# Project 14: Design och konstruktion av grafeditor

# Architectural and Detailed Design

Project submitter:
Swedish National Defence College

Project Group nr 20 Aliquid
Simon Lindholm, Erik Lindström, Andreas Linn, Martin Mützell, Johannes Olegård, Anton Olsson, Viktor Palmkvist, Daniil Pintjuk, Frans Tegelmark, Sean Wenström.

Version number and document status
Version 1.1.0
Date of printing 2014-04-10

## Abstract

STRATMAS v7.5 is simulation software intended for decision aid in crisis situations like war and natural catastrophes. Infrastructure such as road, waterpipes and communications are needed to simulate more advanced countries, war zones and evacuation scenarios. All these features are best represented as graphs and as such need the same kind of abstractions and GUI representations. This document describes how the existing software will be modified and extended to accomplish these additions.

# Table of Contents

Abstract
Table of Contents
1. Introduction
  1.1. Purpose
  1.2. Scope of the software
  1.3. Definitions, acronyms and abbreviations
  1.4. References
  1.5. Overview of the document
2. System Overview
3. System Context
  3.1. Shape files
4. System Design
  4.1. Design method
  4.2. Decomposition description
5. Component Description
  5.1 Toolbox
    5.1.1 Type
    5.1.2 Purpose
    5.1.3 Function
    5.1.4 Subordinates
    5.1.5 Dependencies
    5.1.6 Interfaces
    5.1.7 Resources
    5.1.8 References
    5.1.9 Processing
    5.1.10 Data
  5.2 Map
    5.2.1 Type
    5.2.2 Purpose
    5.2.3 Function
    5.2.4 Subordinates
    5.2.5 Dependencies
    5.2.6 Interfaces
    5.2.7 Resource
    5.2.8 References
    5.2.9 Processing
    5.2.10 Data
  5.3 TacLan
    5.3.1 Type
    5.3.2 Purpose
    5.3.3 Function

5.3.4 Subordinates
5.3.5 Dependencies
5.3.6 Interfaces
5.3.7 Resources
5.3.8 References
5.3.9 Processing
5.3.10 Data
5.4 Scenario
5.4.1 Type
5.4.2 Purpose
5.4.3 Function
5.4.4 Subordinates
5.4.5 Dependencies
5.4.6 Interfaces
5.4.7 Resources
5.4.8 References
5.4.9 Processing
5.4.10 Data
5.5 Scenario.Unit
5.5.1 Type
5.5.2 Purpose
5.5.3 Function
5.5.4 Subordinates
5.5.5 Dependencies
5.5.6 Interfaces
5.5.7 Resources
5.5.8 References
5.5.9 Processing
5.5.10 Data
5.6 Scenario.Graph
5.6.1 Type
5.6.2 Purpose
5.6.3 Function
5.6.4 Subordinates
5.6.5 Dependencies
5.6.6 Interfaces
5.6.7 Resources
5.6.8 References
5.6.9 Processing
5.6.10 Data
6. Feasibility and Resource Estimates
6.1. Toolbox
6.2. Map

6.3. Taclan
6.4. Scenario and sub components
7. User Requirements vs Components Traceability Matrix

# 1. Introduction

## 1.1. Purpose

The purpose of this document is to describe how to achieve the goals and implement the requirements from the user requirement document. It describes the architectural changes we are going to do on the software and changes in the protocol between the server and the client.

## 1.2. Scope of the software

As we extend STRATMAS v7.5, with a graph editor to support infrastructure (roads, power lines, water pipes, etc.) changes are going to take place in three major parts of the codebase. First in the XML protocol Taclan, second in the client to visualize and edit roads and similar infrastructure and third in the server to support the simulation for said roads and infrastructure.

## 1.3. Definitions, acronyms and abbreviations

Here is a list of words that appear in this document and short descriptions of what the intended meaning is.

- STRATMAS: STRATegic MAnagement System. The name of the different simulation software developed at the Swedish National Defence College. There have been many different versions of STRATMAS (not necessarily related to each other except in concept) and this project intends to extend one of these versions (v7.5).
- Scenario - Version of the simulation, often a country or crisis zone.
- Agent - A major mobile entity in the simulation e.g military platoon or terrorist milita.
- Graph: A graph is a set of “nodes” and “edges”. A node is any object and edges describe how these nodes are connected. Graphs are very generic and can be used for describing virtually anything.
- Node - Abstract, connection points for edges, e.g water pumps or road split.
- Edge - Abstract, connections between nodes, e.g the roads, waterpipes or powerlines.
- Graph editor: A graphical user interface for creating and editing graphs (here: for STRATMAS).
- GUI - Graphical user interface, the visual representation of the software and simulation
- Server - The simulator part of the software, calculates the outcome of a scenario for the client to display
- Client - The GUI part of the software, connects to the server
- XML: eXtensible Markup Language. A way of formatting text-data so that it is as easily readable and modifiable by people as by machines. XML is used for storing simulation parameters for scenarios.
- TacLan - TACtical LANguage. An XML-based language for describing the parameters used in the simulations to specify simulation scenarios.
- Cell - Small part of the simulation world. The simulation world is divided into a grid of cells.
- URD - User requirements document.

