Methods, notation and tools for modeling Command and Control Systems

“Metodi, notazioni e strumenti per la modellazione di sistemi di commando e controllo”

Imad Zaza

Attività svolta nel progetto RAISSS: RAILWAY SIGNALLING: SAFETY AND SECURITY

POR CReO FESR 2007 – 2013, LINEA D’INTERVENTO 1.5.a - 1.6, BANDO UNICO R&S ANNO 2012, cofounded by Tuscany region

Referente: Paolo Nesi

Distributed System and Internet Technologies Lab
Distributed Data Intelligence and Technologies Lab
Department of Information Engineering (DINFO)
University of Florence

http://www.disit.dinfo.unifi.it

30/11/2012

Version 1.0

Methods, notation and tools for modeling Command and Control Systems

EXECUTIVE SUMMARY

By the time, computer science and technologies, lie at the heart of our economy, our daily lives, and scientific enterprise.

The railway's domain, being one of the backbone of the world economy, has benefited from this revolution by giving in turn to the academic and to the enterprise research, a wide set of problems to deal with.

One of them is the signaling systems which control and preserve the safety of the transportation.

The introduction of the EN50128 guidelines , issued by the European Committee for Electro-technical Standardization (CENELEC), address the development of "Software for Railway Control and Protection Systems", and constitute the main reference for railway signaling equipment manufacturers in Europe and in future it will be also embraced by other countries.

Formal methods are rated as highly recommended for the specification of systems/components with the higher levels of SIL.

Contextually some European railway companies have constituted a consortium to define a standard interlocking system at a European level: the Euro interlocking project.

Inside this project a trend has developed towards the use of specific formal method such statecharts for modeling interlocking rules because the above cited formalism have been considered suitable to express the sequences of checks and actions typical of an interlocking system.

This report analyze the methods and tools present in the relative literature with the main scope to define the main concerns and past, present and possibly future best practice in developing, verify and validate interlocking software.

The document is structured in the following thematic sections that evolve starting by the domain problem landing to the main objective:

- Introduction
- Domain Problem
- System architecture
- System failure resilience policy
- System software
- Method, notation and tools
- Conclusion

The second section investigate the domain problem with a bird-eye of the main components in terms of functionality and relationship among them. Such components are abstract concepts such signaling and interlocking principles or physic such the trackside elements. The keystone of the above section is the control tables which are the starting point to develop the interlocking logic.

The third and fourth section give a short resume about the architecture point of view, exploiting in structured layers the computational elements involved and a brief tour on the classical solution to preserve safety for a critical system.

The fifth section focuses on the main scope of the showing the State of the Art concerning tools and frameworks for the specification and verification.

The last section explain the method, notation and tools that it will be used in developing Interlocking systems.

Keyword lists

Railway signalization; Interlocking system design; Model Driven Engineering; Formal method; State charts; Ladder Logic; SCADE; PLC; Safety system; SAT solver;

Summary

INTRODUCTION

Railway transportation has many benefits:

- Minimize pollution;
- High density network ;
- Safe movement.

Unfortunately trains cannot avoid things on the track, the way a car can avoid things on the road. Moreover trains cannot avoid things on the track so quickly they need a certain distance ahead of them free of any obstacl e to be able to stop on time.

This is due to their low friction coefficients; during braking action there are limits to the force that can be transf erred to the track. The result is that trains have longer brake distances.

For the sake of clarity, braking from 1000 meters to a full stop from 140 km/h is normal and the above distance is termed the "brake distance".

Therefore trains must run at least one brake distance from each other.

But looking ahead for 1000 meters or during foggy days is quite a challenge so to tell a train driver that tracks ahead is free, signal are placed next to the track.

These signals tell the driver how fast he may go to run his train safe and to ensure he can stop on time. Betwee n those signals it's only allowed to have one train. This is called a "block" or "section".

In the early days of railway signaling, to safeguard the running of trains, railway administrations used timetables and flag officers. However, the increase of the traffic within the railway network corresponds in a growing number of disasters due to human errors so automatic *signaling equipment* became essential.

In such an installation, safety is ensured by creating *interdependences* between the equipment for operation of the points and the equipment for the operation of the signals [1]. These interdependences are called interlockings and from this emerge the name: interlocking system (further on indicated by IXL).

Through the last 150's years circa of railway history there was different types of IXL available like cabin Interlocking System (Mechanical Interlocking), Panel Interlocking System (PI), Relay Interlocking System (RI) and Computer Interlocking System (CIS) sometimes called Solid Sate Interlocking System (SSI) (proposed by Cribbens et al in 1987 for the British railways [2] even if in 1985 it was proposed a microprocessor based in literature [3]). The cabin interlocking system is obsolete and the panel interlocking is slowing becoming obsolete. The relay interlocking system is the widely used system in the world but was slowly being phased out and replaced with SSI's descendent systems.

For the sake of clarity it must be say that the RI is still being used in the small station while the newer systems are used in the main nods of the railway network.

Nowadays , with the development of modern multipurpose computers, second generation processor-based interlockings are known by the term "Computer Based Interlocking" (CBI), which Smartlok (Alstom), HRM9 (ECM), Westlok (Siemens – ex Invensys) are some examples.

The main advantage of the last mentioned systems is that their design can be tested through software simulation before being installed at the trackside.

Conversely, the introduction of microprocessors raises the difficulties of software and hardware safety assessments in order to certify these computer controlled railway systems.

Last years, also, the Movares Eurolocking system has developed interlocking system based on Commercial of the Shelf (COTS) PLC's (commonly used in the process industry). This system was adopted since 2012 by Prorail Netherlands national railway company.

Meanwhile, the specification and validation of system safety is a primary and mandatory task for the approval of railway signaling systems and this implies that the software validation (that have little to do with the problem of interlocking design per se) for such systems are of vital importance.

But the exhaustive testing of railway interlocking systems software is not practical. Further, given the dynamic and continuous nature of the operating environment, some errors are very difficult to reproduce in a testing environment, and as a result faults and malfunctions caused by environmental factors unique to a real system often go undetected.

Indeed, experience has shown that for systems tested and verified in a conventional manner, defects still remain and cause accidents in operation [4].

For that reasons, formal methods have been highly recommended by railway safety standard EN50128 [5].

In related works the formal methods used to establish system specification and verify safety properties were Petri net, CSS, CTL, CSP, B, VDM, SPIN, and lately State Charts.

Although formal methods, such as model checking, can be implemented automatically, they are only applicable to small-scale systems due to the state explosion problem.

On the other hand, the migration in software engineering from Model Driven Approach (MDE) to Model Based Approach reflects on design of IXL with an example in scientific world like tools as UMC (UML on-the-fly Model Checker) [6] .

However in the last decade the railway companies had adopted, to model safety requirements and verify interlocking systems property, special purpose tools like safety critical application development environment (SCADE) , Prover iLock (NP-Tools) of Prover Technologies, IBM Rational Rhapsody.

THE DOMAIN PROBLEM: RAILWAY SIGNALING

What are a railway transportation system and its purpose it is commonly known.

The railway transportation is basically divided in two categories:

- Main line – the oldest;
- Metro line – usually underground.

Today this classification is going to vanish manly because of interoperability.

A consequence as it will be clear further on is that some feature of the metro line it will be carried on the foreground line that is driverless issue.

On the other hand the above system it is making up of many parts of which Signaling is one of the most important.

Next it will be briefly presented the evolution of signaling beginning with the main line.

Long time ago …

For the sake of clarity it will be given an explanation of railway transportation considering one direction of traveling. Further on it will be considered the bidirectional issue as the main “concerns” - safety - is the same.

In the early days of railways transportation there wasn’t any system which the train driver could interact with for state the line ahead. Indeed, the driver have to use his eyes, for example, to get sign of a train in front so could stop before hitting it.

Soon, the railway authorities concluded that the driver’s sighting distance was not enough to prevent accidents.

The motivation varies from human factor as experience to environment factor as bad brakes or tenuous contact which exists between steel wheel and steel rail for traction and braking.

Moreover, The adhesion levels are much lower and vehicle weights much higher on railways than on roads and therefore trains need a much greater distance in which to stop than, say, a car travelling at the same speed.

The next refinement to improve the train driver’s stopping distance was to require:

- time intervals between trains (e.g. 10 minutes);
- Divide the track into sections requiring that only one train was allowed in one section at one time.

The colored flags (red, yellow, green) were used by signal officer to tell drivers how to proceed.

A red flag was shown for the first five minutes after a train had departed. If a train arrived after 5 minutes, a Yellow caution signal was shown to the driver. The full-speed green signal was only shown after the full 10 minutes had elapsed.

The above system introduced was neither safe nor efficient (by the railways' authority point of view) because:

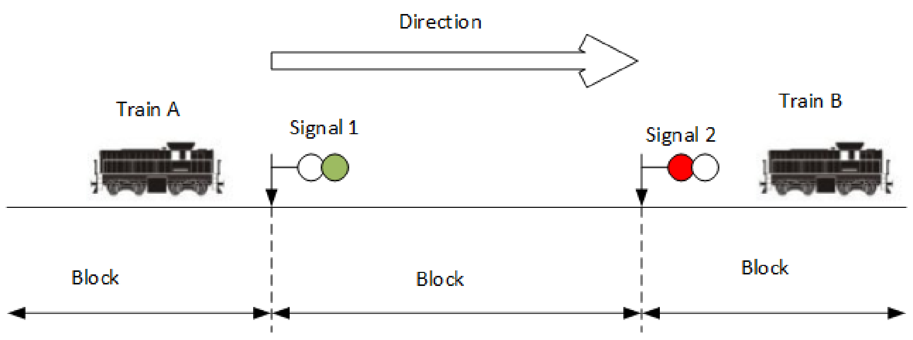
- it didn't consider that a train could broke between stations;
- it require that the trains had same speed;
- The time interval restricted the number of trains which could run per hour over a given line.

As the railways authority needed to run more trains, they gradually began to reduce the time between trains. As they reduced the time, or "headway", the number of trains per hour increased. At the same time too, the number of accidents increased.

The next improvement was fixed signaling where each section (or block as it is often called) is protected by a fixed signal placed at its entrance for display to the driver of an approaching train. If the section is clear, e.g. there is no train in it, the signal will show a "Proceed" indication.

The fixed signal it was usually a raised semaphore arm. The above "Proceed indication it is usually a green light or "aspect". If, however, the section is occupied by a train, the signal will show a "Stop" indication, usually a red aspect.

The next train will be made to wait until the train in front has cleared the section. This is the basis upon which all signaling systems are designed and operated.



Mechanical signals were the first that appeared with signal box with levers controlling remote signals and points.

Originally, the passage of each train through a section was tracked visually by the signalman. When the train had cleared his section, the signalman told the signal box on the approach side that his section was now clear and that he could, if required, "accepts" another train.

The messages between signal boxes were transmitted by a system of bell codes using the electric telegraph.

Compulsory use of the electric "block telegraph" to pass messages and signal interlocking, where points and signals were mechanically prevented from allowing conflicting movements to be set up was introduced soon after.

Another improvement was to use double signals that is a basic stop/go signal used to protect each section of the line was right as long as the driver of an approaching train was able to see the signal in time to stop but as it was stated above the sight of a driver it is not always reactive so "distant" signals was provided.

Distant signals were placed in such a position that the driver could stop in time if the next stop signal was at danger. Positioning depended on the visibility, curvature, maximum permitted line speed and a calculation of the train's ability to stop.

Originally, distant signals were semaphores, like the stop signals mentioned above. They showed a green light at night if their related stop signal was also green (or clear) and yellow if the stop signal was at red.

The red-yellow-green pattern was adopted for color light signals and eventually used to provide a more sophisticated form of train control.

Another safety feature – which the computerized improvement is the matter of this survey – was introduced in the mid-19th Century: mechanical interlocking of points and signals.

The purpose was to prevent the route for a train being set up and its protecting signal cleared if there was already another, conflicting route set up and the protecting signal for that route cleared.

The interlocking was performed by a series of mechanically interacting rods connected to the signal operating levers in the signal box. The arrangement of the rods physically prevented conflicting moves being set up. As the systems developed, some larger signal cabins at complex junctions had huge frames of interlocking levers, which gave the name "lever frame" to the row of operating levers in a signal box.

Eventually, by the time signal levers were being replaced by small (miniature) levers or push buttons, mechanical interlocking frames were superseded by relay interlockings.

Electro-magnetic relays were used in series to ensure the safety of route setting at junctions. Complex "control tables" were drawn up to design the way in which these relays would interact and to ensure safety and integrity.

Conversely as the use of railway transportation on the main line was growing, in the middle of the past century were introduce the metro line.

This type of railway system is differing from the above essentially for this issue:

- many metro routes are in tunnels;
- They are also usually provided with some sort of automatic supervision to prevent a train passing a stop signal.

The metro signaling is based on the same principles as main line signalling but has some difference. For example, the blocks are shorter so that the number of trains using the line can be increased.

Originally, metro signaling was based on the simple 2-aspect (red/green) system. Speeds are not high, so three-aspect signals were not necessary and yellow signals were only put in as repeaters where sighting was restricted.

It has long been the practice of some operators to provide a form of enforcement of signal observation by installing additional equipment. This became known as automatic train protection (ATP). It can be either mechanical or electronic.

The mechanical version is the train stop which consists of a steel arm mounted alongside the track and which is linked to the signal:

- if the signal shows a green or proceed aspect, the train stop is lowered and the train can pass freely;
- If the signal is red the train stop is raised and, if the train attempts to pass it, the arm strikes a "trip cock" on the train, applying the brakes and preventing motoring.

Electronic ATP involves track to train transmission of signal aspects and (sometimes) their associated speed limits.

On-board equipment will check the train's actual speed against the allowed speed and will slow or stop the train if any section is entered at more than the allowed speed.

If a line is equipped with a simple ATP which automatically stops a train if it passes a red signal, it will not prevent a collision with a train in front if this train is standing immediately beyond the signal. Moreover there must be room for the train to brake to a stop.

This is known as a "safe braking distance" and space is provided beyond each signal to accommodate it. In reality, the signal is placed in rear of the entrance to the block and the distance between it and the block is called the "overlap".

