UiO : Department of Informatics
University of Oslo

Railway Interlocking Design Support Tools

AutoCAD Plugin based approach

Master Thesis

Shahzad Ali Khan

Autumn 2016

RAILWAY INTERLOCKING DESIGN SUPPORT TOOLS

Shahzad Ali Khan

Master's Thesis

Informatics: Programming & Networks
Department of Informatics, University of Oslo
Norway

Title:

Railway Interlocking Design Support Tools

AutoCAD Plugin based approach

Author:

Shahzad Ali Khan

Supervisor:

Primary Supervisor: Christian Johansen (University of Oslo, Norway)
Bjørnar Steinnes Luteberget (University of Oslo, Norway)
RailCOMPLETE AS (http://www.railcomplete.no)

Oslo, November 2016

2016

Railway Interlocking Design Support Tools

AutoCAD Plugin based approach

http://www.duo.uio.no/

Printed: Reprosentralen, Universitetet i Oslo

Dedication

To my Parents, without their love and support I couldn't have been able to even write these lines.
To my siblings and Tooba.

Acknowledgement

Firstly, I am thankful to Almighty **Allah,** Who helps me in every single step and stage of my life, bestows me with His blessings without considering my negligence. I also offer praise to the Holy **Prophet Muhammad** ﷺ (Peace Be Upon Him), who is my source of inspiration and a blessing for mankind.

I am in forever debt of my parents for all their love and support, who provided me means to have best education possible. I cannot thank them in limited words for their effortless contributions to my wellbeing, their sacrifices and support for me.

I want to express my heartiest gratitude to my supervisors Christian Johansen and Bjørnar Steinnes Luteberget for their valuable guidance and encouragement in all phases of this research work. Their humbleness has inspired me during stressful days of this project. I am also thankful to the Department of Informatics, University of Oslo (UIO) for offering me opportunity to study in this oldest, largest and leading university of Norway. UIO provided me a peaceful environment and always shown a helpful attitude in my educational activities.

I would also like to thank Claus Feyling for his assistance, help and dedication to make this research fruitful and I am grateful of his IT team at RailCOMPLETE for their input in testing and integration of my project.

Last but not the least, I thank everyone who has given me their valuable time and helped me during my journey of this degree. May Allah bless all. Amen

Abstract

Railways are considered safest of all in land transport system, yet this safety is due to a complex system of interlocking which ensures the safe train movements on track. Data related to interlockings for a railway track is stored manually in (long) tabular forms, which is a time consuming and resource intensive work, since in most cases this approach results in redundant records, data correctness issues and expensive maintenance of interlocking related documents.

With the supplication of computer aided assistance, Engineers are now able to overcome this problem. But due to variance of practices in different railway systems, there is a lack of generic approach in this domain for easy manipulation of Interlocking records. We propose a software implementation of interlocking schema, based on interoperable open XML based data exchange format for railway applications, called railML. Our implementation is directed toward Norwegian railway practices and provides functionality to dynamically create/modify and load interlocking schema from a RailCOMPLETE document, while this XML schema is derived from interlocking specifications presented by Bob Janssen *[Bosschaart, Mark, et al. "Efficient formalization of railway interlocking data in RailML." Information Systems 49 (2015): 126-141.]*

Incorporating a user-friendly editor for interlocking in this prototype performed very promising by limiting excessive work hours and resources being spent without a computer-aided tool and relates improvement in reducing errors in interlocking data. Based on our interaction with railway engineers we present future direction for forthcoming research in this specific domain as more generalized improvements in domain model can make this system reach to wider user base and in turn lead to an easy to understand and configure interlocking design editor for majority of railway systems.

Table of Contents

Index:

S

T

U

W

X

Figures:

CHAPTER 1

INTRODUCTION

1. Introduction

This thesis is concerned with the research and development for the automation of Interlocking data in RailCOMPLETE ® software. This project is done in cooperation with RailCOMPLETE AS, a Norwegian firm developing railway signaling and designing software.

1.1. Initial Motivation and Research Question

Railway is among the safest forms of transport even though trains travel and pass each other at high speeds and through webs of railway crossings. This is made possible by the interlocking, a relay electronics or computer system which controls signals, turnouts, detectors, and other equipment, to ensure that only safe movements will be allowed.

The design of a train station includes the creation of:

- A station infrastructure layout, consisting of tracks, switches and signaling components (such as signals, train detectors, derailers and balises)
- A tabular interlocking specification, giving restrictions and conditions on the use of the train station by referring to the elements of the infrastructure layout

The research in this thesis aims to address the following research question:

Q1 : How can we improve the efficiency of interlocking design?

Hence, improvement in efficiency of interlocking design formulates the goal of this thesis, which is to create an efficient editor for the tabular interlocking specification in user-friendly form of schema, if the infrastructure layout is already available and editable. Currently, engineers experience a lack of specialized software tools to support the design process. The editor is meant to help the engineers by providing a software tool integrated with the CAD tool where the infrastructure layout is designed. Even more, much of the manual process of generating tabular interlocking specifications can be also automated. This would allow the engineer to work much faster with the editor, as many times, upon changes from the engineer, many other small changes would be done automatically by the editor to ensure consistency of the whole tables. Automation aspect of this tool could be investigated in future work.

We address Q1 by researching about current interlocking design process to identify problems. Difference of practices and equipment makes the variance in whole design, and for making our research and tool, as generic as scope of this project allows, railML specifications for interlockings are used. This software , Figure 1, loads Interlocking specifications from a RailCOMPLETE document and populate controls for editing the schema dynamically. Railway engineers makes modifications or generate new schema per their needs and load backs the changes into RailCOMPLETE main document for track or route.

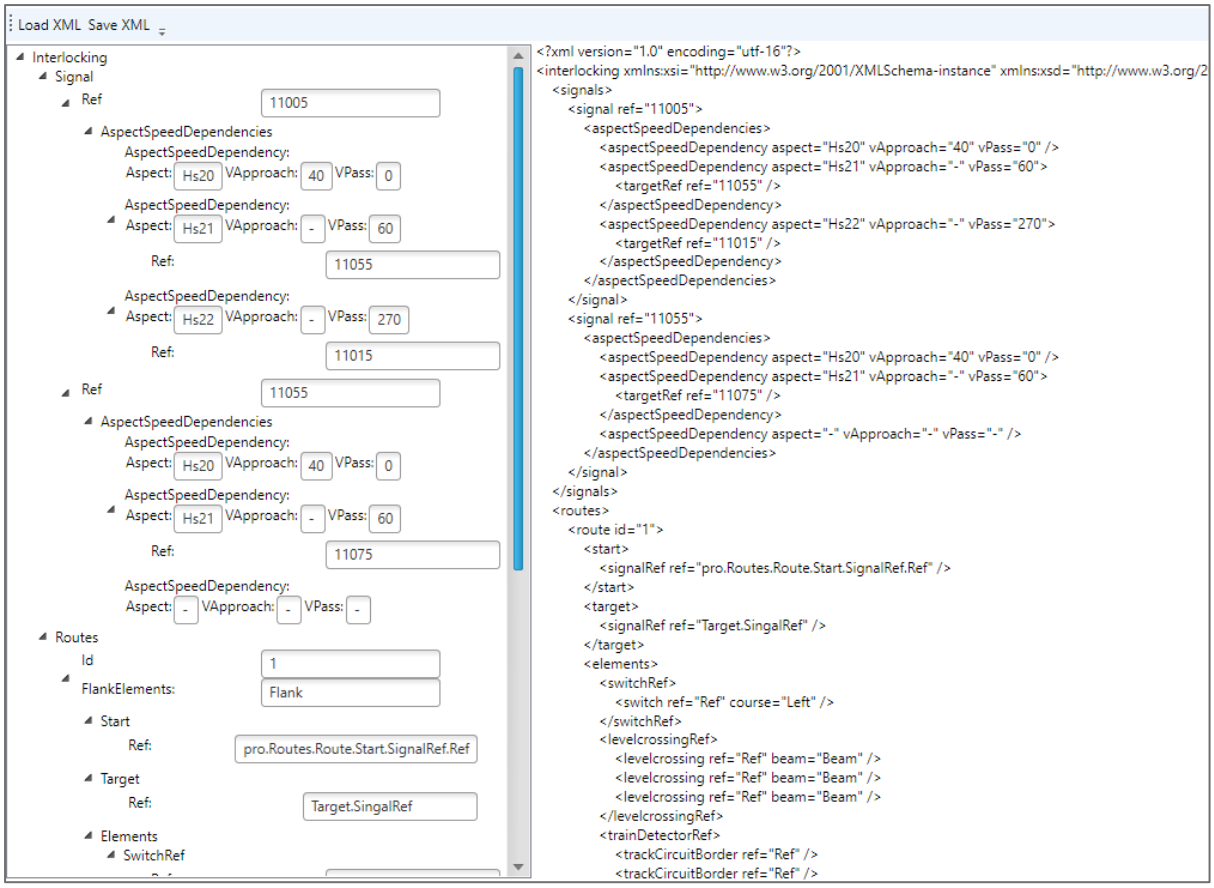


Figure 1 Interlocking Editor Module

1.2. AutoCAD®

AutoCAD [1] is a commercial software application for 2-D, 3-D drafting of designs and layouts. This software is used in many field including but not limited to architecture, electrical, electronics, mechanical, construction and other engineering disciplines for the preparation of blueprints and layout plans. Since this project is about design railway interlocking design support tool development, we are using RailCOMPLETE as a framework for development, which is a plugin for AutoCAD.

1.3. RailCOMPLETE®

Railway lines and stations are continuously constructed or upgraded. Substantial savings and quality improvements are to be made during planning, detail engineering and construction phase with better tools for managing signaling and its related data. Planning of such projects is complicated and it takes more expensive resource allocation to ensure data-corrective and low redundancy measures for carrying out a fine-tuned plan. RailCOMPLETE help engineers and project managers in this task by providing industry specific tooling and assistance in AutoCAD software through RailCOMPLETE plugin. Data flows digitally in standardized railML format in RailCOMPLETE.

RailCOMPLETE is an AutoCAD plugin letting users design tracks, define mileage system throughout the construction stages by parameterized railway objects. It produces high resolution drawings, tables, 3-dimentional views, reports and railML files for easy export and conversion.

In this thesis, we have used RailCOMPLETE framework to integrate our project within AutoCAD. The RailCOMPLETE software consists of an editor for the infrastructure layout, and saves information in the DWG format, a commonly used binary CAD file format. The RailCOMPLETE DWG files also contain a railML representation of the station layout. The aim is to have the Interlocking Editor as a module that can be integrated within RailCOMPLETE.

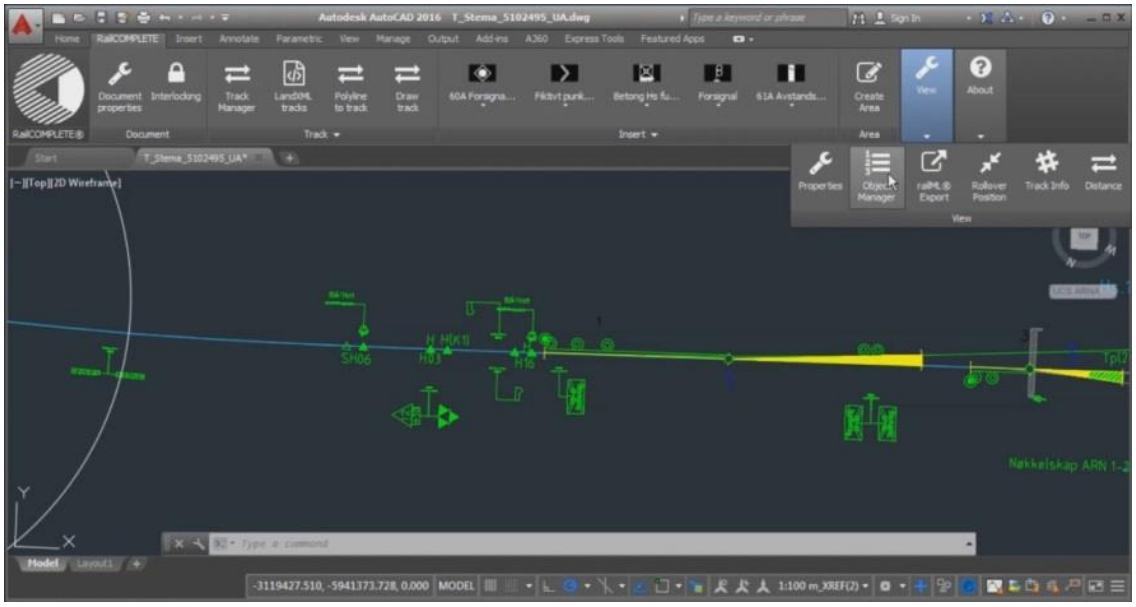


Figure 2: RailCOMPLETE screenshot

RailCOMPLETE have a comprehensive architecture of APIs for different prospects of railway object. It is developed in Microsoft .Net framework, using WinForms and Windows Presentation Framework (WPF). It uses **Microsoft.Office.Interop** [2] libraries as well for data export functionalities.

1.4.Project Structure

In order to achieve the project goal, the project was structured in following fashion:

1. Reading and investigating different key concepts of Railway infrastructure, their role in track designing and operations as discussed in chapter 2.
2. Capturing requirements from railways engineers and RailCOMPLETE developers.
3. Starting to implement object representation of tabular interlocking specification based on RailML extension, discussed in Efficient formalization of railway interlocking data in RailML paper [3]. This object representation was done in Microsoft .Net 4.6 using C-Sharp (C#) and demonstrated in first iteration of prototype.
4. Refined requirements in several sittings with RailCOMPLETE engineers and captured in chapter 3. Iterated further prototypes and gained feedback from actors listed in chapter 3.
5. Documentation regarding evaluation and changes in domain model are done in this step.
6. Integration of Interlocking module with RailCOMPLETE is done in this step. This iteration of prototype is captured in final implementation after testing.

1.5.Project Objectives

The goal of creating an interlocking editor with CAD integration can be divided into the following tasks:

a) Representing interlocking data in object-oriented code structure.
b) Read and write the XML format for the interlocking from a C# program.
c) Create a graphical user interface for editing the interlocking, creating element references to a given railML infrastructure layout.
d) Integrate with RailCOMPLETE CAD framework, saving the interlocking table together with the DWG file using the extension dictionary feature in AutoCAD.

These alone would constitute an improvement for the railway design process. Less time means less resources spent from public funds, but also means better and faster railway infrastructures constructions projects. Having a faster and more user-friendly way of interacting with an existing railway design, and making changes, means that unexpected construction problems can be easier decided and changed for the better. Moreover, this approach may lead to safer designs.

1.6.Organization of the thesis

This thesis contains 5 chapters including introduction. The content of the chapters following, are organized in a fashion mentioned below:

- **Chapter 2**: *Background & Motivation*
  - In this chapter, important railways signaling concepts and a brief history of development in railway infrastructure is discussed. We later relate these concepts with scope of this thesis
- **Chapter 3:** *Planning & Analysis*
  - In this chapter, we present software development methodology we used and our experience during this project, software requirements for interlocking module, use cases, domain model and class diagram of the project
- **Chapter 4:** *Implementation & Software Development Methodology*
  - This chapter contain an overview of RailCOMPLETE framework, implementation details, user feedbacks and technologies used in this project.
- **Chapter 5:** *Summary and Future work*
  - In this chapter, we present an overview of achievements for project objectives, critical reflections, detail summary of this research work, as well as discuss the potential enhancements for further development of Interlocking module.

CHAPTER 2

BACKGROUND & MOTIVATION

2. Background & Motivation

2.1.Background

This chapter provides the necessary background concepts which will enable the reader to have better understanding of the problems and topics brought up during this thesis. I used a very informative book "Railways Operation and Controls" [4] on railways domain by Joern Pachl to discuss railway infrastructure elements and their working. A research project by Andrew Lawrence from Swansea University [5] helped in gathering information about Railway history and interlocking specially. Reader will be able to access the proposed solution in the light of these concepts. This chapter begins with Rail System basics, Interlocking and its applications. Different elements of interlocking control system like conditions, signals, points and flank protectors are further described after wards. Before the final part where I will present a sample route with diagram and its content being filled in RailML format, I will discuss currently used technology and research being done on this domain.

2.1.1. Railway System

Since beginning, railway and its control systems have seen many changes and advancements. Its infrastructure, control and safety has seen manual systems (human aided mechanical and non-mechanical) and todays automatic electronic system with minimum human input. Yet a railway system consists of their essential and most elementary elements in all over the world which are infrastructure such as track work, signals, station and lines etc., rolling stock with cars and locomotives and last but most important driving factor which is operating rules and procedures for a fail-safe operation. According to Joern Pach, Infrastructure and rolling stocks makes the hardware while operating rules are software of railway system [4]

2.1.1.1. History of Railway Signaling and Control Systems

Back in time, in early days, a human source (Police men) was used to provide signals to trains using colored flags during day and lights at night. Purpose of using signals were to provide real-time track information to train driver. But this was not enough for safety as this communication was one way and there was no system to track train location, once it went out of sight. No real schedule was possible as it was before the time of telecommunications and electricity. Hence clocked timers for delay and slow speed were the only key factors progressing toward safety of passengers and equipment.

The first major advancement in railway signaling was the conception of Fixed Signals. [5] Fixed Signals were wooden boards mounted on rotating posts where a visible Board was a stop signal, and if board was not visible, that means Track is clear to proceed. One improvement on the basic fixed signal is the disc and crossbar. Rules were drawn as either disc or the crossbar must be visible at any given time frame, where Disc denotes Proceed Aspect and Crossbar visibility translates into Stop Aspect.

Figure 3 Disc & Crossbar signal

Later a new signal called the semaphore was introduced back in 1841. Semaphore signals were used by Napoleon's army (when coupled with telescopes) which later were adopted by the railways [6] [7] . This signal consists of a moveable signal arm which could be positioned at different angles and an oil lamp for night operations.

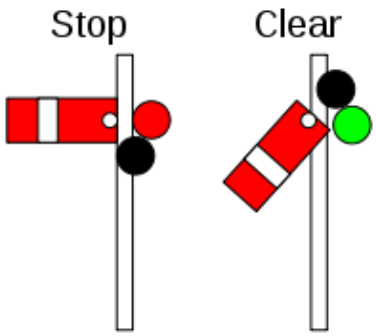


Figure 4 Railway semaphore signal [8]

Up to this point one human resource was needed to control one signal. Then came the era of **signal box** which as a central point for all the signals connections with cables which consist of a system with pulleys, wires and lever. A signaler (one or two) sets the signals using physical levers in signal box. [5] This allowed for the development of the **interlocking**, which was prevention of signaling system from entering an unsafe state. Interlocking provides locks for lever in signal box which ensures that levers can only be moved if it is safe to do so.

During the same time, communication between railway stations was established initially using Morse code transmitting along telegraph lines. Convention of **block** system was introduced. A piece of track between two signal boxes was called a block , defining three stats of operation naming *blocked line* (failsafe default state), *Clear line* or *train on line*. Human intervention and manual inputs even after interlocking was not ensuring the prevention from possibility of human error occurrence until 19th century when track circuits were made electrified which enables the detection of trains automatically. These **electric track circuits** were to be installed along whole segments of track which in turn made automatic signaling achievable. Start and stop aspects were displayed automatically based on train presence in a specific segment of track. This automatic signaling was named track **circuit block signaling**.

Following the general trend of electrification in the railways, points were later electrified. **Electric point machines** enabled points to be controlled remotely as they were moveable with a mere flick of a switch. These electric point machines reduced the large amount of physical interventions performed by signalers(human) allowing for a smart control over large area by each human source. Later, idea of electro-mechanical become more and more popular and mechanical relays were replaced by electro-mechanical versions, reducing space needed for a signal box.

