



DOUBLE DEGREE PROGRAM IN INFORMATION AND COMMUNICATION TECHNOLOGIES ENGINEERING

GRADUATION PROJECT REPORT

AUTOMATED METRO GUIDANCE SYSTEM

Advisers

Dr. Alaa Hamdy
Dr. Claudio Fornaro

Students

Amr Hassan
Mohamed Essam El Din
Mohamed Abdelrazek
Moamen Mahmoud

Academic year 2011/12

Document revision number: 1.2

TABLE OF CONTENTS

| | |
|---|----|
| Summary..... | 7 |
| The Current Situation..... | 8 |
| The Proposed Solution..... | 9 |
| Introduction..... | 11 |
| Hardware Design Problems..... | 13 |
| A Train Needs to Be Able to Determine its Location..... | 14 |
| A Train Needs to Be Able to Determine its Location Underground..... | 15 |
| A Train Needs to Determine Which Track it is On Precisely..... | 16 |
| A Train Needs to Determine When People Stopped Going in and Out so That it would close the Doors..... | 17 |
| A Train Needs to Actuate the Incoming Driving Instructions from the Control Server ... | 17 |
| Emergency Situations Should Be Allowed to Be Reported By the Passengers..... | 18 |
| A Train Needs a Master Computer System..... | 18 |
| Railway Tracks Need to Be Switched Frequently..... | 19 |
| Control Room Operators Will Need a Way to Monitor the Railway Network..... | 19 |
| Trains, Railway Monitors, Railway Switching Booths Need to Communicate with the Central Control Server..... | 20 |
| Hardware System Components..... | 21 |
| The GPS Receiver..... | 22 |
| The UGPS..... | 23 |
| The TrackID Emitter..... | 24 |
| The TrackID Receiver..... | 24 |
| The Train Door Photo-sensor..... | 24 |
| The Emergency Button..... | 25 |

| | |
|---|----|
| The Train Controlling Actuators | 25 |
| The Train On-board Computer | 26 |
| The Railway Switching Hubs..... | 26 |
| Software Design Problems..... | 28 |
| Choosing the Software Platform for Building the Control Server..... | 29 |
| The Structuring of a Railway Layout in a Computer Software Memory | 29 |
| RailwayEntity | 30 |
| Trackpoint..... | 30 |
| Segment (inherits RailwayEntity) | 30 |
| ConjunctionSegment (inherits RailwayEntity)..... | 30 |
| Station (inherits RailwayEntity) | 31 |
| The Generation of Unique Ids | 31 |
| The Serialization of a Mapped-out Railway Layout | 32 |
| The Painting of a Railway Layout on a Computer Terminal Screen..... | 35 |
| Mapping Out the Railway Tracks | 36 |
| Other Objects that are Under the Control of the Control Server | 36 |
| Train..... | 36 |
| SwitchingHub..... | 37 |
| The General Communication Protocol between System Nodes and the Central Control Server | 38 |
| The Client Identification Stage | 39 |
| Securing the Communication between the Trains and the Central Control Server | 41 |
| The Role of Public-key Cryptography | 41 |
| The Role of Secret-key Cryptography..... | 42 |
| The Authentication Stage in the General Communication Protocol..... | 42 |
| The Conversation Communication Stage..... | 43 |

| | |
|---|----|
| The Encrypted Message Format..... | 43 |
| The Compressed Communication Message Format | 43 |
| The Variable-Length Message Format | 44 |
| The Fixed-Length Message Format | 45 |
| The Communication Protocol with a Train Client..... | 45 |
| Client Messages..... | 45 |
| Server Messages..... | 47 |
| The Communication Protocol with a Switching Hub | 48 |
| Server Messages..... | 48 |
| Client Messages..... | 49 |
| The Communication Protocol with a Monitoring Station..... | 49 |
| Request/Response Message Pairs..... | 50 |
| The Communication Protocol with a Controlling Station | 50 |
| Request/Response Message Pairs..... | 51 |
| Software System Components | 53 |
| The Control Server Software..... | 54 |
| Objectives | 54 |
| Platform..... | 54 |
| Connectivity..... | 54 |
| Human Interface..... | 54 |
| Installation and Usage | 54 |
| The Monitoring/Controlling Software | 55 |
| Objectives | 55 |
| Platform..... | 55 |
| Connectivity..... | 55 |
| Human Interface..... | 55 |

| | |
|---|----|
| Installation and Usage | 55 |
| The Schematic Editor | 56 |
| Objectives | 56 |
| Platform | 56 |
| Connectivity..... | 56 |
| Human Interface..... | 56 |
| Installation and Usage | 56 |
| Technologies Used | 63 |
| The Java Runtime Environment | 64 |
| Java 2D | 65 |
| Basic Concepts | 65 |
| Advanced Objects | 66 |
| GPS | 67 |
| Trilateration..... | 68 |
| Photoelectric Sensors..... | 69 |
| Types..... | 69 |
| Sensing modes | 70 |
| Transmission Control Protocol (TCP) | 71 |
| Network function..... | 71 |
| IP Networking..... | 73 |
| Function | 73 |
| Public-key cryptography..... | 73 |
| How it works..... | 75 |
| The RSA Algorithm | 76 |
| Operation..... | 76 |
| Encryption..... | 77 |

| | |
|---|----|
| Decryption | 78 |
| Secret-key Cryptography..... | 78 |
| Types of symmetric-key algorithms | 78 |
| Implementations | 79 |
| Bibliography | 80 |
| Document Revision History | 82 |
| 1.1 | 83 |
| 1.2 | 83 |

SUMMARY

The Current Situation

Currently, the entirety of the Metro railway system depends inefficiently on extensive human intervention. From conducting the individual trains to actually locating the conducted train and its neighboring trains.

Human conductors communicate with other conductors and station operators via a primitive audio communication system on which they receive instructions and report back their locations.

Given the close proximity of the train stops, and the large number of trains near each other on the same railroad, the human conductor tends to almost always be either accelerating or decelerating the train, depending on how near it is from the previous and next stops and the trains ahead and behind it.

The conductors have no way of knowing their exact coordinates but by looking at signs on the rails and at the familiar landscape. In times when weather conditions present viewing obstacles like heavy rain or fog, the distinction of the current location and that of the train ahead becomes very difficult (if not impossible), and the whole system comes to a near halt or to a catastrophe.

At train stops, the train conductors look at giant mirrors or monitors placed conveniently next to tank locomotive to give them full view of the doors and let them know when people are done getting in and out so they could shut the doors and take off.

The Proposed Solution

A more efficient system of conducting would eliminate the dependence on using human sight to locate the trains, and instead, would use a more modern and accurate location triangulation system that isn't affected by weather and other obstacles.

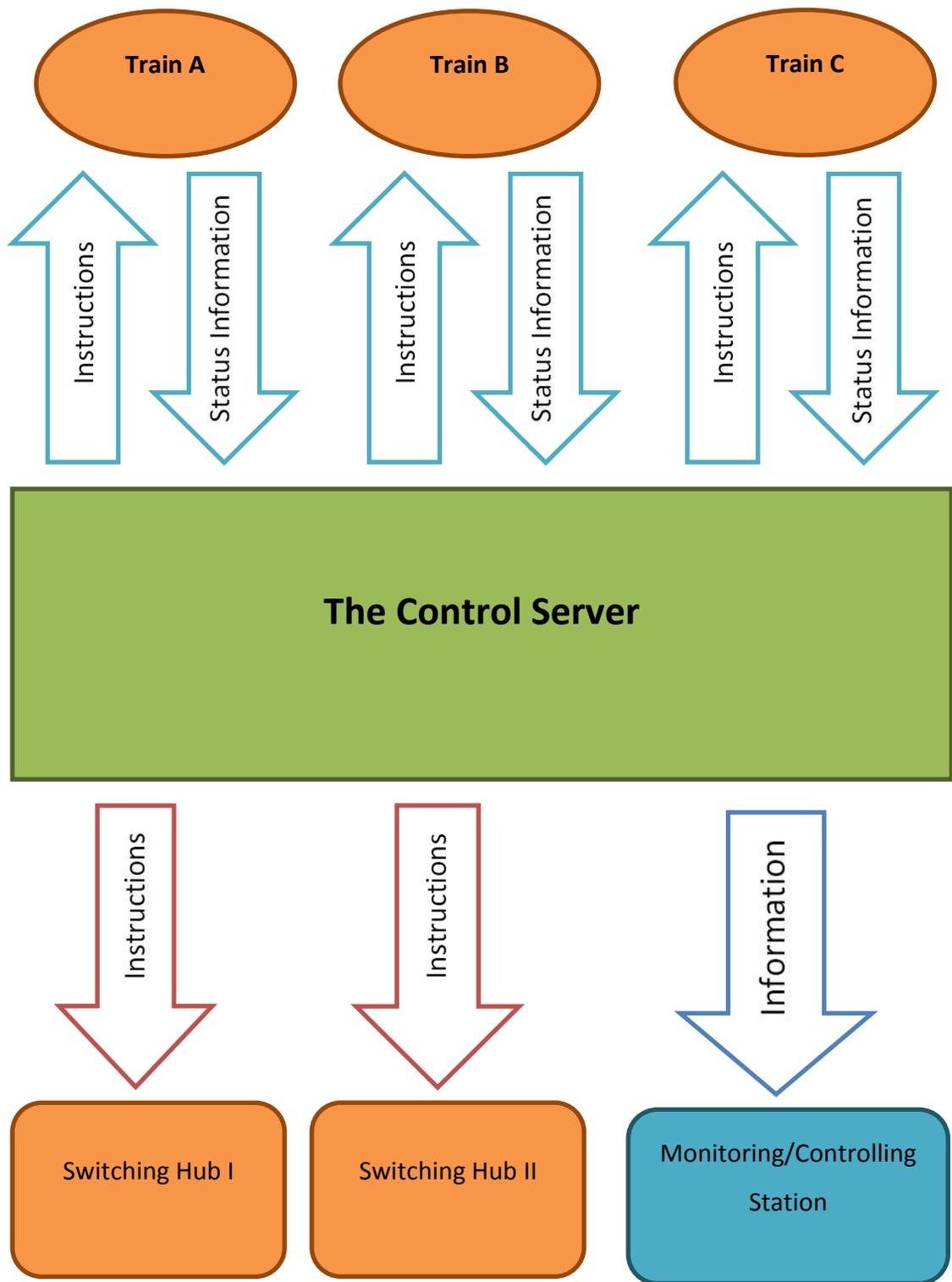
The location of all trains would be reported via a combination of wireless and wired data networks to the central offices and to station operators so that a clear view of the entire railroad and the trains on it is easily obtained and better steering decisions are made.

A central computer system would use the location data from all trains for finding the most efficient and safe steering instructions for every train and send them back on the same wireless network.

A terminal at the conducting booth would use actuators to realize these steering instructions without the actual intervention of any human being.

Sensors at each train door would decide when people stopped going in and out and automatically shut the doors most safely.

The proposed plan adds a lot more efficiency and manageability to the system and excludes the ever-present error factors resulting from relying on humans to do mundane jobs.



INTRODUCTION

Introduction

In the rest of this report, we shall explain the design problems that faced us while realizing the project, followed by a detailed description of each component of the system.

Since our project is mainly about the software side of this system, we don't go into much details about how the hardware components do what they are supposed to do, instead we try to break them down into as many sub-components as possible and describe how each of those sub-components are supposed to function in as much detail as possible so as to ease the problem of building it on the hardware engineer.

The contributions made by each team member:

| | |
|--------------------|--|
| Moamen Mahmoud | Technologies Used |
| Mohamed Abdelrazek | Hardware Design Problems |
| Amr Hassan | Software Design Problems |
| Mohamed Essam | Software System Components Hardware System Components |

HARDWARE DESIGN PROBLEMS

A Train Needs to Be Able to Determine its Location

Several solutions already exist for the problem of electronically determining one's location. We've decided to use an already tried-and-true method for the most reliability; thus, the Global Positioning System was our method of choice.

GPS is a space-based satellite navigation system that provides location and time information in all weather, anywhere on or near the Earth, where there is an unobstructed line of sight to four or more GPS satellites. It is maintained by the United States government and is freely accessible to anyone with a GPS receiver.

GPS is back-boned by a constellation of 27 Earth-orbiting satellites (24 in operation and three extras in case one fails). Each of these 3,000- to 4,000-pound solar-powered satellites circles the globe at about 12,000 miles (19,300 km), making two complete rotations every day. The orbits are arranged so that at anytime, anywhere on Earth, there are at least four satellites "visible" in the sky. A GPS receiver's job is to locate four or



more of these satellites, figure out the distance to each, and use this information to deduce its own location. This operation is based on a simple mathematical principle called trilateration.