## 1.4. References

[1] Full system description
https://github.com/sootn/aliquid-stratmas/blob/master/doc_v7_5/system-description.pdf?raw=true
[2] Model details
https://github.com/sootn/aliquid-stratmas/blob/master/doc_v7_5/reference/stratmas-models.pdf?raw=true
[3] Client documentation
https://github.com/sootn/aliquid-stratmas/blob/master/doc_v7_5/StratmasClient/client_user_manual.pdf?raw=true
[4] Server documentation
https://github.com/sootn/aliquid-stratmas/blob/master/doc_v7_5/StratmasServer/server_user_manual.pdf
[5] Taclan2 xsd
https://github.com/sootn/aliquid-stratmas/blob/master/schemas/stratmasProtocol/taclan2sim.xsd
[6] http://www.qgis.org/en/site/
[7] http://www.statsilk.com/software/statplanet-plus
[8] http://docs.oracle.com/javase/6/docs/api/
[9] https://xerces.apache.org/xerces-c/apiDocs-2/classes.html
[10] https://en.wikipedia.org/wiki/A*_search_algorithm
[11] http://jogamp.org/deployment/jogamp-next/javadoc/jogl/javadoc/
[12] https://en.wikipedia.org/wiki/Shapefile

## 1.5. Overview of the document

This document is divided into 7 major sections:

1. Introduction: This chapter, explaining the project goals and structure.
2. System overview: This chapter briefly explains the current codebase.
3. System context. A short overview of what technical environment and context the system is written for.
4. System design: An overview of our changes to the system and the architectural layout of our new code.
5. Component description: A more in-depth look at each component in the system.
6. Feasibility and resource estimates: An analysis of the feasibility of the planned changes included in the project plan.
7. User Requirements vs Components Traceability Matrix: A more graphical explanation of the project where each component is related to the requirements in the URD, with dependencies, resources and functions.

# 2. System Overview

STRATMAS is a large pre-existing project with very little interaction with external systems. It is divided into a server, a client and a language in between. The server deals with the simulation while the client controls the server and shows the intermediary and final results of the simulation. The interface between these is an XML schema named TacLan. There is also a basic dispatcher that can distribute simulation requests between multiple servers.

### Client

The client consists of GUI widgets and taclan protocol classes. Classes (or groups of classes) that are of interest are mainly the toolbox (created by us) and the map (modified by us).

### Server

The server can be split in three parts, one simulator (which is the most important part), one buffer (where all data from the simulation is saved when sending to/retrieving from clients), and multiple connections (one per client). Intermediary results during simulation are saved in the buffer, from which they are later sent out to all subscribed clients. The simulator has many parts, but few are relevant to the current project. The simulation is contained inside a scenario, to which we will have to add things. Current additions are looking to be graphs of two kinds (paths and effects) and ensuring units follow paths if that would lower the required time spent in transit. The effect graphs will also have to apply their effects to the surrounding simulation.

### External Interactions

STRATMAS is largely independent, with a few exceptions, some obvious, some not. The client is written in Java and requires JOGL, thus the both of these have to be present and work on the computer upon which the client is intended to run. The server has to be built for the specific operating system and architecture and also has some dependencies of its own, which need to exist on the targeted machine. To be able to run client and server on separate machines there has to be an Internet connection available.

STRATMAS uses one user-facing external system; its maps are defined in shapefiles, which can be edited with some external program (it is a standard, so multiple should exist) or obtained from other sources as prebuilt maps.

# 3. System Context

As the system is not used today and it unclear how or why it should be used from the customers side, the system context is hard to identify.

## 3.1. Shape files

The maps in the visuals for Stratmas is made up from shape files with file extension .shp, .shx and/or .dbf. These are created by external programs and only read and displayed by the Stratmas client. The the shapes has no impact on the simulation. Some external programs compatible with shape files are QGIS [6] and StatPlanet Plus [7].