So far semaphore signals were being in use during all these improvements in railway infrastructure, even though with the invention of electric light bulbs, comes the era of scientists and engineers struggling for color light signals which were bright enough to be seen under sunlight. Starting in 1904,

most of these initial iterations of light bulbs were used in low speed applications, but final improvement came in the early 1920s with Corning's "High Transmission" glass colors increasing the range to 1100m under daylight [7] .This leads to replacement of semaphore to light signals in railway. In 1930s the mechanical levers were replaced with an electronic control panel of switches, buttons and indicators. Which later made a way for preconfigured **route settings** where a single press of a buttons would activate signals and points for a route. With the invent of microprocessors in 1980s caused a complete overhaul for railways as how it performs now in term of technology. Replacement of relay and mechanical interlocking with solid state electronic interlocking systems (SSI) was the major breakthroughs in Railway history.

In the following section, we will look through some of these infrastructural elements in details and then relate our motivation and problem statement under the light of these concepts.

2.1.2. Interlocking

Interlocking is one of the most crucial and highly sensitive aspects of these operating rules which ensure the safety of the railway. In simplest words, its job is to process and analyze the commands and requests received in control system, according to predefined set of rules and check whether future state of current railway movement is safe or not. It conveys command to the physical infrastructure if control signal it receives does not violate the safety rules defined in a rail system or conflicts with other routes.

Term interlocking is usually used in two meanings as described in Railway Operation and Control,

"An interlocking is the interlocking plant where points and signals are interconnected in a way that each movement follows the other in a proper and safe sequence. Secondly, the principles to achieve a safe inter connection between points and signals are also generally called interlocking [4]

Interlocking on route must ensure the following condition on a train route.

- All points on track must be set and locked properly and appropriately
- Conflicting routes must be locked
- The track must be clear and not occupied or on fault state

Above mentioned conditions are brought by the usage of following functions.

- Interlocking b/w points and signals,
- Route locking
- Locking conflicting routes
- Flank protection,
- Track clear detection.

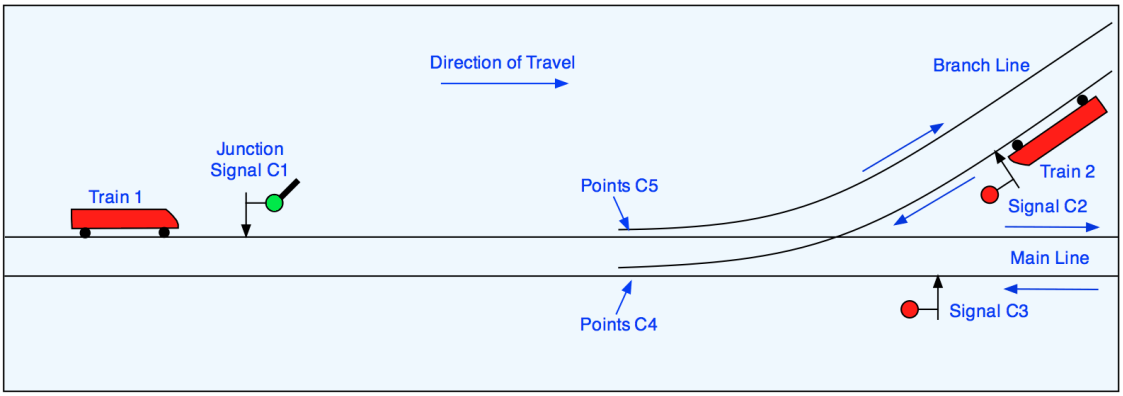


Figure 5 Interlocking. In this case, the interlocking could allow Points C4 to be set for the main line route and permit Signal C3 to show a proceed aspect. [9]

In some places, the term "interlocking" is used to denote an area controlled by a signal cabin or by a computer. As discussed in previous section (history of railway signaling), originally, interlocking was done by a combination of mechanical connections to the operating levers and electro-magnetic relays. Nowadays, most new systems employ "solid state interlocking" (SSI) or computer based interlocking (CBI) using modern electronics instead of old electromagnetic relays.

Railway systems where train movement signals are different from shunt signals, interlocked routes are considered separately from shunting routes as well. Shunt signals are used in shunting movements, which refers to any activity on track, which is used for the management of rolling stock/locomotives without power. Shunt route may execute a movement into an occupied track without flank protection usually. There are examples of railway systems like older German interlocking System [4] where interlocked routes are a requirement for only train movements, while shunt movements are carried out without protection, while in North American railways, train movements are not separated usually from the shunting movements so same interlocking are applied (might be) for both movements.

A train route starts always from an interlocking signal, while its exit can be another interlocking signal for exit /destination or the end of the interlocking/home signal limits.

Following is an example of tabular Control table used in Indian Railways.

S.No.	SIGNAL NO.	ROUTE	ROUTE HELD BY		CONTROLLED BY TRACKS	LOCK & DETECTS POINTS		LOCKS ROUTE POINTS	REMARKS
			APPROACH TRACKS	BACK LOCK TRACK		NORMAL	REVERSE		
1	2W	2WBI	A2WT(120SECS time delay)	2WT A2ET 2T 2BT	2WT, A2ET 2T, 2BT, BT IBT	1	2	4EB 2E	Controlled by closed position of LC gate
2.	2W	2WBII	A2WT (120 SECS time delay)	2WT A2ET 2T 2BT	2WT, A2ET 2T, 2BT, BT IBT, IT A1 WT	–	1,2	4EB 2E 3WB	Controlled by closed position of LC gate

Figure 6 Control table Sample [10]

2.1.3. Route Locking

After resetting the signal to stop position it is required to maintain the points locked because engine/train may not have cleared those points yet. That mean, once a driver has been given a clear signal indicating a route, it is highly important that route must not be changed. It should be certain that route must not only be locked by the signal but also by a locking appliance that works independently from the signal. This second locking must prevent the operation of the points until they have been cleared by the train on track. [4] Electric locking take effects on signal when train passes it and are used to prevent external manipulation of physical levers.

2.1.4. Flank Protection

Flank protection should prevent vehicles from running into a route that is cleared for an approaching train. They shall usually be provided to protect the route and overlap for a signaled train movement [11] .

This could be achieved by

- Operating rules
- Flank protection devices
    - Safety points
    - Derails
    - Stop signals

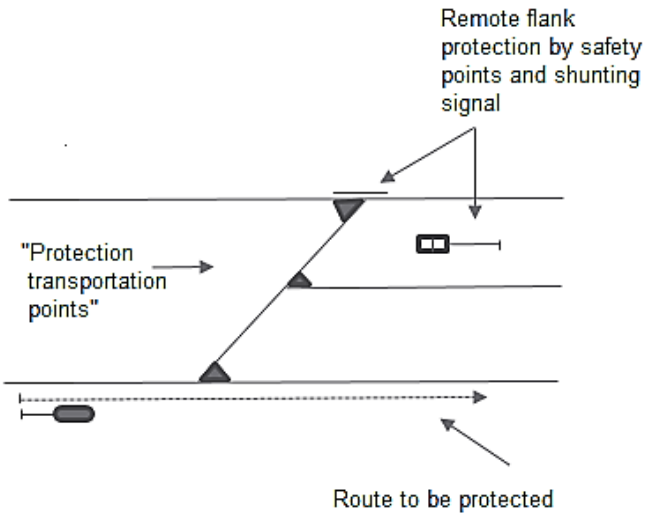


Figure 7 Remote Flank protection [4]

Flank protection by operating rules means that if a train has a movement authority to run through an interlocking any shunting movements on side tracks from where the train is approaching are strictly prohibited. Since this form of protection is not very effective it should only be used when other forms of flank protection are not possible or applicable.

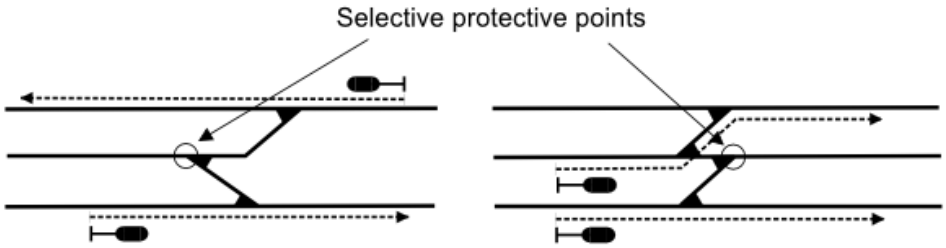


Figure 8 Selective Protective Points [4]

Stop signals are only useful for flank protection against movements controlled by a driver who pays attention to the signals. When it is necessary to have protection against vehicles which could get into motion unintentionally or by accident like stock stored on track, flank protection must be effected by safety points or derails

2.1.5. Crosslock

In a Crosslock, a pair of points is permanently interlocked with a derail or with another pair of points.

Crosslock improve the safety of movements made without locked routes by preventing movement over derails and trailing points which are not properly configured. It also helps by providing flank protection by locking derails and points in a protective position.

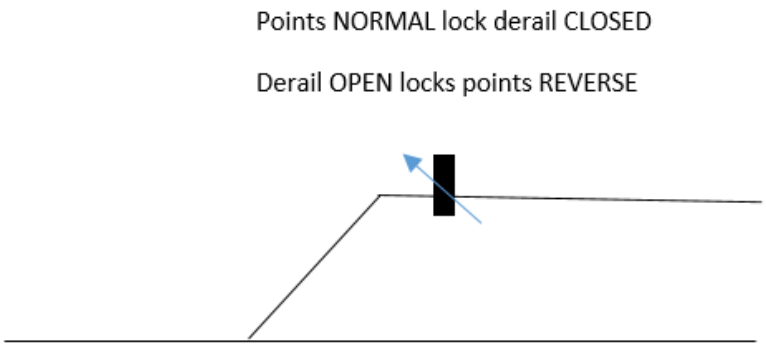


Figure 9 Example of a Crosslock [4]

A multiple Crosslock is a special kind of crosslock where the interlocking between two points depends on the position of a third pair of points or derailers. The first two pairs of points are interlocked to the third element in form of a simple Crosslock [12].

Crosslocks between points are very common in interlocking arrangements following British and North American practices while in German interlocking Crosslocks between points are very rare [4]. But, they are regularly used for interlocking between point and derails.

2.1.6. Intermediate Points

Intermediate points are points which are located between two interlocking signals in a way that a train that is waiting at the signal ahead would not clear the points. (see Figure) [4]. On railways where overlap is used the locking of intermediate points is usually released together with the points inside the overlap behind the exit signal [4]. On lines without overlaps, intermediate points may be equipped with a time release or they are kept locked until they have been cleared again. Intermediate points must be locked when the signal ahead is cleared. Therefore, intermediate points must be interlocked both to the entry and exit signal in front and rear.

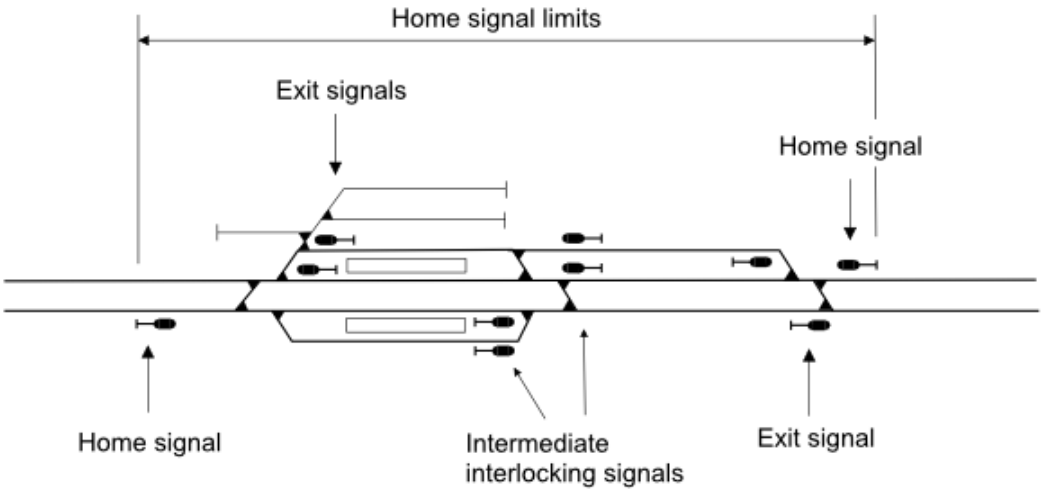


Figure 10 Home signal limits with intermediate interlocking signals [12]

2.1.7. Interlocking Signals

Interlocking signals are used to

- Authorize train movements
- Provide flank protection

Interlocking signals are located at all entrance points of interlocked train routes. An interlocking signal is always placed in the required overlap distance before the first point to be protected by this signal. This point could be: [4]

- Facing points which are not locked when a train approaches this signal
- The fouling point limit of trailing points or crossings on which conflicting movements are possible,
- The rear of a train that has a scheduled stop (this also applies to block signals outside of an interlocking)
- A shunting limit board.

Interlocking signals which are not used for regular train movements are usually dwarf signals which can often only show a restricting proceed aspect. However, this is not a shunt signal in the European sense because it may authorize a train movement. Successive interlocking signals inside one interlocking are not typical on North American railways but can be found in some interlocking at large passenger terminals [12].

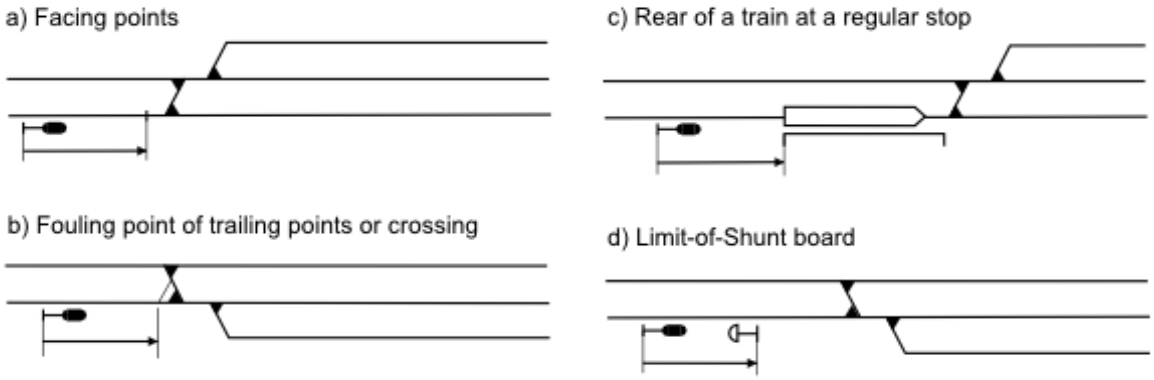


Figure 11 Points to be protected by an interlocking Signal [12]

2.1.8. Shunt Signals

Shunt signal can only be found on railways where the operating procedures require a different signaling for trains and shunting movement. In a typical rail network where shunting signals are employed, following are the key characteristics of shunt signals: [4]

- Authorize shunting movements,

- Provide flank protection
- Shunt signals are placed at all points
- where flank protection is required at a track without an interlocking signal
- At all locations where it is useful for the regulation of shunting operations.

Shunt signals are not only used where shunting movements are governed on main tracks but also in sidings. In modern interlocking, there are a lot of shunt signals in the track layout because all shunting movements are carried out on interlocked shunt routes. [12]

A shunt signal must also be cleared when a train is passed by, so that the stop aspect of the shunt signal is removed, other than authorizing a shunt movement.

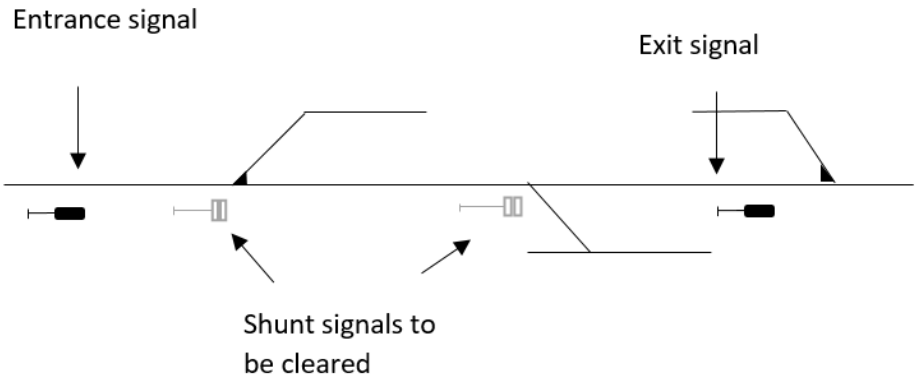


Figure 12 Shunt signals to be cleared for a train route [4]

2.1.9. Internal Logic of Interlocking systems

Following are the two different principles related to the internal logic of an interlocking system.

- Tabular interlocking
- Geographical Interlocking

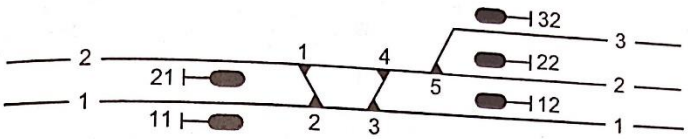
2.1.9.1. Tabular Interlocking

In a tabular interlocking system, the lock between points and signals is achieved in form of a locking sheet that contains the locking conditions for all routes. Tabular interlocking is further differentiated into two very distinct kinds which are

- Cascade locking
- Route-related locking

2.1.9.1.1. Cascade Locking

The cascade locking was originated as a traditional British interlocking practice that was further adopted by North American rail systems. In cascade interlocking, a route is established by a locking cascade which take effects by conditional and permanent locking between different point and further between points and signals. [4]



	IF	LOCKS
11		1 2 3 ③
	3	4 12
	③	④ 5 ⑤
	③ 5	22
	③⑤	32
21		1 ① 3 4
	1	2 5 ⑤
	1 5	22
	1 ⑤	32
	①	② 12
12		2 ② 3 4
	2	1 11
	②	① 21
22		1 2 4 ④ 5
	4	3 21
	④	③ 11
32		1 2 4 ④ 5
	4	3 21
	④	③ 11

2 Points No. 2 locked in normal position
② Points No. 2 locked in reverse position
2 ② Points No. 2 locked in any position

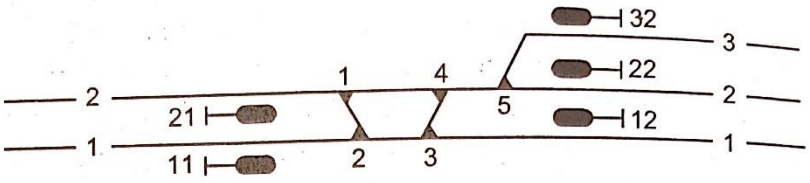
Figure 13 Cascade Locking table [4]

A permanent locking of points and signal is used when clearing the signal requires the points always in same state/position. A conditional locking of points and signal however is dependent on the position of other points. this is being employed that when a signal serve as an entry signal for different routes, the position of the facing points serve as the condition for locking relative points.

2.1.9.1.2. Route-related Locking

In route-related locking, which is more often termed as route-based locking, for each route there is a special route locking element instead of directly connecting points and signals by locking sequences. This special route can only by operated when all points are sets properly for said route. Operating the route locking element will cause lock on all points, throughout the route and release clear signal. Whereas, while there is a clear signal, the route locking elements cannot be released or set to change positions. In this fashion signal indirectly locks all route points.

This type of locking is based on German practice which has been adopted by several European rail systems now.