Signal overlaps are calculated to allow for the safe braking distance of the trains using this route. Of course, lengths vary according to the site; gradient, maximum train speed and train brake capacity are all used in the calculation.

Overlaps are often provided on main line railways too. In the UK, it is the practice to provide a 200 yard (185 m) overlap beyond each main line signal in a color light installation. Back in 1972 when it was decided upon, it was, after a review of many instances where trains had overrun stop signals, considered the maximum normally required. It was a rather crude risk analysis but it was the best they could afford.

In the US, the overlap is considered so important that a whole block is provided as the overlap. It is referred to as "absolute block". This means that there is always a full, vacant block between trains. It's rather wasteful of space and it reduces capacity but it saves the need to calculate and then build in overlaps for each signal, so it's cheaper.

Basic equipment and some terminology

Within the main line railway network, four types of wayside elements can be distinguished:

1. a track-circuit ;
2. an axle counter;
3. a turnout;
4. A signal.

A section of track is typically broken down into track segments – the above mentioned sections or blocks - each containing one or more electrical devises called *track circuits* to detect the presence of a train.

A track circuit is either cleared indicating no train on the track or occupied indicating the possible presence of a train. Typically track circuit become larger on long straight stretches of track without any interesting topological features such as junctions or stations. Likewise track circuit become smaller around junctions and stations where control over train movement is of greater importance.

An *axle counter* detects the presence and traveling direction of wheels at various points along the right of way. The right of way is broken into “blocks” with wheels (axles) being counted into and out of the block. If the same amount of axles is detected departing the block as were previously detected entering it, the block is considered vacant. Axle counter systems and track circuit systems both provide for track vacancy detection but function in significantly different ways. A study discussing the main advantages and disadvantages in terms of safety in track is [5].

Signals are the main means used to communicate information regarding the state of the track ahead of the train. Typically they are placed either on the track side or overhanging the railway. Visual indications known as aspects are used to convey information to the driver. A signal will have many such aspects which can be displayed, each with a particular meaning.

The colored light signal have between one –four aspects each conveying a different indication about the state of the track ahead.

Below is a description of the aspects used for a three light signal:

- Green - If this aspect is displayed it indicates that track ahead is clear for a sufficient distance and the train driver can proceed at full speed to the next signal.
- Yellow (double) - This aspect indicates the track immediately ahead in between this signal and the next is clear however the driver should proceed with caution as a train could be in the track after that.
- Red - This aspect indicates that the track ahead is not clear, the driver should stop and wait at this signal.

The two - four aspect signals are used on tracks with different speeds to convey different stopping distances. The two aspect signal for instance would be used on a low speed track segment where stopping distances are relatively short.

Whereas the four aspect signal would be used on a high speed line where stopping distances are long and the driver needs information for a greater length of track. These signaling schemes are not fixed from country to country, for example, they may use di different conventions, colors and number of lights on each signal so it will not reported example here.

A *turnout* sometimes called pair of points or points is a physical piece of equipment that is used to form a junction. Due to the nature of the rails and trains it is not possible to physically to just join two segments of track. Instead a point is needed to act as physical switch controlling the flow of trains through a junction. A point has two positions which are referred to as normal and reverse. This presents a safety hazard, for example if a train enters the junction it is locked in the position for normal then the train will be derailed.

Railway control System

The main actor of the railway control system is the interlocking system (IXL) which "orchestrates" the trackside devices discussed earlier. Its job is to apply a set of rules (further on interlocking *principles*) to the requests and commands it receives from the control system (further on *Traffic Control Center*) and check whether or not the future state of the railway is safe. If the information it receives as signals do not violate the safety of the railway then these signals are committed to the physical infrastructure otherwise ignores it (eventually, reports to TCC).

The most common technology used in the past was based on relays, namely Route Relays Interlocking (RRI) because; the above mentioned rules are implemented using relays. These relay based circuits implement all types of interlocking logic, that will be discussed further, and take inputs from above cited equipment (i.e. signals, points and track Circuits) in the form of relays. Despite the fact that relays are phasing out, they are still in use in the small stations and nevertheless they had been of great importance because they were used as start point in developing the newest systems as it will be clarify in this work.

The command to set, monitor and clear the route for the train is taken in the form of buttons by the station master’s console (IECC in UK and ACEI in Italy are some example) or more recently by workstation equipped with control software.

The output of the interlocking logic is also a relay, which in turn drive the signals and turnouts associated with points.

The relays circuits are build using the station control table (sometimes called Control Laws or Conditional Tables) as input which is generally paper document and the interlocking principles as the Business Logic (further on *Interlocking Logic).* The control table rules the possible movements of the train inside a station yard and its relationship with other stations.

Conversely, an IXL when built using electronics replacing traditional mechanical levers and electro mechanical relays is called as Computer Interlocking System or nowadays Computer Based Interlocking. The same control table used in RRI forms the basis here also. The relays used to form the logic circuits in RRI are replaced by software variables.

In the CIS, states of trackside equipment are stored in memory. The core of that interlocking system is divided into two distinct parts: configuration and a generic interpreter of that data.

Configuration data is prepared specially for each signaling scheme. The data contains both information and the application of signaling principles to that information. The source data is compiled into executable data which is interpreted at run-time. The source data is written using a special-purpose programming language which was designed by signaling engineers for signaling engineers.

The above way of working is the fact that the first computerized system recalls the electro-mechanical ones which has created a considerable time lag between the new information technology (e.g. object oriented approach) and approaches applied to systems thus constituting a major constraint for railway industries.

This is also a recurrent aspect in the human life that implies low adaption to the new technologies (the first cars, for example, recall buggies without horses).

The language is constructed from expressions and statements. Primitive constructs of the language consist of tests and commands which are used to read and write to the interlocking memory.

The field inputs are collected using digital input cards and outputs are given using digital output cards. The processing is done by a processor where the virtual relays (software variables) are evaluated using the interlocking equations, which are now in digitized form either as algorithms, Boolean equations or state charts in the processor memory. These algorithms now being executed by the processing unit take appropriate action.

The advancements from tried and trusted electro-mechanical solutions - they implements the proven interlocking rules and also since the relays used in RRI are inherently failsafe (due the fact that they drop to safe state due to gravity even when power supply is not available or in any kind of malfunction) - to modern functional equivalents of today was principally due to the next facts:

- improved availability because redundant architectures as it will be presented further on provide inherent redundancy through switchover to duplicate systems – both processor and input/output modules;

- enhanced maintainability as the provision of extensive logging and diagnostics facilities comes as standard across most contemporary products and the most modern solutions provide predictive diagnostics to target behavior trends, eliminating potential failures;

- reduced equipment footprint as for example, in a typical four road station the number of relays used to implement this type of logic would in the order of 1000 relays and wiring is so complex that the time taken to install and commission a RRI is very long. The testing of the system requires the total station to be setup and testing done during normal train operation.

- increased communications capabilities (related to network communications) to allow simpler and higher performance of relational data streaming around the signaling and train control system;
- Power loading improvements as by design, the scale of relay interlockings is significant - up to tens of thousands of relays, individually, the electrical load is insignificant, but when multiplied draws significant current. Besides that, poor power factor and phase imbalances can result in significant heat dissipation and overrated mains supply / UPS.

Conversely, the develop of CSIs has once again brought issue to take care of such:

1. lack of formal methods in developing the Interlocking Logic;
2. Lack of domain knowledge in signaling and traditional RRI Systems, this creates a technological gap between the software programmers and the domain consultants that leads to errors in software, which might lead to unsafe failures of the system.

Interlocking logic

As it was said before, the interlocking system executes the above mentioned interlocking logic that satisfies the interlocking principles and the control table for the particular station layout.

An example of interlocking principles is illustrated here after.

When the railway operator requests the IXL to set and lock a route, the IXL:

1. Check the route signal (signal that indicates whether a train could run from a location to another location within a station) enters the route-requesting state after receiving the route-requesting information from a train;
2. Check whether there are routes with uncompleted settings that may be in collision with the requested route, if not, go to the next step;
3. Check the status of the tracks that the requested route will use, if unoccupied, then reserve those tracks;
4. Check whether the points are in the required position, if not, switch it to the requested position;
5. Lock the conflict routes to guarantee the safety of the requested route;
6. Check whether the tracks and points related to the routes are usable; if usable, turns the route signal green, which means the Trans can enter the route; For the sake of avoiding subsequent trains' entering which may lead to tail collisions, block the signals immediately as soon as the train enters the route.

As and when a train moves along a route, the IXL system releases the locked devices behind the train and lights the aspect of the signal at the entrance of the route forbidding the trains to enter that route.

Another example of IXL principle is the rule that governs the control of points. It is important to distinguish between IXL principles and IXL safety properties: safety properties are global system-level properties of a network (such as collision avoidance) while IXL principles are local rules that apply in individual switching points and signals.

Control tables and interlocking principles

The safety principles and basic rules that underling an interlocking are specified by means of control tables sometimes called conditions table. This important document also acts as an agreement between the railway administrators and the railway companies [7].

More specifically a control table is a structured, tabular presentation of the rules governing route setting on a railway track layout. It is also used as a test specification for the interlocking.

The rules for writing out control tables are derived from the principles of safe working of trains [8]. A control table represents an intermediate level of design between a track plan and a wiring diagram.

The format and content of tables is not standardized, and may vary even within the same railway administration. Diverse safe working practices and signaling technologies drive diverse table formats [9].

A collection of track circuits along the reserved section or path from signal to signal is called a "route" that is a is a key concept in the table.

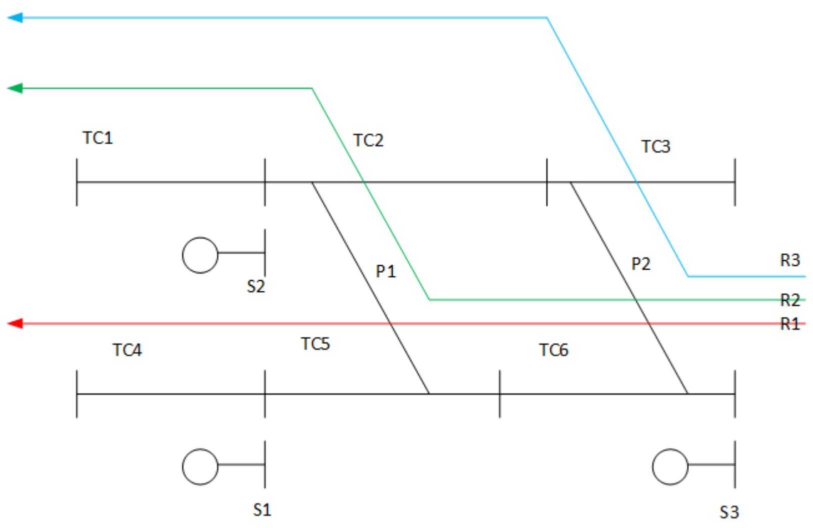
An entry signal shall be clear to let the train enter the route. Although the request to clear the entry signal is issued by the signal man, the route entry permission is decided by the interlocking system using safety rules and control methods specified in the agreed control tables.

Columns of the table indicate the:

- point settings required by the route;
- conditions for locking and releasing other routes;
- tracks required clear in order to clear the entry signal;
- route replacement rules;
- And approach locking acquisition and release.

Some signaling conditions are presented in other tables, subsidiary to the main route table. There is a point control table that indicates the conditions needed to move a point. Since the interlocking connects points and routes, the two tables are, to a degree, converses of one another. Other tables cover approach locking, aspect sequencing, level crossings and dual gauge requirements.

Below it will showed a very simply track layout – with the relative all-in-one control table - over which are present three routes (respectively colored red, green and cyan .



Route	S IN	S OUT	Aspect	S Ahead	Tracks	Point Normal	Point Reverse
1	S2	S1	Y/G	R/Y	TC4, TC5, TC6	P1, P2	-
2	S2	S3	Y/G	R/Y	TC6, TC2, TC1	P2	P1
3	S2	S3	Y/G	R/Y	TC3, TC2, TC1	P1	P2

Data in the first column, “Route”, is the route identifications which are labeled here 1, 2, and 3.

That’s is not a standard, it’s depends by railway authorities for example some tables have the form below

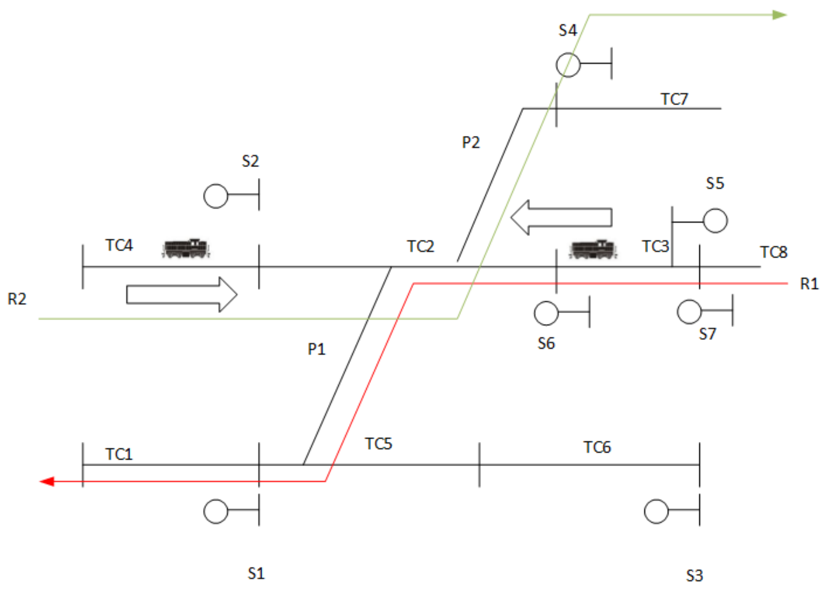
Route	Aspect	S Ahead	Tracks	Point Normal	Point Reverse
1(S2-S1)	Y/G	R/Y	TC4, TC5, TC6	P1, P2	-
2 (S2-S3)	Y/G	R/Y	TC6, TC2, TC1	P2	P1
3(S2-S3)	Y/G	R/Y	TC3, TC2, TC1	P1	P2

Each row in the tables represents the requirement how to set and release each route. For example, route 2 comprises the track circuits TC6, TC2, TC1 and requires that the points P2 are in normal position while P1 in reverse.