## 3.2. Hierarchy files

Stratmas uses hierarchy files named tl2 created in a program called Visto. Stratmas needs to read these files to parse hierarchies.

## 3.3. TacLan

The internal communications protocol of Stratmas, defining where units are, what cells contains what and what shapes populations and other variables take. It is an XML format sent over the network during simulations or written to disk to store scenarios.

## 4. System Design

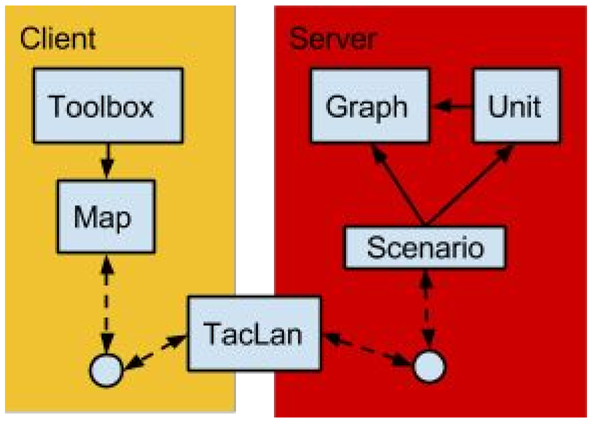
Both the client and the server are large systems with many components, which could be described in more detail, but most of them are of no interest to the current project, given the limited scope of additions, and so we shall refrain from doing so. Instead we will only describe components we will need to modify/add.

### 4.1. Design method

Given the large amount of pre-existing code, coupled with the low amount of time available, the already existing design has been used to a very large extent. All additions have been made based on common sense and basic analysis of the impact of the addition. The protocol for communicating between the server and the client was designed first given its high likelihood of making the rest of the design easily derivable (once again through common sense).

### 4.2. Decomposition description

The big parts of Stratmas is the client and the server, talking with each other through TacLan. The toolbox creates graphs on the map, defining them as roads, water pipes etc. The client then writes it into the taclan blob together with everything else from the old version and sends it to the server for the simulator to process the new data, now with graph support as well as the old simulation variables.



*Circles represent big parts of the project that are irrelevant to the current project.*

# 5. Component Description

This section describes the various components in detail.

## 5.1 Toolbox

### 5.1.1 Type

Changes the existing JPanel (Java class) defining the map controls

### 5.1.2 Purpose

To be able to select tools used to create graphs and coexisting with existing editing tools. This to support requirements from the URD like 1-GRAPH_ROADS, 2-GRAPH_LINES, 8-MAP_CONTROL, 9-EDITOR_SCENARIO .

### 5.1.3 Function

Through the toolbox one should be able to select a tool from multiple different tools. Only one tool at a time should be selected, like a radio button. When you push one of them, the last selected will be unselected.

### 5.1.4 Subordinates

Buttons, panels, tabs etc.

### 5.1.5 Dependencies

The old map controls to be improved on.

### 5.1.6 Interfaces

The selected tool will be sent to the map to do the actual changes to the graph, view or process variables.

### 5.1.7 Resources

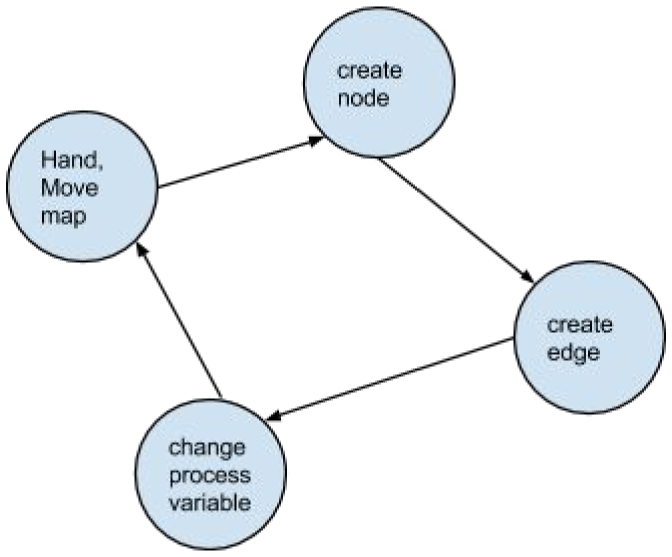
Mouse, keyboard, Java virtual machine, screen, trained user.

### 5.1.8 References

Java, awt and swing docs [8]

### 5.1.9 Processing

The buttons need to know when some other button is pressed to unpress itself. That is, only one state will be active at any time, see graph.