ROUTES	CONFLICTING ROUTES												POINTS				
	11-1	11-2	11-3	21-1	21-2	21-3	12-1	12-2	22-1	22-2	32-1	32-2	1	2	3	4	5
11-1	-	|	|	|			||	|	|		|		+	+	+	+	
11-2	|	-	|	|	|	|	|	|	||	|	|	|	+	+	-	-	+
11-3	|	|	-	|	|	|	|	|	|	|	||	|	+	+	-	-	-
21-1	|	|	|	-	|	|	|	||	|	|	|	|	-	-	+	+	
21-2		|	|	|	-	|		|	|	||	|	|	+	+	+	+	+
21-3		|	|	|	|	-		|	|	|	|	||	+	+	+	+	-
12-1	||	|	|	|			-	|	|		|		+	+	+	+	
12-2	|	|	|	|	|	|	|	-	|	|	|	|	-	-	+	+	
22-1	|	||	|	|	|	|	|	|	-	|	|	|	+	+	-	-	+
22-2		|	|	|	||	|		|	|	-	|	|	+	+	+	+	+
32-1	|	|	||	|	|	|	|	|	|	|	-	|	+	+	-	-	-
32-2		|	|	|	|	||		|	|	|	|	-	+	+	+	+	-

Routes: | Conflicting routes locked by plain locking
|| Conflicting routes locked by special locking
- Main diagonal in route matrix

Points: + Points locked in normal position
- Points locked in reverse position

Figure 14 Route-related Locking table [4]

In this table, each row of the table represents a single route. A row contains the position of all points which are needed to be locked for that specific route and all conflicting routes. Conflict between routes which require points to be locked in alternate position would lock each other by the self-effected plain locking [4]. Figure 13 shows an example for the same route as in Figure 12. In this figure symbols being used are in correspondence to the tradition German locking tables style.

2.1.9.2. Geographical Interlocking

In a geographical system of interlocking, elements related to track such as points, signals, entrance / exit points, track, section and derails etc. are represented in a logical object oriented fashion which are connected to each other in form of the track layout. When a route is set up, the start and exit

elements are marked to start search current from route entrance. This search currents runs in all diverging tracks at facing points and result in a tree structure of search currents. When an exit element is found by search current, a response current is dispatched to the entrance element. Whereas at facing points, other search current branches are deprecated.

2.2.Motivation

2.2.1. Problems and challenges in current design process

As we know in general, manual input takes a lot of dedicated attention and accountable accuracy which, often due to human processing capacity, are limited. Whereas interlocking domain is much more complex than day to day book keeping. One silent error in data can cause catastrophic results, if no other control systems are deployed.

During our meeting with Martin Ruff, one of the railway engineers from Norconsult, he mentioned that since operating a station with several routes and interlocking routines is a complex task, railway staff device these interlocking configurations on tabular forms for easy management. While these tabular forms are one way of keeping track of interlocking, it's not the efficient practice. Usually these forms extend to several sheets and referencing and management of points can be disrupted. Data redundancy and error factors require a considerable amount of time and resources. He said that it takes almost 5-6 working hours for an engineer to sort one interlocking form. While it takes this much, still this work is error-prone usually. Typing error, mismatched references, and inclusion of conflicting routes are common problems in manual work, which further lead to more work force and time.

Additionally, management of interlocking specifications is a delicate time consuming task. Printing, keep and editing of printed tables is not an efficient approach. By just switching this operation from manual to computer aided version, a significant amount of work time can be saved, hence resulting in less expense and low noise.

RailCOMPLETE is an AutoCAD plugin software, which helps engineers in unifying their efforts in transferable data format for designing, visualizing and maintenance of track and signaling component layout. All these operations are being performed by following RailML protocol and set of principles. Right now, RailCOMPLETE lacks the functionality to automate the editing of interlocking aspect of complete route and users are using same traditional format of interlocking tables. This basically leads them to the very same issue of manual process hazards.

CHAPTER 3

PLANNING & ANALYSIS

3. Planning & Analysis

In this chapter, we will briefly present RailCOMPLETE Framework, for which we are developing Interlocking module. We will describe the overview of Interlocking Module, its requirements gathering and limitation of documentation, following afterwards. Furthermore, the following sections will be used to present the actor goal list, use-case description, class diagrams and domain model using RUP framework.

In the following section, we will discuss requirements for this research project and the architecture of our solution.

3.1.Software Development Methodology

The process of splitting software development work into different and distinct stages is known as Software development methodology. A Major motivation behind these methodologies is to manage the software development lifecycle more efficiently, per requirement and within defined constraints.

For this project, I will be using *Rational Unified Process (RUP)* for capturing requirements and implementation of software and used agile rapid prototyping as a method for software requirement rectification in contrast with Scrum methodology being used in RailCOMPLETE development team

In following section, we will briefly describe this framework based on the white paper by Rational Software [13] and our experience with RUP in this project.

3.1.1. IBM Rational Unified Process (RUP)

IBM Rational Unified Process was proposed, developed and implemented in IBM division named Rational. RUP is simply a disciplined approach to achieve software development related tasks strategically which managing the resources based on pre-implementation research on scope, risks, and business analysis [14]. RUP ensures that all team members have access to the same knowledge base, domain and related aspects of software development lifecycle for a project. This is modeled using Unified Modeling Language (UML) instead of textual version. This helps users to quickly assess, qualify and read the document. Additionally, a properly modeled software artifact in UML, could be used to generate automated code and more research is underway to make it more automated.

According to Rational softwares,

"The Rational Unified Process® is a Software Engineering Process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget." [13]

RUP defines the process of software development into four distinct phases. Each phase involves business modeling, analysis & design, implementation, testing and deployment. Following are the four phases of RUP

3.1.1.1. Inception

During this phase, the idea for the project is stated. System business case are established and project scope is defined for both parties (development team and client). The development team determines if the project is worth pursuing and what resources will be needed. While Client gets the estimation of project worth and time. Inception phase produces following complete or partially complete documents/ artifacts

- Vision document: This document scope of project, general requirements, key features and system constraints.
- Initial Use-case model (partial)
- Business use case modeling
- Project glossary, risk assessment, project plan and business model if necessary (complete or partial)
- Prototypes if required

3.1.1.2. Elaboration

In this phase, problem domain is explored and a foundation of architecture is established. This helps in finalizing the project development phase and by using the risk assessment, elimination of the risk factors. The project's architecture and required resources are further evaluated. Developers consider possible applications of the software and costs associated with the development by dividing functional and non-functional requirements, elaborating scope and functionality matrix

Following are the outcome of elaboration phase,

- Use case model (80% complete)
- Supplementary requirements are identified which are not associated with any specific use case
- Architecture description
- Architectural Prototype
- Revision for business use cases and risks
- Overall project development plan
- User manual / Appendix (optional)

3.1.1.3. Construction

During the construction phase, project is developed and completed. The software is designed, written, and tested according to the defined requirements. All components are stitched together and integrated into the product and are thoroughly tested. This phase consists of following outcomes,

- Integrated software project
- User Manuals
- Release version and description

3.1.1.4. Transition

The software is released to the user. Final adjustments or updates are made based on feedback from end users. [15]. Transition phase is based on Construction phase evaluation. If a project is found mature enough and resource expectations are under control, project undergoes this transition phase.

Transition phases is an ongoing phase where system is tested, new requirements or maintenance tasks are recorded, user trainings are conducted and product roll-out are registered.

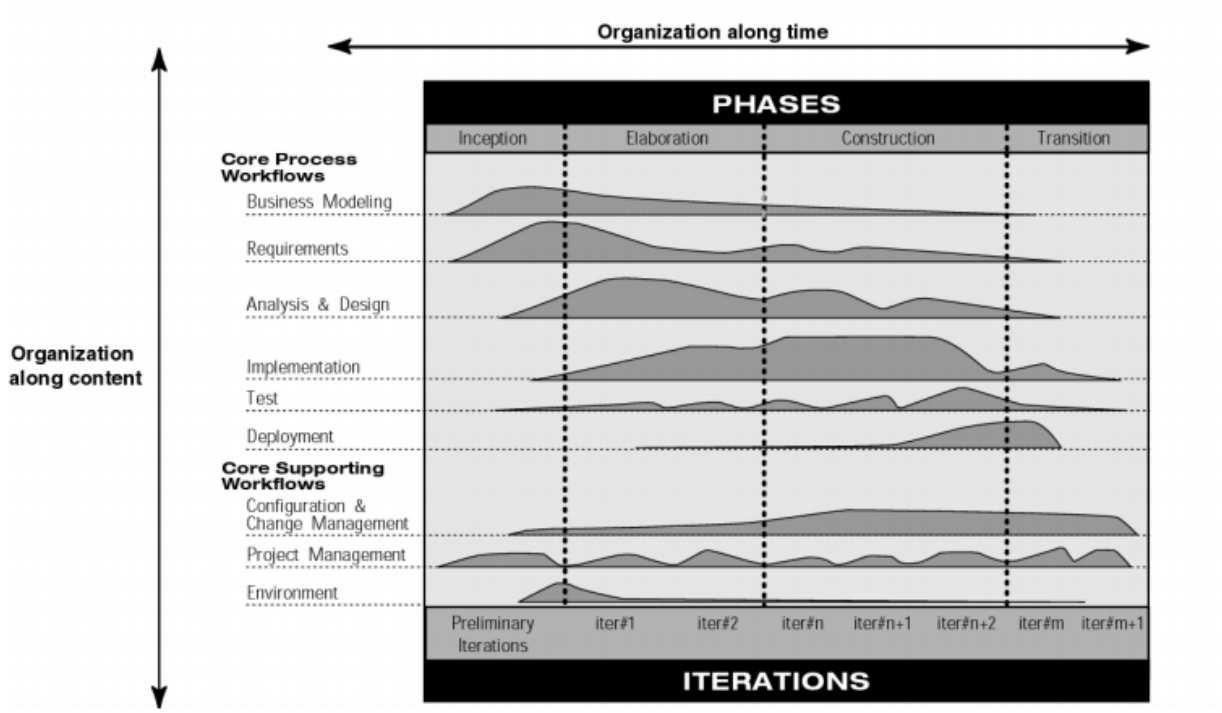


Figure 15 IBM RUP Lifecycle [14]

3.2. Conclusions from using RUP

Unlike RailCOMPLETE where SCRUM is being used for software development, I preferred RUP for this project due the fact that RUP prescribe predefined templates for capturing user requirements, and highly engaging user feedback while focusing on a simple approach of modeling instead of textual description of project during different phases.

RUP is a subset of best software development practices in industry based on their usage, user community and success ratio. [13] With the help of widely available RUP templates, I maintained my meetings with client (RailCOMPLETE) easily which later helped me to convert them into user cases with the fine-grain detail of steps and pre/post conditions of the use case. During this process, I learned that if a project is fully modeled for RUP in UML, we can generate code from those models using tools like IBM® Rational® Software Architect [16] and hence it can save tremendous time for large projects.

However, I do feel that using a full-fledged RUP approach for small teams and projects might become time consuming and difficult to maintain because it requires a thorough procedural execution in every

stage of an iteration. Due to this fact, instead of producing all artifacts of RUP, I preferred to include only those (chapter 3), which can help me capture software requirements for this project since it was not a complex team project and with a single resource. While this approach does come with limitation of time, I experienced that rational process can produce a comprehensive documentation for project owners and user manuals for client. It has a clear way for eliciting architecture of whole project so project maintenance and extensionality are not limited especially in academic projects like this, it also provides necessary foundation for further enhancements in the project in an easy to understand form factor.

3.3.Software Requirements

Since this project is addressing a real-world problem and is to be integrated with an actual product, requirements have been specified during several project meetings involving my supervisors and with development team and our key contact Claus Feyling for RailCOMPLETE in Norconsult AS. In this case, Claus Feyling acted on client role to specify the desired functionality in Interlocking module, while IT team in Norconsult and my supervisor Bjørnar Luteberget provided the technical understanding of RailCOMPLETE Software. Clause and Bjørnar also participated as an active Interlocking user to highlight the functionalities end user want to see in final version. Whereas, in these meetings, I took the role of software requirement engineer to capture the user stories and document them using IBM RUP framework. Since we followed the agile methodology for this project, it involved rapid prototyping based on user feedback and suggestions on functionality, which consecutively I implemented as software engineer.

In the following sub sections, we use RUP form of use cases to specify user intended functionality in as close to client word as we can, without including redundant information.

3.3.1. Stockholders

Following are the stakeholders for Interlocking module of the RailCOMPLETE

- Shahzad Ali Khan (Me) As Student and developer of this module
- RailCOMPLETE Developers
- RailCOMPLETE Testers
- RailCOMPLETE Management
- University of Oslo

3.3.2. Primary Actors

There are two primary actors and one secondary actors of Interlocking design tool, in RailCOMPLETE domain. These actors and their required operations are listed below

3.3.2.1. Interlocking Engineer

Interlocking engineers want to design, formulate, integrate and rectify interlocking segments within a train track design model. They are the one sole responsible for overlooking of interlocking on route. Interlocking Engineers are not necessarily one role. Usually Track Engineers can perform the said task under this role, but since our aim is to produce as loosely coupled software as constraints allows, we will keep the Interlocking Engineer as a user who is solely responsible for Interlocking schema of a route only. Interlocking Engineer is not necessarily deemed to have information related to main RailCOMPLETE document.

3.3.2.2. Track Engineer

Railway track engineers are responsible for overall modeling of track design, and they want to make sure correct interlocking mechanism configurations are applied within a design model. They can access interlocking schema loaded within a RailCOMPLETE document and based on their needed, can use or ask for changes in said interlocking schema of document. Track Engineer can perform Interlocking as well as other RailCOMPLETE functionalities.

3.3.3. Secondary Actors

3.3.3.1. Locomotive Drivers

Locomotive drivers are the ground force, who is responsible for executing railway operations and drive rail based on route from RailCOMPLETE and associated Interlocking. They are not required as an active user of the system, but will be influenced by product of RailCOMPLETE route project.

3.4.Actor Goal List

Actor	Goal
Track Engineer	• Design interlocking • Visualize Interlocking • Synthesis of Interlocking Tables
Interlocking Engineer	• Design Interlocking • Visualize Interlocking • Synthesis of Interlocking Tables
Locomotive Drivers	• Document Reports (No)

3.5.Use Case Modeling

Following is the use case diagram of the system. We can see that Track Engineer scope is not limited to only Interlocking module only and (s)he can interact with RailCOMPLETE object outside interlocking module, while *Interlocking Engineer* is only concerned with Interlocking Module. Functionalities concerned with both actors in our module, are same, but to limit the scope of our module both users are listed separately.

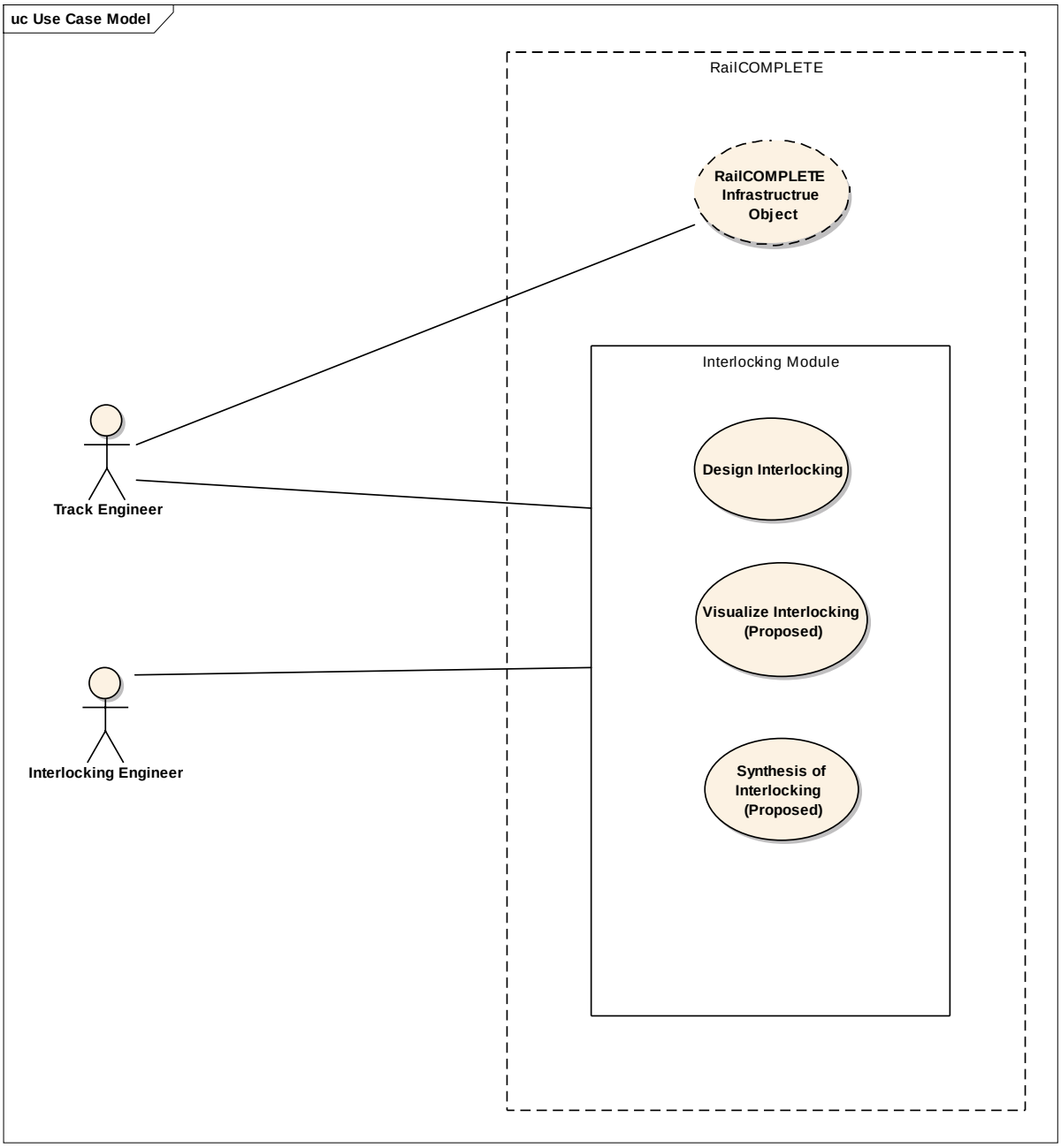


Figure 16 Use case Model

3.6.Brief Use Cases

3.6.1. Design Interlocking

Interlocking Engineer logs into RailCOMPLETE and open the new/required diagram. User opens Interlocking module and enter interlocking related data into XML based structure or User views the interlocking data and adjusts the schema per requirements. Module process newly entered data per standard and regulations specified and inform user regarding warning and constraints. User edit file accordingly and save file.

3.6.2. Visualize Interlocking

User logs into RailCOMPLETE and open the new/required diagram. User opens Interlocking module or create new Interlockings. User loads the interlocking elements from track diagram and system presents elements in interlocking window. User make changes and select to visualize. System visualize the track based on interlockings and predefined rules. User makes modification if need and save file.