Although there is significant difference in the way that interlocking systems are implemented, all currently used systems full the same functions. When observed as black boxes, the behavior of most interlocking types will be the same in most situations. Indeed there are different control methods but four are considered basic which are widely accepted and used among railway companies:

Route locking: Route setting involves a collection of adjacent track circuits, points and signals. A route can be set and reserved for a passage of a train along this route. To assure the safety, firstly, the interlocking system verifies that the route does not conflict with other routes previously set. Secondly, the points along the route are locked in the correct positions. If the related points are not in the correct positions, the controller will attempt to set and lock them in the correct positions. Thirdly, the track circuits along the required route are all clear or unoccupied so that nothing obstructs the passage of the train. Then the entry signal can be cleared (showing yellow or green).

Approach locking. After a route is set, the point is locked and the entry signal is cleared, if the track circuit in front of (approaching) the entry signal is occupied, then the signal man cannot cancel the route and the entry signal by the normal procedure. Approach locking prevents the train driver from the sudden change of signal aspect from green or yellow to red. The importance approach locking is explained in Figure below. Consider that two trains respectively on route R1 and R2 approaching each other. Consider initially signal S2 had been cleared to allow train on R2 to move towards the track TC2- TC7 taking divergence at point P1, The train in route R1 was to wait till train on route R2 goes past the track TC2 and onto track TC7. Consider that due to change of mind the train controller decides to stop the train on route R2 at signal S2 and move train on route R1 to move onto track TC2 and through diverging setting of point P2 onto track T4. To do this the signal S1 is put back to danger and then signal S6 is attempted to be cleared. Because of the delays involved in the carrying out the commands and the indication given by the signals at site it is possible that the train on route R2 passes the signal S1 before it had turned red, lands on track TC2 and starts to progress further not aware that the signal S1, it had passed was intended to be put back to red an it was supposed to have stopped at S2.



Route released. After the passage of the train, the reserved route is released automatically.

Flank protection. The equipment within the surrounding area of the reserved route that may cause an accident shall be protected even if no train is expected to pass such a signal or such points. For example points should be in such positions that they do not give immediate access to the route.

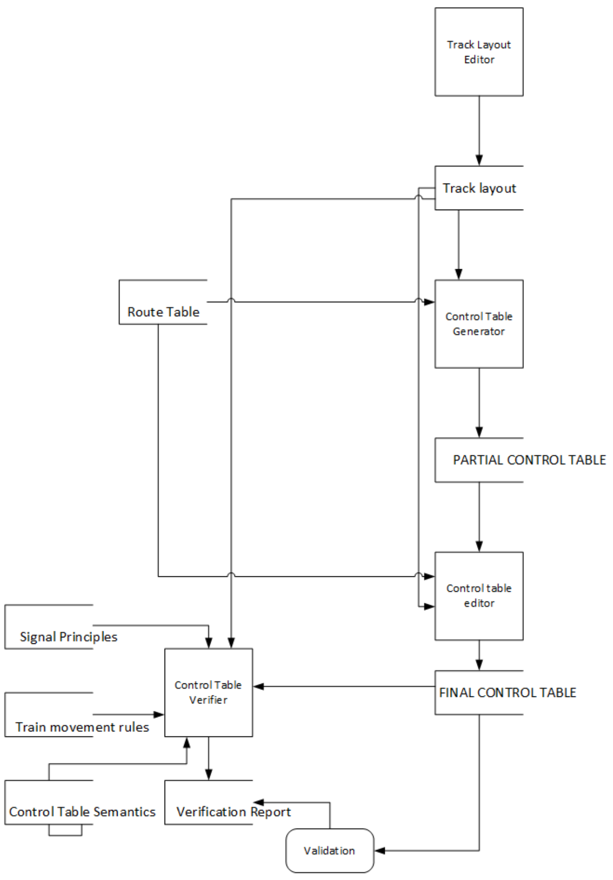
As it was said above a control table is an off-line document that exists with the track layout schema relative to a station that is compiled and update by the railway authorities' engineers.

The pervasive introduction of the computers reflects also on this subject.

It is clear that the process of generating a control tables has some routine elements that are amenable to automation.

However, as stated by [9] in general, it does not appear to be feasible to generate a complete table automatically. Some signaling rules are imprecisely defined, some are difficult to capture in the control table format, and at some locations there are special rules that deviate from the general principles.

Also in the above work they present Signaling Design Toolset (SDT) which it will be reported here to give an idea of the step required to generates a control table automatically (allowing human editing of the table, verifies the control table against a set of signaling principles, etc..).



In the figure above it is showed the process flow of generating the control table.
It is possible to identify 4 main processing units:

- Track Layout Editor;
- Control Table Generator;
- Control Table Editor;
- Control Table Verifier.

The Track Layout Editor is used to produce a graphical representation of a track layout. The user draws the track on the screen and decorates it with the elements needed for a control table (track segments, points, signals etc.). The editor undertakes consistency checks on the drawing (e.g., that precisely three track segments join at a point) and writes out the elements and their connectivity in a formal notation. For reasons of portability and they have chosen Extensible Markup Language.

The Control Table Generator is a tool that generates control table entries for a layout, given its XML description. It generates all entries of the table that can be inferred from the layout. However, in general it is not possible to compute the entire table automatically. Limitations include situations where different options are possible, or where extra commentary or notation is needed to fully specify the meaning of a condition. The output of the tool is a structured description of signaling controls for the layout. It is expressed in XML, in a form that mirrors the railway authorities control table structure.

The Control Table Editor allows the user to add, remove or modify entries in a Control Table. Control Tables are presented in a style similar to a spreadsheet. Edited control tables are stored in the same interchange format produced by the Generator. User editing is essential, for several reasons. First, it is still sometimes desirable to produce a table by hand from scratch. Second, as indicated above, the Control Table Generator will not always produce a complete table. Third, specific local signaling rules apply at some locations. These local rules may be either more or less restrictive than the general principles. From the Control Table Editor, the user can perform basic consistency checks on the Control Table. For example there might be a check that each point control in the route table has a corresponding control in the points table.

The Control Table Verifier is a tool that checks a control table against a set of Signaling Principles. To achieve this it translates a Control Table into a model that defines the behavior of the signaling objects e.g. points and signals) for the layout, based on generic Control Table Semantics, which define the meaning of the Control Tables.
It then puts this model together with a model that captures assumptions on how trains can move through a layout. This final model is exhaustively checked against the signaling principles, using a model checking algorithm (see further on).

Implementation issue

The Boolean equation method is the immediate implementation of the traditional relay interlocking principles. In this method the relay circuits are implemented as Boolean equations, so there is one to one relationship between the relay circuits and the software variables which a railway operator can easily validate the software entrees made and this method gives him sufficient confidence.

This method theoretically has very high safety performance, since the control laws once written remains constant and only the station data is changed for every yard, but the actual implementation of this method has limitations:

- the control laws are not fully tested and they are not generic;
- Typically these Boolean equation are in huge numbers and very difficult to verify these equations.

Most of computer based interlocking systems use (in their implementation and/or formal specification) the above form of centralized database above cited as control tables.

In the 2000's, it was introduced another approaches to interlocking rules mainly well discussed in [10] and [11].

The paper has addressed the problem from a "geographic" and "distributed", point of view.

Fantechi et al call the above table-based approach "functional" because in that approach the rules are generated rules by adopting a design methodology focused on functions, such as the switch points checking function, the routes setting function or the routes verification.

In the Geographical approach the input to the interlocking systems is given as the position of the signals, points, tracks Circuits and Slots. The Interlocking is implemented based on the generic rules such as no part of the track are shared by the two routes at a time, Conflicting routes should not be set at a time etc. This type of implementation requires a great knowledge of the Yard Elements and the interconnection between them. In this method the software does not have one to one relationship to the relay circuits used for RRI and is very difficult validate, so this method has failed to create the necessary confidence in the railway operators.

The also noticed that placement of the yard equipment is ignored in this design methodology, it does not exist a direct correlation between the geographical position on the yard of a device and the function that controls it implemented in the IXL. They also pointed that it is a hard task to identify the parts of the system stimulated by external events.

They also introduced a different approach that can consider distributing the knowledge of the interlocking rules to objects modelling the geographical placement of physical elements.

This approach has broadened the field of design of IXL including now the object oriented approach.

SYSTEM ARCHITECTURE

In the above sections, it has been described the system by identifying the environment involved, and few consideration of which type of computations used was given. Now it’s time to setup the first axis of the case study.

It is well known that a system architecture design allows engineers to exploit recurring organizational patterns. It provides, also, routine solutions for certain class of problems, by supporting the reuse of underlying implementations and by permitting specialized analyses.

A description of system architecture, by very nature is expected to make a complex system intellectually tractable by using abstraction and structural decomposition techniques.

It should expose top-level design decisions and should allow the designers to reason about how to satisfy system requirements in assigning functionality to design elements.

System architectures are generally characterized by:

- *Structure* that is its constituent parts and relationship among them,
- *Functionality* that is its basic functions and rules of their combination
- *Behavior* that is a process view e.g. the sequence of events it may trace.

In the field of computing, the term architecture is often used to mean a common routinely used hardware and software structure. In a broader sense, it is a programmable system structure that comprises active modules, a mechanism to allow interaction among these modules, and a set of rules that govern the interaction with its environment.

Real-time system architectures are too decomposed into hardware and software components. The properties of these components such as interfaces and computing resource (e.g. stack, CPU) requirements are described. Constraints imposed on their design, the standards followed, flexibility and adaptability are also explained.

To return to the paper subject, the aim of the rail operators was to achieve the maximum possible reduction in project execution times, planning outlay and costs, and follow the trend towards ever more high-performance interlockings installed in increasingly short time periods. This resulted in the development of modular interlockings.

The conceptual Layers

An interlocking system is a real-time modular architecture system that is composed by subsystems that they could be structured on three layers:

- The logistic layer
- The functional core layer
- The I/O layer

The logistic layer

The purpose of the logistic layer is to grant to the signal operators:

- setting traffic schedules;
- checking routes occupancy;
- Checking system failures.

At this layer it will be found the Traffic Control System and Diagnostic Control System. The corresponding equipment will be a Video Display Unit with a commercial pc to issue commands and monitor or a NX Panel.

The letters “NX” are for “eNtranceeXit.” This indicates the operation strategy of the panel: on the panel only b uttons need to be pressed where the train starts and where the train should go.

Generally the TCS and DCS are placed in the Traffic Control Center.

At this layer also it will be found a local equipment to interface directly with the interlocking system as a terminal, keyboard and printer.

The functional core layer

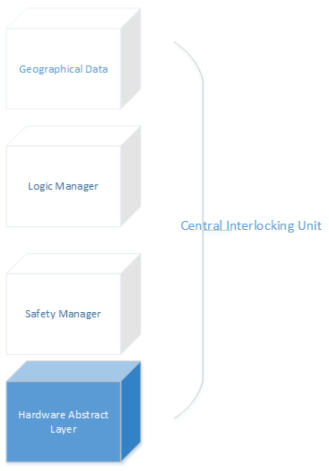
The purpose of the functional core layer is:

- ensure the safety properties on the elements of the i/o layer;
- issue messages to the i/o layer;
- Read messages from the I/O layer.

At this layer it will be found one or more Central Interlocking Unit which is also conceptually structured in:

- Geographical Data;
- Logic Manager;

- Safety Manager;
- Hardware Abstraction Layer.



The processor could be based one CPUs or PLC (depends by the type of the interlocking) and a redundant management unit.

The central interlocking microcomputer processor is mainly characterized by low power consumption. An 8-bit microprocessor with 1 MHz it's enough to control in safety a small station.

Generally it's not mandatory using cisc or risc based architecture. Indeed the most used microprocessor is Motorola m68xx or PowerPCs.

PowerPC is a reduced instruction set computer (RISC) microprocessor architecture created by the 1991 Apple-IBM-Motorola alliance, known as AIM. The PowerPC is designed along RISC principles, and allows for a superscalar implementation, and this technology begins with IBM's POWER (Performance Optimization with Enhanced RISC) architecture. "PC" means Performance Computing, and "PowerPC" means super-high-performance computing.

The M6800 is a general purpose microprocessor which was introduced by Motorola in August 1974. It was initially developed as an enhancement of Intel's 8008 but is comparable to Intel's 8080 microprocessor which had been launched in early 1974. Both were 8 bit microprocessors with 16 bit address buses. The 6800 and its follow-ons have proved to be quite popular and continue to be used is a variety of applications.

The 6800 is a basic general purpose microprocessor. It comes in a 40-pin package that requires additional elements to make up a useful circuit. The normal clock signal speed is 1MHz, though the 6800 does not have an internal clock, ROM or RAM. These must be supplied together with any I/O circuits to use it in a practical application. A range of support devices have been developed by Motorola for the 6800 family of microprocessors, including clocks, interrupt controllers, I/O adapters, etc. The 6800 has no I/O capability within itself; this must be provided by I/O chips.

PLCs were developed in the 1960s to replace the complex electrical relay circuits widely used for machine control in industrial automation. Users program PLCs to define a series of actions that a machine will perform. The PLC works by reading input signals from devices such as sensors or switches and, depending on the value(s), turning output on or off.

In its simplest form, a PLC replaces relay logic. Instead of mechanical devices like interconnected relays or timers providing the logic for a machine or other device, the PLC is a single boxed device performing the same function. Because it is programmable and in essence a computer, it is more flexible, easier to change than physical relay wiring and a great deal smaller than the relay equivalent. It can also perform arithmetic and other functions, such as servo control and analog measurement.

Traditional PLC programming resembles a relay diagram, and for simple diagrams it has the same meaning. PLCs are usually programmed off-line, and then text-based programs or ladder diagrams are compiled to an intermediate language that is downloaded to the PLC and interpreted.

