any obstacle object the acceleration should be off.

Table1 Safety functional requirements assertions

Safety Assertion	From	To	Path Name	Path Notation
A1	Reserve Segment	Request Segment	Display Train Attribute, Display Track Attribute	2 → 1.2 → 1.1 → 1
A2	Permit Enter	Reserve Segment	Listen Signal, Store OR Listen Message, Store OR Listen Broadcast, Store	3 → 2.4 → 2.1 → 2 OR 3 → 2.4 → 2.2 → 2 OR 3 → 2.4 → 2.3 → 2
A3	Permit Pass	Permit Enter	Display Arrival Time, Display Waiting Time	4 → 3.2 → 3.1 → 3
A4	Permit Leave	Permit Pass	-	5 → 4
A5	Release Segment	Permit Leave	Display Message, Display Broadcast	6 → 5.2 → 5.1 → 5

Table2 Safety behavioural requirements assertions

A6	Alert Driver	Release Segment	Display Arrival Time, Display Waiting Time	7 → 6.2 → 6.1 → 6
A7	Obstacle Report	Alert Driver	-	8 → 7
A8	Permit Rewrite	Obstacle Report	Permit Speed OR Listen Signal	9 → 8.1 → 8 OR 9 → 8.2 → 8
A9	Permit Rewrite	Request Segment	All nodes	9 → 1 If all the above paths are safe and correct

Table3 Safety Checks for fault free requirements

Path Notation	Type of Errors	Safety Check Mechanism
2 → 1.2 → 1.1 → 1	Type error, Value error	Sort or Kind Check
3 → 2.4 → 2.1 → 2 OR 3 → 2.4 → 2.2 → 2 OR 3 → 2.4 → 2.3 → 2 OR	Communication error, Timing error, Updation error	Rules, Function, Object
4 → 3.2 → 3.1 → 3	Timing error	Condition Rules check
5 → 4	Function error	Function check
6 → 5.2 → 5.1 → 5	Declaration error	Format check
7 → 6.2 → 6.1 → 6	Timing error	Condition Rules check
8 → 7	Function error	Rules Check

The rewrite based distributed railway signaling system is modeled using Timed Petri Net for various safety checks against communication errors and timing errors. The system is simulated with one station having multiple track segments and another with multiple stations having multiple trains as shown in Fig 5 and Fig 6 respectively. The requirements are managed using the term rewriting and conditional rewriting along with the supportive management modules and tabulated in Tables 1 and 2. The requirements are asserted for the basic functionality of the system and also for its safety behavior using interacting state machine model as shown in Fig 4. The rewrite based requirements are satisfied and managed with risk management techniques.