3.6.3. Synthesis of Interlocking tables

User logs into RailCOMPLETE and open the new/required diagram. User opens Interlocking module or create new Interlockings. User make modifications in interlocking and system process the modifications based on track layout diagram or predefined rules. System warns user on violations and display messages accordingly. User makes modification and save file

3.7.Fully Dressed Use Cases

3.7.1. Design Interlocking (for already created track layout)

<table>
<tr><td rowspan="3">Design Interlocking</td><td>Version</td><td>1.0</td></tr>
<tr><td>Date:</td><td>18/06/2016</td></tr>
<tr><td>Scenario 1:</td><td>Interlocking in already created track layout</td></tr>
<tr><td>Use Case Section</td><td colspan="2">Comment</td></tr>
<tr><td>Scope</td><td colspan="2">RailCOMPLETE Interlocking Module
Business use case</td></tr>
<tr><td>Level</td><td colspan="2">User Goal</td></tr>
<tr><td>Primary Actor</td><td colspan="2">Interlocking Engineer</td></tr>
<tr><td>Stakeholders and Interests</td><td colspan="2">• Interlocking Engineer: wants to create interlocking section for already created Track design
• Track Engineer: requires interlocking section for his/her designed Track</td></tr>
<tr><td>Preconditions</td><td colspan="2">• User is logged into RailCOMPLETE
• Track design is already created per standards predefined</td></tr>
<tr><td>Success Guarantee</td><td colspan="2">User successfully creates interlocking sections for integration into main track design</td></tr>
<tr><td rowspan="4">Main Success Scenario</td><td>Primary Actor</td><td>System Response</td></tr>
<tr><td>1. User request search dialogue box</td><td>2. System presents search dialogue box</td></tr>
<tr><td>3. User selects Track design</td><td>4. System presents interlocking editor for track design</td></tr>
<tr><td>5. User edit interlocking</td><td>6. System checks the input per rules defined</td></tr>
</table>

	7. User save the file	8. System checks rules and updates file status
Extensions/Alternate Flow	**6e/8e-1:**	**Rule violations**
	Actor Action	**System Response**
		System inform user about rule violation
	8e-2:	**Fail to save file**
	Actor Action	**System Response**
		1. System inform user about error reason.
	2. User fix the error and save file	3. System checks rules and updates file status
Special Requirements	TBD	
Technology and data variation	N/A	
Frequency	Moderate	
Miscellaneous	N/A	
Data/ Field requirements	TBD	
Prototype	TBD	

3.7.2. Design Interlocking (for an empty track layout)

Design Interlocking	Version	1.0
	Date:	**18/06/2016**
	Scenario 2:	**Interlocking in an empty Track layout**
Use Case Section	**Comment**	
Scope	Interlocking Module Business use case	
Level	User Goal	
Primary Actor	Interlocking Engineer	
Stakeholders and Interests	• **Interlocking Engineer:** wants to create interlocking section for already created Track design • **Track Engineer:** requires interlocking section for his/her designed Track	
Preconditions	RailCOMPLETE document is created	
Success Guarantee	Interlocking schema is successfully loaded into empty track	
Main Success Scenario	**Primary Actor**	**System Response**
	1. User click new RailCOMPLETE document button	2. System presents new document dialogue options
	3. User configure options and click create	4. System presents an empty RailCOMPLETE document
	5. User enters Interlocking module command in command window	6. System loads Interlocking Module
	7. User selects options to create new interlocking file	8. System presents options for creating new interlocking file

	9. User enters interlocking data and clicks save schema	10. System checks rules and updates file status
Extensions/Alternate Flow	**Step # 10e**	**Rule Violation**
	Actor Action	**System Response**
		1. System inform user about error reason.
	2. User fix the error and save file	3. System checks rules and updates file status
Special Requirements	TBD	
Technology and Data Variation	N/A	
Frequency	Moderate	
Miscellaneous	N/A	
Data/ Field requirements	TBD	
Prototype	TBD	

3.7.3. Visualize Interlocking

Interlocking Integration	Version	1.0
	Date:	**18/06/2016**
	Scenario 1:	**Visualizing interlocking file**
Use Case Section	**Comment**	
Scope	RailCOMPLETE Interlocking Module Business use case	
Level	User Goal	
Primary Actor	Interlocking Engineer, Track Engineer	
Stakeholders and Interests	• **Interlocking Engineer:** wants to visualize the interlockings. • **Track Engineer:** wants to validate and visualize the interlockings in track diagram.	
Preconditions	• User is logged into RailCOMPLETE • Track design and Interlocking xml is already created as per standards predefined • Main Track design is open	
Success Guarantee	User successfully integrates interlocking sections into main track design	
Main Success Scenario	**Primary Actor**	**System Response**
	1. User click new RailCOMPLETE document button	2. System presents new document dialogue options
	3. User configure options and click load	4. System loads the RailCOMPLETE document
	5. User enters Interlocking module command in command window	6. System loads Interlocking Module
	7. User selects options to load interlocking file	8. System presents the interlocking file options
	9. User loads elements for interlocking file from track	10. System load the elements from RailCOMPLETE object

	11. User selects Visualization option	12. System process the command and visualize the interlocking
	13. User make modifications as required and selects save file	14. System save interlockings in track diagram
Extensions/Alternate Flow	**6e/8e/12e/14e-1:**	**Rule violations**
	Actor Action	**System Response**
		System inform user about rule violation
	12e-2:	**Fail to save file**
	Actor Action	**System Response**
		1. System inform user about error reason.
	2. User fix the error and save file	3. System checks rules and updates file status
Special Requirements	TBD	
Technology and Data Variation List	N/A	
Frequency	Moderate	
Miscellaneous	Extension Project	
Data/ Field requirements	TBD	
Prototype	NA	

3.7.4. Synthesis of Interlocking

Interlocking Integration	Version	1.0
	Date:	**18/06/2016**
	Scenario 1:	**Synthesis of interlocking file**
Use Case Section	**Comment**	
Scope	RailCOMPLETE Interlocking Module Business use case	
Level	User Goal	
Primary Actor	Interlocking Engineer, Track Engineer	
Stakeholders and Interests	• **Interlocking Engineer:** wants automated synthesis of interlocking file • **Track Engineer:** wants to automate the safety of track by synthesizing interlocking information	
Preconditions	• User is logged into RailCOMPLETE • Track design and Interlocking xml is already created as per standards predefined • Main Track design is open	
Success Guarantee	User successfully integrates interlocking sections into main track design	
Main Success Scenario	**Primary Actor**	**System Response**

	1. User click new RailCOMPLETE document button	2. System presents new document dialogue options
	3. User configure options and click load	4. System loads RailCOMPLETE document
	5. User enters Interlocking module command in command window	6. System loads Interlocking Module
	7. User selects options to load interlocking file	8. System loads the interlockings
	9. User modify the interlocking file	10. System process the file
	11. User selects synthesis option	12. System process the file and present the analysis
	13. User make modifications as required and selects save file	14. System save interlockings in track diagram
Extensions/Alternate Flow	**6e/8e/12e/14e-1:**	**Rule violations**
	Actor Action	**System Response**
		System inform user about rule violation
	14e-2:	**Fail to save file**
	Actor Action	**System Response**
		4. System inform user about error reason.
	5. User fix the error and save file	6. System checks rules and updates file status
Special Requirements	TBD	
Technology and Data Variation List	N/A	
Frequency	Moderate	
Miscellaneous	Extension Project	
Data/ Field requirements	TBD	
Prototype	NA	

3.8.Domain Model

Following is the conceptual domain model of the interlocking schema and how it is used in Interlocking Editor. This domain model is derived from interlocking specification presented by Bon Janssen in his research paper on efficient formalization of railway interlocking data in RailML [3]

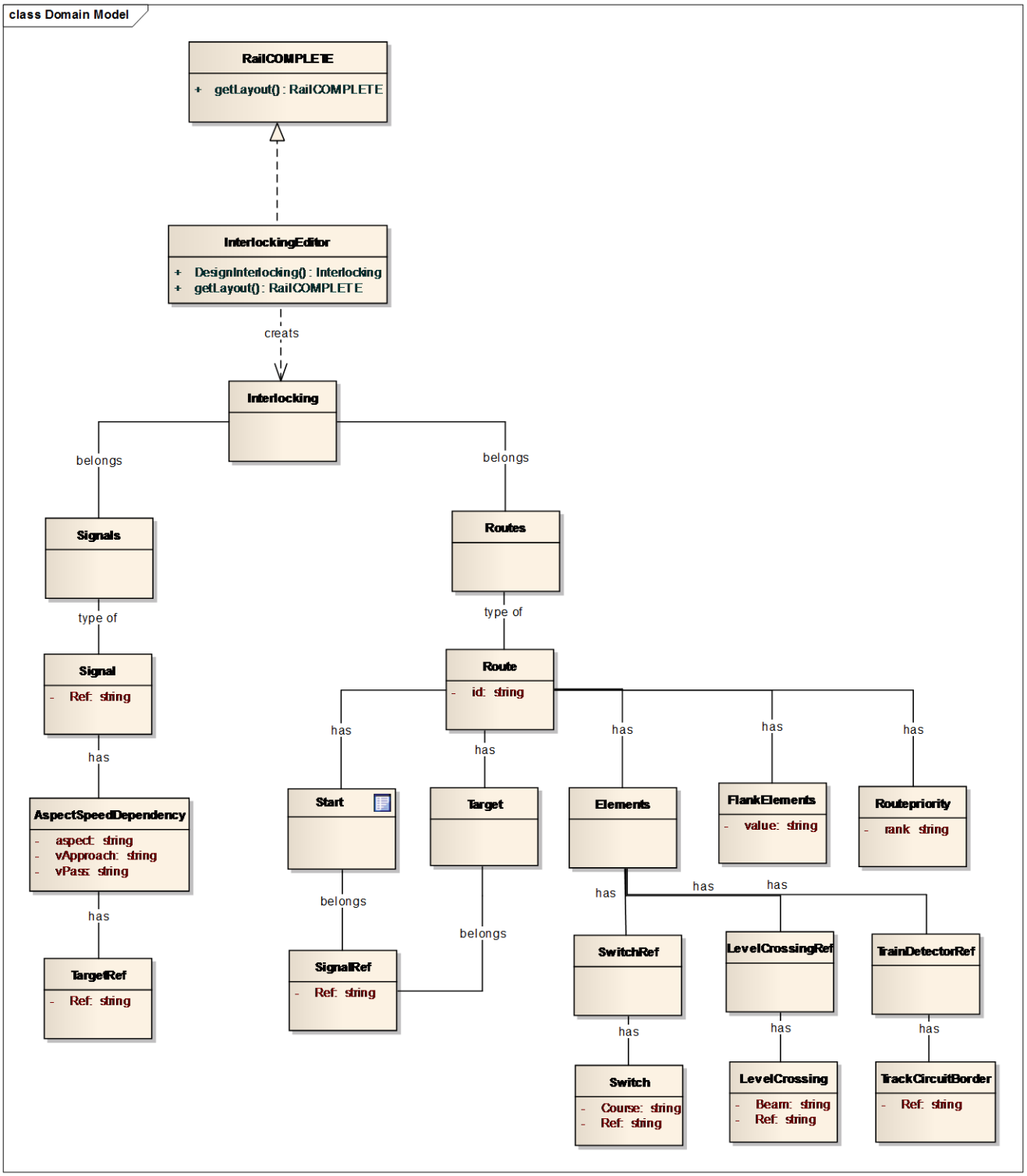


Figure 17 Domain Model

3.9.Component Diagram

In following diagram, we present different components of Interlocking editor as a plugin. Interlockings processing and interface interaction is carried out in InterlockingEditor class while worker class is responsible for RailCOMPLETE Integration.

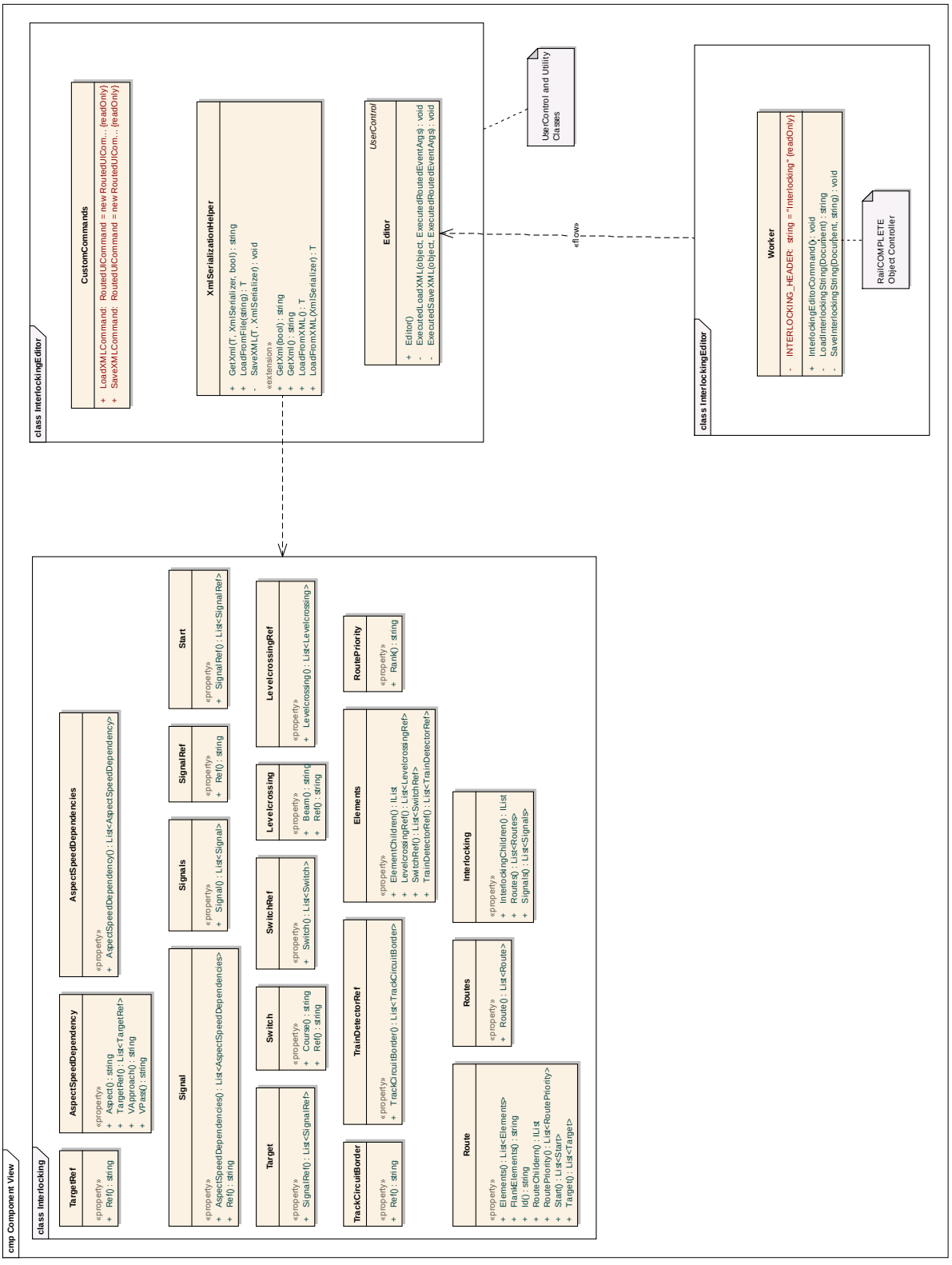


Figure 18 Component Diagram

3.10. Class Diagram (Interlocking)

Class diagram describes the structure of the system by presenting its classes and their relationships.

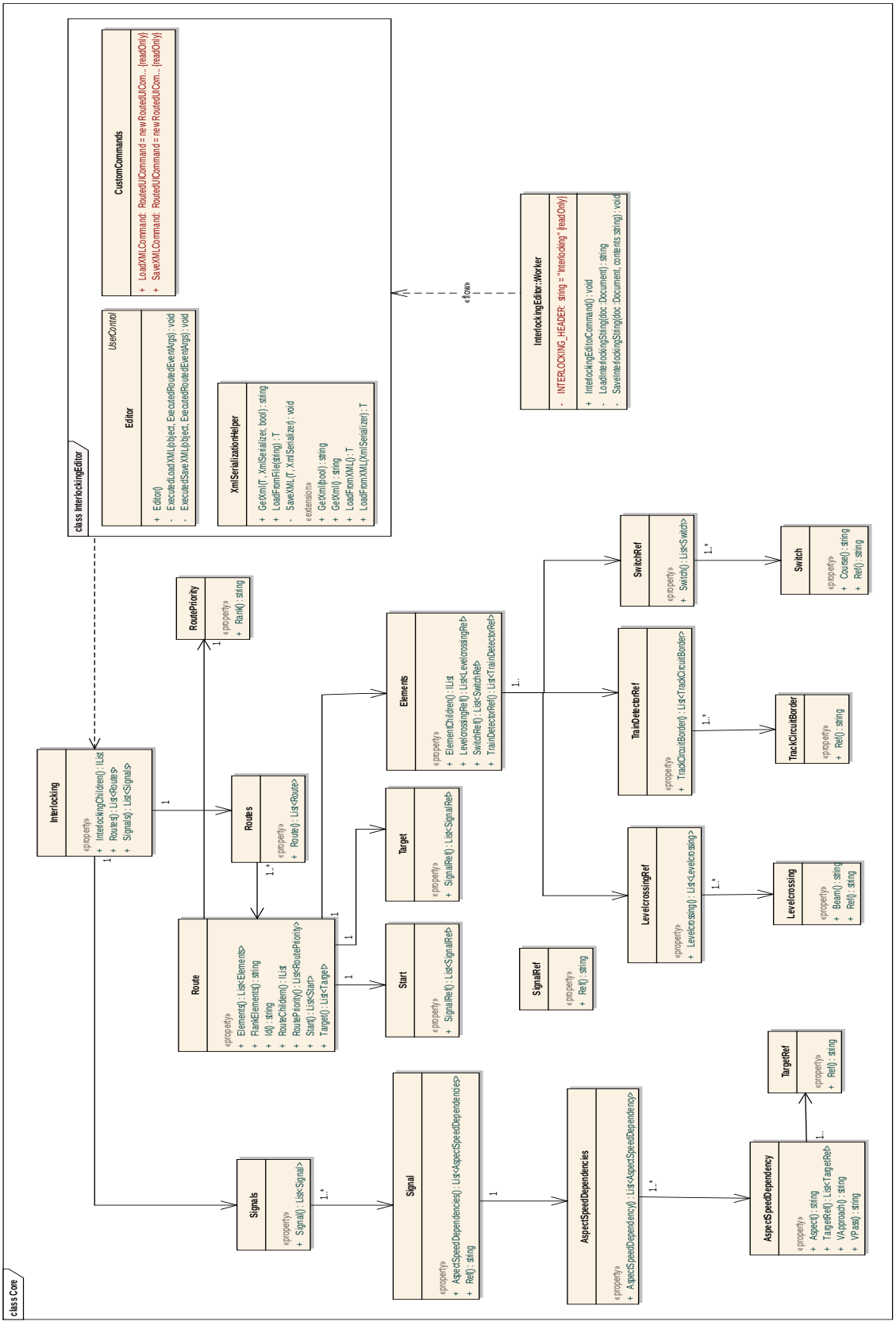


Figure 19 Class Diagram

CHAPTER 4

IMPLEMENTATION
&
SOFTWARE DEVELOPMENT METHODOLOGY

4. Implementation

4.1. Overview of RailCOMPLETE framework

RailCOMPLETE is developed in Microsoft Visual Studio, using .Net Framework and C# to utilize AutoCAD.net API so that we can provide Railway operations functionality right into the AutoCAD software. Project consist of 18 Core modules. Test drive approach is being followed in throughout the project, and for Window Presentation Framework (WPF), Model-View-View Model (MVVM)

There is one object Manager module responsible for managing RailCOMPLETE Object in AutoCAD software. Core module provide the base projects for different resources and functionalities for the whole plugin including Containers, Context Menus, Database rules, Database Reactors, Area Elevation, Export functionalities, Object definition and management, Document commands, plugin management, Railway Object and document, Transactions, Schematic views, styles, graphics and utilities. Rest of the modules are sole responsible for graphical user interface and functionalities for base defined in Core module.

4.2. Developing the Interlocking Module

Interlocking Module is part of the framework which lets the user dynamically create/load, edit and save the interlocking schema of an active RailCOMPLETE document, based on Interlocking schema devised by Bob Janssen [3] for including interlocking specifications into the RailML format. This module will further act as a base for advance features for Interlocking management, which will be discussed in future work section of Chapter 6. The Interlocking module is implemented using Microsoft .Net framework over AutoCAD.net API. User Interface is written in XAML using Windows Presentation Framework (WPF) and for code behind C-Sharp(C#) is used.

Interlocking Module is embedded in RailCOMPLETE solution as standalone plugin, which can be used on demand through runtime commands and can be loaded along other core modules in AutoCAD. It is using RCTransaction and Core Module from RailCOMPLETE to get Loaded into AutoCAD for saving and retrieving Interlocking file from an active RailCOMPLETE document. In following sections of this chapter, software development of Interlocking module prototype is discussed.

4.3. Interlocking Elements and XMLSerialziation

The infrastructure layout is modeled using a CAD program, and can be exchanged through the RailML format. An extension of RailML for describing tabular interlocking specifications is described in the paper Efficient formalization of railway interlocking data in RailML. An (incomplete) example of the interlocking format is shown in Figure 17.