The GPS system guarantees a positioning accuracy of 7.8 meters at a 95% confidence level. The actual accuracy depends on factors including atmospheric effects and receiver quality. Real-world data show that some high-quality GPS receivers currently provide better than 3 meter horizontal accuracy¹, which is very adequate for our use-case here.

¹ <http://www.gps.gov/systems/gps/performance/accuracy/>

A Train Needs to Be Able to Determine its Location Underground

The only problem with regular GPS in our project is that Metro trains spend about third of its time in underground tunnels, and GPS – by design - cannot be used underground because the ground dirt hinders the propagation of the GPS radio signals.

In 2004, researchers from the Swiss Institute of Speleology and Karstology (ISSKA²) were working on a system for underground positioning, and in 2010 a commercial company named InfraSurvey³ was established to sell the product of that research.

The Underground GPS (UGPS) sold by InfraSurvey consists of:

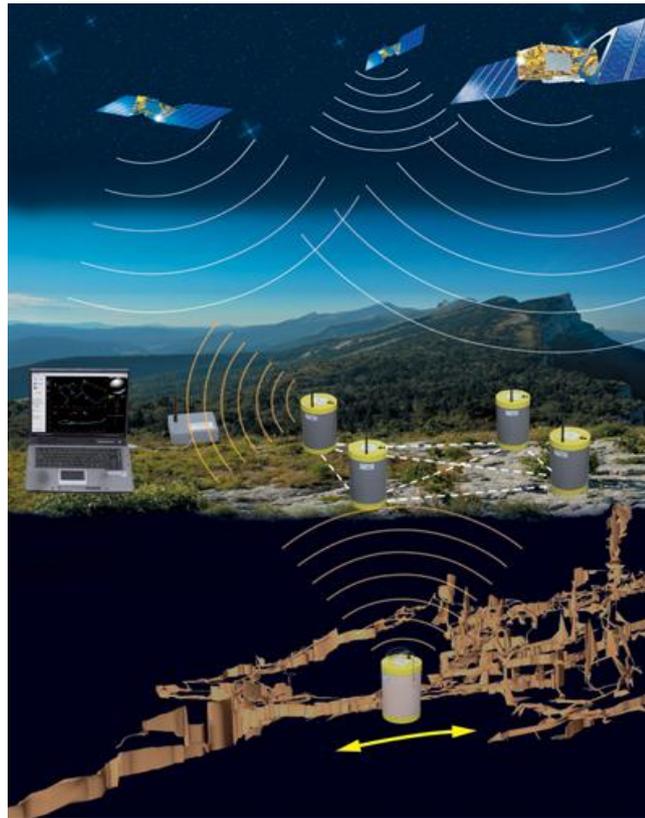
- An underground mobile emitter
- Four receivers stationed at fixed positions on the surface
- A radio link connecting the surface-mounted receivers and a computer.



² <http://www.isska.ch>

³ <http://www.infrasurvey.ch/?lang=en/>

It works as follows:



The UGPS uses an induced magnetic field created by the underground emitter, which is detected and analysed by four receiver stations at the surface. The receivers are static and positioned automatically by GPS. They communicate by a radio link with a surface computer system. The subsurface position and orientation of the emitter can be figured out by software on the computer system on the surface.

A Train Needs to Determine Which Track it is On Precisely

Since that railway roads usually go in pairs that are very close to each other, one for coming trains and one for going ones, GPS may not be accurate enough to determine which railway track the train is actually on.

Therefore a way of identifying the certain track beneath the train is needed. Such a way could be a tiny close-range radio transmitter attached to every railroad track segment,

emitting the unique id of that track segment so that trains that are passing over it can pick it up using a close-range radio receiver attached to the bottom of every train tank locomotive.

A Train Needs to Determine When People Stopped Going in and Out so That it would close the Doors

Simple photoelectric sensors installed at each door could easily tell if there is movement at the door area, which would mean that people are still going in and out.

A photoelectric sensor, or photoeye, is a device used to detect the distance, absence, or presence of an object by using a light transmitter, often infrared, and a photoelectric receiver. There are three different functional types: opposed (through beam), retroreflective, and proximity-sensing (diffused).⁴

A Train Needs to Actuate the Incoming Driving Instructions from the Control Server

This is the part of the system that has to deal with actually delivering the actuating signals to the train locomotives. Since it will probably be difficult to realize, we minimized the what is actually needed from the actuating system component to very primitive simple commands that include:

- Accelerating the train
- Decelerating the train
- Applying the emergency brakes
- Opening the doors on either side and closing them
- Reading the train speed

⁴ https://en.wikipedia.org/wiki/Photoelectric_sensor

The actuators should do whatever hacking necessary to the train driving console and control systems to be able to realize these requirements.

Emergency Situations Should Be Allowed to Be Reported By the Passengers

Due to the full human-absence in the conducting booths (where the train is controlled), and the full autonomy of the entire system, emergency situations can only be detected by the human passengers. So a way to report emergencies by passengers must be implemented and a procedure for handling such emergencies must be defined.

A Train Needs a Master Computer System

Since that it'd be a fairly complex network if each door sensor, railroad track receiver, on-board GPS receiver, UGPS receiver and driving actuators communicated with the central control server individually.

A better idea is to have one master embedded computer system on the train with software logic to handle all these external devices while connecting to them via an on-board communication network, and interface with the control server directly and relay the needed control messages between it and the various components on the train.

The design and the realization of this component is irrelevant, along with the way it communicates with the train hardware system components as long as it adheres to the communication protocol to be explained in detail later while communicating with the control server.

Railway Tracks Need to Be Switched Frequently

Parts of the railway tracks need to be switched frequently to allow the forming of complex railway systems. Currently, the switching is done manually by workers from switching booths that pull certain levers directly. A way to remotely manipulate these switches is needed. It should be via a layer of actuators on top of the current equipment that controls these switches and communicates with the control server as do the trains.

Also as are the train actuators, it is irrelevant to us how they are realized, as long as they communicate with the control server in the specified communication protocol.

Control Room Operators Will Need a Way to Monitor the Railway Network

Of course, complete exclusion of human interaction in this system is desired, but for reasons including safety of the trains and the entire railway system, humans still need to be able to monitor the system up close to be able to make sure everything is going smoothly and be able to intervene at any time to pull emergency procedures in case anything goes wrong.

A monitoring system should be set up running special monitoring software that will be written as a part of this project that will allow the human observers to watch up close the state of the entire railway system and intervene at any time to perform several pre-configured procedures, as long as the needed security credentials are provided.

Trains, Railway Monitors, Railway Switching Booths Need to Communicate with the Central Control Server

Since that trains will not perform any driving logic, each train needs to periodically communicate with the central control server in order to:

- Report location as received from the GPS device
- Report the track segment ID as picked up from the track close-range transmitters
- Report current speed
- Receive simple driving instructions such as accelerate, decelerate, stop, or any other pre-configured instructions such as an emergency door-open.
- Railway switching booths need to be able to:
 - Report the status of the switches they are responsible for switching
 - Receive the instruction of turning on or off a certain switch.

Monitoring stations need to perform a complex set of features including pulling data about the system from the control server and pushing instructions to it.

This type of two-way communication can be done seamlessly over a combination of wireless and wired IP networks. Nodes themselves will be talking in TCP.

All the communication will be solely between each individual node and the control server. No other two nodes will be allowed to talk to each other for security reasons.

HARDWARE SYSTEM COMPONENTS

The GPS Receiver

A GPS receiver module will be used in our on-board computer system on each train for determining its location.

A typical GPS module outputs information in a standard NMEA format⁵, like this:

```
$GPGGA,174233.551,4003.8861,N,10512.5838,W,1,04,5.8,1593.0,M,
-20.7,M,,0000*51
$GPGSA,A,3,30,22,31,12,,,,,,,,,10.6,5.8,8.9*0B
$GPGSV,2,1,08,31,68,170,31,22,30,119,36,30,22,079,36,12,11,04
6,32*7D
$GPGSV,2,2,08,14,51,047,20,32,38,309,,11,28,285,,20,11,306,*7
3
$GPRMC,174233.551,A,4003.8861,N,10512.5838,W,0.07,84.92,20051
0,,,A*4A
$GPVTG,84.92,T,,M,0.07,N,0.1,K,A*3C
$GPGGA,174234.551,4003.8370,N,10512.6372,W,1,04,2.8,1285.8,M,
-20.7,M,,0000*54
$GPGSA,A,3,30,22,31,12,,,,,,,,,4.1,2.8,3.0*3C
$GPRMC,174234.551,A,4003.8370,N,10512.6372,W,0.62,352.44,2005
10,,,A*70
```

Each line of data is a self-contained NMEA sentence, independent from other sentences. The data convey the complete PVT (position, velocity, time) solution computed by the GPS receiver.

A NMEA sentence is a string of ASCII characters of 80 characters long maximum. Each sentence begins with a '\$' character and ends with a CR\NL sequence. The standard defines the first word of each sentence to start with the two letters GP followed by three more letters to determine the type of information conveyed in the rest of the sentence. Each sentence has its own interpretation and is decoded uniquely to extract the fix data.

The most important NMEA sentence includes the GPGGA which provides 3D location and accuracy data.

⁵ <http://www.gpsinformation.org/dale/nmea.htm>

If this configuration seems rather complex, the implementing engineers are welcome to replace this with any other underground positioning system that works and fulfils the requirements of reporting the latitude and longitude of each train underground at all time.

The TrackID Emitter

In order to identify the track upon which the train is moving, a close-range radio emitter is installed at each track segment as planned by the system configurer.

The unique TrackID will be broadcasted continuously for any passing train over it to pick it up.

The TrackID will be constant and should never be changed, so this device should be rather simple to build and install. The only condition it must adhere to is the uniqueness of the TrackID value which is discussed as a software design problem of the system.

The TrackID Receiver

One of these receivers should be installed on the bottom of each train. It should be connected to the on-board train computer and its only job is to read the unique TrackID value from a TrackID Emitter when the signal strength is past a certain calibrated threshold to guarantee that the train is right on that track and not passing on a track near it.

The Train Door Photo-sensor

A photo-sensor should be installed at a convenient position from the inside of each door so as to be able to tell when people stop going in and out.

All the sensors should be connected to the on-board train computer and let it know the status of each door in a simple binary signal that indicates either that people are still going in and out or not.

The Emergency Button

One of these emergency buttons should be placed conveniently next to every door. Pressing it would raise the emergency signal at the on-board computer and start the executing emergency procedure that is defined in the software.

The Train Controlling Actuators

The hardware system that would hack the train controls in order to execute incoming signals from the on-board train computer.

The actuators should work in two modes: BACKWARD and FORWARD. Each actuating input signal should be accompanied by one of the two modes signal.

The implementation of these actuators is irrelevant as long as it provides the following black-box functions of input and output signals:

- Input:
 - ACCELERATE: accelerates the train by 5km/h
 - DECELERATE: decelerates the train by 5km/h
 - BRAKE: apply the brake
 - OPEN_DOOR_RIGHT: open the doors on the right relative to the FORWARD direction
 - OPEN_DOOR_LEFT: open the doors on the left relative to the FORWARD direction

- Output:
 - ACK: acknowledges the successful completion of the last received instruction
 - SPEED: the speed of the train in km/h

The Train On-board Computer

The computer system controlling all the functions of a single train, it should receive all the signals from the following train components as inputs:

- The GPS Receiver
- The TrackID Receiver
- The UGPS Position Receiver
- Each Train Door Photo-sensor
- Each Emergency Button
- The actuator's ACK and SPEED signals

And generate the following signals as output:

- All the actuator's input signals

The system should have an active IP connection with the control server when it is booted up.

The software requirements of this system will be explained in the software components section.

The Railway Switching Hubs

These embedded computers will be scattered wherever there is a railway switch to be activated or deactivated. It will be very simply comprised of an actuator to realize the switching on or off, and whatever that is needed to make it connected to the same IP network that the control server is connected to.

The software requirements of this system will be explained in the software components section.

SOFTWARE DESIGN PROBLEMS

Choosing the Software Platform for Building the Control Server

As its name implies, the control server is going to be in charge of running the whole system. Every node in the system will only do its job upon direct instructions from the CS with minimal to no thinking on the executing side, so a lot of effort must go into making the software system running on the CS as reliable and scalable as possible.

Unlike on most of the other system components where embedded systems are forced to run on limited-resources, the CS software should not be crippled by these limitations; so more system resources can be traded off in favor for more reliability.

In the light of these considerations, our choice has settled on using the Java Virtual Machine as our software platform and the Java programming language for building the system.