The I/O layer

The purpose of the I/O layer is to perform the required action to the trackside elements. At this layer it will be found the track circuits, axel counters, turnouts, signals and the corresponding actuators. There is also a dedicated unit that is Remote Processing Unit which acts as a concentrators for a grouped of trackside elements.

Flexibility

As it was noticed before the railway transportation was born in the field of Mechanical and Electrical engineering but now it could be addressed as an Engineering System due the fact that the latter is a mode of through about large scale systems.

The benefit of engineering systems is that takes a relatively optimistic view of ways dealing with change. One way of managing change is to consider those aspects of the system that will remain relatively stable for example the macro architecture.

That is called holism perspective which lends to thinking about appropriate abstractions of describing and analyzing engineering systems as whole.

Another key feature of those systems is the life-cycle perspective. One usually optimizes function, performance, and cost of the system for its intended initial use. In engineering systems, one is also concerned with these issues over the lifetime of the system. [12]

Thus engineered interlocking system properties that arise is a life cycle view include:

- Flexibility issue ;
- Safety issue.

The latter will be discussed deeply further on in this document.

Here the flexibility implies the capacity of offering interlocking solutions for a wide range of environments and requirements.

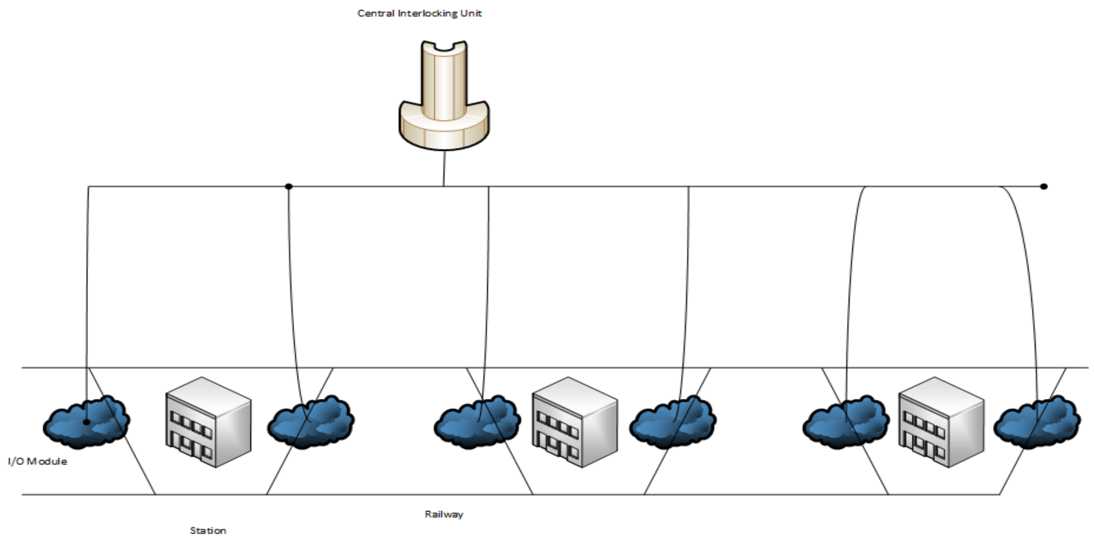
By these criteria, system architecture can be grouped into three categories [13]:

- Centralized IXL (Centralized interlocking , centralized IO);
- Decentralized IXL (Centralized interlocking, distributed IO) ;
- Distributed IXL (Distributed interlocking, distributed IO).

These types of architecture are easily distinguishable from each other and - although classification of specific applications may not neatly fall into one or the other category - subjective classification of systems as defined by the essence of their functional capabilities is easily accomplished.

Centralized architectures are prevalent in products with a European heritage. Due to the dense population centers, and relatively large amount of controlled equipment necessary for interlocking control, large and powerful controllers are utilized. These controllers typically have the implicit capability of interfacing to large amounts of controlled equipment over large distances (several miles).

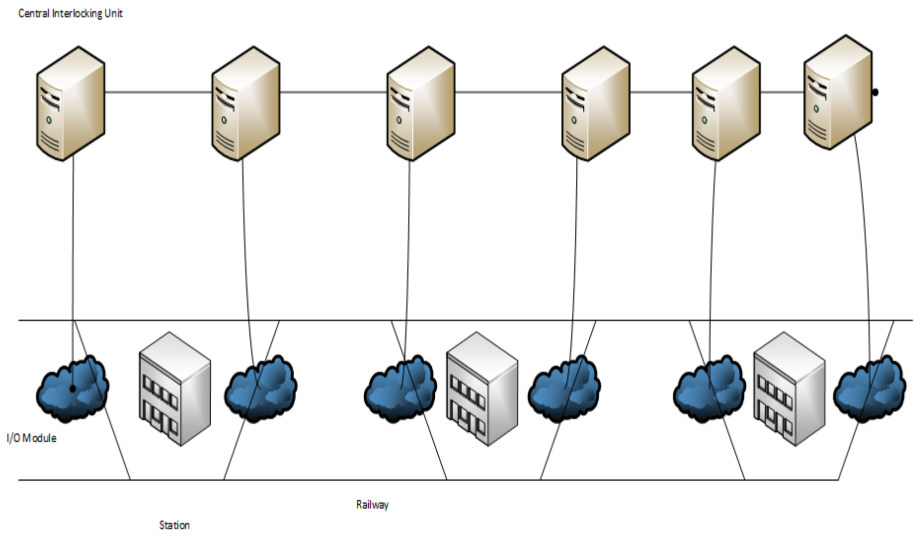
The systems generally have a higher unit cost per controller due to their functional requirements, but can be cost effective when applied in an environment which requires their capabilities. Additionally, these systems often have implicitly combined availability and vitality in their primordial executive functions due to the obvious effects of low availability on a controller with a high concentration of controlled peripheral equipment spread over large geographical distances. Thus these systems are often described as “two-out-of-three or briefly 2oo3”, or, “two times two-out-of-two or briefly 2x2oo2” systems. The installed base of these systems, with several notable exceptions, is concentrated in Eastern Europe and Asia.



The defining characteristics of these systems are:

- a powerful central processor capable of controlling a relatively large amount of elements (for example more than 50 switch machines and associated periphery)
- the capability of controlling equipment over large distances (several miles)
- an internal combination of availability and vitality (“two-out-of-three”)
- a per controller hardware cost relatively higher than that of the other types of systems

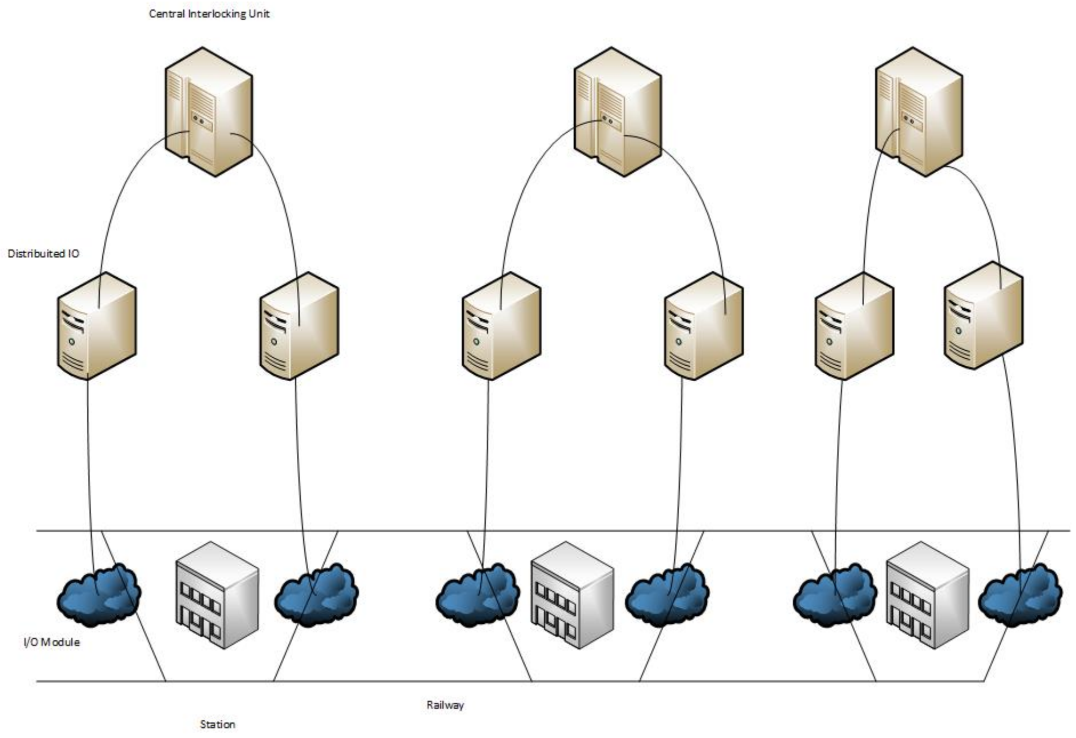
Decentralized controller architectures have their geographical heritage in countries with large land areas and relatively sparse populations such as North America, Australia, and Russia. These systems tend to control only a limited amount of elements, and those are limited to within relatively short distances of the controller itself. These systems are typically installed in relative close proximity to the controlled equipment (for example next to switches). These systems generally maintain a clear separation between vitality and availability (i.e. they are “mono-coded” processors). The most prominent geographic installed base of this classification of controller is in the United States. These systems have the disadvantage of relatively difficult maintenance and troubleshooting, due to their dispersed nature. Additionally the systems do not degrade gracefully in failure conditions.



The defining characteristics of these systems are:

- limited to a relatively small number of controlled elements (typical applications less than 50 switch machines)
- do not typically have the capacity to control field elements over long distances (control distances typically less than 1 mile)
- do not combine availability and vitality ("mono-coded")
- have a relatively low per controller hardware cost compared with centralized systems

Systems with distributed architectures are a relatively recent development, owing their development to the ever increasing technological innovations in computing power and telecommunications. These systems have only begun to emerge into the railroad signal environment as a vital wayside platform, but they have been in use for some time in the industrial automation field. The defining hallmark of these systems is that they have the capability of being applied in a centralized architecture, a decentralized architecture, or a mixture of the two. The application of these systems – in comparison with decentralized systems – allows for a reduction of the total quantity of required controllers, a reduction of cabling, and significantly fewer design constraints. Additionally the systems can be arranged to overcome some of the drawbacks of a decentralized architecture. The systems offer elegant troubleshooting functionalities, and graceful degradation characteristics.



The defining characteristics of these systems are:

- they can control a relatively large number of controlled elements (more than 50 switch machines)
- can control field elements of large distances (several miles)
- do not combine availability and redundancy (“mono-coded processors”)
- have a relatively low per controller hardware cost compared with centralized systems

The benefits of a distributed I/O arrangement are obvious but not always easy to realize. The architecture is ideal for long thin layouts, where direct cabling from a few centralized interlockings is extremely wasteful in terms of cable and associated ducting.

SYSTEM FAULT RESILIENCE POLICY

It’s clear now that modern computer based interlocking systems are embedded systems because they are a computer systems with a dedicated function within a larger mechanical or electrical system, often with real-time constraints.

It’s also clear that they are safety-critical systems because they could cause injury or loss of human life if they fail or encounter errors.

Also the (vital) communication systems, such as Leaky Coaxial Cable system [14] are safety-critical.

In the railway industry, reliability is intricately related with safety. In that foreseeable failure conditions, systems are generally operated manually and, as a consequence, they are more prone to human errors. Further, poor reliability reduces efficiency and availability, which can lead to poor performances and loss of revenue.

Safety critical architectures have evolved to provide routine solutions for safety critical applications, by supporting the reuse of underlying implementations and by permitting specific safety techniques which will be discussed further on.

First, faults are defects or situations that can lead to failures. They may be quite small, such as a frozen memory bit, an uninitialized variable in software, or a cosmic ray ionizing its way through the embedded system.

A fault generally leads to an error. An error is a manifestation of a fault as an unexpected behavior within the system. It might be something like an incorrect result of a calculation or a mistaken value of a state variable. Errors generally lead to failure. A failure is a situation in which a system (or part of it) is not performing its intended function.

A low-level failure in some small part of a system can be viewed as a fault at another level, which can lead to errors at that level that can trigger failures that can themselves, be viewed as faults at yet a higher level. If these faults are allowed to "avalanche" to system-level failures, they can lead to hazards that have the potential to threaten injury or loss of life.

On the other hand, safety-critical systems don't always strive to maximize uptime. In fact, they may intentionally take themselves down or bring some subsystems down in situations where there is a threat of injury or loss of life.

These systems take themselves to a safe state in order to break the fault-error-failure.

For example one of safe state for interlocking system is to immediately turn to stop aspect (red light).

As with any embedded system, design is preceded by a system requirements definition, covering physical and functional specification. For safety-critical systems, a thorough hazard analysis and risk analysis must also be done.

The objective of hazard analysis is to systematically identify the dangers to human safety that a system may pose, including an evaluation of the likelihood of an accident resulting from each hazard. A popular technique for doing hazard analysis is the well-known fault tree analysis.

It takes a top-down hierarchical that involves decomposing undesired system events in order to identify which combinations of hardware, software, human, or other errors could cause safety-threatening hazards.

After hazard analysis, the next step is risk analysis. Risk is the combination of the probability of an undesired event occurring and the severity of its consequences. It might be expressed in units such as "deaths per 100 years of system operation."

Once the greatest risks posed by a system have been identified, they must be dealt with in the system design: if possible, the underlying hazards should be avoided or removed. This can often be done using:

- hardware overrides to bypass risky software components
- lockouts to prevent entry into risky states
- locking to ensure remaining within safe states
- Interlocks to constrain sequences of events in order to avoid hazards.

If it's not possible to totally avoid or remove the hazards, the risk of accident must be minimized; if an accident does occur the risk of loss of life must be minimized.

Together with the system requirements, the results of the hazard analysis and risk analysis will guide a safety critical system's architectural design.

All the type of safety critical embedded system has to deal with the shutdown system which is a dedicated unit with responsibility for identifying dangerous situations. It will force the entire system into a safe state (in other words, off) whenever a hazard is detected and thus lock the system out of a life-threatening state.