Interlocking Module is based on Serialization of Interlocking schema of the tabular data used in Railway, based on Bon Janssen work [6]. To utilize his schema, we used XMLSerialziation classes from .Net framework. We converted XML file to XSD schema using Microsoft XML Schema Definition (XSD.exe) tool [17]. Later, using the same tool, XSD schema is used to generate .Net Classes as shown in following figure.

```xml
<?xml version="1.0"?>
<interlocking xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <signals>
    <signal ref="11005">
      <aspectSpeedDependencies>
        <aspectSpeedDependency aspect="Hs20" vApproach="40" vPass="0" />
        <aspectSpeedDependency aspect="Hs21" vApproach="-" vPass="60">
          <targetRef ref="11055" />
        </aspectSpeedDependency>
        <aspectSpeedDependency aspect="Hs22" vApproach="-" vPass="270">
          <targetRef ref="11015" />
        </aspectSpeedDependency>
      </aspectSpeedDependencies>
    </signal>
    <signal ref="11055">
      <aspectSpeedDependencies>
        <aspectSpeedDependency aspect="Hs20" vApproach="40" vPass="0" />
        <aspectSpeedDependency aspect="Hs21" vApproach="-" vPass="60">
          <targetRef ref="11075" />
        </aspectSpeedDependency>
        <aspectSpeedDependency aspect="-" vApproach="-" vPass="-" />
      </aspectSpeedDependencies>
    </signal>
  </signals>
  <routes>
    <route id="1">
      <start>
        <signalRef ref="pro.Routes.Route.Start.SignalRef.Ref" />
      </start>
      <target>
        <signalRef ref="Target.SingalRef" />
      </target>
      <elements>
        <switchRef>
          <switch ref="Ref" course="Left" />
        </switchRef>
        <levelcrossingRef>
          <levelcrossing ref="Ref" beam="Beam" />
          <levelcrossing ref="Ref" beam="Beam" />
          <levelcrossing ref="Ref" beam="Beam" />
        </levelcrossingRef>
        <trainDetectorRef>
          <trackCircuitBorder ref="Ref" />
          <trackCircuitBorder ref="Ref" />
          <trackCircuitBorder ref="Ref" />
        </trainDetectorRef>
      </elements>
      <flankElements>Flank</flankElements>
      <routePriority rank="1" />
    </route>
  </routes>
</interlocking>
```

Figure 20 Interlocking XML Schema

This Interlocking Schema object contains reference for all object types we need for this schema. We take/generate this object based on user input and provide an interactive editor in RailCOMPLETE for Interlocking.

4.4.Prototypes

In following section, we will discuss the different prototypes we implemented during this research period

4.4.1. Version 1.0

Prior to first prototype, we researched for possible representations of Interlocking data and compared it with original interlocking tables which are usually used in railway domain. Usually with some nominal changes, most elements are common in different railway system. railML framework provides functionality for interoperability between railway centric applications, so specifications from Bob Janssen [3] help us conceiving a standardized format for Interlocking schema.

With thorough discussion on the layout of those tables with my supervisors and Clause Feyling from RailCOMPLETE, we proposed to use the railML specification for interlocking. Since there is no complete standardized interlocking domain model, so each railway administration is likely to have their own additions and customizations to the railML model. We are using the elements intended within Norwegian rail since RailCOMPLETE is currently supporting only Norwegian standards of track layouts.

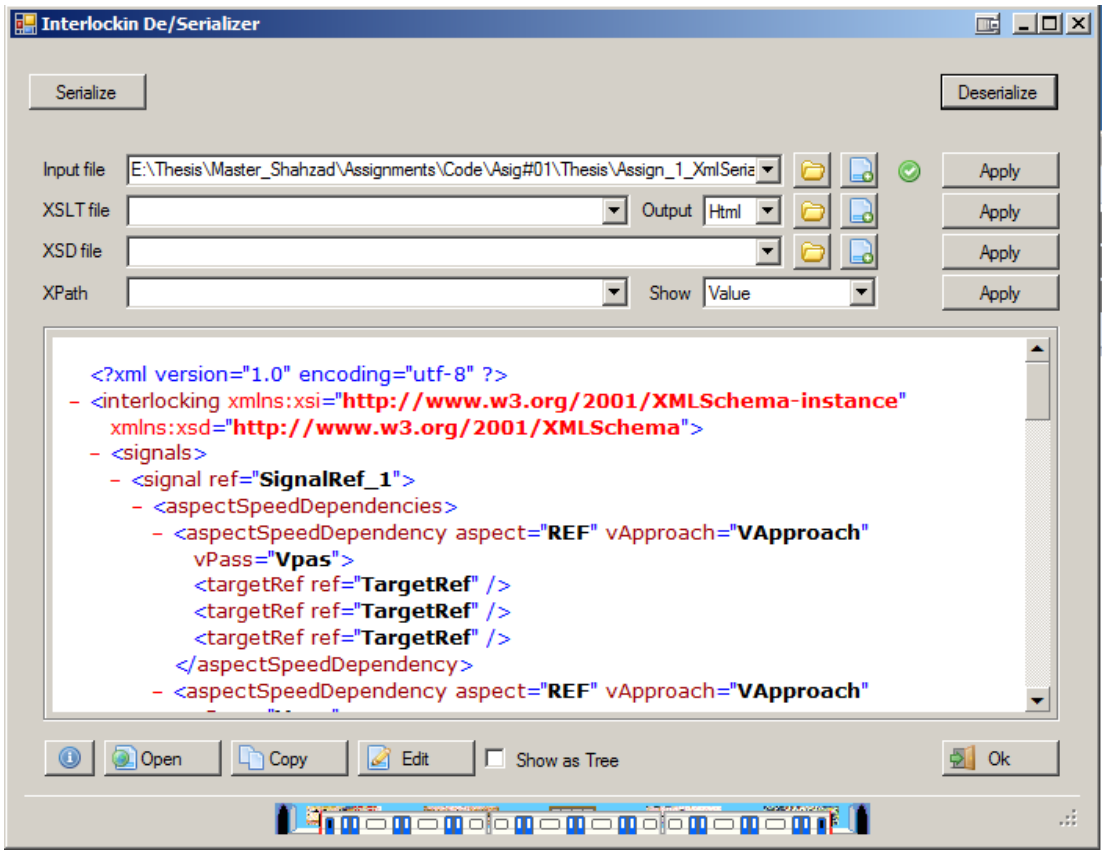


Figure 21 Prototype Version 1.0 Serializing XML Schema

In this version, we converted the XML schema proposed in railML to .net based class structure. We tested the XMLSerialziation and deserialization to access the data stored in classes from object to xml and xml to objects.

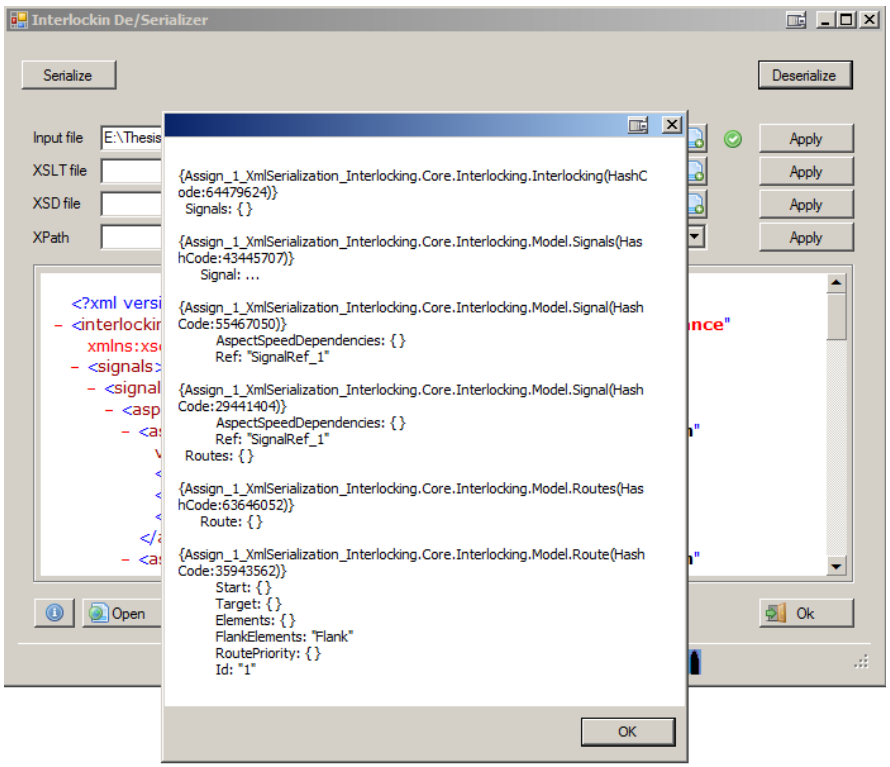


Figure 22 Prototype Version 1.0 Deserializing of schema

Following table provides overview of functionality achieved in this prototype.

Legend	Information
Version	1.0
Date	08/04/2016
User	• Developer of Module • RailCOMPLETE Developers • Bjørnar Steinnes Luteberget
Functionalities	• Defining interlocking schema in XML • writing class representation of that data in C# • Converting XML to C# object • Converting C# object to XML file • Writing and editing file on a specific location
Technical	• Microsoft .Net • C# • XML Serialization classes • WinForms

This prototype help us to understand the schematic specifications much better from implementation point of view. Our primary objective during this phase was to understand the problem complexity and

to define and agree on schema for Interlockings. This prototype is developed in .Net WinForms and .Net Serialization classes are used to create XML to/fro Object Classes.

4.4.2. Version 2.0

In version 1.0 we used XML specifications we proposed for interlocking in object-oriented classes. Schemas generated from Interlocking tables can be of varying length and nested elements, so we cannot present a static interface for different schemas. For making it editable, a dynamically populated user interface with appropriate controls is necessary. Another key aspect of this prototype is development in Windows Presentation Foundation Framework (WPF), since RailCOMPLETE is making the development heavily in WPF, to make our final module integrable and ready for future research / development, we used WPF. WPF gives us the flexibility to design interfaces completely detached from business logic or backend functionality.

In Version 2.0, we designed the UI and created binding for XML schema with WPF tree view. User can load the XML schema from an already created xml file, edit it and save it in the file. Results can be observed immediately in adjacent section of the prototype

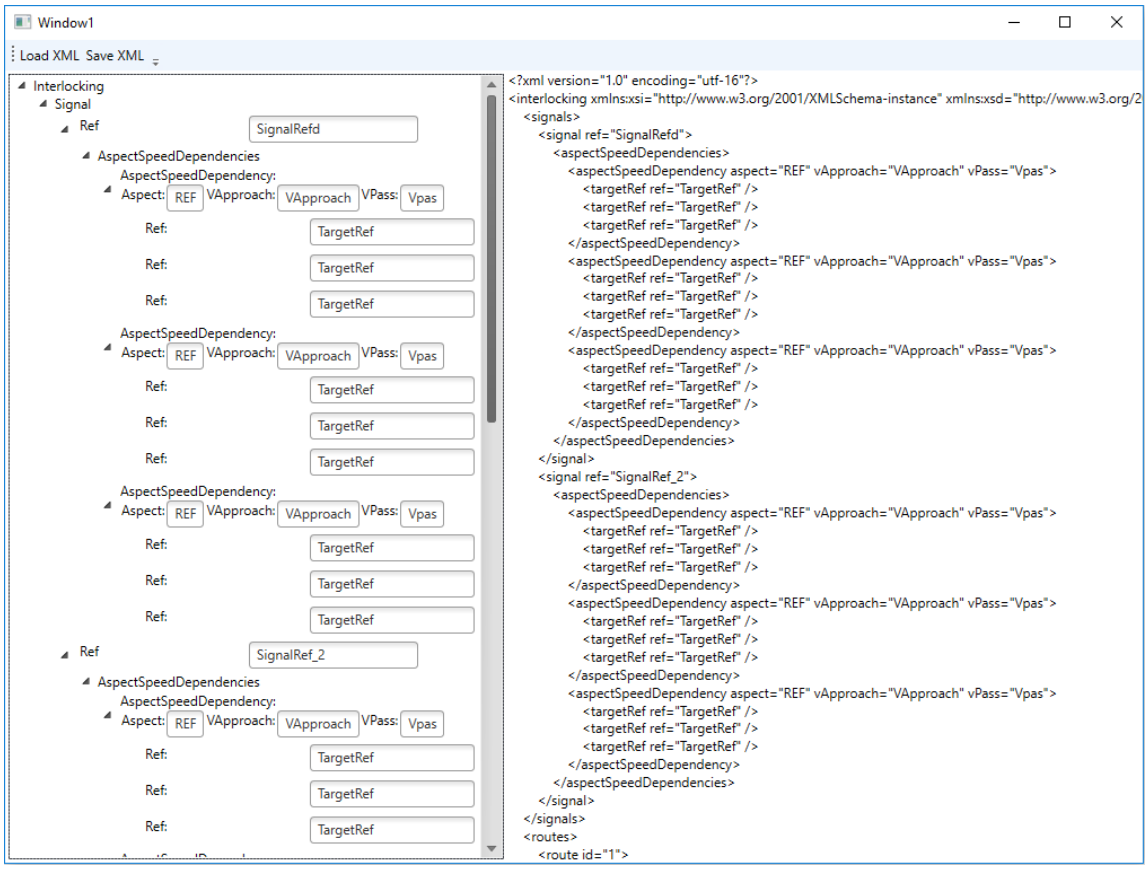


Figure 23 Prototype Version 2.0

We can solve the problem of dynamic control loading by Model-View-ViewModel (MVVM) in WPF, where model refers either to a domain model or to the data access layer representation, view constitutes for layout or user interface and view model is the abstraction or binding element for view and model.

Implemented functionalities in version Version 2.0 are presented below.

Legend	Information
Version	2.0
Date	20/05/2016
User	• Developer of Module • RailCOMPLETE Developers • Claus Feyling • Bjørnar Steinnes Luteberget • Christian Johansen
Functionalities	• Binding XML representation with Input/output controls • Loading data from XML schema • Saving input in XML Schema
Technical	• Microsoft .Net • C# • XML Serialization classes • Windows Representation Foundation (WPF)

4.4.3. Version 3.0

In Version 3.0 Interface is revamped and file loading/unloading process is rewritten per our meeting with RailCOMPLETE. Aspect dependencies and References for dependencies were discussed and a new requirement for creating file from scratch was captured. This version can serve as a stand-alone tool to edit any XML based schema for Interlocking as well

Below given screenshot of Interlocking Module is with redesigned interface and input controls.

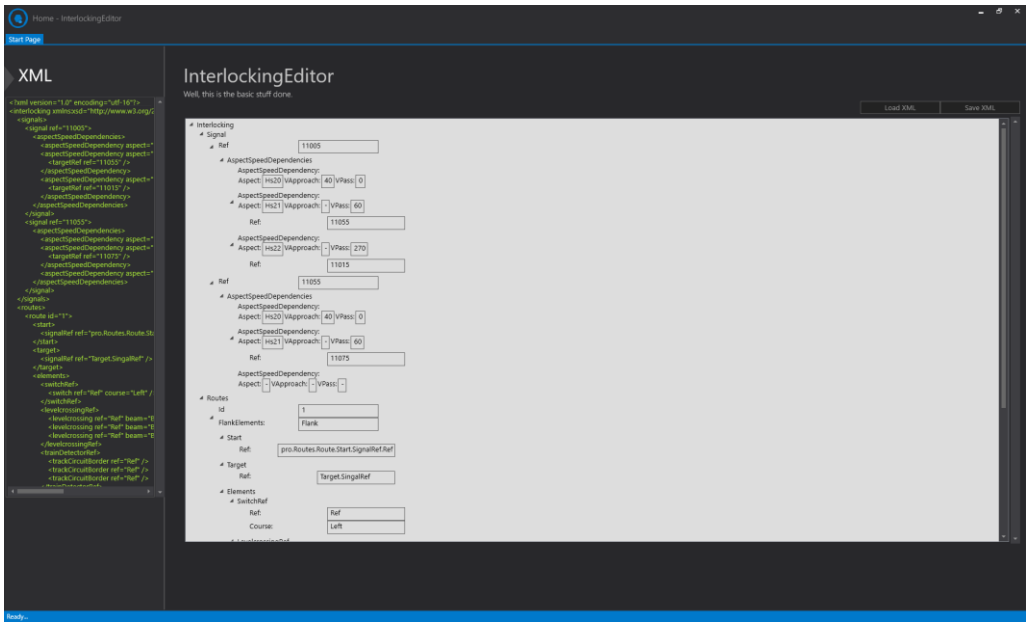


Figure 24 Prototype Version 3.0

Following tables shows information about functionalities implemented in this prototype.

Legend	Information
Version	3.0
Date	26/08/2016
User	• Developer of Module • RailCOMPLETE Developers • Claus Feyling (As Interlocking Engineer) • Bjørnar Steinnes Luteberget • Christian Johansen
Functionalities	• Redefining XML Schema • Editing the value and presenting changes in file • Saving files • Thematic changes for RailCOMPLETE design consistency
Implementation	• Microsoft .Net • C# • XML Serialization classes • Windows Representation Foundation (WPF)

4.4.4. Version 4.0

In Version 4.0 Interlocking module from Version 3.0 is integrated with RailCOMPLETE and successfully loaded and used in AutoCAD. This version can be installed along with RailCOMPLETE software and based on needs can be loaded/unloaded from active plugins directory. In order to access the functionalities, we are using the same pattern which is being established in RailCOMPLETE, as of using command bar. On submitting "SHAHZAD-INTERLOCKING-EDITOR", our module is loaded and presents schema definition and modification tool.

For this demonstration, we have produced a file within the AutoCAD plugin Directory for Interlocking schema, using the XML serializers.

Following is the overview of functionalities in Version 4.0.

Legend	Information
Version	4.0
Date	06/09/2016
User	• Developer of Module • RailCOMPLETE Developers (As developers and Layout engineers) • Claus Feyling (As Interlocking Engineer) • Bjørnar Steinnes Luteberget • Christian Johansen
Functionalities	• Integration of Interlocking module in RailCOMPLETE • Editing XML structure based on input • Loading module from command AutoCAD command prompt • Loading/Unloading Interlocking module from Module window
Implementation	• Microsoft .Net • C#

	• XML Serialization classes • Windows Representation Foundation (WPF) • AutoCAD 2016 • RailCOMPLETE • AutoCAD.Net

Following screenshots demonstrates the use case of creating interlocking for a new route document. RailCOMPLETE is providing support tools in designing a track for different rules and regulation by different rail administrations (JBV and Sporveien).

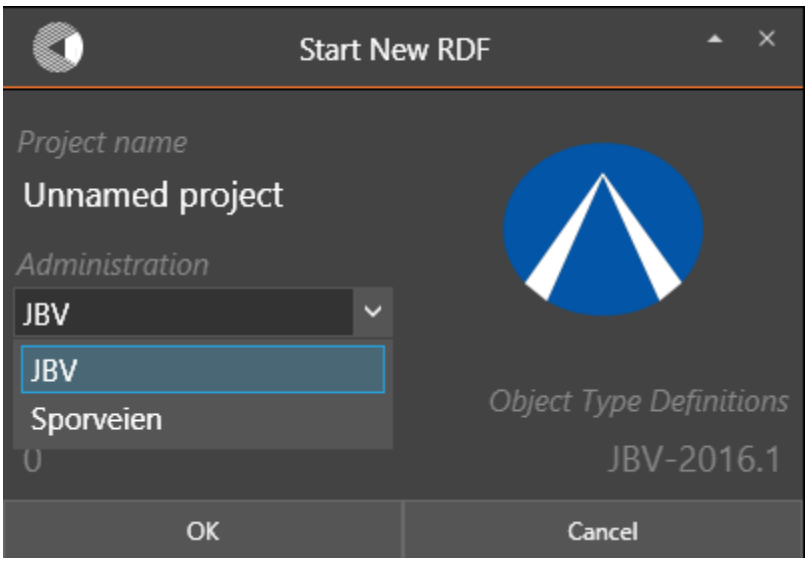


Figure 25 RailCOMPLETE new project administration options

We can click on New RailCOMPLETE document and create one, in this example according to JBV standards and press OK. This will open a New RailCOMPLETE document.