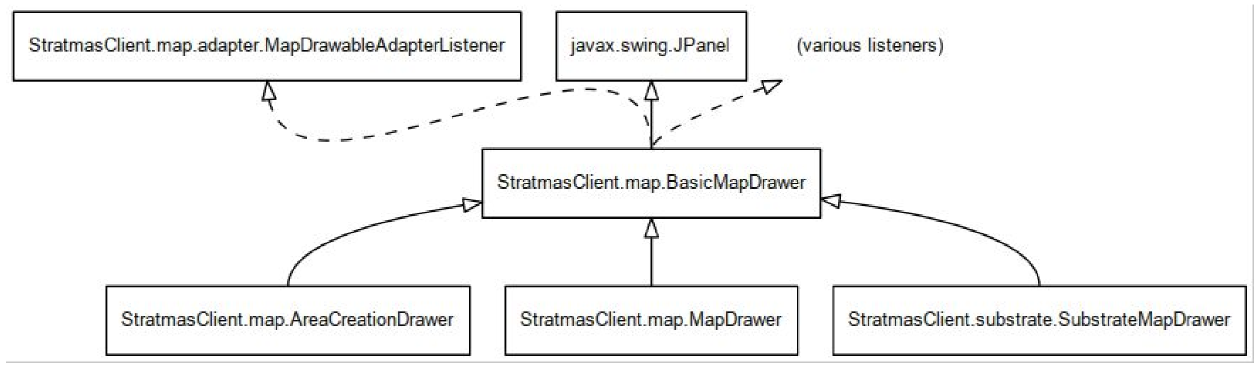
### 5.1.10 Data

StratmasClient/src/StratmasClient/map/DisplayControl.java

## 5.2 Map

### 5.2.1 Type

Mostly a class. The main class of the map component is the BasicMapDrawer, which extends JPanel. Though most parts of the client that uses a map extends the BasicMapDrawer to add additional behaviour.