The shutdown system is independent of the primary system that is normally in control, and operates in parallel with it. To ensure its complete independence, the shutdown system has its own separate sensor(s).

Generally, a diagnostic subsystem is used to ensure the integrity of operation of the shutdown system itself. If the diagnostic subsystem determines that the shutdown system's decisions may be untrustworthy, it can bring the entire system to an immediate safe state rather than allowing the primary system to continue to operate without trustworthy shutdown monitoring going on in parallel.

So in fact, some safety-critical systems have dual shutdown systems working in parallel (with either "AND" or "OR" logic for deciding when to shut down the primary). In extreme instances, a safety-critical system can be designed with three shutdown systems working in parallel using TMR-style voting among them. In this way, a faulty shutdown system can be identified and itself be shut down while the remaining shutdown systems can continue to operate in redundant and trustworthy fashion and the primary system continues to provide its services.

The idea of a shutdown system can also be applied on a smaller scale within a primary system itself. A basic primary system is structured by the simple design pattern of Input-Process-Output, as the sequence labeled "Data Acquisition," "Processing/Transformations," "Output/Control."

To lower costs, the primary system and the sensor data integrity checking "shutdown" monitoring activity share the same input sensor(s).

The idea of shutdown monitoring can also be extended to the output side of a system. This is called actuation monitoring. Actuation monitoring can be done in a number of ways, each with a different balance of costs versus benefits.

The most basic form of actuation monitoring is end-around monitoring. It simply checks the commands to the output actuators for validity before they reach the actuators themselves. A more stringent form is wraparound monitoring, which checks that the output actuators are actually producing valid outputs that will soon reach the patient under treatment.

A third, usually more costly, form is actuation-results monitoring that uses an independent set of sensors to verify that the system is actually producing the results it's intended to provide.

A significant weakness of both the shutdown system and the single-channel architectures is that they cannot continue to operate safely in the presence of faults. They have single points of failure.

This means that these architectures can only be used in safety-critical systems that have an immediate safe state

For safety-critical systems without an immediate safe state, dual-channel architectures can be used to allow a system to continue operation even when one of its channels has "fail stopped."

In dual-channel architecture one of the channels serves as the primary, or active, channel and the other is a standby or backup channel, ready to take over system operation if the current primary channel suffers faults or failure. Depending on the needs of the specific safety-critical system, the standby channel when becoming active could either continue normal operation of the system or it could take the system through a possibly long and complex sequence of steps to bring it to its eventual safe state.

Dual-channel architecture is going to have higher unit costs than previous architectures discussed. There will be redundant embedded processing channels using redundant hardware and redundant sensors.

But the big benefit of paying this price is the ability to continue to operate in the presence of a fault. Dual-channel architecture has a number of popular variants.

That is two channels that use the same replicated software and hardware, the architecture can handle random faults well but it can't handle systematic faults such as software design or coding defects that would be reproduced in both channels.

If this is of concern in the treated system, heterogeneous dual-channel architecture is preferable.

This kind of architecture would consist of two channels implemented in totally different ways. For example, software for the two channels could be implemented by separate software-development teams working from the same software requirements specification, in what is called "n-version programming" or "dissimilar software."

Clearly, the development costs as well as the unit cost for doing this would be high. Another variant of the dual-channel architecture is multi-channel voting architecture.

This extends the TMR approach discussed earlier for sensor error detection into the realm of entire replicated processing channels. In this architecture, three (or more) channels operate in parallel. A "voter" compares the outputs of the channels: if a majority of channel outputs agree, this will become the system's output. If some channels disagree, they will be fail-stopped.

Safety critical digital architectures also provide means to manage the complexities of the applications. For example, they provide facilities:

- to minimize the safety critical elements of complex applications by providing standard measures for well-known problems;
- to segregate critical and non-critical parts, referred as vital or non-vital, in the implementation of the system;
- To minimize design and implementation errors by providing design guidelines and implementation tools and techniques.

All safety critical digital architectures are programmable systems which basic capabilities such as:

- real time processing,
- input-output control through programming,
- computing resources, e.g. CPU and memory, management,
- meaningful interfaces with the human operators,
- Built-in-Test and self-checking hardware and software.

In railway industry applications, further specific capabilities are required which:

- recognize and handle input and output errors,
- recognize and control internal faults, such as faults due to transmission and CPU or memory faults,
- implement standard safety measures, such fail-safe and emergency measures,
- improve reliability of the over-all system,
- increase the system availability,
- Implement specific types of safety critical functions, such as digital transmission or interlocking logic.

In safety operation of microprocessor architectures, the safety design approach can use the following concepts:

- The information redundancy concept which consists in detecting faults due to failures or perturbations by the very contents of information is made as a basic technique. This involves mainly a single microprocessor architecture using information redundancy like the coding techniques, the software diversity and the self-checking techniques.
- The hardware redundancy concept, based on several hardware running simultaneously, is adopted when the safe state is the end of the mission; this is the case particularly of air transport or space programs.

These approaches can be jointly used in a given architecture as it will be shown further on in the railway's vendor interlocking system section.

The information redundancy can be integrated into the software or installed in the very processor. The principle consists in giving to the processed information a certain redundancy. Validation is made by controlling the information belonging to the chosen redundancy.

- The signature microprocessors: the partial signatures, collected during the processing, are bound to unveil the errors when the final signature is verified. The signature is independent of the type of microprocessor.
- The software diversity (N version programming) means that N independently developed functionally equivalent versions are executed in parallel. Their outputs are adjudicated by a voter.
- The self-checking processors are designed and developed in order to facilitate tests of different processor blocks. They integrate into their architecture the devices necessary for implementing operational tests. Special operations are added to the conventional instructions, in order to facilitate the testing of the microprocessor. These specialized processors are capable of detecting immediately their own faults.

The Hardware redundancy architectures contain two, three and more distinct most often homogeneous processors (for reasons of maintenance costs) accompanied by their own environment equipment. These architectures differ by the nature of comparator and the techniques of its putting into safety. They often integrate internal functional tests for protecting themselves against latent failures. The difficulty inherent in these tests is the evaluation of the coverage rate of failures.

- The Dual microprocessor architecture can be found in on-board automatic devices.
- The Triple Modular redundancy architectures are based on the redundancy of three microprocessors to which a system of comparison and majority vote is associated. The result of each processor is compared with those of the other ones and the system output is the result of a "Two out of three" vote. The suspected processor can then either be put out of action (thus no longer a participating voter) or penalized and put out of action if its failure persists. The replacement of the failed part may be made without stopping the system. These architectures are well adapted to ground equipment.

In a distributed architecture the overall functionality is partitioned/decomposed into sub-functions which are allocated to logically/physically separate subsystems that communicate and collaborate to synthesize/achieve the required functionality. The concept of distributed architecture is not binary in the sense that a given architecture is either distributed or not. Instead, there is a wide spectrum of distribution and related issues which should be considered in the state and time ‘dimensions’ (e. g. physical distribution, communication delays, transmission errors, etc.). In a distributed system each component has limited capabilities to acquire knowledge about the status and timing of other components (e.g. What its state is and when the state has changed).

The dependability of any distributed system relies on the dependability of

- the separate subsystems,
- the architecture itself and
- The communication.

This is one of the reasons why the architectures are so different and why it is not trivial to compare them.

Distribution may result from different design considerations such as:

- adaptation to a geographically distributed process
- increase of dependability by redundancy or diversity
- functional partitioning or functional autonomy
- Performance considerations (increase of system time response through local intelligence at peripheral levels, load sharing, etc.).

The first design consideration means that geographical needs strongly influence digital architecture structures. The second directly improves system safety and/or reliability and availability. The graceful degradation behavior, achieved by the third consideration also leads to increased system dependability. The fourth consideration is not usually driven by the distributed nature of the process or by dependability requirements and will here not be expanded further, therefore.

On the other hand, distribution also introduces additional complexity (e. g. of communication), might cause synchronization problems and requires sophisticated solutions for data storage and data management. This increase in complexity and underlying functionality to support distribution leads to an increase in failure modes that have to be additionally considered within the system dependability analysis. Additionally there might be cost, weight and space problems. The benefits and drawbacks of distribution thus have to be carefully analyzed for each application.

Some geographically distributed systems (such as a continuous cab signaling and speed control system) require a distributed digital architecture. Other systems (such as interlocking systems or nuclear power stations) might use distributed digital architectures to avoid long power lines. All these systems have long communication lines. As the vulnerability both to interruption of communication and faults in the information content are a function of the length of the line, these play an important role in dependability considerations. Interruptions will be handled using fail-safe mechanisms (which might impair the availability). Faults in the information content (e. g. caused by electromagnetic interference) should at least be detected by the receiver.

Apart from satisfying geographically distributed processes the main incentive for distributed architectures is to increase the dependability (e.g. Safety, reliability and/or availability) of the system by redundancy and/or diversity. Distribution is an important means to making a system fault-tolerant (by avoiding system failure due to single faults), and to increase the dependability of the system, if the quality of single components is not sufficient to satisfy the overall dependability requirements.

Redundancy (using identical components in parallel) will decrease the system vulnerability caused by non-systematic hardware faults, diversity (using different components in parallel) will be used against systematic faults (design faults, especially in the software). Diversity can be hardware diversity, software diversity (n-version programming) or using hardware or software or human action for the same function in parallel. Both in redundancy and diversity it is necessary to consider possible common cause failures (e. g. by a common power supply, electromagnetic interference or common requirements).

A crucial element of redundant and diverse systems is the voting system, which compares the output of the different parts. Voting can be done in hardware or software. It may compare only the final results of redundant or diverse elements; in redundant systems it can also include the comparison of intermediate states (such as the results of each step of an algorithm).

The increase of dependability provided by the distributed architectures in this group might concern both safety and availability or only one of them. As these two aims often oppose each other, it can be difficult to obtain an acceptable compromise. It should be noted, however, that the unavailability of a system, even if it leads to a fail-safe state, can also harm the overall safety of a system, for instance when a train stops in a tunnel or on a bridge. Furthermore the necessary recovery to normal operation - often needing seldom practiced human interaction - poses an additional threat to safety.

The first answer to safety requirements is of course the fail-safe principle. In redundant and diverse systems this concerns in particular the voting systems. This is the reason why safety-relevant voting systems are usually realized in fail-safe hardware.

Besides the deterministic fail-safe principle there is also the probabilistic approach, which is - at least implicitly - used in the development of most safety-relevant systems. This approach is especially valuable when assessing a compromise between safety and availability.

If the influence of the availability on the safety of a system is not taken into account, a NooN-configuration (N out of N - the total result is only true, when all N results are identical) satisfies the safety requirements (Common cause failures have been discussed above). The most common realization is a 2oo2-system.

A major concern in safety is design faults, especially software faults. In distributed architectures (single channel architectures are discussed elsewhere) diversity is a powerful answer to this problem. While it cannot be proven, that e. g. n-version programming will result in 100% fault-free software, the probability of a fault will decrease with each or the following: different programs, different development teams, different languages, different language types (including expert systems), different requirements, different processors, etc.

Like in localized systems, to satisfy high availability requirements, redundant and diverse MooN-systems (M out of N, M < N: the total result is true, if at least M of the N results are identical - majority voting) are used. If an item has failed, it can be inspected, taken out of service and replaced without interrupting the operation. A common realization of MooN is a 2oo3-system.

Single items, redundancy and diversity can be combined in a number of ways to arrive at a distributed architecture which satisfies high safety and availability requirements. If a single microprocessor already meets the safety requirements, it might only be necessary to make it redundant to achieve the required availability. Towards the other end of the spectrum one might use two channels with diverse software, each on its own computer, to satisfy the safety requirements, and triple these computers to satisfy the availability requirements. In this case there could be majority voting (2oo3) in each channel and finally fail-safe voting (2oo2) for the final output.

Compared with a centralized system functional partitioning or functional autonomy might also increase the dependability, because in this case it might be possible to continue (at least a limited) operation, when the central system breaks down (graceful degradation). Examples are substations in interlocking systems or emergency control rooms in nuclear power stations.

A fail-safe system is a control system that either continues to function safely after the occurrence of a fault in the system or lapses into a predefined condition known to be safe. For the railway interlocking system, keeping switches stable and changing signals to red axe considered to be safe. It avoids catastrophic accidents even when critical situations occur in the field. The faults under which the system guarantees safety belong to a fault set, called a fault model, either at the gate level or higher levels.

Different philosophies have been espoused as the ideal method of design of fail-safe processor based equipment. These philosophies fall basically into two categories:

- redundant or composite safety systems, using multiple sets of equipment with facilities for voting and shut-down of any failed system;
- Single channel systems employing a single set of unique equipment.

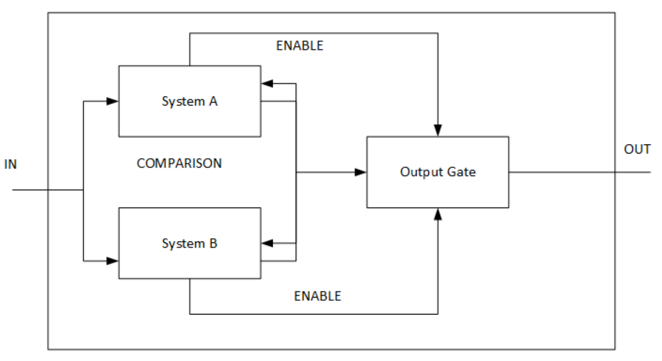
It is important not to confuse the differences with standby property as both categories may have capability to operate as a standalone or redundant hot standby system.

There are two essential requirements for a redundant system:

- two or more identical systems;
- Fail-safe fault detection and negation.

At least two systems must be operational but additional systems may be included to enhance the availability.

The operation of two or more systems in parallel will not, in itself, increase the system safely unless means is provided to detect and positively isolate all faulty systems. The philosophy behind redundant systems is based on the concurrence of outputs of at least two systems and the probability that identical wrong-side failures will not occur simultaneously.