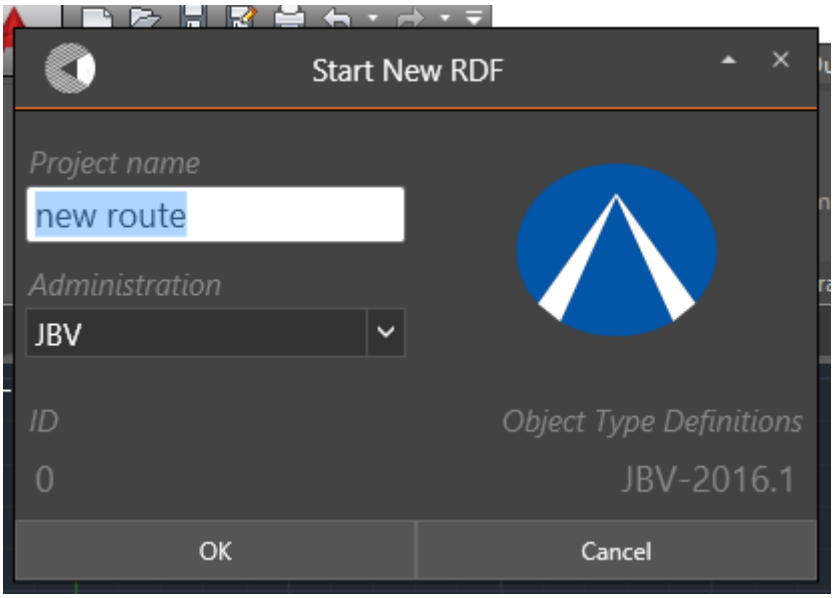


Figure 26: Creating New RailCOMPLETE Project

Interlocking module can be loaded from command line as well, as heavily used practice throughout in RailCOMPLETE. Here we are using load on user request and from command prompt parameters of the LOADCTRL defined by AutoCAD framework.

Figure 27 Using command to load Interlocking Module

When a user try to provoke Interlocking module, we make sure that there is an active RailCOMPLETE document opened/created for which we need to make interlocking schema. Module can only be loaded in an active RailCOMPLETE document or otherwise it will display error that this is not a RailCOMPLETE document as shown in following figure.

Figure 28 Loading Interlocking Module without a RailCOMPLETE document.

Interlocking can be loaded/unloaded based on preferences and can act as a hot pluggable Module for RailCOMPLETE. Program may ask for permission to load executable binary, which we can confirm to load. This specific dialogue box may appear if binaries are modified and not loaded from AutoCAD after modifications.

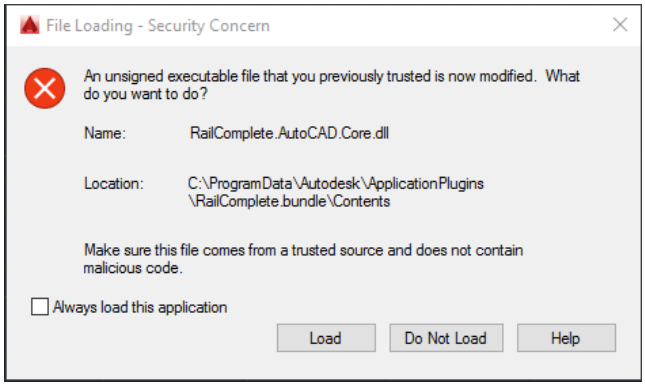


Figure 29 Loading Unsigned modules

Following screenshot displays all the available module for RailCOMPLETE. Interlocking Module uses the same module structure which is being used in RAILCOMPLETE. Users are already familiar with this approach so loading/unloading our module will be effortless for users.

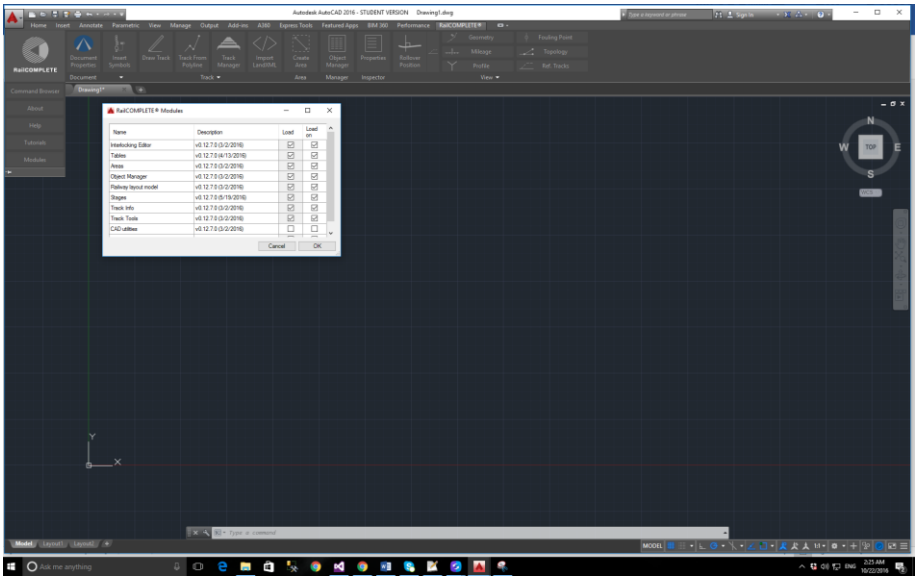


Figure 30 Using Module management to load Interlocking pluging

One of the worth noting point here is to not require from user to load modules manually each time they run AutoCAD, so our module must be ready to use and should be in a seamless position. For manually loading .Net assembly, a user needs to use NETLOAD command at the command prompt or within AutoLISP file. However, for auto-loading of modules, there are wide variance of ways how to enable automatic module loading mechanism like acad.lsp, acad.ads acad.rx etc but most sustainable and neat way to do this is demand-loading mechanism [18]. Demand-loading mechanism enable auto-load of Microsoft .Net framework programs/ applications.

Implementation of Demand load is easy in recent versions of AutoCAD. We must define a key specific to the application (Interlocking Module) we want to load at startup. This key must be placed under Application key for AutoCAD release we are targeting and want to load our application in. Following parameter needs to be included in application key [19].

i. **Description:** .Net assembly description (optional) [string]
ii. **LOADCTRLS:** Control parameter for loading behaviors [DWORD numeric]
    a. 1 = Load application on proxy object detection
    b. 2 = Load at startup
    c. 4 – Load at start of a command
    d. 8 – Load at request from user or application
    e. 16 – Do not load
    f. 32 – Transparently application loading
iii. **LOADER:** Path to the module [string]
iv. **MANAGED:** Specifies the file (.Net assembly or ObjectARX) to be loaded.
    [1= .Net Assembly files] [DWORD]

This information key can be stored in either under HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER registry for Microsoft platform of operating systems. HKEY_LOCAL_MACHINE key writing requires appropriate privileges, since this key is at machine level and not for current user.

We have used both methods, where application is loaded automatically as well as if module is unloaded, a user can load it through module management or from command prompt. Below is the code in C# for the management of application key. This code sample is provided from AutoCAD user manuals [19] and being used in RailCOMPLETE for module loading.

```
using Microsoft.Win32;
using System.Reflection;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("RegisterMyApp")]
public void RegisterMyApp()
{
  // Get the AutoCAD Applications key
  string sProdKey = HostApplicationServices.Current.RegistryProductRootKey;
  string sAppName = "MyApp";

  RegistryKey regAcadProdKey = Registry.CurrentUser.OpenSubKey(sProdKey);
  RegistryKey regAcadAppKey = regAcadProdKey.OpenSubKey("Applications", true);

  // Check to see if the "MyApp" key exists
  string[] subKeys = regAcadAppKey.GetSubKeyNames();
  foreach (string subKey in subKeys)
  {
      // If the application is already registered, exit
      if (subKey.Equals(sAppName))
      {
          regAcadAppKey.Close();
          return;
      }
  }

  // Get the location of this module
  string sAssemblyPath = Assembly.GetExecutingAssembly().Location;

  // Register the application
  RegistryKey regAppAddInKey = regAcadAppKey.CreateSubKey(sAppName);
  regAppAddInKey.SetValue("DESCRIPTION", sAppName, RegistryValueKind.String);
  regAppAddInKey.SetValue("LOADCTRLS", 14, RegistryValueKind.DWord);
  regAppAddInKey.SetValue("LOADER", sAssemblyPath, RegistryValueKind.String);
  regAppAddInKey.SetValue("MANAGED", 1, RegistryValueKind.DWord);

  regAcadAppKey.Close();
}

```

```
43. [CommandMethod("UnregisterMyApp")]
44. public void UnregisterMyApp()
45. {
46.   // Get the AutoCAD Applications key
47.   string sProdKey = HostApplicationServices.Current.RegistryProductRootKey;
48.   string sAppName = "MyApp";
49.
50.   RegistryKey regAcadProdKey = Registry.CurrentUser.OpenSubKey(sProdKey);
51.   RegistryKey regAcadAppKey = regAcadProdKey.OpenSubKey("Applications", true);
52.
53.   // Delete the key for the application
54.   regAcadAppKey.DeleteSubKeyTree(sAppName);
55.   regAcadAppKey.Close();
56. }
```

Figure 31 .Net application demand-load code [19]

Once Interlocking module is loaded, we can use the command prompt to display Interlocking editor interface. This interface is being used to display interlocking file elements or for creating the desired elements of interlocking for a track, view changes and save the file. We are using the interface from prototype 2.0, since AutoCAD application limits some of the advanced feature to draw UI through XAML.

Every interface is treated as user interface type, due to which resource dictionaries are hard to load without workarounds or patches. Since our goal for this project was to attain maximum functionality in limited time, we dropped the interface developed in Version 3.0 and used the normal and simple interface without any resource dictionaries or theme packages.

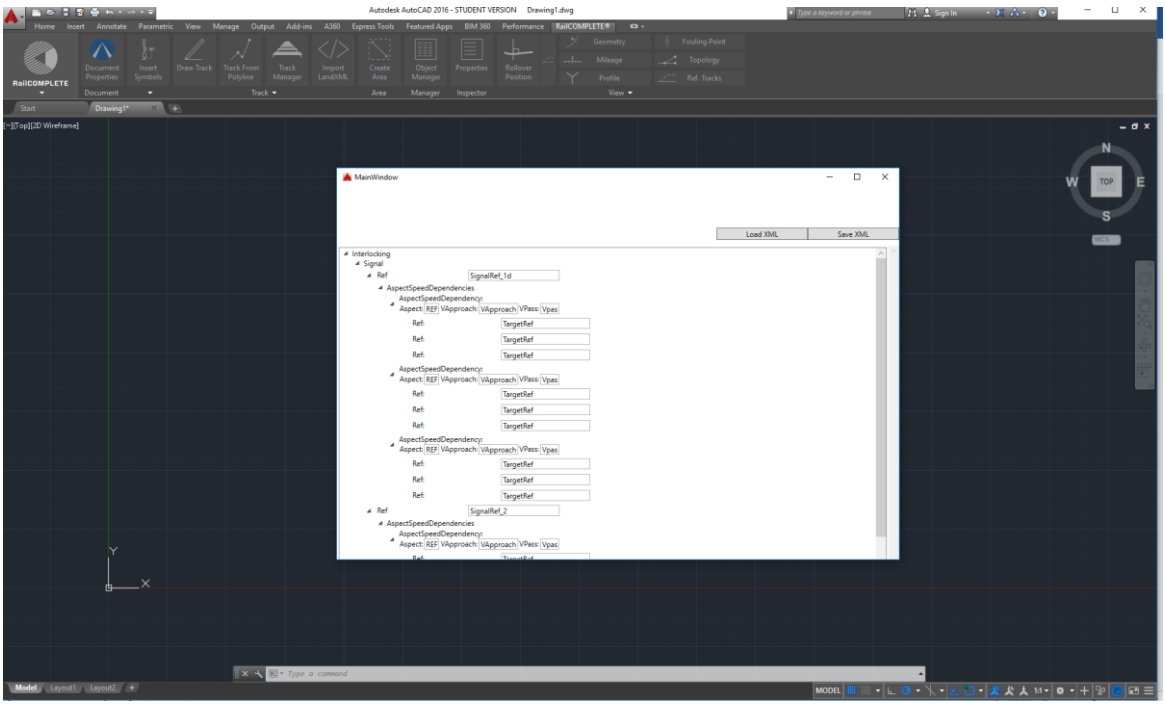


Figure 32: Editing Interlocking schema

4.5.User Feedback

In this section, we will go through user feedback we received during the iteration of project.

4.5.1. Version# 1 feedback

4.5.1.1. Supervisor's feedback

Legend	Information
Version	1.0
Place	University of Oslo
Date	08/04/2016
Users	• Developer of Module • Bjørnar Steinnes Luteberget
Level	Internal (Supervisor's feedback)
Goal	• Assessing technical challenges • Defining project technical expectancy • Assessing programming skillset
Implementation:	• Defining interlocking schema in XML • writing class representation of that data in C# • Converting XML to C# object • Converting C# object to XML file • Writing and editing file on a specific location
Remarks	• Schema needed to be independent of built-in markup and can be assessable for better textual representation in GUI • Classes are generate correctly aligned within the scope of interlocking schema

4.5.1.2. Supervisor's feedback

Legend	Information
Version	1.1
Place	University of Oslo
Date	14/04/2016
Users	• Developer of Module • Bjørnar Steinnes Luteberget
Level	Internal (Supervisor's feedback)
Goal	• Schema needed to be independent of built-in markup and can be assessable for better textual representation in GUI
Implementation:	• Classes are data bounded with WPF tree view control and string representations are handled outside of core interlocking classes, independent of making changes in XML schema
Remarks	• Goal achieved

4.5.1.3. *RailCOMPLETE Developer's feedback*

Legend	Information
Version	1.1
Place	University of Oslo
Date	15/04/2016
Users	• RailCOMPLETE Developers
Level	External (RailCOMPLETE developer's feedback)
Implementation:	• Defining interlocking schema in XML • writing class representation of that data in C# • Converting XML to C# object • Converting C# object to XML file • Writing and editing file on a specific location
Remarks	• Classes are independent and can be reused in RailCOMPLETE code as a .dll code library

4.5.2. Version#2 Feedback

4.5.2.1. *Supervisor's feedback*

Legend	Information
Version	2.0
Date	20/05/2016
User	• Developer of Module • Bjørnar Steinnes Luteberget • Christian Johansen
Functionalities	• Binding XML representation with Input/output controls • Loading data from XML schema • Saving input in XML Schema (N/A)
Remarks	• GUI controls must be scrollable • Saving input in XML schema in next iteration • Output must be more user-friendly • Get Claus Feyling feedback on Interlocking data

4.5.2.2. *Claus Feyling feedback*

Legend	Information
Version	2.0
Date	24/05/2016
User	• Claus Feyling
Functionalities	• Binding XML representation with Input/output controls • Loading data from XML schema • Saving input in XML Schema (N/A)
Remarks	• Files loaded successfully

	• Train restrictions can be more than one constants in interlocking schema. • "Local release area" elements for interlocking can be useful (Norwegian rail).

4.5.2.3. RailCOMPLETE Developers feedback

Legend	Information
Version	2.0
Date	24/05/2016
User	• RailCOMPLETE Developers
Functionalities	• Binding XML representation with Input/output controls • Loading data from XML schema • Saving input in XML Schema (NA)
Remarks	• Goal achieved for binding schema with controls • For later stages, if possible, we can use MVVM approach to bind the schema on code behind.

4.5.2.4. Supervisor's feedback

Legend	Information
Version	2.1
Date	25/05/2016
User	• Developer of Module • RailCOMPLETE Developers • Claus Feyling • Bjørnar Steinnes Luteberget • Christian Johansen
Functionalities	• Saving input in XML Schema
Remarks	• Goal achieved

4.5.3. Version#3 feedback

4.5.3.1. Supervisor's feedback

Legend	Information
Version	3.0
Place	University of Oslo
Date	26/08/2016
Users	• Christian Johansen • Bjørnar Steinnes Luteberget
Level	Internal (Supervisor's review)
Implementation:	• Multiple constant elements on train movements • GUI reworked

	• Logic made generic for any level of xml tree
Remarks	• Goal achieved • After integration, MVVM can be implemented

4.5.3.2. RailCOMPLETE developer's feedback

Legend	**Information**
Version	3.0
Place	University of Oslo
Date	01/09/2016
Users	• RailCOMPLETE Developers (additionally as Track Engineers) • Claus Feyling (As Interlocking Engineer)
Level	External (RailCOMPLETE)
Implementation:	• Multiple constant elements on train movements • GUI reworked • Logic made generic for any level of xml tree
Remarks	• Goal achieved

4.5.4. Version#4 feedback

4.5.4.1. Supervisor remarks

Legend	**Information**
Version	4.0
Place	University of Oslo
Date	06/09/2016
Users	• RailCOMPLETE Developers (additionally As Track Engineers) • Claus Feyling (As Interlocking Engineer)
Level	External (RailCOMPLETE)
Implementation:	• Integration with RailCOMPLETE • GUI changes due to AutoCAD limitation
Remarks	• Goal achieved

4.5.4.2. Clause Feyling remarks

Legend	**Information**
Version	4.0
Place	University of Oslo
Date	10/09/2016
Users	• Claus Feyling (As Interlocking Engineer)
Level	External (RailCOMPLETE)
Implementation:	• Integration with RailCOMPLETE • GUI changes due to AutoCAD limitation
Remarks	• Goal achieved • Add/Delete button for tree elements dynamically (extended functionality)

4.5.4.3. RailCOMPLETE Developers remarks

Legend	Information
Version	4.0
Place	University of Oslo
Date	17/09/2016
Users	• RailCOMPLETE Developers (additionally As Track Engineers)
Level	External (RailCOMPLETE)
Implementation:	• Integration with RailCOMPLETE • GUI changes due to AutoCAD limitation
Remarks	• Goal achieved

4.6. Technologies

As this project was a part of RailCOMPLETE, some of the used technologies were pre-requisite for implementation, while other framework or technologies are used for supporting the vision of project.

Microsoft .Net 4.6: RailCOMPLETE is being developed in .Net framework, and although using .Net framework was not requirement but due to compatibility benefits, I choose to carry development in Microsoft .Net framework.

AutoCAD 2016: AutoCAD is design tool being used in RailCOMPLETE for design Railway routes. I used AutoCAD API for communicating with AutoCAD model of infrastructure, so I can populate or create Interlocking file from real data.

I am using AutoCAD.Net API for this project to access module management features to integrate Interlocking module in AutoCAD environment.

RailCOMPLETE: RailCOMPLETE is an AutoCAD plugin which provide the functionality to draw and define the railway related drawings, tables, 3D views, reports and railML® files.

RailML: An open xml based data exchange format for railway applications interoperability. [20]

WPF/XAML: UX design of the project is being done in Windows Presentation Foundation which takes XAML syntax to interpret and draw design components. WPF gives flexibility of controls and their customization. As RailCOMPLETE will be upgraded to WPF in future, keeping that prospective in mind, I used WPF for this project.

CHAPTER 5

SUMMARY & FUTURE WORK

5. Summary and Future Work

In this last chapter, we briefly write about possible future work directions to improve this framework. Certainly, this project has fair bit of room for improvement in term of extensibility and functionality. We will close this chapter with a detailed summary of our work in this project.

5.1. Achievement of project objectives

In the beginning of this thesis, we stated preliminary objectives for this project. During the process of requirement gathering and our interaction with railway engineers, we acquired the understanding of how these goals stands in term of usability for users and achievable in term of project duration.