Since that the Java Runtime Environment comes already packed with code for doing most of the utility functions like TCP/IP networking and cryptography, this will make us focus more easily on the design and the integration of the system itself.

The Structuring of a Railway Layout in a Computer Software Memory

In the spirit of the OOP philosophy which lies in the heart of the Java programming language, a sound model representation of the railway layout needed to be constructed first in order to have a full grasp on how can the system think about these components and how they react with each other.

The railway tracks were broken down into segments of 200m minimum length each that are laid out in sequence with some segments more privileged than others, that they get to switch between one of two possible end points.

The layout also includes such fixed entities as each train station and their locations, and some attributes that are useful in deducing the driving instructions that are instructed at trains.

In the rest of this subsection we introduce a break-down of the structure of these entities and how they are related to each other.

RailwayEntity

A RailwayEntity is the supertype of all the railway entities that are identified by a unique ID value.

Attributes:

- Id : UUID

Trackpoint

A Trackpoint is an entity comprising a location on a railway track. It is defined by its (latitude, longitude) pair.

Attributes:

- latitude : real numerical value
- longitude : real numerical value

Segment (inherits RailwayEntity)

This entity is used to represent a railway segment of 200m maximum length. Its beginning and end are identified by two TrackPoint entities.

Attributes:

- beginning : TrackPoint value
- end : TrackPoint value

ConjunctionSegment (inherits RailwayEntity)

This object represents a segment of a railway track that has one fixed beginning and can switch from one of two possible ends. The one beginning and the two possible ends are defined as three TrackPoint entities. One of the two ends could be eliminated to indicate that this railway track segment can either connect to a point or be a dead end.

Attributes:

- beginning : TrackPoint value
- primary_end : TrackPoint value
- alternative_end (optional) : TrackPoint value

Station (inherits RailwayEntity)

This entity represents one of the train stations of the system. A station is where trains usually make stops and people get in and out. Each station is defined by its own name and the track segments that it encloses. A station usually encloses two or more segments, each identified by their respective IDs.

Attributes:

- name : character string
- enclosed_segments : array of Segment

All these entities are implemented as Java classes and included in the railwayschematics module. Please see the chapter about it for more details.

The Generation of Unique Ids

Since that the control server is going to be dealing with a multitude of instances of the same type, representing many of the entities it is in control of, it needs a way of identifying each entity by a value that is unique to it.

During our mapping of the railway tracks, we have used the Universally Unique Identifier (UUID) standard to generate unique collision-free IDs for each entity.

A universally unique identifier (UUID) is an identifier standard used in software construction, standardized by the Open Software Foundation (OSF) as part of the Distributed Computing Environment (DCE).

The intent of UUIDs is to enable distributed systems to uniquely identify information without significant central coordination. In this context the word unique should be taken to mean "practically unique" rather than "guaranteed unique". Since the identifiers have a finite size it is possible for two differing items to share the same identifier. The identifier

size and generation process need to be selected so as to make this sufficiently improbable in practice. Anyone can create a UUID and use it to identify something with reasonable confidence that the same identifier will never be unintentionally created by anyone to identify something else. Information labeled with UUIDs can therefore be later combined into a single database without needing to resolve identifier (ID) conflicts.

UUIDs are documented as part of ISO/IEC 11578:1996 "Information technology – Open Systems Interconnection – Remote Procedure Call (RPC)" and more recently in ITU-T Rec. X.667 | ISO/IEC 9834-8:2005. The IETF has published Standards Track RFC 4122 that is technically equivalent with ITU-T Rec. X.667 | ISO/IEC 9834-8.

The Serialization of a Mapped-out Railway Layout

The structural mapping of the layout of the railway system need to be serialized in a form suitable for storing to a computer filesystem for loading back later, and for transferring across a data network.

For that, an XML schema was created to describe the entities comprising the railway layout and all their mapped-out respective attributes.

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards.⁶

An XML schema is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic syntactical constraints imposed by XML itself. These constraints are generally expressed using some combination of grammatical rules governing the order of elements, Boolean predicates that the content must satisfy, data types governing the content of

⁶ <https://en.wikipedia.org/wiki/XML>

elements and attributes, and more specialized rules such as uniqueness and referential integrity constraints.⁷

Our format schema is composed of a root node:

```
<schematic version='3'>  
...  
</schematic>
```

The root node encapsulates all the component definitions in of the layout. The attribute version represents the spec version just to alert the parser, and so far our spec has been bumped up till version 3 but could go higher before the project is actually finished.

The root node contains zero or one of each of the following nodes:

```
<segments>  
...  
</segments>  
  
<conjunctionsegments>  
...  
</conjunctionsegments>  
  
<stations>  
...  
</stations>
```

The <segments> node contains one or more of nodes that define a track segment each:

```
<segment id="xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx">  
  <beginning>  
    <latitude>[LATITUDE]</latitude>  
    <longitude>[LONGITUDE]</longitude>  
  </beginning>  
  <end>  
    <latitude>[LATITUDE]</latitude>  
    <longitude>[LONGITUDE]</longitude>  
  </end>  
</segment>
```

⁷ http://en.wikipedia.org/wiki/XML_schema

The <beginning> and <end> nodes each describe a geographical point; a geographical point is described by a decimal real value for latitude and another one for longitude.

The id attribute has the value of the unique schematic-scope id.

A conventional latitude or longitude value is represented in angular degrees, minutes and seconds. The decimal real value representation of that is computed by:

$$[decimal\ value] = [degrees] + [minutes] / 60 + [seconds] / 3600$$

This is the total value of angular degrees.

The <conjunctionsegments> node contains one or more nodes that define a conjunction segment each:

```
<conjunctionsegment>
  <beginning>
    <latitude>[LATITUDE]</latitude>
    <longitude>[LONGITUDE]</longitude>
  </beginning>
  <primaryend>
    <latitude>[LATITUDE]</latitude>
    <longitude>[LONGITUDE]</longitude>
  </primaryend>
  <alternativeend>
    <latitude>[LATITUDE]</latitude>
    <longitude>[LONGITUDE]</longitude>
  </alternativeend>
</conjunctionsegment>
```

The nodes <beginning>, <primaryend> and <alternativeend> represent the locations of the three ends of the ConjunctionSegment, each consisting of a latitude and longitude real decimal value as explained earlier.

The <primaryend> node is not required.

The <stations> node contains one or more nodes that describe a train station each:

```
<station id="xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx">
  <name>[STATION_NAME]</name>
  <segment id="xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx" />
  <segment id="xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx" />
  <segment id="xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx" />
</station>
```

The <name> node contains the textual name of the station.

There must exist one or more enclosed segments in the defined station, each segment is identified by its id attribute from one of the already defined segments.

The Painting of a Railway Layout on a Computer Terminal Screen

For our monitoring client system component, from which humans will be watching the status of the railway system closely and making sure everything is running smoothly, the system layout will have to be painted on a computer screen somehow with figures that represent each component in the system and its status.

For the same reasons which have lead us to choose the Java Standard Edition platform for building the control server software, we chose it for building our monitoring system software.

Among the rich set of functionality provided the JSE runtime environment, there's the excellent Java 2D API for producing advanced 2D graphics and images, encompassing line art, text, and images in a single comprehensive model. The API provides extensive support for image compositing and alpha channel images, a set of classes to provide accurate color space definition and conversion, and a rich set of display-oriented imaging operators.⁸

⁸<http://java.sun.com/products/java-media/2D/index.jsp>

Mapping Out the Railway Tracks

The process of actually mapping out the railway tracks is undefined in our project. As long as the final mapping output is in XML format and conforms to our XML Schema specification, it is fine to hack it out whichever way the system maintainer wishes.

The most accurate way is to traverse the real railway tracks while having a portable computer system with GPS receiving capabilities that stores location information every few moments configured relative to the traversing speed and then ironing out these locations to track segments that conform to our recommended length of 200m each, and configuring conjunction segments whenever is necessary.

However, for our demonstration purposes, we have built a simple WYSIWYG editor that creates these schematics and the system maintainer is free to use it. It has the capability of creating new schematics and editing already created ones. It could be used by the system maintainer as a final step in editing after performing the traversing method mentioned earlier.

Other Objects that are under the Control of the Control Server

The control server's controlling side is going to be in charge of guiding the trains and controlling the switching booths that control each ConjunctionSegment. So besides having the entire mapped-out railway layout structured in its memory, the control server is going to need object structure to represent trains and switching booths as well.

Train

An object of this type represents only one train that is several cars connected to two tank locomotives on each end. The train can move forwards and backwards, can have either of the doors on its side open or closed, and have an associated speed value in km/h.

Attributes:

- id : UUID

- doors_left_opened : boolean
- doors_right_opened : boolean
- speed : integer
- orientation : {BACKWARDS, FORWARDS}

SwitchingHub

A switching hub is a computer system connected to the control server via the IP network and to the hardware that can switch conjunction segments between their primary and alternative ends. It receives orders from the control server and actuates them.

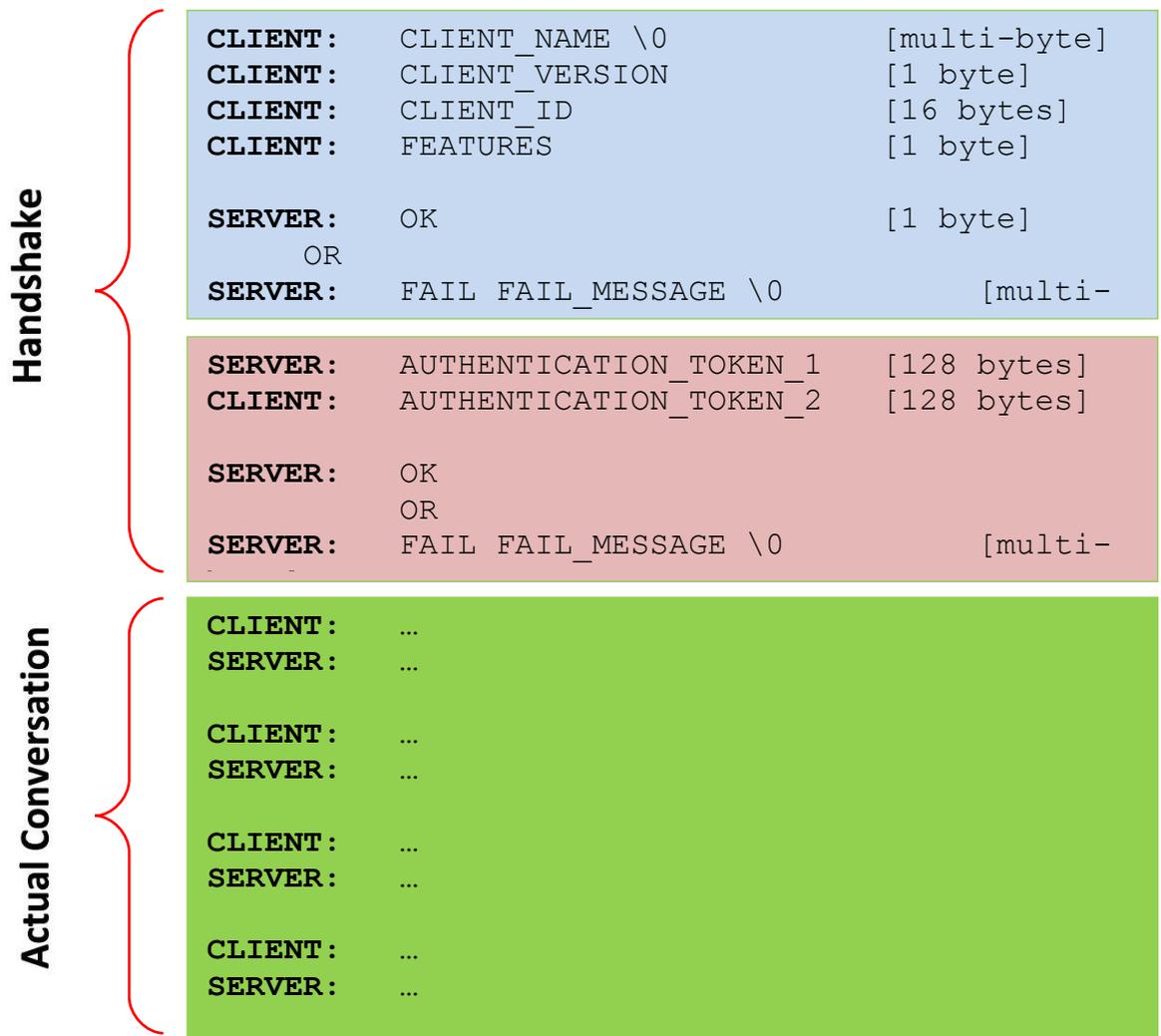
Attributes:

- id : UUID
- controlled_conjunction_segments : UUID[]

The General Communication Protocol between System Nodes and the Central Control Server

The communication between the system nodes and the control server are all going to be over TCP connections established between each node and the control server. TCP will provide the functionality of creating and terminating connections between the communicating nodes, and also provide reliable message delivery and integrity checks on behalf of them.