Fault detection may be incorporated in each system through monitoring the outputs of other sets of equipment. Voting is generally based on a simple majority with the facilities to shut down any equipment out of step.

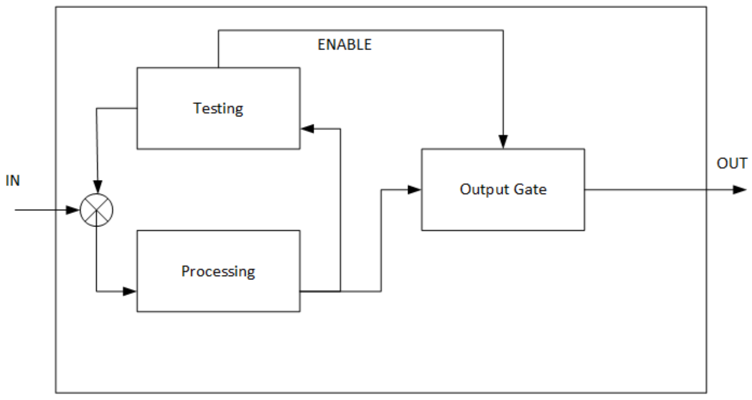
Technician intervention is required to restart equipment. Provision of redundancy does not obviate the need for detailed attention commencing from a correct and complete specification all the way the proving with that specification.

There are a number of factor that need to be considered in the design of safety redundant systems. Some of these include:

- the need to ensure that all active systems utilize identical input data and process it in such a way as to give identical outputs;
- consideration of separating the reading of inputs and of processing in the time domain to overcome the effects of noise interference and the effect this will have on output comparison;
- the ability to update off-line equipment prior to connection online;
- prevention of common mode errors caused by such things as interferences on the input or software errors;
- the provision of a fail-safe means to detect and isolate any faulty equipment;
- The reduction of (Mean Time between Failure) MTBF through provision of additional equipment.
- Fleeting wrong-side outputs which may occur until faulty equipment is shut down.

A single channel system incorporates a single, inherently fail-safe, set of equipment. A single channel system may be duplicated for availability. Single channel systems must adhere to the failsafe design principles such that any possible fault that may occur will be detected, the output prevents from going wrong-side and the fault reported.

Single channel systems often necessitate custom design or each functional block to include the means of testing and negation. Modern approach is to provide two diverse defenses against failure with at least one be fail-side. The next figure illustrates the concept, showing processing between inputs and outputs using a failsafe system and stimulating the same processing function with test inputs.



The fail-safe system will not generate provocative outputs under fault conditions and outputs will be inhibited should incorrect outputs be detected by processing of the test function. Some consideration of single channel systems is:

- often requires custom design hardware to facilitate inherent fail-safety;
- may require some additional equipment to provide diverse proving paths (although this is far less than total duplication);
- Possibility of fleeting wrong side outputs if reliant on secondary fault negation path.

SYSTEM SOFTWARE

SPECIFICATION AND VERIFICATION OF INTERLOCKING SYSTEMS IN THE LITERATURE

In the above sections it was introduced briefly the domain of the problem and a short view on the architecture point of view. Further on it will focus on software part.

The computer based interlocking repeatedly executes an application that executes the interlocking logic over some discrete time interval.

Each time it uses the set of rules it contains to process a new set of inputs before committing them as outputs. The outputs calculated in the previous cycle are then passed on to various places including the physical railway.

It is well known that an application life-cycle can be divided in three phases:

- Modeling;
- Validating;
- Implementing.

The requirements on an interlocking system can be divided into two categories: *generic requirements* and *specific requirements*.

An example of a generic requirement is a condition that must hold whenever a route is cleared:

"A route must be cleared only if every switch in the route is proved in the position required by the route"

A specific requirement applies to a single interlocking system. An example of such a requirement is the configuration of the actual routes in the system:

"Route XYZ requires that switches 1X and 1Y are normal and that switches 2X and 2Y are reverse"

A clear and complete generic requirements specification gives a precise description of the functionality of a range of systems and can be used when validating that the developed system meets all requirements.

However generic requirements specifications are typically written in natural language. Natural language may seem easy to understand at first glance, but experience shows that it tends to be interpreted differently by different people.

Indeed, in reality the quality of generic requirements specifications is often unsatisfactory. Although requirements often are captured in documents or perhaps even in a requirements management tool, they are typically not complete, and the successful development relies on implicit requirements, existing only in the minds of the engineers.

As a consequence, it is hard to guarantee that systems are *designed in a consistent way*.

An efficient way to resolve possible ambiguities in requirements is to instead express them in a formal language. A formal language is a mathematical language, in which every statement has exactly one, well defined, meaning, much as in any well-defined programming language.

The conventional verification methodology based on factory and on-site tests discloses most of the errors but it cannot claim that it discloses all the errors so, also, only a formal verification can claim to do so.

Confidence results from the application of a certified process compliant with CENELEC standards EN50126, EN50128 and EN50129.

This process involves:

- Management plans, including quality, engineering, requirements, configuration, documentation, change control, safety, verification and validation plans;
- Safety Management, including safety verification and validation strategy, safety analyses, hazard log;
- Document Management;
- Requirements Management, including requirements identification, capture and traceability;
- Configuration management;
- Change control management;
- Tool Management;
- Test Management, including regression test strategy, factory and on-site tests.

The above cited EN50128 guidelines for example, address the development of "software for Railway Control and Protection Systems". It, also, constitute the main reference nowadays for every proposal for railway signaling equipment.

The EN50128 define the software safety Integrity Level (SSIL or SIL) by the means of level of risk associated to process.

Briefly the SIL level is:

- 4 – Very High
- 3 – High
- 2 – Medium
- 1 – Low
- 0 – Not Safety Related

Despite the fact for that the above guidelines is not mandatory a precise development methodology or programming technique, it classify commonly adopted technique in terms of a rating like forbidden to recommended.

Initially formal methods were promoted aggressively by idealistic academics. Formal methods were seen as the solution to the "software crisis" and implied the uncompromising use of mathematics and rigor for the whole development life cycle where entire software systems were developed from scratch.

Hence, the development process should start with an abstract specification of the system and proceed by formal refinements and proofs of correctness towards a final implementation.

From an industrial point of view this is not realistic because:

1. systems are rarely developed from scratch;
2. only parts of the systems would benefit from a formal model
3. The general skill level in industry is not adequate to cope with the techniques for fully formal development.

Many companies were reluctant to throw away current practice but they began to allocate massive investments required by such a revolutionary approach [15]. With this approach formal methods are used more as a defect detection technique in the early stages of the software development life cycle. Hence the abstraction inherently associated with formal modeling is a powerful means of reducing complexity and improving a development team's understanding of the requirements.

Conversely formal notations provide a structured way of expressing requirements which makes inspections more effective and reliable and allow different sorts of checking to be performed by automation tools.

Currently there are two main approaches to the problem of verifying the correctness of safety critical systems namely model and theorem proving. The model checking is the approach most widely used by industry. The model checking problem is as follows: given a model and formula expressing some property we would like the model to have, under what conditions does the model satisfy the formula.

The two main techniques typically used in model checking are SAT-based model checking and binary decision diagrams.

Using model checking has the advantage over deductive methods of verification that it is automatic.

However not every system or property can be verified by this method due to the limits of what is decidable /computable.

One further advantage is that a model checking algorithm will typically provide a counter example trace. These traces show the exact sequence of states which brought about an error. This property is useful because it can be used by engineers to find faults in the system.

Early model checking procedure verified models formalized in CTL. Computation tree logic introduced by Clarke et al [16] is branching-time logic, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realized. Nowadays, time logic is used for formal specification, validation and verification for real time systems. A notably review of that logic languages that discern by expressiveness, order metric of time is [17].

One the most well- known problems in the field of model checking is state space explosion. The number of states in a model of a concurrent system increases exponentially in relation to the increase in number of components in the system.

The main focus of the further research has been to reduce the number of states. This has been achieved through the use of state based reduction, path based reduction and compositional reasoning. A simple form of state based reduction would be to remove a state from the search space if it is bisimilar to another state already encountered.

Other methods include program slicing which simplifies the model with respect to the safety property being verified.

The model formed only contains information about the system that is relevant to the safety property.

Progress has also been made in reducing the amount of physical storage needed for a model using caching and compression. While these cannot prove an entire system correct they are useful for finding errors.

Specification issue

Before formal verification of interlocking system software, the system must first be formalized. This requires a computational model of the interlocking. The model includes a language (syntax and semantics) used to program the interlocking and relevant hardware. The hardware that needs to be formalized depends on the interlocking design. For example, timers are part of the hardware, but not necessarily directly supported by the language.

Over the past 30 years, there have been many attempts to formally model various aspects of railways.

Some, attempt to capture as much information as possible while creating the models, these models are typically written using a specification language such as VDM, RSL, B/Event-B or Casl.

A well-known example of this algebraic specification approach is the Paris metro, Metro EST Ouest Rapide (METEOR) [18]. METEOR is a driverless metro line that was partly built by Matra Transport International. Matra Transport decided to use the B-Method during development B is a formal language which is the successor to Z. It is a complex language to use partly because the specification and implementation are deeply intertwined which requires successive refinements to obtain an implementation.

Perhaps Matra Transport decided to use the B-Method as it was developed in France, so they had a wealth of local expertise. Using the B-Method Matra Transport was able to verify 100% of the safety and liveliness requirements of the ATP/ATO systems. It was claimed that no bugs were found during the validation, in house testing, on-site testing and since METEOR went live. This is clearly an exceptionally meritorious result for such a complicated, critical system.

Other work relating to the specification of railways acutely focus on the topology and operational aspects. For example, the approach taken by uses graph theory [19]. The basic idea is that track segments are vertices, and the edges are connections between the segments. The edges are directed; hence they must be doubly linked. The reason they are directed is that it allows the edges to be annotated in only one direction. For example, signals are added to the model by annotating edges, this allows the model to represent that the signal is only visible in one direction (which is a requirement in the railway domain).

Hansen specified Danish interlocking systems using the Vienna Development Method (VDM) specification language. This specification was simulated so that domain experts could validate its correctness. The first attempt omitted manual overrides that are used for shunting and controlling sets of points. The second attempt was augmented to provide these manual overrides, then verified against the safety requirement that the interlocking could not allow trains to collide or derail. Although safety conditions that are more precise were not verified, this high-level requirement is the ultimate goal of the interlocking with respect to safety (not liveliness) properties. It is clear from Hansen's work that using a graph as an underlying data-type to represent the topology is flexible and extensible.

There are a number of different operations that can be applied, for example, the vertices relating to routes can be colored to determine incompatible combinations, or the graph can be traversed to determine whether the topology is well-formed.

Another approach is to use predicates. They express a richer view of information than the graph based approach. Predicate logic has been successfully applied in several projects to model railways [20]. It is straightforward to define n-ary relations between objects: a Prolog-like syntax is suited well to this. For example, this is the approach taken by Eriksson when modelling the Swedish rail topology. Using objects and relations between these objects, the physical world and abstract concepts (such as routes) can be formally modelled. Provided the model adheres to a suitable nomenclature the model will be readable by humans. Having a formal model of the topology allows for information to be automatically deduced, e.g. pairs of incompatible routes.

Verification issue

The formal verification of railway interlocking systems software has been studied a number of times over the past 20 years. Most research into verification of interlocking systems has focused on using exhaustive techniques: SAT-solving, symbolic model checking and explicit state model checking. Typically, case studies focus on a specific interlocking installation and try to prove that the interlocking installation will never allow train movements that might lead to collisions or derailments.

An approach used in multiple case studies is to translate the interlocking logic to a model (for model checking) or a Boolean equation (for SAT-solving). Properties are defined as formulae, representing requirements over the inputs and outputs of the system. These properties can mostly be derived directly from the track layout.

Model checkers or SAT-solvers are then used to prove that the desired properties hold for the models or equations.

The first verification attempts focused on determining whether an interlocking system satisfied concrete safety properties, i.e. formulae represented in terms of the input and output variables of the system.

During the mid-nineties, a notable body of work relating to the verification of the interlocking systems at Hoorn-Kersenboogerd and Heerhugowaard (Netherlands) was undertaken by Fokkink [21].

Years later, the same stations/problem sets of Hoorn-Kersenboogerd and Heerhugowaard were revisited by Eisner in [22], where above cited CTL model-checking was successfully applied in-place of SAT solving.

It is noted that the SAT based techniques were successfully applied in Sweden by Eriksson, where a serious aw in the interlocking system was identified.

Orthogonally during the mid-nineties an effort was made to apply model-checking to a more complicated class of interlocking systems that are programmed using the Geographic Data language [23].

This notable piece of work applies the Calculus of Communicating Systems (CCS) and mu-calculus to verify safety properties of Westinghouse's Solid State Interlocking (SSI). Morley defined how the interlocking and its interpreter can be modelled using CCS and also defined a translation from the SSI's command set into CCS, thus creating a complete model of the interlocking.

Then using mu-calculus and the Concurrency Workbench, he verified safety properties. The project also verified the communications between interlocking systems. An example of this interlocking communication occurs when a train route is spanned through. Mu-calculus can also be used for liveliness properties.

Multiple interlocking systems and the interlocking systems must communicate information regarding the whereabouts of the train and route status.

The transition from verifying a single interlocking system, to verifying the communications between multiple interlocking systems was greatly simplified through the use of CCS.

The verifications described thus far do not make safety verification tractable. They only provide a mechanism to perform verification of safety and/or liveliness properties for interlocking systems.

In Fantechi et al [24] applied the state charts formalism to perform safety verification on distributed interlocking systems. Notably, in that work the high-level formalizations of distributed interlocking systems are introduced.

Swansea Railway Verification Group at Swansea University, we have formed a group of researchers dedicated to exploring verification in the railway domain. The projects range from safety verifications, to safely optimizing capacity. These areas have been explored using a multitude of techniques, some of which include SAT solving and model-checking for the verifications, and creating formal models of the railway domain for the specifications.

The techniques of the safety verifications were explored and honed, in respective order, they used: inductive SAT solving, SAT based model- checking and the SCADE tool set.