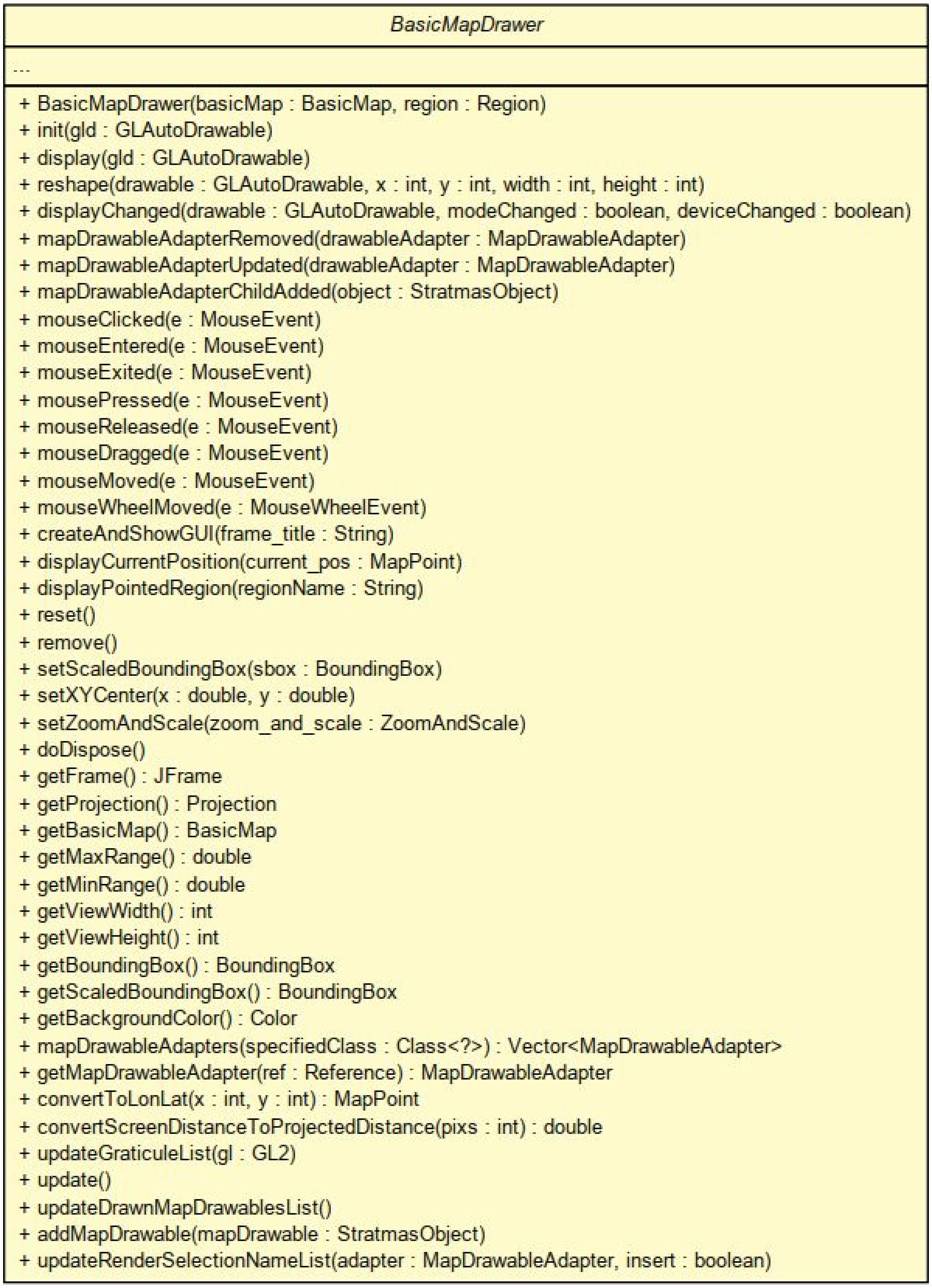

StratmasClient.map.adapter.MapDrawableAdapterListener
javax.swing.JPanel
(various listeners)
StratmasClient.map.BasicMapDrawer
StratmasClient.map.AreaCreationDrawer
StratmasClient.map.MapDrawer
StratmasClient.substrate.SubstrateMapDrawer

### 5.2.2 Purpose

The map is needed to visualize the scenario and simulation results to the user. It is related to requirement 3.1.2 (and its various sub requirements) in the URD. This includes 5-MAP_VIEWS, 6-MAP_MOREMAPS, 7-MAP_UNITS, 8-MAP_CONTROL. Visualization of various kinds of graphs also make this component relevant to 1-GRAPH_ROADS and 2-GRAPH_LINES.

### 5.2.3 Function

An overview of the various methods in BasicMapDrawer is shown here in simplified UML. Note that only public members are shown. The functions of the drawer could be divided into two categories: GUI widget functions and camera functions (i.e. controls the perspective the map is viewed in). The functions of the latter category are more relevant to this project, and include SetXYCenter(double,double)  (used to move the camera), the various get-functions (used to get the camera details). Some widget functions may also be worth mentioning: the various mouseXXX(..) functions are used to implement the mouse-wheel zoom and mouse-drag move on the map (8-MAP_CONTROL).



*BasicMapDrawer*

...

```
+ BasicMapDrawer(basicMap : BasicMap, region : Region)
+ init(gld : GLAutoDrawable)
+ display(gld : GLAutoDrawable)
+ reshape(drawable : GLAutoDrawable, x : int, y : int, width : int, height : int)
+ displayChanged(drawable : GLAutoDrawable, modeChanged : boolean, deviceChanged : boolean)
+ mapDrawableAdapterRemoved(drawableAdapter : MapDrawableAdapter)
+ mapDrawableAdapterUpdated(drawableAdapter : MapDrawableAdapter)
+ mapDrawableAdapterChildAdded(object : StratmasObject)
+ mouseClicked(e : MouseEvent)
+ mouseEntered(e : MouseEvent)
+ mouseExited(e : MouseEvent)
+ mousePressed(e : MouseEvent)
+ mouseReleased(e : MouseEvent)
+ mouseDragged(e : MouseEvent)
+ mouseMoved(e : MouseEvent)
+ mouseWheelMoved(e : MouseWheelEvent)
+ createAndShowGUI(frame_title : String)
+ displayCurrentPosition(current_pos : MapPoint)
+ displayPointedRegion(regionName : String)
+ reset()
+ remove()
+ setScaledBoundingBox(sbox : BoundingBox)
+ setXYCenter(x : double, y : double)
+ setZoomAndScale(zoom_and_scale : ZoomAndScale)
+ doDispose()
+ getFrame() : JFrame
+ getProjection() : Projection
+ getBasicMap() : BasicMap
+ getMaxRange() : double
+ getMinRange() : double
+ getViewWidth() : int
+ getViewHeight() : int
+ getBoundingBox() : BoundingBox
+ getScaledBoundingBox() : BoundingBox
+ getBackgroundColor() : Color
+ mapDrawableAdapters(specifiedClass : Class<?>) : Vector<MapDrawableAdapter>
+ getMapDrawableAdapter(ref : Reference) : MapDrawableAdapter
+ convertToLonLat(x : int, y : int) : MapPoint
+ convertScreenDistanceToProjectedDistance(pixs : int) : double
+ updateGraticuleList(gl : GL2)
+ update()
+ updateDrawnMapDrawablesList()
+ addMapDrawable(mapDrawable : StratmasObject)
+ updateRenderSelectionNameList(adapter : MapDrawableAdapter, insert : boolean)
```

### 5.2.4 Subordinates

Display of graphs, functions of the toolbox tools.