A typical communication scenario will be between two nodes with one them being The Control Server and thus we will call it SERVER from now on, the other node will be called CLIENT. The connection will always be initiated by CLIENT.



The protocol can be broken down into three separate blocks; the first two blocks comprise the initial mandatory handshake, followed by the actual communication stage which is basically a conversation between the server and the client node in a protocol that is client-specific, so we will go through each client-specific communication protocol on its own, later in this chapter.

The Handshake stage is divided into two sub-stages:

1. The Client Identification Stage:

In which the client identifies its type, its unique instance ID that was assigned to it by the system maintainer, along with the specific communication features that are requested from the server to comply to.

2. The Authentication Stage:

In which the client authenticates its identity to the server by using a mixture of public-key cryptography and cryptographically-strong hashing algorithms that are described later in this chapter.

The Client Identification Stage

This is the very first stage of communication between a system node and the control server. It begins by the system node identifying its type, protocol version, and its instance ID followed by the specific features requested in this communication session.

The control server will be expecting a CLIENT_NAME value indicating its type as a multi-byte character string terminated by a null character (ASCII code: 0).

The CLIENT_VERSION is an unsigned one-byte integer indicating the protocol revision that the system node uses. The value can range from 1 to 255. The zero value is not permitted as a CLIENT_VERSION.

The CLIENT_ID is the 16-byte UUID value representing the unique ID of that system node assigned to it by the system maintainer. The ID should be already in the control server's database associated with a record of that node.

The FEATURES value is one byte of 8 flags, designated as follows:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-----------------|-----------------|------------|---|---|---|---|---|
| CMPRS_SRVR_MSGS | CMPRS_CLNT_MSGS | ENCRYPTION | 0 | 0 | 0 | 0 | 0 |

CMPRS_SRVR_MSGS: If this bit is set then the messages sent from the control server are compressed using the DEFLATE algorithm, level 9 with no headers or checksum.

CMPRS_CLNT_MSGS: If this bit is set then the messages sent from the client are compressed using the DEFLATE algorithm, level 9 with no headers or checksum.

ENCRYPTION: If this bit is set then the messages from the control server and from the client node are both encrypted using a shared secret-key cipher that is explained later in the security section.

It should be noted however that not all these features are available to all the connecting nodes. Enabling a feature on a connection with a node that doesn't support it produces a FAIL message from the control server.

At the end of the Identification Stage, the server replies either with an OK message designated by a 0x01 byte or a FAIL message designated by a 0x00 byte followed by a null-terminated ASCII character string indicating some useful message that would help the system maintainer figure out what was going wrong. After the FAIL message the server would terminate the TCP connection. After an OK message the communication transitions on to the next stage.

The Authentication Stage is explained in the next section.

Securing the Communication between the Trains and the Central Control Server

Security is probably the most important aspect of designing this system. With going fully automatic, the risk of eavesdroppers, hijackers and interceptors increase significantly.

All the sensitive communications between our system nodes is done at the transport layer over a regular IP network. So the system maintainer is free to implement whatever she finds necessary at the Network Layer or the Data Link Layer.

Going the extra mile for extra protection, and assuming that no other means of security exist on the network, we've implemented security mechanisms deep within our communication protocol between the control server and each other system node.

Using a mixture of public-key and secret-key cryptographic algorithms we provide means for the system nodes to authenticate their identity, their messages and protect the confidentiality of the communications.

The Role of Public-key Cryptography

Our public-key cryptographic algorithm of choice is the tried-and-true RSA. Every communicating node on the system has its own 1024-bit public/private key pair that is generated by the system maintainer upon installation.

Each system node other than the control server only knows its own private key and the public key of the control server. The control server however knows the public key of every node on the system plus its own private key.

All the public keys should be distributed discretely by the system maintainer using methods of physical storage media for maximum security, since that these keys will be in use for probably many years to come, treating them as secret keys is best for minimizing the risks of brute-force attacks on the system.

The public-key cryptosystem in place here will provide means guaranteeing the authenticity of each node.

The Role of Secret-key Cryptography

We can't do away with asymmetric ciphers for everything, since they are relatively slow to encrypt and decrypt messages. We use the asymmetric cryptosystem – like everyone else does – to securely exchange symmetric cipher keys that last for the duration of that one connection.

We use the Blowfish algorithm as our symmetric cipher with a 128-bit key, the standard 64-bit block size, and the CFB8 mode that will effectively turn this block cipher into an 8-bit self-synchronizing stream cipher. The Cipher Feedback Mode needs to be initialized with a 64-bit initialization vector value that is shared between the two parties.

The Authentication Stage in the General Communication Protocol

The Authentication Stage is where the client node authenticates its identity and exchanges the secret-key and the initialization vector needed for the symmetric ciphering between the two communicating parties.

The control server generates a random 192-bit (24-byte) chunk of data using a pseudo-random number generator, that we'll call RAW_TOKEN. The RAW_TOKEN is then encrypted with the client's public key, and the resulting value is the 128 byte (1024 bit) AUTH_TOKEN_1 that gets sent to the client.

At the client side, after receiving the AUTH_TOKEN_1, it is then decrypted using the client's private key into the original 24-byte RAW_TOKEN.

Still at the client side, the RAW_TOKEN is then hashed using the SHA1 algorithm into a digest of 16 bytes, and the hash digest is then encrypted with the client's private key and is sent back to the server as AUTH_TOKEN_2.

At the server, the SHA1 hash digest of the RAW_TOKEN is computed and is then compared with the value that is received from the client (AUTH_TOKEN_2) after decrypting it with the client's public key.

The server then decides if the identity of the client is authenticated or not based on the result of that comparison. If both values are equal, then the Authentication Stage is a success and the server replies to the client with a 0x01 byte. If not, then the

Authentication Stage has failed and the server then responds with a 0x00 byte followed by a null-terminated ASCII character string explaining the problem to the client.

Assuming success, the two parties now have authenticated the identity of each other given that only they possess their respective private keys.

If ENCRYPTION was requested using the FEATURES byte, and encryption is enabled for communication with this client type, then both parties should hold on to that 24-byte RAW_TOKEN because it will be used as the 128-bit (16 bytes) symmetric session key and the 64-bit (8 bytes) initial control vector for the symmetric cipher, in that order.

The Conversation Communication Stage

This is whatever data that gets exchanged between the client and the server after the Handshake is complete. The data is exchanged in the form of messages, and their structure should be known to both parties as part of the client-specific communication protocol.

The Encrypted Message Format

Encryption will not add any overhead to exchanged messages, so it's not considered a message format on its own, but rather a more primitive layer below the messages layer that handles the exchange of individual bytes. Sent and received bytes will be encrypted and decrypted 1-byte at a time using the Blowfish cipher in CFB8 mode and sent as-is.

At the client side, two symmetric cipher instances should be instantiated, both with the same secret-key and initialization vector and the agreed-upon mode. One cipher instance will be used for encrypting and the other for decrypting. At the server-side, the same thing is achieved.

The Compressed Communication Message Format

If compression is requested on either side of the conversation, then the messages sent from the side with the compression enabled will be in the following format.

| LEAD | LNTH_SIZE | | CMPRSD_LNGTH | NRMAL_LNGTH | CONTENT |
|------------|-----------|-----|--------------|-------------|---------|
| 0x0F, 0xF0 | 0x0 | 0x0 | | | |

The message starts with the LEAD byte; its value is either 0x0F or 0xF0. 0x0F means that CONTENT is not actually compressed for some reason, probably because it's too short and would yield a longer message if compressed. 0xF0 means that CONTENT is compressed.

The following LNTH_SIZE byte is split in half, the most significant four bits represents the number of bytes in the CMPRSD_LNGTH value, and the least significant four bits represent the number of bytes in the NRMAL_LNGTH.

The values CMPRSD_LNGTH and NRMAL_LNGTH are each multi-byte unsigned integers in big-endian order.

If the value of LEAD indicates that CONTENT is compressed, then the length of CONTENT in bytes is the value of CMPRSD_LNGTH. After it is decompressed, the length of the uncompressed CONTENT should be that of NRMAL_LNGTH.

If the value of LEAD indicates that CONTENT is not compressed, then the length of CONTENT in bytes is the value of NRMAL_LNGTH. The four most significant bits in LNTH_SIZE then should be zeros and the value of CMPRSD_LNGTH should not be expected.

The Variable-Length Message Format

This format is used when each message is arbitrarily-sized and compression is not requested

Each message is structured as follows:

| LENGTH_SIZE | LENGTH | CONTENT |
|-------------|--------|---------|
| | | |

The LENGTH_SIZE is a 1-byte unsigned integer that indicates the size of the LENGTH value.

The LENGTH is a multi-byte big-endian unsigned integer which expresses the length of the CONTENT value in bytes.

The CONTENT is the actual message data.

The Fixed-Length Message Format

This is the most simple message format because it comprises no headers or footers or any overhead to the data. Instead, the two parties must be expecting messages in certain sizes based on turn or on the expected response to the last sent message.

The Communication Protocol with a Train Client

| Server-side Compression | Client-side Compression | Encryption |
|-------------------------------|-----------------------------|------------|
| No | No | Available |
| CLIENT_NAME | Trainclient | |
| CLIENT_VERSION | 1 | |
| Server Message Format: | Fixed-Length Message Format | |
| Client Message Format: | Fixed-Length Message Format | |

This is the description of the protocol of communication with a Train client. It has been taken into consideration that this type of communication will be relayed by wireless means mostly on the count of the moving trains, and these connections tend to be slow and flaky, so the messages were designed to be as brief as possible.

Since the number of possible messages is small, a Fixed-Length Message format is sufficient for the exchange of data between the two nodes.

Client Messages

The client only sends two kinds of messages; the first is a fixed-length status report of 33-byte length, formatted as follows:

| | | | | |
|-----------|--------|-------|----------|--------------|
| TIMESTAMP | STATUS | SPEED | LOCATION | LAST_TRACKID |
|-----------|--------|-------|----------|--------------|

The `TIMESTAMP` value is a 4-byte signed integer comprising the Unix timestamp of the second the message is packed together.

The `STATUS` value is 1 byte of 8 bit-flags:

| | | | | | | | |
|-----------------|------------------|-----------|---------|---|---|---|---|
| LEFT_DOORS_OPEN | RIGHT_DOORS_OPEN | EMERGENCY | FORWARD | X | X | X | X |
|-----------------|------------------|-----------|---------|---|---|---|---|

The `LEFT_DOORS_OPEN` and `RIGHT_DOORS_OPEN` are set when the left doors or the right doors are open, respectively. The `EMERGENCY` bit is set when someone on the train had pushed on an emergency button. The `FORWARD` bit is set when the train is being driven by the locomotive labeled as the forward one, it is reset when the train is on the backward mode. The least significant four bits are not used.

The `SPEED` value is the train's speed in km/h represented as a 4-byte big-endian IEEE-754 floating point number.

The `LOCATION` is the latitude value followed by the longitude value, each represented as a big-endian IEEE-754 floating point number.

The `LAST_TRACKID` is the 16-byte UUID value of the `LastTrackID` stored on the train's memory.

The second kind of message is an acknowledgement consisting of either 1-byte or a multi-byte message:

| Name | Value | Size |
|-------------------|-------------------|--------------|
| OK | 0x01 | [1 byte] |
| FAIL FAIL_MESSAGE | 0x00 FAIL_MESSAGE | [multi-byte] |

The OK message indicates that the control instruction received has been executed successfully.

The FAIL message indicates that the control instruction received has failed to executed. It should be followed by a null-terminated character string message that indicates the problem.