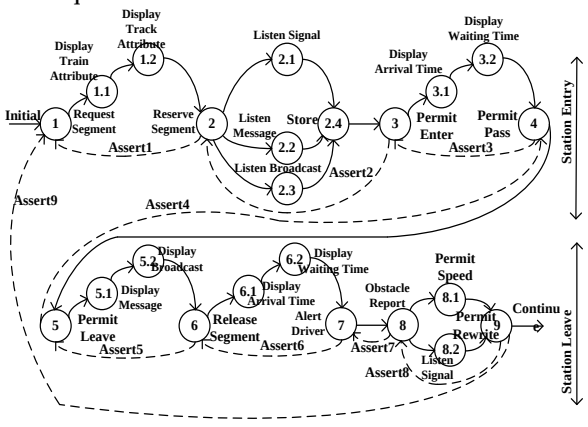


Fig.4 Safety Behavioral Requirements Assertions

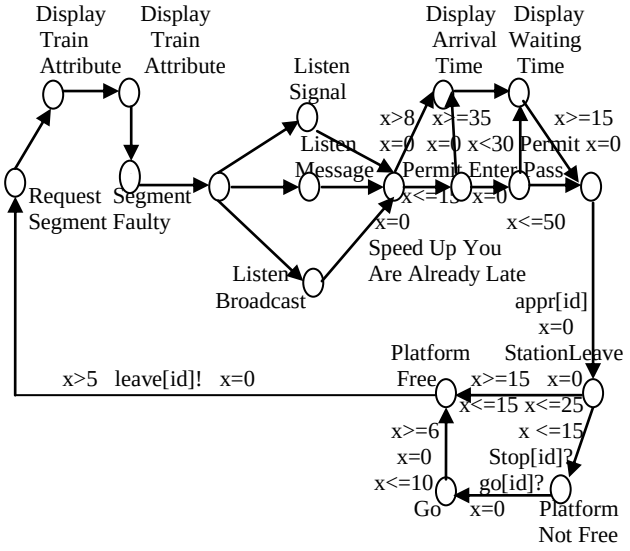


Fig.5 Rewrite based Timed Automata for single train with multiple tracks

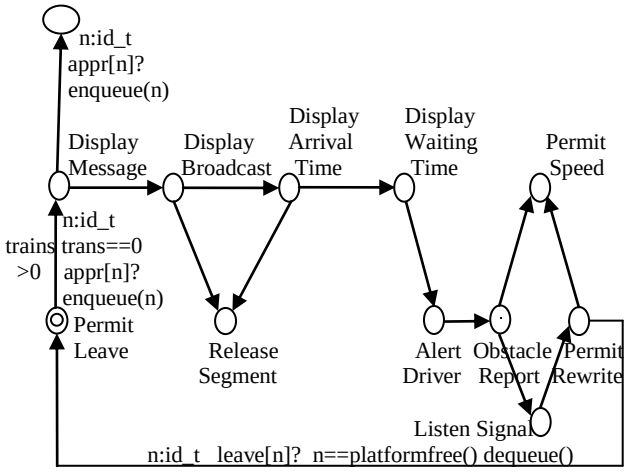


Fig.6 Rewrite based Timed Automata for multiple trains with multiple tracks

6 Conclusion

The railway signaling system requirements are identified and categorized into hardware, software and user ware requirements and engineered so as to ensure safety of applications. The user ware requirements in terms of awareness and the regulations are more dominating in the system safety and the proposed rewrite techniques minimize the safety risks. The requirement phases are focused towards safety starting from the inception to feedback stage which is more applicable even in agile software developments. The MAUDE specifications are verified for using different equations, rules and conditions. The signaling system is modeled and simulated using timed automata for various safety checks during the track reservation and late arrival time conflicts between multiple trains. The functional and behavioral requirements are engineered and also managed through a number of supportive management functions. The safety is ensured once the requirements based on the hardware and infra structure is met. The software safety requirements are asserted in the simulation model of a distributed railway system. The size or the capacity of the individual stations and the frequency of trains are the limiting factors in the proposed requirement model but these may be incorporated in the rewrite engine. The application is being a concurrent networked application, the inter operability and the portability factors are also the other limitations in the work. The safety in any signaling software application will not evolve by simple add on functions but emerge from the software requirement inception phase to safety design and continue till the software maintenance phase through safe coding.

References:

[1] Software Safety, NASA Technical Standard, NASA-STD-8719.13A1. 1997.

[2] Jeffery Voas, "Fault Tolerance", *IEEE on Software*, Vol.18, No.4, 2001, pp. 54-57.

[3] S.Chandrasekaran, T.J.Madumathy, M.Aparna, "A Safety Enhancement Model of Software System for Railways", *International Conference on System Safety*, London, 2009.

[4] A.Felleca, E.Spinicci, "Modeling and Validating a Multi-Configuration Railway Signaling System Using SDL", Electronic Notes in Theoretical Computer Science, Vol.82, Issues 6, 2003, pp. 66-76.

[5] John Favoro, "Issues in Object Orientation and Software Safety", *International Workshop on Software Reuse and Safety, Italy, 2006.*

[6] S.Chanadrasekaran, H.Ravi, S.Aishwarya, M.Dipesh "A Fault Tolerant Pervasive Model for Intelligent Transport System", *International Conference on Computer Technologies in Agriculture Engineering*, China, 2010.

[7] A.Chakaraborthy, "Fault Tolerance Safe System for Railway Signalling", *World Congress on Engineering and Computer Science (WCECS), U.S.A.*, 2009, pp. 1177 – 83.

[8] O.Peter, M.Sigurd, "Specification of network protocols in rewriting logic", *Norwegian Conference of Informatics*, Norway, 1998.

[9] G.Amulya, "A poor track record", http:// www. frontlineonnet.com/fl2015/200306911900.htm.

[10] Fangmei Wu, Meng Li, "Railway Signaling Safety-Critical Software Based on Decision Table", Asian Test Symposium , 1999, pp.247.

[11] A.Rumsey, "What can signalling do to rail operations",http://www.irseitc.net/index.phpoption=com_content&view=article&id=78, 2010.