Another project is to explore the use of domain specific languages in the railway domain. This includes defining a graphical language that the topology can be represented within, then automatically deriving from one of these diagrams the required constraints for the network to be safe. This information is then automatically translated into a control table, i.e. a high-level specification of an interlocking system. Typically, a specification of the railway is made using a formal specification language such as RSL, and then some refinement method is applied to produce a working system that fulfils the specification. Assuming the specification has been validated to be correct, and then the control system will meet these conditions, as well. This is the procedure followed by the previously mentioned METEOR project.

Verification can then proceed by deciding which states of the track-side functional modules were in conflict, e.g. two opposing signals should not both show the proceed aspect. All possible transitions into these bad states are computed from the state machines, these transitions are then used to build a “safety monitor" state machine.

The formal development of the distributed railway control systems was performed in a more traditional manner. The specifications were made executable, then using these high-level specifications of the distributed railway domain, safety requirements such as “no collisions” and “no derailments” were verified. Subsequent refinements were made preserving these properties until a working system was produced. Verification is carried out on multiple levels, one level of interest are control tables. It is suggested that model checking could be used for this verification in. This article also suggests that the generated code can be automatically verified against the control tables. Fokkink et al. demonstrate how this can be implemented.

In recent years there have been a number of frameworks aimed at bringing formal methods within the reach of the standard critical system developer (JACK) [25].

A framework is characterized as a collection of tools that support the development of verified software, for instance the functionality should include, but is not limited to: type-checking, formal methods, pretty printing, interactive proving, support for automation and obtaining fully functioning programs. Another works of this kind was the Prototype Verification System (PVS). Although VDM predate PVS as development methods, PVS was a framework, as opposed to a collection of tools. The goal of PVS was to provide an integrated interface that allowed developers to specify their control systems at a high-level during an early phase of the design. The system then facilitated specifying properties about the control system; essentially proving that the models fulfilled the specifications is performed using an interactive proof system. The system supported a number of tools that would automatically attempt to discharge proof obligations. These obligations typically arose from the use of refinement types, for example, the divide operator has a refined type such that the denominator is not 0, then everywhere that division is used a proof obligation must be discharged, namely that the provided denominator is not 0. The basic techniques used in PVS have been repeated in subsequent frameworks.

One of the more recent frameworks of this style is the Rodin tool set. It is based on Event-B and is still under active development at the time of writing.

ENGINEREING LANGUAGE

In the first era of computerized interlocking system were developed specific domain languages.

Some of them are still in use meanwhile the ongoing trend is to use model code translation to multipurpose languages as ADA, C or Lustre.

GDL

Geographic Data Language for Solid State Interlocking (SSI) systems detail site-specific behavior of the railway interlocking.

Moreover, the generic SSI program relies on configuration data specifying the conditions under which routes can be set for the specific track layout controlled by the installation. Therefore, each SSI interlocking module keeps a database with the necessary geographic information, which is referred to as Geographic Data.

Errors in the Geographic Data could lead to conflicting routes being set at the same time, which would allow trains to proceed to the same track segment. Other data errors could allow points to be moved while a train is passing over them, which could lead to derailment of the train. Therefore, Geographic Data are safety-critical. For each installation, the data set has to be verified carefully.

CHILL

CHILL is a strongly typed, block structured language designed primarily for the implementation of large and complex embedded systems.

CHILL was designed to:

- enhance reliability and run time efficiency by means of extensive compile-time checking;
- be sufficiently flexible and powerful to encompass the required range of applications and to exploit a variety of hardware;
- provide facilities that encourage piecewise and modular development of large systems;
- cater for real-time applications by providing built-in concurrency and time supervision primitives;
- permit the generation of highly efficient object code;
- Be easy to learn and use.

The expressive power inherent in the language design allows engineers to select the appropriate constructs from a rich set of facilities such that the resulting implementation can match the original specification more precisely.

Because CHILL is careful to distinguish between static and dynamic objects, nearly all the semantic checking can be achieved at compile time. This has obvious run time benefits. Violation of CHILL dynamic rules results in run-time exceptions which can be intercepted by an appropriate exception handler (however, generation of such implicit checks is optional, unless a user defined handler is explicitly specified).

CHILL permits programs to be written in a machine independent manner. The language itself is machine independent;

However, particular compilation systems may require the provision of specific implementation defined objects. It should be noted that programs containing such objects will not, in general, be portable.

- a description of objects;

STERNOL

A STERNOL program is a system of equations, with one equation—really, a guarded command—for each value a variable in the program can take. One group of equations may refer, for example, to the aspect of a particular signal.

EURIS

European Railway Interlocking Specification (EURIS) is a graphical method of designing interlocking systems. It consists of four sub-languages; three of these languages are used to model the topology and routes.

The final language models how the actual entities, such as a signal, operate. The semantics of EURIS has not been fully defined, thus proving theorems about a EURIS specification is difficult due to ambiguities.

An attempt has been made to verify safety properties of a EURIS program by translating the program into a Petri net which has a precise mathematical meaning.

The lack of a formal definition of EURIS resulted in a second language being defined. This language is called: Language for Railway Interlocking Systems (LARIS). LARIS 1.0 was developed by Fokkink et al., at the CWI, Amsterdam.

LARIS modularizes the interlocking. A module sends messages to other modules or to the real world. Modules also receive messages from the real world. All these modules are assumed to be asynchronous; therefore, the order the messages are processed in is not fixed. The language is designed to be easily formulated in process calculi to aid in later verifications.

GRACE

From EURIS Siemens derived the toolkit GRACE (Graphical Requirement Analysis CENELEC Engineering). GRACE is based on the graphical, OO representation mode of EURIS, which perfect y matches the geographical approach of SIMIS-W and allows any signaling engineer to become instantly familiar with the tool due the similarity of the method with rely design.

GRACE, also, provides a simulation component for verification of the implemented structures. This can be done in a consultation with the client to efficiently work out the necessary adjustments, which can then be instantly performed and tested with another simulation.

On the validation process is complete, GRACE allows a direct transfer of the structured signaling rules into the target software code. The code transformation process is vital and validated which means that a check of the generated target code is not necessary.

LADDER DIAGRAM

Some interlocking systems realized use ladder logic, a graphical representation of a Boolean circuit, specially tailored for reactive systems.

This low-level language of Boolean equations is particularly amenable to off-the-shelf verification without much effort. The actual formalization of the implementation of an interlocking system is then an instance of this model.

The international standard for programmable logic controllers IEC 61131 [61103] describes the graphical language ladder logic. It gets its name from its graphical "ladder" like appearance which was chosen to suit the control engineers responsible for their design. Each rung of the ladder is used to compute an output variable from one or more input variables in the rung. In the railway industry these input variables are referred to as contacts and the output variables are referred to as coils. A briefly description of the entities representing these variables is as follows:

- Coils - These are used to represent values that are both stored for later use and output from the program. The value of a coil is calculated when a rung res making use of the current set of inputs, the previous set of outputs and any outputs already computed for this cycle. The coil is always the right most entity of the rung and its value is computed by executing the rung from left to right.
- Open Contacts - This entity represents the value of an un-negated variable;
- Closed Contacts - This entity represents the value of a negated variable.

A Ladder logic rung is built using these entities and connections between them. The shapes of the connections between the contacts determine how the value of the coil is computed from them. Using propositional logic for comparison, a horizontal connection between two contacts represents logical conjunction and a vertical connection between two contacts represents logical disjunction.

METHODS, NOTATION AND TOOLS

METHODS

Model driven development (MDD) is a software development process that has been gaining in popularity in recent years. MDD focuses on the models, or abstractions of the software system, rather than on the final programs [26].

These models are transformed, automatically or manually, into code. Executable models are a key component of MDD, as well as such concepts as automatic transformation of models, validation of models, and standardization to enable interoperability of different MDD tools (e.g., OMG's Model Driven Architecture initiative).

NOTATION

Within MDD, state machines are a popular way of modelling the behavior of systems.

With respect to state machines, the most popular formalisms, as represented in the research literature, are UML statechart diagrams (as specified in UML 2.0 [18]),

The Unified Modeling Language (UML) [27] is a widely accepted modeling standard in industry. Without extensions, however, UML does not allow modeling and evaluating of properties like timeliness, throughput, or fault tolerance.

The modelling facilities provided by the conventional state machine modelling technique are used frequently to model real time systems.

Interlocking systems are based upon events happening to which the system responds, and then transitions to another state. This directly maps onto the state machine model for simpler systems.

State Machine Modelling is concerned with modelling the system which it describes as a collection of discrete sates. State machines model these states in a language independent manner. The state model of a system is always in one of its set of possible states, the model transitions from one state to another system state when stimuli are received.

There are limitations to the use of conventional state machine modelling.

One of the limitations inherent in conventional state machine modelling techniques is the complexity of the state diagram increases dramatically as the number of possible states increases. The state machine model then becomes unwieldy to use, this detracts from the intended purposes of state machine models, to allow the system to be modelled in a more comprehensible manner.

Another of the limitations of modelling a computer system as a conventional state machine is the lack of support for concurrent constructs. Traditional state machine modelling is based on sequential transitions from one state to the next. Concurrent systems cannot be modelled in this manner as various aspects of the system may be in different states.

It is seen that the limitations inherent in state machine models include the inherent complexity which occurs as the number of states increases, and also the modelling of concurrent systems.

State machine modelling is the basis for various real-time methods such as that proposed by Ward and Mellor [28] and Harel [29].

Statechart modelling, overcomes the limitations of conventional modelling.

These limitations which have been shown to be present in state machine models are, the complexity of the model and the inability to model of concurrent systems.

Statecharts overcome the limitations of state machines by providing a construct known as the and-state. In the Mealy-Moore state machine model, states were an or-state; they were either in one state or another, never in two states concurrently.

The and-state overcomes this limitation of or-states by allowing the state chart to have substates of a higher level state active at the same time. The state machine model could be decomposed into lower states, however these lower states were still sequential not contemporaneous, in the state chart modelling system proposed by Hare these substates may be concurrent. Each of these substates model an aspect of the object of which they are a part.

These substates are also allowed to communicate with each other in well-defined manners. These communication and synchronization methods have been enumerated by Douglass [30]

1. All orthogonal regions of the chart accept events sent to the object
2. One region may create an event as a result of a transition that is consumed by another orthogonal region.
3. A guard may be used to test if another region is in a certain state before allowing a transition to occur to the guarded state.

The third enumerated point introduces the concept of a guard. This is a construct introduced to Statecharts which adds additional features not available in a state machine model, conditional event responses.

To overcome the second limitation of conventional state machine modelling, that of a quickly escalating level of complexity as the number of states increased, orthogonal regions are all used.

These regions allow the modeler to detail highly complex specification logic in a condensed manner. Harel [31] stated “Modelers can use orthogonal states to decompose large state spaces naturally into independent (or almost independent) parts.”

This flexibility enable statecharts to be far less complex as the number of states increases the modeler can use some of the more powerful facilities provided by statecharts, such as orthogonal regions and guards, to succinctly specify the required logic in a more comprehensible manner.

A brief overview of statecharts has now been provided, showing how the statechart method of system modelling overcomes the difficulties that may be experienced when attempting to model real-time systems using state machines.

CSP (Communicating Sequential Process) is another commonly used method of specifying the behavior of modelled systems. It is also used to model the flow of control from one “state” to another, and also supports concurrency.

A fundamental aspect of CSP as defined by Hoare [32] involves the synchronization of concurrent processes on their inputs and outputs. The synchronization only occurs when two processes wishing to synchronize to send a message, are both in a sending and receiving state respectively.

Statecharts also make provision for the synchronization of states or processes with the functionality provided by guards. These guards are placed at the transition from one state to another. The transition may only occur when the transition event occurs and the condition on the guard is met, a conditional event response. This condition may be that another orthogonal region needs to be in a particular state. Thus these two separate states may be synchronized.

Both techniques are based upon mathematical foundations. This enables a degree of formalism to be inherent in designs produced according to these two methods, formal reasoning is therefore possible with both of these practices.

CSP and statecharts do provide similar functionality in many areas; this is to be expected due to their predominant use in developing real time systems. There are some differences within each of their respective paradigms.

Contrasting CSP and statecharts on a paradigmatic approach statecharts are concerned primarily with clearly and succinctly express complex logic relationships between states.

Hilderink [33] showed that CSP is primarily concerned with the specification of communicating concurrent processes, and describing complex communication patterns. CSP is not suitable for describing the internals behavior of processes, whereas statecharts can address this.

The method by which CSP and statecharts describe the specification is also different. Statecharts provide a graphical representation of the system. CSP provides a character based representation. These two representations each have different advantages and disadvantages, a statechart may be more comprehensible to many people, a CSP description may be easier to include as a comment in source code thereby enabling the programmer easy access to the specification to which he is writing in an easy and unambiguous manner.

Statecharts are also unaware of the concept of time in the same respect as timed CSP. Although statecharts may recognize that transitions take some time (unlike state machines), they do not include the notion of states taking a measured amount of time in the same respect as timed CSP does. When developing real time systems timed CSP provides a better set of tools for dealing with “real time” than statecharts do.

Statecharts were introduced as a development over modelling techniques such as traditional state machine modelling. The weaknesses inherent in state machine modelling such as overly complex diagrams for larger systems, and the lack of concurrent support were addressed by the statechart method.

Statecharts overcame these weaknesses by introducing concepts such as orthogonal regions, and-states and also guards. These concepts allow for the specification of far more complex real time systems than traditional state machine modelling.

Other techniques for developing complex real time systems include CSP. These techniques provide slightly different aspects than statecharts at which systems can be specified. The basis of all these techniques in formal mathematics enables them to be used to comprehensibly specify the system in a provably correct manner. As well as this benefit, work is currently ongoing [31] into developing tools by which automatic synthesis of efficient code can be achieved from a suitably rigorous statechart specification.

Hoare [34] also describes the relationships which can exist between formal models such as programming languages and specification languages.

Statecharts are a methodology by which complex real time systems can be specified in an intuitive graphical manner. They enable complex relationships between concurrent states to be formed, through synchronization techniques and decomposition of states.