Server Messages

The server sends only control messages that are either 5 or 9 bytes long. The client should be able to tell by the leading byte the length of the message. Each control message is constructed from a 4-byte TIMESTAMP comprising the big-endian unsigned integer Unix timestamp followed by the 1 or 5-byte instruction.

| Name | Value | Size |
|-------------------------|--------------------|-----------|
| BRAKE | 0x01 | [1 byte] |
| ACCELERATE_BY_1 | 0x02 | [1 byte] |
| ACCELERATE_BY_2 | 0x03 | [1 byte] |
| ACCELERATE_BY_3 | 0x04 | [1 byte] |
| ACCELERATE_BY_4 | 0x05 | [1 byte] |
| ACCELERATE_BY_5 | 0x06 | [1 byte] |
| DECELERATE_BY_1 | 0x07 | [1 byte] |
| DECELERATE_BY_2 | 0x08 | [1 byte] |
| DECELERATE_BY_3 | 0x09 | [1 byte] |
| DECELERATE_BY_4 | 0x0A | [1 byte] |
| DECELERATE_BY_5 | 0x0B | [1 byte] |
| SET_SPEED [SPEED_VALUE] | 0x0C [SPEED_VALUE] | [5 bytes] |
| OPEN_DOORS_LEFT | 0x0D | [1 byte] |
| OPEN_DOORS_RIGHT | 0x0E | [1 byte] |
| CLOSE_DOORS_LEFT | 0x0F | [1 byte] |
| CLOSE_DOORS_RIGHT | 0x10 | [1 byte] |

The BRAKE message instructs the client to apply the brakes.

The ACCELERATE_BY_* messages instruct the client to accelerate by the specified amount in the message name, in km/h.

The DECELERATE_BY_* messages work just like the ACCELERATE_BY_* only it instructs to decelerate the speed in km/h.

The SET_SPEED message is one instruction byte followed by 4-bytes bearing a big-endian IEEE 754 floating point number. The message instructs the client to change the speed to the floating point value.

The OPEN_DOORS_RIGHT, OPEN_DOORS_LEFT, CLOSE_DOORS_RIGHT, CLOSE_DOORS_LEFT messages instruct the client to open or close the right or left doors of the train. The right and left doors are designated as such according to the forward orientation of the train.

The Communication Protocol with a Switching Hub

| | | |
|--------------------------------|--------------------------------|-------------------|
| Server-side Compression | Client-side Compression | Encryption |
| No | No | Available |
| CLIENT_NAME | switchinghub | |
| CLIENT_VERSION | 1 | |
| Server Message Format: | Fixed-Length Message Format | |
| Client Message Format: | Fixed-Length Message Format | |

This is a very simple fixed-length protocol with only two server messages and two acknowledgement client messages.

Server Messages

| Name | Value | Size |
|--------------------------|---------|-----------|
| SWITCH_TO_PRIMARY ID | 0x00 ID | [17 byte] |
| SWITCH_TO_ALTERNATIVE ID | 0x01 ID | [17 byte] |

The SWITCH_TO_PRIMARY message instructs the hub to switch the conjunction segment identified by the id ID to its designated primary end.

The SWITCH_TO_ALTERNATIVE message does the same only with the alternative end.

Client Messages

| Name | Value | Size |
|-------------------|-------------------|--------------|
| OK | 0x01 | [1 byte] |
| FAIL FAIL_MESSAGE | 0x00 FAIL_MESSAGE | [multi-byte] |

The OK message indicates that the control instruction received has been executed successfully.

The FAIL message indicates that the control instruction received has failed to be executed. It should be followed by a null-terminated character string message that indicates the problem.

The Communication Protocol with a Monitoring Station

| Server-side Compression | Client-side Compression | Encryption |
|-------------------------------|--|------------|
| Available | Available | Available |
| CLIENT_NAME | monitor | |
| CLIENT_VERSION | 1 | |
| Server Message Format: | Variable-Length Message Format/Compressed Message Format | |
| Client Message Format: | Variable-Length Message Format/Compressed Message Format | |

This is the most complicated communication protocol. It consists of variable-size messages between the two nodes.

If compression is used then the message originating from the node with the enabled compression should bear the message format of compressed messages, as described earlier.

The communication process is text oriented, and is always in the form of a request from the client followed by a response from the server.

Request/Response Message Pairs

The messages from the client are in the form of method names and optional arguments all separated by the space character, all encoded in 8-bit ASCII.

The responses are always XML documents encoded in UTF-8.

QUERY_SCHEMATIC

Response: The schematic of the railway system that the control server is in charge of

QUERY_TRAINS

Response: A sequence of train the train objects on the server side

QUERY_CHANGES

Response: A sequence of the different objects that have had their attributes changed since the client made a connection with the server. These include Train and ConjunctionSegment objects.

The Communication Protocol with a Controlling Station

| Server-side Compression | Client-side Compression | Encryption |
|-------------------------------|--|------------|
| Available | Available | Available |
| CLIENT_NAME | controller | |
| CLIENT_VERSION | 1 | |
| Server Message Format: | Variable-Length Message Format/Compressed Message Format | |
| Client Message Format: | Variable-Length Message Format/Compressed Message Format | |

This is the exact same protocol as the monitoring station, only it provides different requests to the server side that should require different authorization keys than that needed for just monitoring since they produce effects on the state of the railway system.

You should note that all these actions override the automatic control algorithms of the control server, so they should only be used when absolutely necessary.

Request/Response Message Pairs

TRAIN_SET_SPEED TRAIN_ID TRAIN_SPEED

Response: An acknowledgement message.

Arguments:

- train_id: the id of the train
- train_speed: the desired speed in km/h

Effect: Changes the speed of the desired train to the specified value, overriding the automatic control algorithms.

TRAIN_OPEN_DOORS_LEFT TRAIN_ID

Response: An acknowledgement message.

Arguments:

- train_id: the id of the train

Effect: Opens the left doors of the train.

TRAIN_OPEN_DOORS_RIGHT TRAIN_ID

Response: An acknowledgement message.

Arguments:

- train_id: the id of the train

Effect: Opens the right doors of the train.

TRAIN_CLOSE_DOORS_LEFT TRAIN_ID

Response: An acknowledgement message.

Arguments:

- train_id: the id of the train

Effect: Closes the left doors of the train.

TRAIN_CLOSE_DOORS_RIGHT TRAIN_ID

Response: An acknowledgement message.

Arguments:

- train_id: the id of the train

Effect: Closes the right doors of the train.

TRAIN_SET_ORIENTATION TRAIN_ID ORIENTATION

Response: An acknowledgement message.

Arguments:

- train_id: the id of the train
- orientation: {forward | backward}

Effect: Switches the orientation of the train to the specified value and stops its movement until started up again.

TRAIN_BRAKE TRAIN_ID

Response: An acknowledgement message.

Arguments:

- train_id: the id of the train

Effect: Applies the brakes on the specified train. Figuring Out the Routes that Each Individual Train Should Take

SOFTWARE SYSTEM COMPONENTS

The Control Server Software

Objectives

- Provides the main functionality of our project, which is the autonomous control of the railway system entities like the trains and the switchable tracks
- Provides real time information about the status of the railway system

Platform

Runs on any computer capable of running the JRE7, on top of a Java Virtual Machine

Connectivity

Needs to be connected to the same IP network that is connecting all the other system computer nodes

Human Interface

Does not provide a direct human interface, but its operation can be observed through performance logs and it can be ordered to execute commands using a controller machine

Installation and Usage

Check the manual that comes with the software.

The Monitoring/Controlling Software

Objectives

- Provides real time information about the status of the railway system to the observing humans
- Provides means for emergency remote controlling the system entities

Platform

Runs on any computer capable of running the JRE7, on top of a Java Virtual Machine

Connectivity

Needs to be connected to the same IP network that is connecting all the other system computer nodes

Human Interface

- Provides graphical diagrams representing the status of the railway system
- Provides GUI means of controlling the controllable entities of the system

Installation and Usage

Check the manual that comes with the software.

The Schematic Editor

Objectives

- Provides a way for editing railway schematic maps in a WYSIWYG manner

Platform

Runs on any computer capable of running the JRE7, on top of a Java Virtual Machine

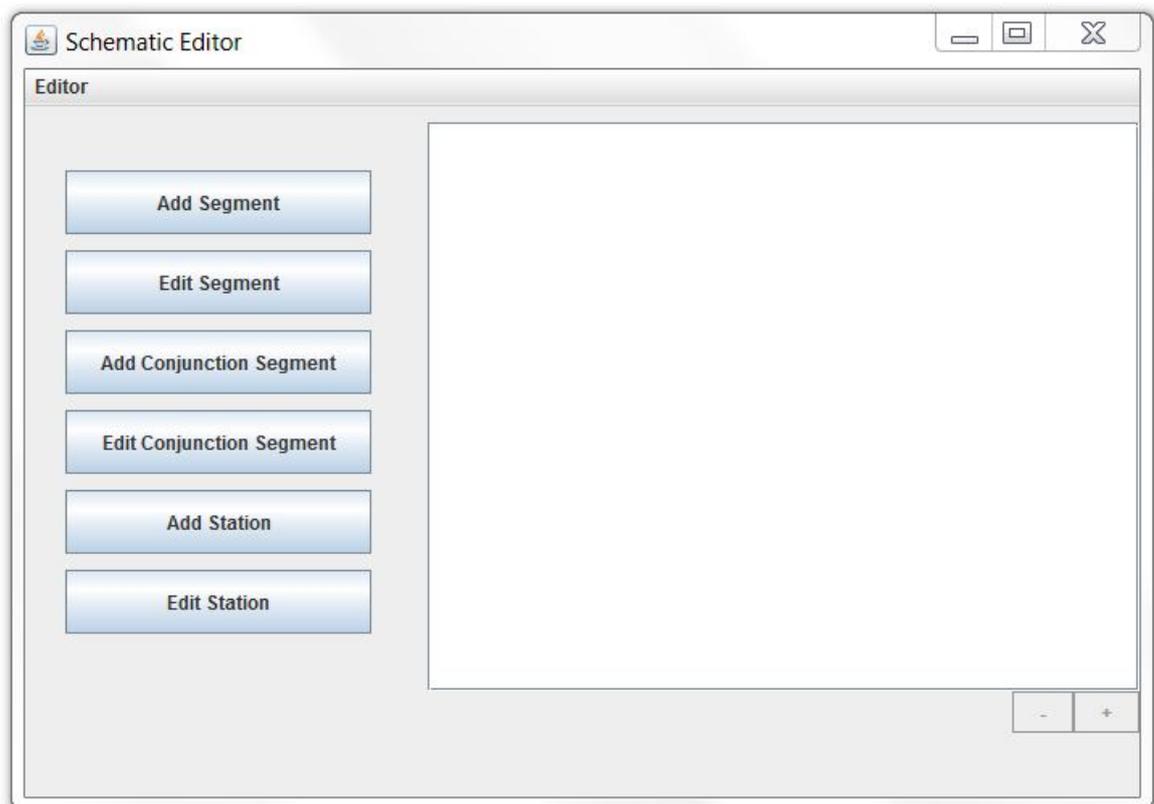
Connectivity

Not required

Human Interface

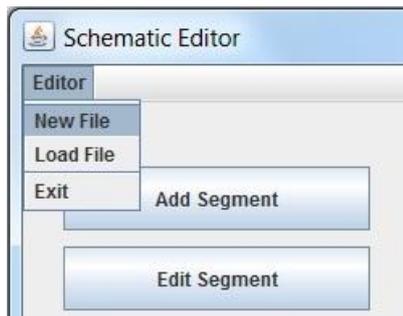
- Provides graphical diagrams representing the structure of the railway system
- Provides graphical means of editing the railway schematic

Installation and Usage

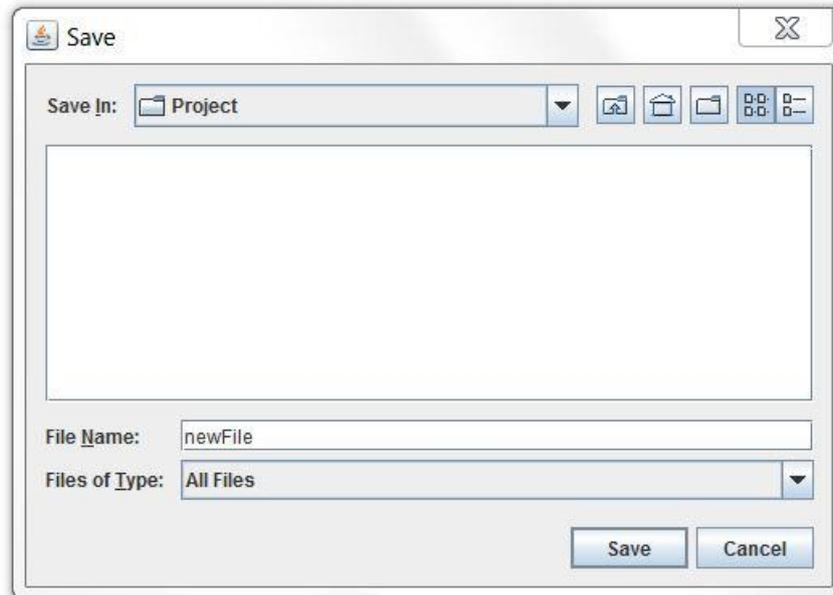


Creating a New Schematic File

1. Click on “Editor” and select “New File”.



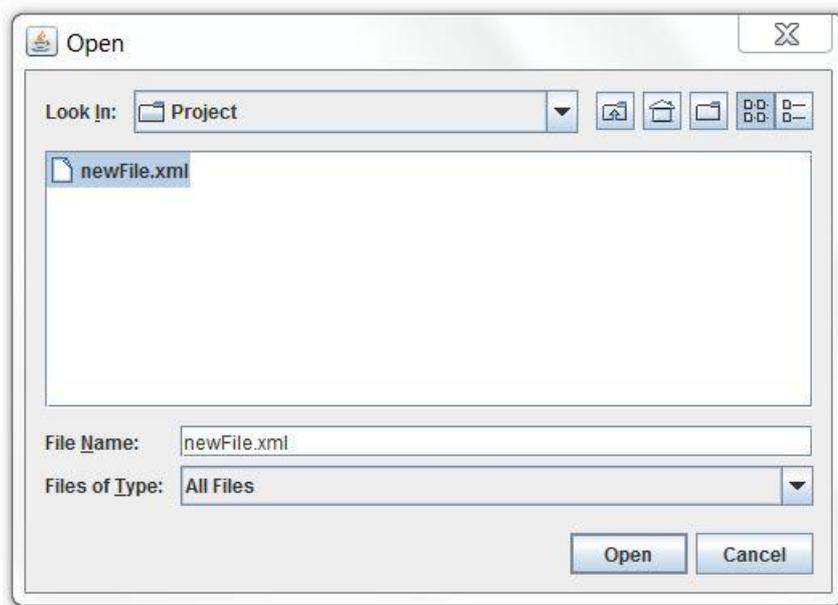
2. Browse for the directory to save the new file and enter file name.