### 5.2.5 Dependencies

The loaded shapefile, graphs and process variables, the Toolbox (see 5.1).

### 5.2.6 Interfaces

Process variables to be sent to the server, read shapefiles.

### 5.2.7 Resource

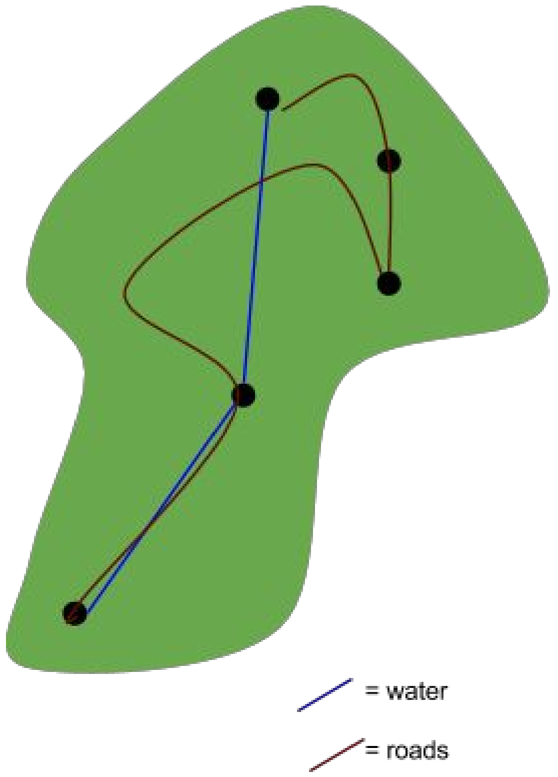
Sample shapefiles from the samples folder [1]

### 5.2.8 References

JOGL documentation [11], shapefiles [12]

### 5.2.9 Processing

Added render layers for the roads, waterpipes and other graphs on top of the existing process variable and agent visualization. Something like this (concept).



### 5.2.10 Data

This is too complicated to explain in detail here. In short the map contains objects to maintain and control the gui, and objects to retrieve data to display as the map.

## 5.3 TacLan

### 5.3.1 Type

Part of XSD namespace. Changes to the existing xml schema definitions, the xsd files and small structural changes to the client and servers parsing of these files.

### 5.3.2 Purpose

To send information between the client and server about the infrastructure graphs. To convey the simulation to the client to solve 1-GRAPH_ROADS, 2-GRAPH_LINES and others.

### 5.3.3 Function

Verify and define how the xml over the network and in file should look like.

### 5.3.4 Subordinates

Not applicable

### 5.3.5 Dependencies

The existing XSD's to make the changes in.

### 5.3.6 Interfaces

The server and the client's parsing of the TacLan files. Data is passed both to and from both endpoints.

### 5.3.7 Resources

XMLparsing libraries already existent in the client like java standard library and xerces c++

### 5.3.8 References

Java [8] and Xerces [9] docs

### 5.3.9 Processing

Below is a pseudo markup of the extensions for the XSD and TacLan protocol to demonstrate a proof of concept. An XSD version exists, but is a work in progress and unnecessarily complicated to display as-is here.

```
CommonScenario < sp.Scenario {
  ...
  path: sp.PathGraph
  graphs*: sp.EffectGraph
}
NodeReference < sp.Reference <target: string fixed="Node"> {}
abstract Graph < sp.ComplexType {}
EffectGraph < Graph {
  nodes+: sp.EffectNode
  edges+: sp.EffectEdge
  nodeConnected?: sp.Effect
  nodeDisconnected?: sp.Effect
  edgeConnected?: sp.Effect
  edgeDisconnected?: sp.Effect
}
PathGraph < sp.Graph {
  nodes+: sp.PathNode
  edges+: sp.PathEdge
}
Effect < sp.ComplexType {
  continuos: bool //not sure if this is a good idea, but it might allow for both triggered
effects and continuous
  radius: float
  //TODO: represent the changes that can be applied in a useful way
}
abstract Node < sp.ComplexType {
  point: sp.Point
}
EffectNode < sp.Node {
  power: float //how much is provided. negative for drain.
  connected?: sp.Effect
  disconnected?: sp.Effect
}
PathNode < sp.Node {}
abstract Edge < sp.ComplexType {
  origin: sp.NodeReference
  target: sp.NodeReference
  connected: bool
}
EffectEdge < sp.Edge {
  connected: sp.Effect
  disconnected : sp.Effect
}
PathEdge < sp.Edge {
  travelSpeed: float //A factor of original travelspeed perhaps?
}
```

### 5.3.10 Data

See 5.3.9 Processing, above.