Following is the list of objective and description of our progress for the accomplishments of these objectives.

1. First objective was to present interlocking tabular data in a format which can be created and modified easily in comparison to tabular sheets. We derived interlocking schema from specification presented in “Effective formulization of Interlocking data in RailML” [3].
2. Our next goal was to translate this interlocking XML schema in Object oriented code structure so we can manipulate and perform processing on future usage. We accomplished this task by using Microsoft .Net framework.
3. Once we created the library (.dll) for interlocking schema in object-oriented classes, we started designing interface for the representation of these specifications. We first created WinForms based UI but after discussion with RailCOMPLETE developers, we started designing interface in Windows Presentation Foundation. Graphical User Interface was the tricky part of the project since interlocking specifications can vary and so does the does in those specification files. So, we must create a dynamic interface based on a specific XML structure. We solve this problem by defining a generic hierarchy of tree-view control and assigning XML nodes programmatically to tree-view childs. Due to this, we slightly changed the domain model to meet our requirements, as an interlocking file can have multiple childs nodes under single root for example **Interlocking** root can have both Signals and Routes. We introduced Collection Containers as Composite Collection object to overcome this issue.
4. Next step was to integrate our module with RailCOMPLETE framework. We acquired the code for framework and studied its architecture. Since RailCOMPLETE is a plugin for AutoCAD itself, we had to understand AutoCAD.net API as well. We created a global string constant to save Interlocking specifications created in second step, as discussed earlier.

5.2.Critical Reflections

Through above mentioned stages, based on my struggle I realized that I have invested much time on interface designing of this project. Since XML specifications for interlockings can differ so we needed a generic approach to bind user interactive controls with specifications. For solving this issue, I had to sketch templates for binding, which are serving the purpose but later during my research I found that If I had used Model-View-ViewModel approach from beginning, it could have saved a deliberate amount of time for the development of future functionalities, given that learning of MVVM in WPF takes as much time as the approach I used. As in right now, our controls can get data from XML structure but in the backend, we are serializing and deserializing all information in one long step. While MVVM could have provided us a two-way binding which can store data in models and those models

are further linked with XML nodes. In this fashion, a very useful feature for dynamically creating/modifying new nodes of elements in already created interlockings, could have been achieved. Furthermore, by using this technique we could have saved time for the enhancement of this project and more user feedback.

One another important aspects of findings throughout this research is, I used IBM Rational Unified Process methodology for the management of software development and requirement gathering process. I found that if I had attempted to implement this methodology in its complete form for this project work, that would have leads this project to further delay. As RUP is very extensive model and for small teams and projects It might not be beneficial to use if implemented in full scope. During the analysis phase, based on duration of our project and efforts for requirement gathering, I decided to only use RUP for the system requirement gathering during our sessions with users and to capture those requirements in UML diagrams. This has saved us going off-track from our project plan. This is, however, admittedly, a difficult balance to negotiate in practice for projects and resources at this scale, but the I do realize that RUP can provide some baselines and directions to students and researchers in their projects for a systematic approach to solve a problem.

5.3.Summary

In this manuscript for thesis, we tried to describe the work done in partial fulfillment of the requirement for degree of Masters in Informatics: Programming & Networks. We started this work with a limited scope of an integrated module for RailCOMPLETE, for the automation of interlocking tabular data. To achieve this goal, we developed the said module for RailCOMPLETE with basic functionalities and integrated successfully with already developed RailCOMPLETE plugin for AutoCAD tool.

In chapter 1 we stated the initial motivation for this project, briefly introduced RailCOMPLETE framework and scope of functionality for our thesis as well as structure of this essay.

In chapter 2 we give a thorough briefing on railway signals and its development through the history. Key concepts and elements being used in railway tracks are also discussed with figures where applicable. Furthermore, Motivation behind this project, problem statement for the need of Interlocking editor are also discussed in this chapter.

Software project requirements and details of RailCOMPLETE framework are discussed in planning and analysis chapter. We used IBM RUP [21] framework to illicit user requirement first which followed by the implementation in later stage of the development. RUP requires that before an application is build, a through business modelling and requirement gathering must be done. Based on our meetings with users (RailCOMPLETE employees), we proposed the functionality of project in RUP templates using UML.

Later we presented the detail of implementation throughout its several prototype versions and technologies being used. We also discussed the framework we used for software development.

Evaluation comments of this tool are captured and presented in chapter 4, as we demonstrated the tool after each iteration of prototype. The evaluation of Interlocking editor showed good acceptance results as by using a built-in Interlocking editor, management of different key elements from interlocking schema, users could achieve the desired results with much more ease than the traditional

method of recording in tabular sheets. However since, advance features of the project were not implemented due to time constraints and their level of complexity, full potential of project cannot be gauged. We pointed toward those extensions of functionalities in following section**.** We encouraged forthcoming, research students interested in railways information systems and graph search to improve this project. Outcome can be refined by conducting further studies on different railways systems and by including more user types and their requirements in this project. However, since Interlocking is a sensitive and essential part of railway operations, safety must not be neglected as advance features includes prediction of rail movements based on user input.

5.4.Future Work

As discussed in chapter 1, there are some advance aspects of functionality where are a desirable extension to current program including but not limited to visualization of interlocking and synthesis of interlocking tables from track layout. In this section these aspects are briefly discussed.

5.4.1. Interlocking Visualization

AutoCAD design file contains many different symbols and a RailCOMPLETE document is a complex drawing of whole rail track. While working in Interlocking Editor, a highlighted route when selected an element in interlocking editor, can help user in readability of the design.

Furthermore, visualization of train movements as per interlocking data, where a user can manipulate information regarding interlocking element and a simulator simulates this movement on said track can further enhance the safety of the design and help user in observe changes in real time.

5.4.2. Synthesis of Interlocking tables

Much of the contents of the tabular interlocking is easily derived by looking at the station layout. For example, overlapping routes must be listed in the interlocking table, and this information can be automatically deduced by a program checking overlaps between all combinations of routes.

A processing of rules related to a specific rail system, and based on those constraints processing track diagram to list overlapping routes, suggesting elements on track, violations of rules and automation of consistency in user input can enhance the usability of the program. Graph search and logic encoding may be the techniques that can be used for this functionality.

5.4.3. Domain Model extensions

In one of the session with RailCOMPLETE engineers, Clause Feyling(C.E.O) mentioned Local release area elements being used widely in Norway than other railways. This opens the possibility to further enhance the domain model presented in this project based on railML specifications, to meet different railway system's specification. Since implementation for conversion of interlocking schema and classes is generic, any extension and further work on domain model will increase the usability of the project and help make it regulation specific for special needs.

Bibliography

[1] "AutoCAD," AutoDesk, [Online]. Available: http://www.autodesk.com/products/autocad/overview. [Accessed 8 11 2016].

[2] "Office Primary Interop Assemblies," [Online]. Available: https://msdn.microsoft.com/en-us/library/15s06t57.aspx. [Accessed 21 September 2016].

[3] M. B. E. Q. B. J. and R. M. G. , "Efficient formalization of railway interlocking data in RailML," 2014.

[4] "Railway Operation and Control “by Joern Pachl".

[5] A. Lawrence, Verification of Railway Interlockings in Scade, Swansea, Wales: Swansea University, 2011.

[6] H. Schofield, "How Napoleon's semaphore telegraph changed the world," [Online]. Available: http://www.bbc.com/news/magazine-22909590. [Accessed 19 July 2016].

[7] "Railway signal," [Online]. Available: https://en.wikipedia.org/wiki/Railway_signal. [Accessed 18 July 2016].

[8] "Railway Semaphore Signal," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Railway_semaphore_signal. [Accessed 22 September 2016].

[9] "ROUTE SIGNALLING," [Online]. Available: http://www.railway-technical.com/sigtxt5.shtml. [Accessed 19 July 2016].

[10] "OVERHAULING AND TESTING OF INTERLOCKING," Indian Railways, [Online]. Available: http://www.indianrailways.gov.in/railwayboard/uploads/codesmanual/SEM-II/SignalEngineering%20ManualIICh13_data.htm. [Accessed 22 September 2016].

[11] M. Marks, "provision of Overlaps, Flank Protection and Trapping," *Railway Group Standard,* vol. GK/RT0064, no. One, pp. 9-10, 2000.

[12] J. Pachl, "Railway Knowledge from Jörn Pachl," 16 9 2016. [Online]. Available: http://www.joernpachl.de/German_principles.htm. [Accessed 16 9 2016].

[13] R. Software, "Rational Unified Process, Best Practices for Software Development Teams," Rational Software , 2001.

[14] P. Kruchten, "A Rational Development Process," vol. 9, no. 7, pp. 11-16, 1996.

[15] P. Christensson, ""RUP Definition." TechTerms," Sharpened Productions, 2006, [Online]. Available: http://techterms.com/definition/rup. [Accessed Web. 26 September 2016].

[16] "Rational Software Architect Designer," IBM, [Online]. Available: http://www-03.ibm.com/software/products/en/ratsadesigner. [Accessed 17 10 2016].

[17] Microsoft, "XML Schema Definition Tool (Xsd.exe)," Microsoft Corporation, [Online]. Available: https://msdn.microsoft.com/en-us/library/x6c1kb0s(v=vs.110).aspx. [Accessed 7 March 2016].

[18] "Automatic loading of .NET modules," Through the Interface, [Online]. Available: http://through-the-interface.typepad.com/through_the_interface/2006/09/automatic_loadi.html. [Accessed 17 October 2016].

[19] "AutoCAD Help," Autodesk, [Online]. Available: http://help.autodesk.com/view/ACD/2016/ENU/?guid=GUID-70D60274-57E0-4B22-8D0C-3C7F212A7CAF. [Accessed 17 October 2016].

[20] "RailML Concept," RailML.org, [Online]. Available: https://www.railml.org/en/introduction/concept.html. [Accessed 14 October 2016].

[21] "Rational Unified Process," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Rational_Unified_Process. [Accessed 2 10 2016].

Appendices

Appendix A

Installation for Windows 10

This section contains the instruction to setup a Microsoft Windows 10 based device for the installation of Microsoft Visual Studio, AutoCAD, AutoCAD.Net API packages through Package manager console. This installation also works for Windows 7 and Windows 8.x

A.1 Microsoft Visual Studio 2015

Microsoft Visual Studio is an Integrated Development Environment (IDE) by Microsoft which enables developers to write code efficiently without losing the context. It helps to utilize different frameworks easily, build, debug and create different project extensions based on preferences of the project on a variety of platforms including but not limited to Windows, Linux, Mac etc.

A.1.1 Installation

a) Gain Administrator Access on machine
b) Download (Buy) the appropriate installation version per your system specifications, from https://www.visualstudio.com
c) Start the installation by running Setup.exe in administrative mode, which is usually located in the root directory of file where program was downloaded and follow the on-screen instructions.
d) Finish the software installation and restart computer to start using MS Visual Studio.

A.2 AutoCAD 2016

A.2.1 Installation

a) Download (Purchase) the product version specific to your system specification from https://www.autodesk.com/
b) Launch the Autodesk AutoCAD product installer by .exe or .dmg file you have downloaded
c) Select the desire language on top-right corner and press install.
d) Follow the instruction and Choose Autodesk AutoCAD 2016 from list of products.
e) Finish installation.

A.3 Nuget Package for AutoCAD 21.02

A.3.1 Installation

1. To install AutoCAD.Net API, run the following command in Package Manager Console or search in Nuget Packet Manager for AutoCAD.Net

```
PM> Install-Package AutoCAD.NET
```

Appendix B

CODE

Microsoft .Net C# and XAML code for Interlocking Object representation of railML interlocking schema and RailCOMPLETE Module

B.1 Components

This project consists of three main components to accomplish the goal specified in this thesis. We are getting the RailCOMPLETE document from AutoCAD using AutoCAD.net API and then serializing/deserializing interlocking data based on actions, in XML or C# object for further processing. GUI components are specified in XAML syntax using WPF framework using custom templates, while its processing is implemented in code behind file using C#. Interlocking schema is translated in Interlocking.cs file in multiple nested classes using XML tags, so that the correctness of data must meet the railML schema standards.

Code written in this project is given in following section.

B.1.1 Editor.cs (C#)

Class responsible for management of RailCOMPLETE document with our module.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using RailCOMPLETE.Common;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices.Core;
using RailCOMPLETE.AutoCAD;
using Autodesk.AutoCAD.ApplicationServices;



[assembly: RailCOMPLETE.Common.Module("Interlocking Editor", true, "2016-03-02", "Interlocking", true)]
[assembly: ExtensionApplication(typeof(InterlockingEditor.Editor))]
[assembly: CommandClass(typeof(InterlockingEditor.Editor))]
namespace InterlockingEditor
{
    public class Editor : RailCOMPLETEModule
    {
        private static readonly string INTERLOCKING_HEADER = "Interlocking";

        [CommandMethod("SHAHZAD-INTERLOCKING-EDITOR")]
        public void InterlockingEditorCommand()
        {
            var doc = Autodesk.AutoCAD.ApplicationServices.Core.Application.DocumentManager.MdiActiveDocument;
            var serializer = new System.Xml.Serialization.XmlSerializer(typeof(Interlocking));
            if (!doc.IsRailCompleteDocument())
            {
                doc.Editor.WriteMessage("\nNot a RailComplete document.\n");
                return;
            }
            else
            {
                var existingObjectAsString = LoadInterlockingString(doc);
                Interlocking interlocking = null;
                if (existingObjectAsString != null)
                {
                    interlocking = (Interlocking)XmlTools.DeserializeString(serializer, existingObjectAsString);

                }
                else
                {
                    interlocking = new Interlocking();
                }

                var myCon = new MainWindow();
                Autodesk.AutoCAD.ApplicationServices.Core.Application.ShowModalWindow(myCon);

                var objectAsString = XmlTools.SerializeToString(serializer, interlocking);
                SaveInterlockingString(doc, objectAsString);
            }
```

```
53.
54.
55.         }
56.
57.         public override void Terminate()
58.         {
59.         }
60.
61.         protected override void Init()
62.         {
63.         }
64.
65.         private string LoadInterlockingString(Document doc)
66.         {
67.             string s = null;
68.             using (var transaction = RCTransaction.StartOpenCloseTransaction(doc))
69.             {
70.                 s = doc.Database.GetDataString(transaction, INTERLOCKING_HEADER);
71.                 transaction.Commit();
72.             }
73.
74.             return s;
75.         }
76.
77.         private void SaveInterlockingString(Document doc, string contents)
78.         {
79.             using (var transaction = RCTransaction.StartOpenCloseTransaction(doc))
80.             {
81.                 doc.Database.SetData(transaction, INTERLOCKING_HEADER, contents);
82.                 transaction.Commit();
83.             }
84.         }
85.     }
86. }
```

B.1.2 MainWindow.xaml (XAML)

```
<Window x:Class="InterlockingEditor.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:o="clr-
namespace:InterlockingEditor" mc:Ignorable="d" Title="MainWindow" Height="350" Width="525">
    <Window.CommandBindings>
        <CommandBinding Command="o:CustomCommands.LoadXMLCommand" Executed="ExecutedLoadXML" />
        <CommandBinding Command="o:CustomCommands.SaveXMLCommand" Executed="ExecutedSaveXML" />
    </Window.CommandBindings>
    <Window.Resources>
        <HierarchicalDataTemplate DataType="{x:Type o:Interlocking}" ItemsSource="{Binding
Path=InterlockingChildren}">
            <TextBlock Text="Interlocking"></TextBlock>
        </HierarchicalDataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:Signals}" ItemsSource="{Binding Path=Signal}">
            <TextBlock Text="Signal"></TextBlock>
        </HierarchicalDataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:Signal}" ItemsSource="{Binding
Path=AspectSpeedDependencies}">
            <Grid Margin="3" MinWidth="300">
                <Grid.RowDefinitions>
                    <RowDefinition />
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition />
                    <ColumnDefinition />
                </Grid.ColumnDefinitions>
                <TextBlock Text="Ref" Grid.Column="0" Grid.Row="0" />
                <TextBox Text="{Binding Path=Ref, Mode=TwoWay}" Grid.Column="1" Grid.Row="0" />
            </Grid>
        </HierarchicalDataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:AspectSpeedDependencies}" ItemsSource="{Binding
Path=AspectSpeedDependency}">
            <TextBlock Text="AspectSpeedDependencies"></TextBlock>
        </HierarchicalDataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:AspectSpeedDependency}" ItemsSource="{Binding
Path=TargetRef}">
            <Border BorderThickness="1" MinWidth="300">
                <StackPanel Height="auto" Width="auto">
                    <TextBlock Text="AspectSpeedDependency:" />
                    <StackPanel Orientation="Horizontal">
                        <TextBlock Text="Aspect:" Margin="1" />
                        <TextBox Text="{Binding Path=Aspect, Mode=TwoWay}" Margin="1" />
                        <TextBlock Text="VApproach:" Margin="1" />
                        <TextBox Text="{Binding Path=VApproach, Mode=TwoWay}" Margin="1" />
                        <TextBlock Text="VPass:" Margin="1" />
                        <TextBox Text="{Binding Path=VPass, Mode=TwoWay}" Margin="1" />
                    </StackPanel>
                </StackPanel>
            </Border>
        </HierarchicalDataTemplate>
        <DataTemplate DataType="{x:Type o:TargetRef}">
            <Border BorderThickness="1" MinWidth="300">
```

```
                <Grid Margin="3">
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="Ref:" Grid.Column="0" Grid.Row="0" />
                    <TextBox Text="{Binding Path=Ref, Mode=TwoWay}" Grid.Column="1" Grid.Row="0" />
                </Grid>
            </Border>
        </DataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:Routes}" ItemsSource="{Binding Path=Route}">
            <TextBlock Text="Routes"></TextBlock>
        </HierarchicalDataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:Route }" ItemsSource="{Binding
Path=RouteChildern}">
            <Grid Margin="3" MinWidth="300">
                <Grid.RowDefinitions>
                    <RowDefinition />
                    <RowDefinition />
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition />
                    <ColumnDefinition />
                </Grid.ColumnDefinitions>
                <TextBlock Text="Id" Grid.Column="0" Grid.Row="0" />
                <TextBox Text="{Binding Path=Id, Mode=TwoWay}" Grid.Column="1" Grid.Row="0" />
                <TextBlock Text="FlankElements:" Grid.Column="0" Grid.Row="1" />
                <TextBox Text="{Binding Path=FlankElements, Mode=TwoWay}" Grid.Column="1" Grid.Row="1"
/>
            </Grid>
        </HierarchicalDataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:Start}" ItemsSource="{Binding Path=SignalRef}">
            <TextBlock Text="Start"></TextBlock>
        </HierarchicalDataTemplate>
        <DataTemplate DataType="{x:Type o:SignalRef}">
            <Border BorderThickness="1" MinWidth="300">
                <Grid Margin="3">
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="Ref:" Grid.Column="0" Grid.Row="0" />
```