3. Click “Save”.

Loading a Schematic

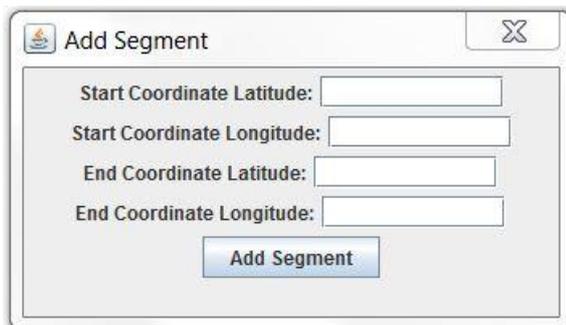
1. Click on “Editor” and select “Load File”.



2. Browse for the file, select it and click “Open”.

Adding a New Segment

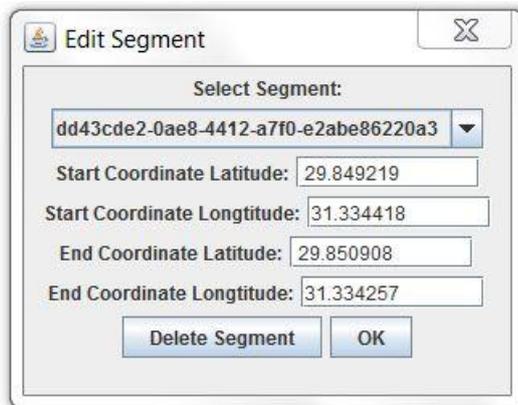
1. Select “Add Segment” from the editor options.
2. Enter the new segment's data:



3. Click “Add Segment” to save the new segment.

Editing an Existing Segment

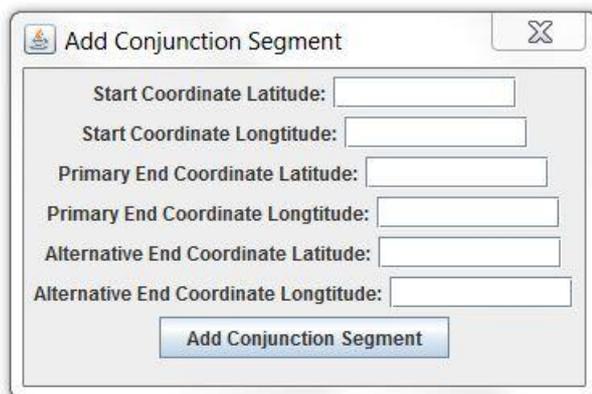
1. Select "Edit Segment" from the editor options



2. Select the segment ID of the segment you want to edit from the drop down list.
3. Edit the segment's data then click "OK" to save the segment with the updated data.
4. To delete the segment from the system, click "Delete Segment".

Adding a New Conjunction Segment

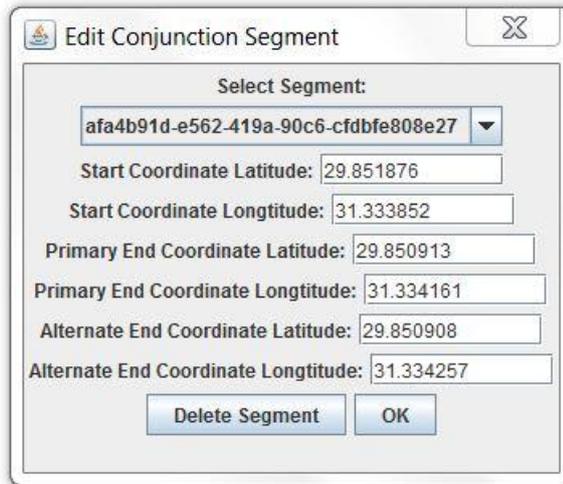
1. Select "Add Conjunction Segment" from the editor options.
2. Enter the new segment's data.



3. Click "Add Conjunction Segment" to save the new conjunction segment.

Editing an Existing Conjunction Segment

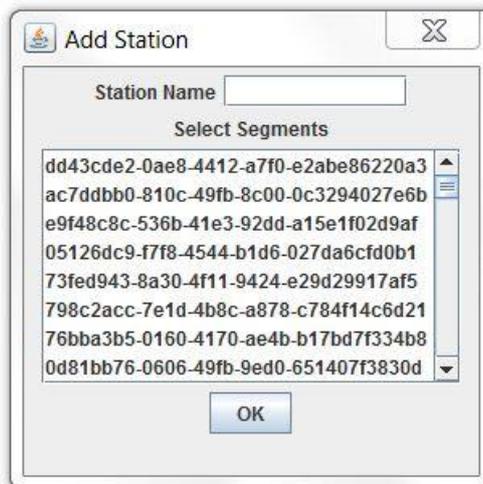
1. Select “Edit Conjunction Segment” from the editor options.



2. Select the segment ID of the conjunction segment you want to edit from the drop down list.
3. Edit the segment's data then click “OK” to save the segment with the updated data.
4. To delete the segment from the system, click “Delete Segment”.

Adding a New Station

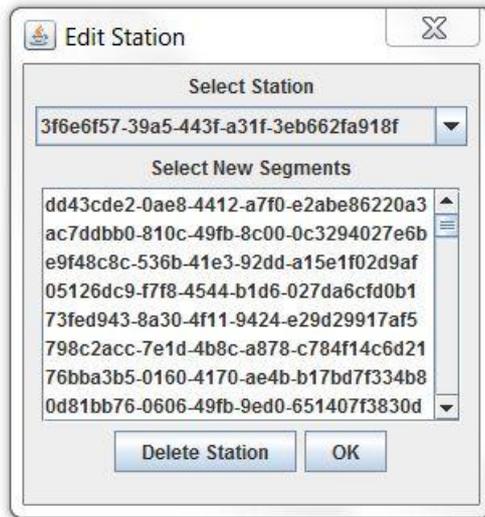
1. Select “Add Station” from the editor options.



2. Enter station name.
3. Select the segments that cover the stations from the segments list.
4. Click “OK” to save the new station to the schematic.

Editing an Existing Station

1. Select "Edit Station" from the editor options.



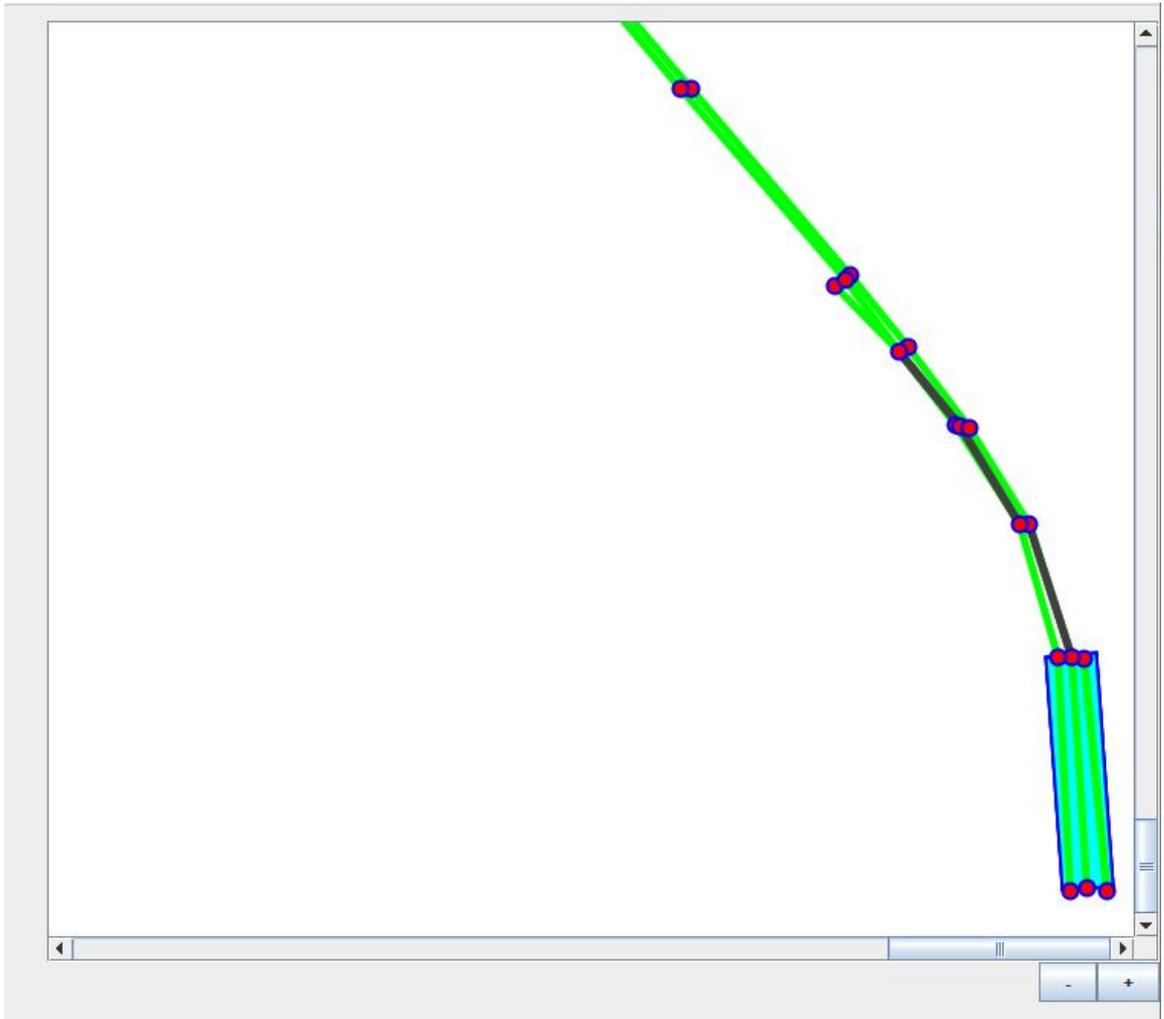
2. Select the station ID of the station you want to edit from the drop down list.
3. Select the new segments that cover the station.
4. Click "OK" to save the new segments to the station.
5. To delete the selected segment, click "Delete Segment".

Viewing Area

The viewing area will display the map of the schematic system using the data entered using the schematic editor.

- Segments are represented by Green Lines.
- Track Points are represented by Red Dots.
- Conjunction Segments are represented by Black Lines.
- Stations are represented by Blue Rectangles parallel to the segments that cover them.

The user can zoom in and out in the viewing area for a more clear view of the railway tracks using the "+" & "-" buttons.



Sample from a Railway Schematic Map

TECHNOLOGIES USED

The Java Runtime Environment

Java is a programming language and computing platform first released by Sun Microsystems in 1995. It is the underlying technology that powers state-of-the-art programs including utilities, games, and business applications. Java runs on more than 850 million personal computers worldwide, and on billions of devices worldwide, including mobile and TV devices. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java is as of 2012 one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users.