## 5.4 Scenario

### 5.4.1 Type

The top-level container class that drives the simulation.

### 5.4.2 Purpose

Simulate and process the graphs from 1-GRAPH_ROADS and 2-GRAPH_LINES (as well as everything else related to simulation).

### 5.4.3 Function
The scenario contains all simulation objects and drives the simulation. All our changes will be inside this component, by changing one subcomponent and adding some subcomponents. Other workings of this component are irrelevant and will not be described.

step(Time) : void. Simulates until the given time. Will have to simulate graphs as well, but will otherwise remain unchanged.

### 5.4.4 Subordinates
Scenario.Graph and Scenario.Unit (as well as several irrelevant components that will not be described).

### 5.4.5 Dependencies
The existing simulation code.

### 5.4.6 Interfaces
Receives data from and sends data to Buffer.

### 5.4.7 Resources
A scenario, defined in TacLan.

### 5.4.8 References
The existing server documentation [4].

### 5.4.9 Processing
All simulation and most of the semantics of the TacLan definitions are initiated from here, but very little of it has relevance to the current project. It passes data defined in TacLan to the subcomponents that require it as well as making sure they are all set up.

### 5.4.10 Data
Scenario contains everything, but relevant to this project are:
- mForces, a list of all units
- mPath, a graph representing the roads in the simulation
- mGraphs, a list of all effect graphs

## 5.5 Scenario.Unit

### 5.5.1 Type
A class in the server describing units, including subordinates.

### 5.5.2 Purpose
For this project the only relevant function is dealing with unit movement, which is related to 1-GRAPH_ROADS.

### 5.5.3 Function
Relevant functions are:

- setGoal(Location) : void. Sets the location to which the unit should move. Will ask the path graph for the fastest path between the current position and the goal.
- move() : void. Moves the unit. Will move it along the previously computed path.

#### 5.5.4 Subordinates

Not applicable.

#### 5.5.5 Dependencies

The only relevant dependency is the path graph.

#### 5.5.6 Interfaces

Once again the majority of the data flow is irrelevant to the current project. Relevant is that unit will get access to the path graph from scenario.

#### 5.5.7 Resources

Unit will require the path graph.

#### 5.5.8 References

The existing server documentation [4].

#### 5.5.9 Processing

Move will follow the precomputed path one edge at a time, similarly to what it does now. SetGoal will request a path, which is a very simple method call.

#### 5.5.10 Data

Unit will have to store the calculated path.

### 5.6 Scenario.Graph

#### 5.6.1 Type

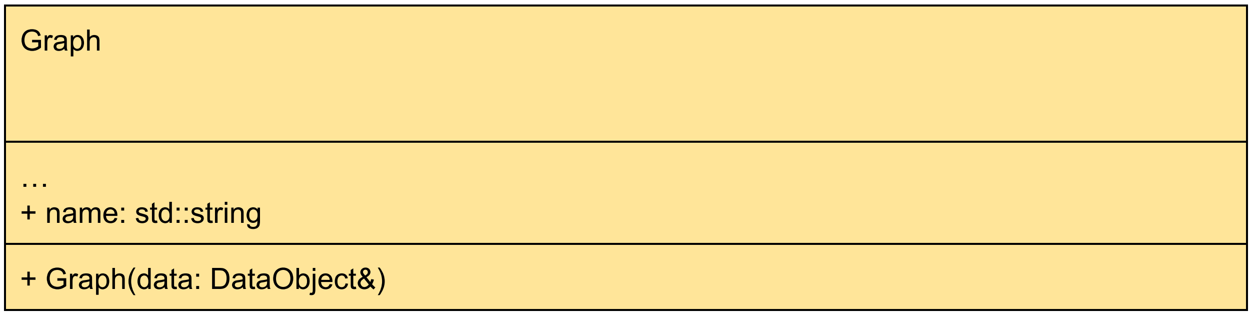
Set of classes in the server project.