TOOLS

There are two major and one minor tools for Statecharts developments:

- I-Logix Statemate tool [35];
- IBM Rational Rhapsody [36].
- Itemis Yakindu [37]

However this tool implements their own dialect of statecharts. Moreover classical Harel statecharts are implemented in Statemate and an object-oriented version of Harel's statecharts [38] in implemented in Rhapsody and Yakindu.

These two dialect appear to be very similar. For instance, at first glance, a model written in one could be easily ported to the other.

However, there are some subtle syntactic and semantic differences between the formalisms which can lead to pitfalls. This is well discussed by [39]

I-Logix Statemate

In the I-Logic Statemate tool the supports the editing of the graphical statecharts notation, it also allows the complete system specification to be executed and graphically simulated, permitting to explore any scenarios determining the system correctness, and evaluating whether the specification meets the requirements.

The Statemate simulator allows to execute the model, permitting to (not exhaustively) verify the behavior examining the animation of the system and producing test scenarios that may be used in order to test the target system. Furthermore it allows to animate panels to have an evidence of the model behavior, so that we can generate easily “what-if” scenarios.

Statemate provides also the automatic generation of a C-based or Ada-based prototyping source code based on the model.

Another important recent add-on to Statemate is a powerful model checker, which is obviously an advantage in terms of the confidence that can be acquired on the specification correctness.

IBM Rational Rhapsody

As in the UML standard, all model elements defined in a diagram such as functions, classes or use-cases have a package to which they belong.

All the information present in the diagrams is also present and modifiable in the packages. This means that it is possible to build a model by only modifying the packages, but this would take away much of the UML’s advantage, to have a graphical representation of the model.

Each executable file or library that the tool may generate is represented by a component. It has a scope indicating which packages it should compile. External files, not generated by Rhapsody, may also be included in the component and they can optionally be compiled together with the rest.

There are also settings, the most important ones concern operating system of the target, instrumentation (see below), include files and switches to compiler and linker. Each component normally has its own directory in the file system to which source code, object files and executables are generated.

UML has a unified syntax for all diagrams, but at first sight it is not obvious how different diagrams are related to each other.

In Rhapsody, the diagrams are connected as follows. Statecharts must belong to a class, use case or actor. The same is true for activity diagrams, with one addition: they may also belong to an operation or function. No element may own both a statechart and an activity diagram. It is possible to associate a sequence diagram with one or several use cases.

The process of going from a system model to implementation code in some programming language is called forward engineering. In Rhapsody this is done automatically; code is generated from the UML model. The other way around is called reverse engineering.

Old source code is analyzed and somehow integrated in the model. Roundtrip engineering is a combination of both the others. The development is done partly in the model and partly in the source code. Forward and reverse engineering are used to synchronize the model and code.

The generated code is based on the contents of the packages, which also reflect the diagrams as described above. The structure is thus automatically generated, involving for example how source files include each other.

Behavior is translated into code for statecharts and activity diagrams associated directly to classes or actors, but not to functions. Implementation code written for functions or object methods are automatically placed in the source files. The sequence of messages in collaboration or sequence diagrams does not affect the code.

Rhapsody enables the program model to be executed and tested in the development environment. A kind of instrumentation referred to as design level debugging may be used. This means that the information from the debugging is related directly to the model instead of to the source code generated from it. Animation is an instrumentation, which enables the developer to follow events being sent, state transitions taken, and attributes changed in the model during execution of the generated program. Tracing is a form of textual instrumentation.

Properties may be set for the entire project down to individual model elements, such as components, classes or functions. They regulate how the notations of the model translate to code, how Rhapsody integrates with other programs and how the tool generally functions.

ITEMIS Yakindu

Yakindu is an open-source-toolkit built on Eclipse for the model-driven development of embedded systems. Through the systematic use of models, it aims at an integrated development process and an increase in quality and maintainability.

Yakindu is not a monolithic application but a set of independent language modules. These modules currently comprise the following list:

- SCT - statechart tools
- Damos - data-flow oriented modelling
- Mscript - math oriented scripting

- CReMa - (requirements) traceability
- CoMo - component model

The language modules are not bound to any specific methodology. They are extendable and can be used in any domain specific tool chain independent from each other.

The only common basis of the language modules is a type system implementation with respect to the SI-units standard.

The statechart tools (SCT) module is used to model reactive systems. They are designed to continuously interact with the environment the system is embedded in.

Beside the fact that events from the environment asynchronously trigger the transitions of the statecharts, their core semantics is cycle driven. Thus, processing of concurrent events is enabled.

A model interpreter is used to simulate the behavior of the statecharts. During simulation, transitions and current states are visualized and events from the embedding environment can be triggered by the developer. Various code generators exist to bring the statecharts onto the embedded device.
Anyway, the model interpreter for simulation and the generated code follow the same core semantics.

In contrast to SCT, Damos is a data-flow oriented modelling tool. It uses block diagrams as syntax to describe system component functions and data-flow connections between them.

Data could be for example physical quantities like current or voltage. Damos is used to model control systems or digital signal processing. Damos supports a "two-dimensional" structuring of models.

In the first dimension, subsystems can be used to hierarchically structure the models for enabling various implementations. A subsystem defines its provided interface. Subsystem realization models implement those interfaces.

Thus, product line engineering for example is supported by implementing different realizations depending on the various product's specifications. The second dimension is established by system fragments which can be used to divide the model into multiple parts.
For example, to support simulation of the model, it might be divided into a functional fragment and a simulation fragment.

Thus, the "outer" world the model should be embedded in can be encapsulated into a separate fragment which in turn can be removed once the model is brought into the productive environment.

The Mscript module is a mathematical DSL to define functions independently from the type system and its type ranges or type conversions. It can be used for example in Damos to model system component functions. The code generation maps the Mscript functions to the concrete type system used on the embedding environment.

The type system the Yakindu toolkit builds upon handles numeric data types by incorporating their units of measurement with respect to the SI-units standard.
When no units are specified, a dimensionless value is assumed. Units are automatically converted and calculated. The type system is used among others for model validation and simulation.

The Cross Relational Manager (CReMa) is a tool that brings a tracing infrastructure into any development environment. It connects pairs of so called trace points to define traces between two development artifacts. A trace point binds an artifact of any kind -a piece of source code, a requirement, a document, a model element, etc. - to a trace.

Artifacts might be assigned to multiple trace points and therefore to multiple traces. Thus, traces for example from a requirement specification via some model element to a piece of source code can be established to follow some specification to its implementation.

CReMa manages these traces and provides a user interface to create and follow the traces. Its extendable architecture allows for implementation of new trace point providers. They are a non-invasive means to enable traceability also for other development artifacts.

CReMa already ships with a requirements model based on ReqIf, an OMG standard for requirements specification.

The component model (CoMo) module is currently under development. It can be used to specify architectural elements of the system to be modelled. In the context of the Yakindu toolkit, components might be implemented by statecharts or block diagrams.

Anyway, the component model will be open to also be used by any other tool in a tool chain that incorporates Yakindu.

CONCLUSIONS

This reports resumes the sufficient knowledge in modeling railway interlocking systems.

Being SIL4 typed system the specification and the verification have to be exploited by using formal language. In literature there was a wide amount of works that study the effort and drawbacks of different formal languages on well-established interlocking system as SSI or modern system as Alstom proposal.

Moreover, developing interlocking system model it could be addressed from different points of view:

- functional description;
- geographical description.

The formal language which is suitably for the domain object is Harel 'statecharts and the relative tool is the IBM Rational Rhapsody however Yakindu is an interesting counter-choice.

BIBLOGRAPHY

[1] L. Zigterman, A New Approach to Specification and Design of a Railway-interlocking: The "route" Approach. An Exercise in Top-down Design with Stepwise Refinement, Nederlandse spoorwegen, 1984.

[2] Cribbens, Furniss e Ryland, «The solid state interlocking project,» *IEE Conf. Publ. 203,* pp. 1-5, 1981.

[3] K. Akita, T. Watanabe, H. Nakamura e I. Okumura, «Computerized Interlocking System for Railway Signalling Control: SMILE,» *IEEE Transactions on Industry Applications,* Vol. %1 di %2IA-21-4, pp. 826-834, 1985.

[4] wikipedia, «Lists of rail accidents http://en.wikipedia.org/wiki/Lists_of_rail_accidents,» 2012.

[5] C. E. 50128, «Railway applications—Communications, signalling & processing,» 2001.

[6] F. M. Stefania Gnesi, «On the fly model checking of communicating UML State Machines,» 2001.

[7] S. Vanit-Anunchai, «Modelling Railway Interlocking Tables Using Coloured Petri Nets,» in *Coordination Models and Languages*, Springer, 2010, pp. 137-151.

[8] W. Fokkink e P. Hollingshead, «Control Tables to Ladder Logic Diagrams,» in *Proceedings of the 3rd Workshop on Formal Methods for Industrial Critical Systems (FMICS 1998)*, Amsterdam, 1998.

[9] David Tombs (SVRC); Neil Robinson (SVRC); George Nikandros (QR), «Signalling control and table generation,» RTSA, Wollongong, 10-13 November 2002.

[10] A. Fantechi e M. Banci, «Geographical Versus Functional, Modelling by Statecharts of Interlocking Systems,» *ENTCS,* vol. 133, pp. 3-19.

[11] A. Fantechi, «Distributing the Challenge of Model Checking Interlocking Control Tables,» *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies,* vol. 7610, pp. 276-289, 2012.

[12] J. Moses, «Foundational issues in engineering systems: A framing paper,» MITesd, March 29-31, 2004.

[13] Blake Kozol, Siemens ; Bob Banks, TriMet; Rande Bird, TriMet, «The application of a distributed architecture vital interlocking platform on the TriMet I-205,» AREMA, 2009.

[14] Tetsuya Yuge ; Shin Sasaki Railway Technical Research Institute, Japanese National Railways, «Train radio system using leaky coaxial cable,» Tokyo, 1984.

[15] G. Dipoppa, G. D'Alessandro, R. Semprini e E. Tronci, «Integrating Automatic Verification of Safety Requirements in Railway Interlocking System Design,» in *Proceedings of the 6th IEEE International Symposium on High Assurance Systems Engineering (HASE'01)*, Boca Raton, Florida, USA, 22-24 October 2001.

[16] Clarke, Emerson e Sistla, «Automatic verification of finite-state concurrent systems using temporal logic specifications,» *ACM Transactions on Programming Languages and Systems 8,* p. 244–263, 1986.

[17] P. Bellini, R. Mattolini e P. Nesi, «Temporal logics for real-time system specification,» *ACM Computing Surveys (CSUR) Volume 32 Issue 1,* pp. 12-42, March 2000.

[18] T. Lecomte, T. Servat e G. Pouzancre, «Formal Methods in Safety-Critical Railway Systems,» 2007.

[19] K. M. Hansen, «Formalizing railway interlocking system,» 1998.

[20] Eriksson e Johansson, «Using formal methods for quality assurance of interlocking systems,» 1998.

[21] W. Fokkink, «Safety criteria for Hoorn-Kersenboogerd railway station,» 1995.

[22] C. Eisner, «Using Symbolic CTL Model Checking to Verify the Railway Stations of Hoorn-Kersenboogerd and Heerhugowaard,» 2006.

[23] M. Morley, Safety Assurance in Interlocking Design, University of Edinburgh, Department of Computer Science, 1996.

[24] M. Banci e A. Fantechi, «Some experiences on Formal specification of Railway Interlocking Systems using statecharts».

[25] A. Anselmi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi e F. Torielli, «An Experience in Formal Verification of Safety Properties of a Railway Signalling Control System,» *Safe Comp 95,* pp. pp 474-488, 1995.

[26] B. Selic, «The pragmatics of model-driven development,» *IEEE Software,* vol. 20, n. 5, pp. 19-25, 2003.

[27] Object Management Group. , «Unified Modelling Language Specification http:// www.uml.org,» 2003.

[28] P.Ward e S.Mellor, «Structered Development for Real-Systems,» in *Structered Development for Real-Systems*, Prentice Hall, 1985, p. 86 & 302.

[29] D. Harel, «Statecharts: A visual Formalism for complex systems,» *The Science of Computer Programming,* n. 8, pp. 231-274, 1987.

[30] B. Douglass, «UML Statecharts».

[31] D. Harel, «Executable Objetc Modeling with Statecharts,» *IEEE COMPUTER,* vol. 30, n. 7 July, pp. 31-42, 1997.

[32] C. Hoare, «Communicating Sequentatial Process,» *Prentice Hall International Series in Computer Science,* 1985.

[33] G. Hilderink, A.Bakkers e J.Broenik, «A Distribuited Real-Time Java System Based on CSP,» *The Third IEEE International Symposium on Object-Oriented Real-Time Distribuited Computing ISORC 2000,* pp. 400-407, 15-17 March 2000.

[34] C. Hoare, «Mathematical Models for Computer Science,» 1994.

[35] I-Logic Inc, «Statemate Magnum Simulation Reference Manual,» Burlington, MA USA, 2003.

[36] IBM, «IBM Rational Rhapsody, http://www-01.ibm.com/software/awdtools/rhapsody».

[37] YAKINDU, «Yakindu, http://statecharts.org/».

[38] D. Harel e H. Kugler, «The Rhapsody semantics of statecharts (on the executable core of the UML),» in *Softspez Final Report, LNCS 3147*, Springer, 2004, pp. 325-354.

[39] M. L. Crane e J. Dingel, «UML vs Classical vs. Rhapsody Statecharts: Not All Models are Created Equal,» School of Computing, Queen's University, Kingston, Ontario, Canada, 2005.

[40] Russo e Ladenberger, «A Formal Approach to Safety Verification of Railway Signaling Systems,» 2012.

[41] Thales Canada Transportation Solutions C. Frase, «The three r's of modern signalling education – remove, recycle & repeat,» AMIRSE, 2012.

[42] RFI, «SCC e CTC per il telecomando della circolazione».

[43] A. Fantechi e M. Banci, «Instantiating Generic Charts for Railway Interlocking Systems,» in *FMICS* , Lisbon, 2005.