The Java Runtime Environment (JRE) is what you get when you download Java software. The JRE consists of the Java Virtual Machine (JVM), Java platform core classes, and supporting Java platform libraries. The JRE is the runtime portion of Java software, which is all you need to run it in your Web browser. When you download Java software, you only get what you need - no spyware, and no viruses.

Java 2D

The Java 2D API is a set of classes for advanced 2D graphics and imaging, encompassing line art, text, and images in a single comprehensive model. The API provides extensive support for image compositing and alpha channel images, a set of classes to provide accurate color space definition and conversion, and a rich set of display-oriented imaging operators.

Basic Concepts

These objects are a necessary part of every Java 2D drawing operation.

Shapes

A *shape* in Java 2D is a boundary which defines an inside and an outside. Pixels inside the shape are affected by the drawing operation, those outside are not.

Trying to fill a straight line segment will result in no pixels being affected, as such a shape does not contain any pixels itself. Instead, a thin rectangle must be used so that the shape contains some pixels.

Paints

A *paint* generates the colors to be used for each pixel of the fill operation. The simplest paint is **java.awt.Color**, which generates the same color for all pixels. More complicated paints may produce gradients, images, or indeed any combination of colors. Filling a circular shape using the color yellow results in a solid yellow circle, while filling the same circular shape using a paint that generates an image produces a circular cutout of the image.

Composites

During any drawing operation, there is a *source* (the pixels being produced by the paint) and a *destination* (the pixels already onscreen). Normally, the source pixels simply overwrite the destination pixels, but the *composite* allows this behavior to be changed.

The composite, given the source and destination pixels, produces the final result that ultimately ends up onscreen. The most common composite is **java.awt.AlphaComposite**, which can treat the pixels being drawn as partially transparent, so that the destination pixels show through to some degree.

Filling

To *fill* a shape, the first step is to identify which pixels fall inside the shape. These pixels will be affected by the fill operation. Pixels that are partially inside and partially outside the shape may be affected to a lesser degree if anti-aliasing is enabled.

The paint is then asked to generate a color for each of the pixels to be painted. In the common case of a solid-color fill, each pixel will be set to the same color.

The composite takes the pixels generated by the paint and combines them with the pixels already onscreen to produce the final result.

Advanced Objects

These objects can be viewed as performing their duties in terms of the simpler objects described above.

Transform

Every Java 2D operation is subject to a transform, so that shapes may be translated, rotated, sheared, and scaled as they are drawn. The active transform is most often the **identity transform**, which does nothing.

Filling using a transform can be viewed as simply creating a new, transformed shape and then filling that shape.

Stroke

In addition to the *fill* operation, Java 2D provides a *draw* operation. While fill draws the interior of a shape, draw draws its outline. The outline can be as simple as a thin line, or as complicated as a dashed line with each dash having rounded edges.

The object responsible for generating the outline is the *stroke*. Given an input shape, the stroke produces a new shape representing its outline. For instance, an infinitely thin line segment (with no interior) might be stroked into a one-pixel-wide rectangle.

A draw operation can therefore be described as creating a new, stroked object and then filling that object.

Technically speaking, the stroke is only required to accept an input shape and produce a new shape. The stroke implementation provided with Java 2D implements the outline rules described above, but a custom-written stroke could produce any shape it wished.

GPS

The **Global Positioning System (GPS)** is a space-based satellite navigation system that provides location and time information in all weather, anywhere on or near the Earth, where there is an unobstructed line of sight to four or more GPS satellites. It is maintained by the United States government and is freely accessible to anyone with a GPS receiver.

The GPS program provides critical capabilities to military, civil and commercial users around the world. In addition, GPS is the backbone for modernizing the global air traffic system.

The GPS project was developed in 1973 to overcome the limitations of previous navigation systems, integrating ideas from several predecessors, including a number of classified engineering design studies from the 1960s. GPS was created and realized by the U.S. Department of Defense (DoD) and was originally run with 24 satellites. It became fully operational in 1994.

Advances in technology and new demands on the existing system have now led to efforts to modernize the GPS system and implement the next generation of GPS III satellites and Next Generation Operational Control System (OCX). Announcements from the Vice President and the White House in 1998 initiated these changes. In 2000, U.S. Congress authorized the modernization effort, referred to as GPS III.

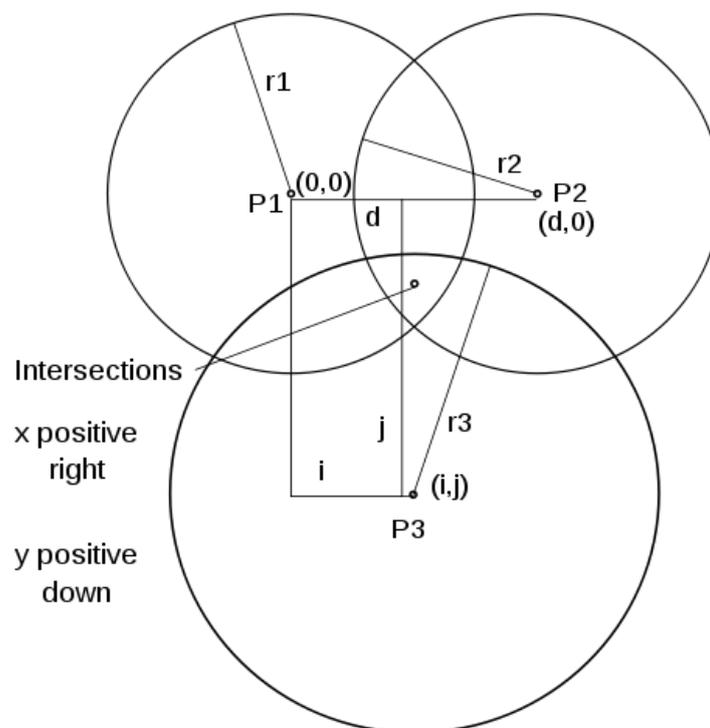
Trilateration

In geometry, **trilateration** is the process of determination absolute or relative locations of points by measurement of distances, using the geometry of circles, spheres or triangles.

In addition to its interest as a geometric problem, trilateration does have practical applications in surveying and navigation, including global positioning systems (GPS). In contrast to triangulation it does not involve the measurement of angles.

In two-dimensional geometry, when it is known that a point lies on two curves such as the boundaries of two circles then the circle centers and the two radii provide sufficient information to narrow the possible locations down to two. Additional information may narrow the possibilities down to one unique location.

In three-dimensional geometry, when it is known that a point lies on three surfaces such as the surfaces of three spheres then the centers of the three spheres along with their radii provide sufficient information to narrow the possible locations down to no more than two. If it is known that the point lies on the surface of a fourth sphere then knowledge of this sphere's center along with its radius is sufficient to determine the one unique location.



The plane, $z=0$, showing the 3 sphere centers, P1, P2, and P3; their (x, y) coordinates; and the 3 sphere radii, r_1 , r_2 , and r_3 . The two intersections of the three sphere surfaces are directly in front and directly behind the point designated intersections in the $z = 0$ plane.

Photoelectric Sensors

A **photoelectric sensor**, or photoeye, is a device used to detect the distance, absence, or presence of an object by using a light transmitter, often infrared, and a photoelectric receiver. They are used extensively in industrial manufacturing. There are three different functional types: opposed (through beam), retroreflective, and proximity-sensing (diffused).

Types

A self-contained photoelectric sensor contains the optics, along with the electronics. It requires only a power source. The sensor performs its own modulation, demodulation, amplification, and output switching. Some self-contained sensors provide such options as built-in control timers or counters. Because of technological progress, self-contained photoelectric sensors have become increasingly smaller.

Remote photoelectric sensors used for remote sensing contain only the optical components of a sensor. The circuitry for power input, amplification, and output switching are located elsewhere, typically in a control panel. This allows the sensor, itself, to be very small. Also, the controls for the sensor are more accessible, since they may be bigger.

When space is restricted or the environment too hostile even for remote sensors, fiber optics may be used. Fiber optics are passive mechanical sensing components. They may be used with either remote or self-contained sensors. They have no electrical circuitry and no moving parts, and can safely pipe light into and out of hostile environments.

Sensing modes

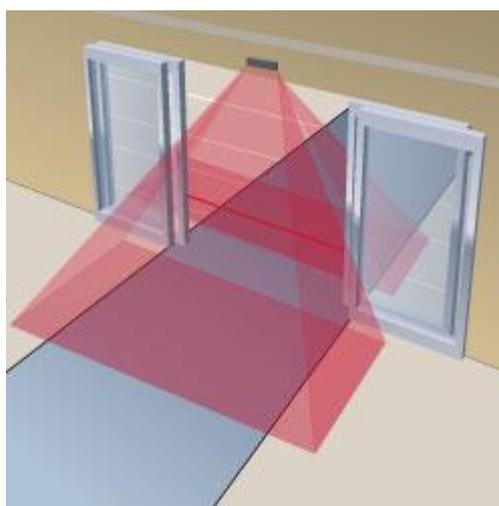
An opposed (through beam) arrangement consists of a receiver located within the line-of-sight of the transmitter. In this mode, an object is detected when the light beam is blocked from getting to the receiver from the transmitter.

A retroreflective arrangement places the transmitter and receiver at the same location and uses a reflector to bounce the light beam back from the transmitter to the receiver. An object is sensed when the beam is interrupted and fails to reach the receiver.

A proximity-sensing (diffused) arrangement is one in which the transmitted radiation must reflect off the object in order to reach the receiver. In this mode, an object is detected when the receiver sees the transmitted source rather than when it fails to see it.

Some photoeyes have two different operational types, light operate and dark operate. Light operate photoeyes become operational when the receiver "receives" the transmitter signal. Dark operate photoeyes become operational when the receiver "does not receive" the transmitter signal.

The detecting range of a photoelectric sensor is its "field of view", or the maximum distance the sensor can retrieve information from, minus the minimum distance. A minimum detectable object is the smallest object the sensor can detect. More accurate sensors can often have minimum detectable objects of minuscule size.



Illustrating image of photoelectric sensor that detect if there is an object in front of it or not.

Transmission Control Protocol (TCP)

The **Transmission Control Protocol (TCP)** is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite, complementing the Internet Protocol (IP), and therefore the entire suite is commonly referred to as *TCP/IP*. TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. TCP is the protocol used by major Internet applications such as the World Wide Web, email, remote administration and file transfer. Other applications, which do not require reliable data stream service, may use the User Datagram Protocol (UDP), which provides a datagram service that emphasizes reduced latency over reliability.

Network function

The protocol corresponds to the transport layer of TCP/IP suite. TCP provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). That is, when an application program desires to send a large chunk of data across the Internet using IP, instead of breaking the data into IP-sized pieces and issuing a series of IP requests, the software can issue a single request to TCP and let TCP handle the IP details.

IP works by exchanging pieces of information called packets. A packet is a sequence of octets and consists of a *header* followed by a *body*. The header describes the packet's destination and, optionally, the routers to use for forwarding until it arrives at its destination. The body contains the data IP is transmitting.

Due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets can be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data, and even helps minimize network congestion to reduce the occurrence of the other problems. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the application program. Thus, TCP abstracts the application's communication from the underlying networking details.

TCP is utilized extensively by many of the Internet's most popular applications, including the World Wide Web (WWW), E-mail, File Transfer Protocol, Secure Shell, peer-to-peer file sharing, and some streaming media applications.

TCP is optimized for accurate delivery rather than timely delivery, and therefore, TCP sometimes incurs relatively long delays (in the order of seconds) while waiting for out-of-order messages or retransmissions of lost messages. It is not particularly suitable for real-time applications such as Voice over IP. For such applications, protocols like the Real-time Transport Protocol (RTP) running over the User Datagram Protocol (UDP) are usually recommended instead.

TCP is a reliable stream delivery service that guarantees that all bytes received will be identical with bytes sent and in the correct order. Since packet transfer is not reliable, a technique known as positive acknowledgment with retransmission is used to guarantee reliability of packet transfers. This fundamental technique requires the receiver to respond with an acknowledgment message as it receives the data. The sender keeps a record of each packet it sends. The sender also keeps a timer from when the packet was sent, and retransmits a packet if the timer expires before the message has been acknowledged. The timer is needed in case a packet gets lost or corrupted.

TCP consists of a set of rules: for the protocol, that are used with the Internet Protocol, and for the IP, to send data "in a form of message units" between computers over the Internet. While IP handles actual delivery of the data, TCP keeps track of the individual units of data transmission, called *segments*, that a message is divided into for efficient routing through the network. For example, when an HTML file is sent from a Web server, the TCP software layer of that server divides the sequence of octets of the file into segments and forwards them individually to the IP software layer (Internet Layer). The Internet Layer encapsulates each TCP segment into an IP packet by adding a header that includes (among other data) the destination IP address. Even though every packet has the same destination address, they can be routed on different paths through the network. When the client program on the destination computer receives them, the TCP layer (Transport Layer) reassembles the individual segments and ensures they are correctly ordered and error free as it streams them to an application.