```
                    <TextBox Text="{Binding Path=Ref, Mode=TwoWay}" Grid.Column="1" Grid.Row="0" />
                </Grid>
            </Border>
        </DataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:Target}" ItemsSource="{Binding Path=SignalRef}">
            <TextBlock Text="Target"></TextBlock>
        </HierarchicalDataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:Elements}" ItemsSource="{Binding
Path=ElementChildren}">
            <TextBlock Text="Elements"></TextBlock>
        </HierarchicalDataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:SwitchRef}" ItemsSource="{Binding Path=Switch}">
            <TextBlock Text="SwitchRef"></TextBlock>
        </HierarchicalDataTemplate>
        <DataTemplate DataType="{x:Type o:Switch}">
            <Border BorderThickness="1" MinWidth="300">
                <Grid Margin="3">
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="Ref:" Grid.Column="0" Grid.Row="0" />
                    <TextBox Text="{Binding Path=Ref, Mode=TwoWay}" Grid.Column="1" Grid.Row="0" />
                    <TextBlock Text="Course:" Grid.Column="0" Grid.Row="1" />
                    <TextBox Text="{Binding Path=Course, Mode=TwoWay}" Grid.Column="1" Grid.Row="1" />
                </Grid>
            </Border>
        </DataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:LevelcrossingRef}" ItemsSource="{Binding
Path=Levelcrossing}">
            <TextBlock Text="LevelcrossingRef"></TextBlock>
        </HierarchicalDataTemplate>
        <DataTemplate DataType="{x:Type o:Levelcrossing}">
            <Border BorderThickness="1" MinWidth="300">
                <Grid Margin="3">
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="Ref:" Grid.Column="0" Grid.Row="0" />
                    <TextBox Text="{Binding Path=Ref, Mode=TwoWay}" Grid.Column="1" Grid.Row="0" />
                    <TextBlock Text="Beam:" Grid.Column="0" Grid.Row="1" />
```

```
                    <TextBox Text="{Binding Path=Beam, Mode=TwoWay}" Grid.Column="1" Grid.Row="1" />
                </Grid>
            </Border>
        </DataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:TrainDetectorRef}" ItemsSource="{Binding Path=TrackCircuitBorder}">
            <TextBlock Text="TrainDetectorRef"></TextBlock>
        </HierarchicalDataTemplate>
        <DataTemplate DataType="{x:Type o:TrackCircuitBorder}">
            <Border BorderThickness="1" MinWidth="300">
                <Grid Margin="3">
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="Ref:" Grid.Column="0" Grid.Row="0" />
                    <TextBox Text="{Binding Path=Ref, Mode=TwoWay}" Grid.Column="1" Grid.Row="0" />
                </Grid>
            </Border>
        </DataTemplate>
        <HierarchicalDataTemplate DataType="{x:Type o:RoutePriority}">
            <Border BorderThickness="1" MinWidth="300">
                <Grid Margin="3">
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                        <RowDefinition />
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="Route Priority:" Grid.Column="0" Grid.Row="0" />
                    <TextBlock Text="Rank:" Grid.Column="0" Grid.Row="1" />
                    <TextBox Text="{Binding Path=Rank, Mode=TwoWay}" Grid.Column="1" Grid.Row="1" />
                </Grid>
            </Border>
        </HierarchicalDataTemplate>
    </Window.Resources>
    <Grid>
        <StackPanel VerticalAlignment="Top" Margin="0,68,0,0">
            <DockPanel HorizontalAlignment="Right">
                <Button Command="o:CustomCommands.LoadXMLCommand" Content="{Binding RelativeSource={RelativeSource Self}, Path=Command.Text}" Width="150" />
                <Button Command="o:CustomCommands.SaveXMLCommand" Content="{Binding RelativeSource={RelativeSource Self}, Path=Command.Text}" Width="150" />
            </DockPanel>
```

```
201.         </StackPanel>
202.         <StackPanel Margin="0,97,0,0">
203.             <ScrollViewer>
204.                 <Grid Height="800" DockPanel.Dock="Bottom">
205.                     <TreeView Margin="3" Name="treeView1" Height="auto" DockPanel.Dock="Left"
    Background="{DynamicResource LightColorBrush}">
206.                         <TreeView.ItemContainerStyle>
207.                             <Style TargetType="{x:Type TreeViewItem}">
208.                                 <Setter Property="IsExpanded" Value="True" />
209.                             </Style>
210.                         </TreeView.ItemContainerStyle>
211.                     </TreeView>
212.                 </Grid>
213.             </ScrollViewer>
214.         </StackPanel>
215.     </Grid>
216. </Window>
```

B.1.3 MainWindow.cs (C#)

```
1.  using RailCOMPLETE.Common;
2.  using System;
3.  using System.Collections.Generic;
4.  using System.Diagnostics;
5.  using System.IO;
6.  using System.Linq;
7.  using System.Reflection;
8.  using System.Text;
9.  using System.Threading.Tasks;
10. using System.Windows;
11. using System.Windows.Controls;
12. using System.Windows.Data;
13. using System.Windows.Documents;
14. using System.Windows.Input;
15. using System.Windows.Media;
16. using System.Windows.Media.Imaging;
17. using System.Windows.Navigation;
18. using System.Windows.Shapes;
19. using System.Xml;
20. using System.Xml.Linq;
21. using System.Xml.Serialization;
22.
23.
24. namespace InterlockingEditor
25. {
26.     /// <summary>
27.     /// Interaction logic for MainWindow.xaml
28.     /// </summary>
29.     public partial class MainWindow : Window
30.     {
31.         public MainWindow()
32.         {
33.             InitializeComponent();
34.         }
35.
36.         private void ExecutedLoadXML(object sender, ExecutedRoutedEventArgs e)
37.         {
38.
39.             string executableLocation =
System.IO.Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
40.             string xslLocation = System.IO.Path.Combine(executableLocation, "Interlocking.xml");
41.
42.             XDocument xmlData = XDocument.Load(xslLocation, LoadOptions.None);
43.             var Interlocking = XmlSerializationHelper.LoadFromXML<Interlocking>(xmlData.ToString());
44.
45.             var children = new List<Interlocking>();
46.             children.Add(Interlocking);
47.
48.             treeView1.ItemsSource = null;
49.             treeView1.Items.Clear();
50.             treeView1.ItemsSource = children;
51.         }
52.
53.         private void ExecutedSaveXML(object sender, ExecutedRoutedEventArgs e)
```

```
54.         {
55.             var planList = treeView1.ItemsSource as IList<Interlocking>;
56.             if (planList != null && planList.Count > 0)
57.             {
58.                 // Peding edit's logic
59.                 treeView1.Focus();
60.
61.                 // Replace with display code!
62.                 //
63.                 //
64.                 string s = planList[0].GetXml();
65.
66.             }
67.         }
68.
69.     }
70.     public static class CustomCommands
71.     {
72.         public static readonly RoutedUICommand LoadXMLCommand = new RoutedUICommand("Load XML",
    "LoadXML", typeof(MainWindow));
73.
74.         public static readonly RoutedUICommand SaveXMLCommand = new RoutedUICommand("Save XML",
    "SaveXML", typeof(MainWindow));
75.     }
76.     public static class XmlSerializationHelper
77.     {
78.         public static string GetXml<T>(T obj, XmlSerializer serializer, bool omitStandardNamespaces)
79.         {
80.             using (var textWriter = new StringWriter())
81.             {
82.                 XmlWriterSettings settings = new XmlWriterSettings();
83.                 settings.Indent = true;         // For cosmetic purposes.
84.                 settings.IndentChars = "    "; // For cosmetic purposes.
85.                 using (var xmlWriter = XmlWriter.Create(textWriter, settings))
86.                 {
87.                     if (omitStandardNamespaces)
88.                     {
89.                         XmlSerializerNamespaces ns = new XmlSerializerNamespaces();
90.                         ns.Add("", ""); // Disable the xmlns:xsi and xmlns:xsd lines.
91.                         serializer.Serialize(xmlWriter, obj, ns);
92.                     }
93.                     else
94.                     {
95.                         serializer.Serialize(xmlWriter, obj);
96.                     }
97.
98.                     SaveXML(obj, serializer);
99.                 }
100.
101.                return textWriter.ToString();
102.            }
103.        }
104.
105.        private static void SaveXML<T>(T obj, XmlSerializer serializer)
```

```
106.        {
107.
108.            string executableLocation =
    System.IO.Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
109.            string xslLocation = System.IO.Path.Combine(executableLocation, "Interlocking.xml");
110.            System.IO.FileStream file = System.IO.File.Create(xslLocation);
111.
112.            serializer.Serialize(file, obj);
113.            file.Close();
114.        }
115.
116.        public static string GetXml<T>(this T obj, bool omitNamespace)
117.        {
118.            XmlSerializer serializer = new XmlSerializer(obj.GetType());
119.            return GetXml(obj, serializer, omitNamespace);
120.        }
121.
122.        public static string GetXml<T>(this T obj)
123.        {
124.            return GetXml(obj, false);
125.        }
126.
127.        public static T LoadFromXML<T>(this string xmlString)
128.        {
129.            return xmlString.LoadFromXML<T>(new XmlSerializer(typeof(T)));
130.        }
131.
132.        public static T LoadFromXML<T>(this string xmlString, XmlSerializer serial)
133.        {
134.            T returnValue = default(T);
135.
136.            using (StringReader reader = new StringReader(xmlString))
137.            {
138.                object result = serial.Deserialize(reader);
139.                if (result is T)
140.                {
141.                    returnValue = (T)result;
142.                }
143.            }
144.            return returnValue;
145.        }
146.
147.        public static T LoadFromFile<T>(string filename)
148.        {
149.            XmlSerializer serial = new XmlSerializer(typeof(T));
150.            try
151.            {
152.                using (var fs = new FileStream(filename, FileMode.Open))
153.                {
154.                    object result = serial.Deserialize(fs);
155.                    if (result is T)
156.                    {
157.                        return (T)result;
158.                    }
```

```
159.                 }
160.             }
161.             catch (Exception ex)
162.             {
163.                // Debug.WriteLine(ex.ToString());
164.                 throw;
165.             }
166.             return default(T);
167.         }
168.     }
169. }
```

B.1.4 Interlocking Classes (C#)

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Data;
using System.Xml.Serialization;

namespace InterlockingEditor
{


    [XmlRoot(ElementName = "targetRef")]
    public class TargetRef
    {
        [XmlAttribute(AttributeName = "ref")]
        public string Ref { get; set; }
    }

    [XmlRoot(ElementName = "aspectSpeedDependency")]
    public class AspectSpeedDependency
    {
        [XmlElement(ElementName = "targetRef")]
        public List<TargetRef> TargetRef { get; set; }
        [XmlAttribute(AttributeName = "aspect")]
        public string Aspect { get; set; }
        [XmlAttribute(AttributeName = "vApproach")]
        public string VApproach { get; set; }
        [XmlAttribute(AttributeName = "vPass")]
        public string VPass { get; set; }
    }

    [XmlRoot(ElementName = "aspectSpeedDependencies")]
    public class AspectSpeedDependencies
    {
        [XmlElement(ElementName = "aspectSpeedDependency")]
        public List<AspectSpeedDependency> AspectSpeedDependency { get; set; }
    }

    [XmlRoot(ElementName = "signal")]
    public class Signal
    {
        [XmlElement(ElementName = "aspectSpeedDependencies")]
        public List<AspectSpeedDependencies> AspectSpeedDependencies { get; set; }
        [XmlAttribute(AttributeName = "ref")]
        public string Ref { get; set; }
    }

    [XmlRoot(ElementName = "signals")]
    public class Signals
    {
        [XmlElement(ElementName = "signal")]
        public List<Signal> Signal { get; set; }
```

```
55.     }
56.
57.     [XmlRoot(ElementName = "signalRef")]
58.     public class SignalRef
59.     {
60.         [XmlAttribute(AttributeName = "ref")]
61.         public string Ref { get; set; }
62.     }
63.
64.     [XmlRoot(ElementName = "start")]
65.     public class Start
66.     {
67.         [XmlElement(ElementName = "signalRef")]
68.         public List<SignalRef> SignalRef { get; set; }
69.     }
70.
71.     [XmlRoot(ElementName = "target")]
72.     public class Target
73.     {
74.         [XmlElement(ElementName = "signalRef")]
75.         public List<SignalRef> SignalRef { get; set; }
76.     }
77.
78.     [XmlRoot(ElementName = "switch")]
79.     public class Switch
80.     {
81.         [XmlAttribute(AttributeName = "ref")]
82.         public string Ref { get; set; }
83.         [XmlAttribute(AttributeName = "course")]
84.         public string Course { get; set; }
85.     }
86.
87.     [XmlRoot(ElementName = "switchRef")]
88.     public class SwitchRef
89.     {
90.         [XmlElement(ElementName = "switch")]
91.         public List<Switch> Switch { get; set; }
92.     }
93.
94.     [XmlRoot(ElementName = "levelcrossing")]
95.     public class Levelcrossing
96.     {
97.         [XmlAttribute(AttributeName = "ref")]
98.         public string Ref { get; set; }
99.         [XmlAttribute(AttributeName = "beam")]
100.        public string Beam { get; set; }
101.    }
102.
103.    [XmlRoot(ElementName = "levelcrossingRef")]
104.    public class LevelcrossingRef
105.    {
106.        [XmlElement(ElementName = "levelcrossing")]
107.        public List<Levelcrossing> Levelcrossing { get; set; }
108.    }
```

```
109.
110.     [XmlRoot(ElementName = "trackCircuitBorder")]
111.     public class TrackCircuitBorder
112.     {
113.         [XmlAttribute(AttributeName = "ref")]
114.         public string Ref { get; set; }
115.     }
116.
117.     [XmlRoot(ElementName = "trainDetectorRef")]
118.     public class TrainDetectorRef
119.     {
120.         [XmlElement(ElementName = "trackCircuitBorder")]
121.         public List<TrackCircuitBorder> TrackCircuitBorder { get; set; }
122.     }
123.
124.     [XmlRoot(ElementName = "elements")]
125.     public class Elements
126.     {
127.         [XmlElement(ElementName = "switchRef")]
128.         public List<SwitchRef> SwitchRef { get; set; }
129.         [XmlElement(ElementName = "levelcrossingRef")]
130.         public List<LevelcrossingRef> LevelcrossingRef { get; set; }
131.         [XmlElement(ElementName = "trainDetectorRef")]
132.         public List<TrainDetectorRef> TrainDetectorRef { get; set; }
133.
134.         [XmlIgnore]
135.         public IList ElementChildren
136.         {
137.             get
138.             {
139.                 return new CompositeCollection()
140.             {
141.                 new CollectionContainer() { Collection = SwitchRef },
142.                 new CollectionContainer() { Collection = LevelcrossingRef },
143.                 new CollectionContainer() { Collection = TrainDetectorRef }
144.             };
145.             }
146.         }
147.     }
148.
149.     [XmlRoot(ElementName = "routePriority")]
150.     public class RoutePriority
151.     {
152.         [XmlAttribute(AttributeName = "rank")]
153.         public string Rank { get; set; }
154.     }
155.
156.     [XmlRoot(ElementName = "route")]
157.     public class Route
158.     {
159.         [XmlElement(ElementName = "start")]
160.         public List<Start> Start { get; set; }
161.         [XmlElement(ElementName = "target")]
162.         public List<Target> Target { get; set; }
```

```
163.         [XmlElement(ElementName = "elements")]
164.         public List<Elements> Elements { get; set; }
165.         [XmlElement(ElementName = "flankElements")]
166.         public string FlankElements { get; set; }
167.         [XmlElement(ElementName = "routePriority")]
168.         public List<RoutePriority> RoutePriority { get; set; }
169.         [XmlAttribute(AttributeName = "id")]
170.         public string Id { get; set; }
171.
172.         [XmlIgnore]
173.         public IList RouteChildern
174.         {
175.             get
176.             {
177.                 return new CompositeCollection()
178.             {
179.                 new CollectionContainer() { Collection = Start },
180.                 new CollectionContainer() { Collection = Target },
181.                 new CollectionContainer() { Collection = Elements },
182.
183.                 new CollectionContainer() { Collection = RoutePriority }
184.
185.             };
186.             }
187.         }
188.     }
189.
190.
191.     [XmlRoot(ElementName = "routes")]
192.     public class Routes
193.     {
194.         [XmlElement(ElementName = "route")]
195.         public List<Route> Route { get; set; }
196.     }
197.
198.     [XmlRoot(ElementName = "interlocking")]
199.     public class Interlocking
200.     {
201.         [XmlElement(ElementName = "signals")]
202.         public List<Signals> Signals { get; set; }
203.         [XmlElement(ElementName = "routes")]
204.         public List<Routes> Routes { get; set; }
205.         //[XmlAttribute(AttributeName = "xsi", Namespace = "http://www.w3.org/2000/xmlns/")]
206.         //public string Xsi { get; set; }
207.         //[XmlAttribute(AttributeName = "xsd", Namespace = "http://www.w3.org/2000/xmlns/")]
208.         //public string Xsd { get; set; }
209.
210.         [XmlIgnore]
211.         public IList InterlockingChildren
212.         {
213.             get
214.             {
215.                 return new CompositeCollection()
216.             {
```

```
                new CollectionContainer() { Collection = Signals },
                new CollectionContainer() { Collection = Routes }
            };
            }
        }
    }
}
```

B.1.5 BuildEvents

```
 <PostBuildEvent>
if "$(ConfigurationName)" == "Debug" (
  mkdir "$(AutoCADBundleDir)"
  xcopy /E /Y /D "$(SolutionDir)RailCOMPLETE.bundle\*.*" "$(AutoCADBundleDir)"
  xcopy /E /Y /D "$(TargetDir)*" "$(AutoCADBundleDir)\Contents\"
) ELSE (
  xcopy /E /Y /D "$(TargetDir)*.dll" "$(SolutionDir)RailComplete.bundle\Contents\"
  "C:\Program Files (x86)\NSIS\makensis.exe" "$(SolutionDir)Installer\RailCOMPLETE.nsi"
)
</PostBuildEvent>
```

B.1.6 References

```
<Reference Include="AcCoreMgd, Version=21.0.0.0, Culture=neutral, processorArchitecture=MSIL">
  <HintPath>..\packages\AutoCAD.NET.Core.21.0.2\lib\45\AcCoreMgd.dll</HintPath>
  <Private>False</Private>
</Reference>
<Reference Include="AcCui, Version=21.0.0.0, Culture=neutral, processorArchitecture=MSIL">
  <HintPath>..\packages\AutoCAD.NET.21.0.2\lib\45\AcCui.dll</HintPath>
  <Private>False</Private>
</Reference>
<Reference Include="AcDbMgd, Version=21.0.0.0, Culture=neutral, processorArchitecture=MSIL">
  <HintPath>..\packages\AutoCAD.NET.Model.21.0.2\lib\45\AcDbMgd.dll</HintPath>
  <Private>False</Private>
</Reference>
<Reference Include="AcDbMgdBrep, Version=21.0.0.0, Culture=neutral, processorArchitecture=MSIL">
  <HintPath>..\packages\AutoCAD.NET.Model.21.0.2\lib\45\AcDbMgdBrep.dll</HintPath>
  <Private>False</Private>
</Reference>
<Reference Include="AcDx, Version=21.0.0.0, Culture=neutral, processorArchitecture=MSIL">
  <HintPath>..\packages\AutoCAD.NET.21.0.2\lib\45\AcDx.dll</HintPath>
  <Private>False</Private>
</Reference>
<Reference Include="AcMgd, Version=21.0.0.0, Culture=neutral, processorArchitecture=MSIL">
  <HintPath>..\packages\AutoCAD.NET.21.0.2\lib\45\AcMgd.dll</HintPath>
  <Private>False</Private>
</Reference>
<Reference Include="AcMr, Version=21.0.0.0, Culture=neutral, processorArchitecture=MSIL">
  <HintPath>..\packages\AutoCAD.NET.21.0.2\lib\45\AcMr.dll</HintPath>
  <Private>False</Private>
</Reference>
<Reference Include="AcTcMgd, Version=21.0.0.0, Culture=neutral, processorArchitecture=MSIL">
  <HintPath>..\packages\AutoCAD.NET.21.0.2\lib\45\AcTcMgd.dll</HintPath>
  <Private>False</Private>
</Reference>
<Reference Include="AcWindows, Version=21.0.0.0, Culture=neutral, processorArchitecture=MSIL">
  <HintPath>..\packages\AutoCAD.NET.21.0.2\lib\45\AcWindows.dll</HintPath>
  <Private>False</Private>
</Reference>
<Reference Include="AdWindows, Version=2015.11.2.0, Culture=neutral, processorArchitecture=MSIL">
  <HintPath>..\packages\AutoCAD.NET.21.0.2\lib\45\AdWindows.dll</HintPath>
  <Private>False</Private>
</Reference
```