IP Networking

The **Internet Protocol (IP)** is the principal communications protocol used for relaying datagrams (also known as network packets) across an internetwork using the Internet Protocol Suite. Responsible for routing packets across network boundaries, it is the primary protocol that establishes the Internet.

IP is the primary protocol in the Internet Layer of the Internet Protocol Suite and has the task of delivering datagrams from the source host to the destination host solely based on the addresses. For this purpose, IP defines datagram structures that encapsulate the data to be delivered. It also defines addressing methods that are used to label the datagram source and destination.

Function

The Internet Protocol is responsible for addressing hosts and routing datagrams (packets) from a source host to the destination host across one or more IP networks. For this purpose the Internet Protocol defines an addressing system that has two functions: identifying hosts and providing a logical location service. This is accomplished by defining standard datagrams and a standard addressing system.

Public-key cryptography

Public-key cryptography refers to a cryptographic system requiring two separate keys, one of which is secret and one of which is public. There are three primary kinds of public key systems: public key cryptosystems, public key distribution systems, and digital signature systems. Diffie–Hellman key exchange is the most widely used public key distribution system, and allows two users to securely agree on a shared secret, even if all their communications are compromised. The shared secret thus generated is then typically used as the key in a symmetric cipher. The Digital Signature Algorithm is the most widely used digital signature system. It can generate a digital signature, but has no privacy features.

A public key cryptosystem is more versatile and can be used both as a public key distribution system and a digital signature algorithm. A public key cryptosystem has two keys, one to lock or encrypt the plaintext, and one to unlock or decrypt the cyphertext. Neither key will do both functions. One of these keys is published or public and the other is kept private. If the lock/encryption key is the one published then the system enables private communication from the public to the unlocking key's owner. If the unlock/decryption key is the one published then the system serves as a signature verifier of documents locked by the owner of the private key. Although in this latter case, since encrypting the entire message is relatively expensive computationally, in practice just a hash of the message is encrypted for signature verification purposes.

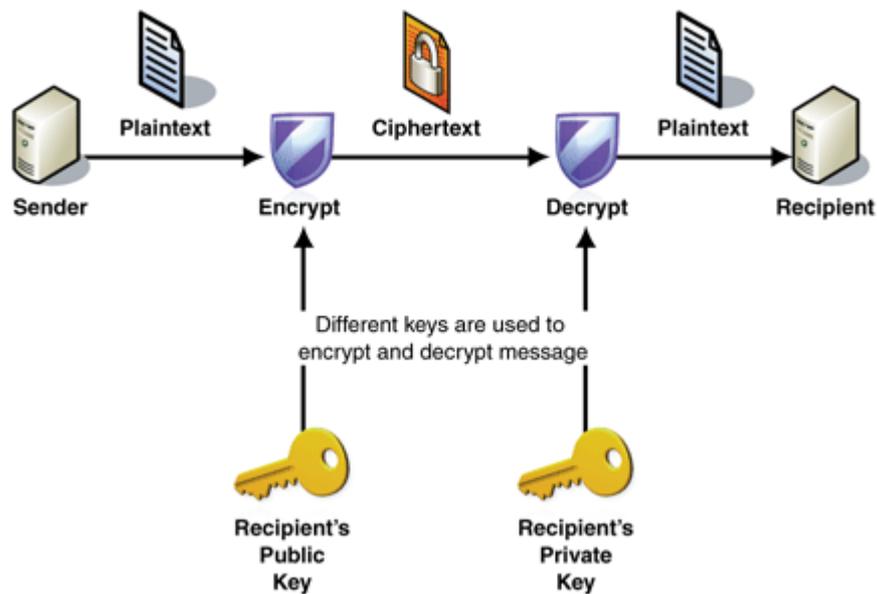
Because this cryptographic approach uses asymmetric key algorithms such as RSA its more general name is "asymmetric key cryptography." Some of these algorithms have the public key/private key property—that is, neither key is derivable from knowledge of the other—but not all asymmetric key algorithms do. Those without derivable keys are particularly useful and have been widely deployed; hence, they are often the meaning behind the name. Although different, the two parts of the key pair are mathematically linked. The public key is used to transform a message into an unreadable form, decryptable only by using the (different but matching) private key. By publishing the public key, the key producer empowers anyone who gets a copy of the public key to produce messages only s/he can read—because only the key producer has a copy of the private key (required for decryption). When someone wants to send a secure message to the creator of those keys, the sender encrypts it (i.e., transforms it into an unreadable form) using the intended recipient's public key; to decrypt the message, the recipient uses the private key. No one else, including the sender, can do so.

Thus, unlike symmetric key algorithms, a public key algorithm does not require a secure initial exchange of one, or more, secret keys between the sender and receiver. These algorithms work in such a way that, while it is easy for the intended recipient to generate the public and private keys and to decrypt the message using the private key, and while it is easy for the sender to encrypt the message using the public key, it is extremely difficult for anyone to figure out the private key based on their knowledge of the public key. They are based on mathematical relationships (the most notable ones

being the integer factorization and discrete logarithm problems) that have no efficient solution.

The use of these algorithms also allows authenticity of a message to be checked by creating a digital signature of a message using the private key, which can be verified using the public key.

Public key cryptography is a fundamental and widely used technology. It is an approach used by many cryptographic algorithms and cryptosystems. It underpins such Internet standards as Transport Layer Security (TLS) (successor to SSL), PGP, and GPG.



How it works

The distinguishing technique used in public key cryptography is the use of asymmetric key algorithms, where the key used to encrypt a message is not the same as the key used to decrypt it. Each user has a pair of cryptographic keys — a public encryption key and a private decryption key. The publicly available encrypting-key is widely distributed, while the private decrypting-key is known only to the recipient. Messages are encrypted with the recipient's public key and can be decrypted only with the corresponding private key. The keys are related mathematically, but parameters are chosen so that determining the private key from the public key is prohibitively expensive. The discovery of algorithms that could produce public/private key pairs revolutionized the practice of cryptography beginning in the mid-1970s.

In contrast, symmetric-key algorithms, variations of which have been used for thousands of years, use a single secret key — which must be shared and kept private by both sender and receiver — for both encryption and decryption. To use a symmetric encryption scheme, the sender and receiver must securely share a key in advance.

Because symmetric key algorithms are nearly always much less computationally intensive, it is common to exchange a key using a key-exchange algorithm and transmit data using that key and a symmetric key algorithm. PGP and the SSL/TLS family of schemes do this, for instance, and are thus called hybrid cryptosystems.

The RSA Algorithm

RSA is an algorithm for public-key cryptography that is based on the presumed difficulty of factoring large integers, the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described it in 1978. A user of RSA creates and then publishes the product of two large prime numbers, along with an auxiliary value, as their public key. The prime factors must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message. Whether breaking RSA encryption is as hard as factoring is an open question known as the RSA problem.

Operation

The RSA algorithm involves three steps: key generation, encryption and decryption.

Key generation

RSA involves a **public key** and a **private key**. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key.

The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q .
 - For security purposes, the integers p and q should be chosen at random, and should be of similar bit-length. Prime integers can be efficiently found using a [primality test](#).
2. Compute $n = pq$.
 - n is used as the [modulus](#) for both the public and private keys
3. Compute $\phi(n) = (p - 1)(q - 1)$, where ϕ is Euler's totient function.
4. Choose an integer e such that $1 < e < \phi(n)$ and [greatest common divisor](#) of $(e, \phi(n)) = 1$; i.e., e and $\phi(n)$ are [coprime](#).
 - e is released as the public key exponent.
 - e having a short [bit-length](#) and small [Hamming weight](#) results in more efficient encryption - most commonly $0x10001 = 65,537$. However, small values of e (such as 3) have been shown to be less secure in some settings.[\[4\]](#)
5. Determine d as:
 - $d \equiv e^{-1} \pmod{\phi(n)}$
 - i.e., d is the [multiplicative inverse](#) of $e \pmod{\phi(n)}$.
 - This is more clearly stated as solve for d given $(de) \pmod{\phi(n)} = 1$
 - This is often computed using the [extended Euclidean algorithm](#).
 - d is kept as the private key exponent.

Encryption

[Alice](#) transmits her public key (n, e) to [Bob](#) and keeps the private key secret. Bob then wishes to send message \mathbf{M} to Alice.

He first turns \mathbf{M} into an integer m , such that $0 < m < n$ by using an agreed-upon reversible protocol known as a [padding scheme](#). He then computes the ciphertext \mathbf{C} corresponding to

$$c = m^e \pmod{n}$$

This can be done quickly using the method of [exponentiation by squaring](#). Bob then transmits \mathbf{C} to Alice.

Note that at least nine values of m could yield a ciphertext c equal to m , but this is very unlikely to occur in practice.

Decryption

Alice can recover m from c by using her private key exponent d via computing

$$m = c^d \pmod{n}.$$

Given m , she can recover the original message M by reversing the padding scheme.

(In practice, there are more efficient methods of calculating c^d using the pre computed values below.)

Secret-key Cryptography

Symmetric-key algorithms are a class of algorithms for cryptography that use the same cryptographic keys for both encryption of plaintext and decryption of ciphertext. The keys may be identical or there may be a simple transformation to go between the two keys. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link. This requirement that both parties have access to the secret key is one of the main drawbacks of symmetric key encryption, in comparison to public-key encryption.

Types of symmetric-key algorithms

Symmetric-key encryption can use either stream ciphers or block ciphers.

- Stream ciphers encrypt the digits (typically bits) of a message one at a time.
- Block ciphers take a number of bits and encrypt them as a single unit, padding the plaintext so that it is a multiple of the block size. Blocks of 64 bits have been commonly used. The Advanced Encryption Standard (AES) algorithm approved by NIST in December 2001 uses 128-bit blocks.

Implementations

Examples of popular and well-respected symmetric algorithms include Twofish, Serpent, AES (Rijndael), Blowfish, CAST5, RC4, 3DES, and IDEA.

BIBLIOGRAPHY

Bibliography

- English Wikipedia: Java (programming language) - [http://en.wikipedia.org/wiki/Java_\(programming_language\)](http://en.wikipedia.org/wiki/Java_(programming_language))
- JAVA: What Is Java - http://www.java.com/en/download/faq/whatis_java.xml
- English Wikipedia: Java 2D - http://en.wikipedia.org/wiki/Java_2D
- English Wikipedia: Global Positioning System - http://en.wikipedia.org/wiki/Global_Positioning_System
- English Wikipedia: Trilateration - <http://en.wikipedia.org/wiki/Trilateration>
- English Wikipedia: Photoelectric sensor - http://en.wikipedia.org/wiki/Photoelectric_sensor
- English Wikipedia: Switch - <http://en.wikipedia.org/wiki/Switch>
- English Wikipedia: Transmission Control Protocol - http://en.wikipedia.org/wiki/Transmission_Control_Protocol
- English Wikipedia: Internet Protocol - http://en.wikipedia.org/wiki/Internet_Protocol
- English Wikipedia: Public-key cryptography - http://en.wikipedia.org/wiki/Public-key_cryptography
- English Wikipedia: Symmetric-key algorithm - http://en.wikipedia.org/wiki/Symmetric-key_algorithm
- English Wikipedia: IEEE floating point - http://en.wikipedia.org/wiki/IEEE_floating_point
- BinaryEssence: Deflate: Basic Algorithm - <http://www.binaryessence.com/dct/imp/en000241.htm>
- English Wikipedia: RSA (algorithm) - [http://en.wikipedia.org/wiki/RSA_\(algorithm\)](http://en.wikipedia.org/wiki/RSA_(algorithm))
- English Wikipedia: Blowfish (cipher) - [http://en.wikipedia.org/wiki/Blowfish_\(cipher\)](http://en.wikipedia.org/wiki/Blowfish_(cipher))
- English Wikipedia: Railroad switch - http://en.wikipedia.org/wiki/Railroad_switch
- English Wikipedia: Track circuit - http://en.wikipedia.org/wiki/Track_circuit
- GPS.gov: GPS Accuracy - <http://www.gps.gov/systems/gps/performance/accuracy/>
- InfraSurvey: UnderGround Positioning System - <http://www.infrasurvey.ch/?lang=en/>

DOCUMENT REVISION HISTORY

1.1

- Fixed diagram on page 9

1.2

- Changes to the communication protocols between the control server and the various clients