#### 5.6.2 Purpose

Simulate and represent graph data in the server. user requirements: 1-GRAPH_ROADS, 2-GRAPH_LINES

#### 5.6.3 Function

Create a main class to represent the graph, and help classes. according to the diagrams:



Graph

…
+ name: std::string

+ Graph(data: DataObject&)

| |
|---|
| + act(): void<br>+ getPath(a: longlat, b: longlat) : NavigationPlan |

| Edge |
|---|
| + origin: Node*<br>+ target: Node*<br>+ conected: bool |
| |

| Node |
|---|
| + point: LongLat |
| |

| NavigationPlan |
|---|
| + on: LongLat<br>+ off: LongLat<br>+ path: List<Edge*> |
| … |

A list of graphs will be created from the appropriate child of the data object that is passed to the scenario. Every execution of scenario::step will call graph::act on every graph.
Graph::act will simulate the graphs and execute activities contained within the graphs.
Graph::getPath(...) will find the best way along the graph from point a to point b and return it represented as NavigationPlan.

The project will have two different kinds of graphs, one for paths (roads) and one for effects. These will get different classes, but given the high similarity only the superclasses are discussed here. The contents of these subclasses are defined in TacLan above, and should be fairly straight-forward to translate.

### 5.6.4 Subordinates

Subordinates include Edge, Node and NavigationPlan as described above. NavigationPlan is created by a path graph after pathfinding is done, but is not contained within it.

### 5.6.5 Dependencies

The class depends on Classes and interfaces in existing simulation.

This class needs a valid DataObject object created from valid taclan according to the expanded taclan definition.

#### 5.6.6 Interfaces

When the scenario is created a DataObject object is passed to the constructor of every Graph that is created. The data object should follow the new taclan definition.

The graphs will be able to change the state of the Grid with Cell::setPV(..) or Cell::setAllPV or Cell:setAllPVR or git Cell:setPVR or Grid::expose (const Action &inA) .

#### 5.6.7 Resources

Needs dataobjects from the scenario.

#### 5.6.8 References

Stratmas Documentation [4], Taclan extension pseudomarkup (see 5.3.9).

#### 5.6.9 Processing

The Graph should be mainly immutable, except for the edge field “connected” that will be changed by activities, agencies and units externally and effect only pathfinding.
function getPath is the only function that will do any “processing”, and it will be done using an appropriate pathfinding algorithm that we will iterate upon to determine an appropriate balance between accuracy, speed and simplicity

#### 5.6.10 Data

Internal data is as shown in the UML above, though the exact details will likely change and will be determined iteratively.

# 6. Feasibility and Resource Estimates

## 6.1. Toolbox

This toolbox implementation is estimated to be comparably unproblematic and will approximately take one day to implement and another day to write the accompanying documentation and instructions. This part is deemed very feasible and thought not a high priority, completing it early adds utility for manual testing during the development process.

## 6.2. Map

The update of the map implementation is estimated to be moderately complicated to complete. Due to flaws in the existing codebase, a clean implementation is deemed beyond the realistic scope of the project and would require a complete rewriting of the current codebase. Instead the development team has decided to instead focus on a solution based on the current codebase and while it is likely to inherit some existing flaws it is deemed adequate for the project scope.

This update of the already implemented map is deemed to approximately take one week of work for implementation and two additional days for documentation and review. This is considered a medium-high priority due to the fact that it will convey the updated functionality to the user and is deemed feasible within the project scope.

## 6.3. Taclan

The updates to the Taclan protocol laid out in 5.3.9 is approximated to take approximately one weeks work, writing of sample documents and documentations included. Due to it's vital role in communicating the new functionality from the server to the client, this is considered the second-highest priority and is deemed feasible within the project scope.

## 6.4. Scenario and sub components

The updates to the server simulation is the most complex part of the project. The complexity is dramatically increased by the flaws in the existing codebase. The update may in turn inherit a number of problems and flaws of the existing codebase and is estimated to take approximately three weeks to implement. This is considered the top priority and focus of the project. The simulation updates is considered feasible although the analysis is preliminary and may come to change due to encountered problems in the existing code. If any insurmountable difficulties are encountered the team will instead focus on updating the client and Taclan protocols at the cost of the server development.

## 7. User Requirements vs Components Traceability Matrix

| | 1 Toolbox | 2 Map | 3 Taclan | 4 Scenario | Need |
|---|---|---|---|---|---|
| 1-GRAPH_ROADS | x | x | x | x | required |
| 2-GRAPH_LINES | x | x | x | x | required |
| 3-GRAPH_COMS | x | x | | | nice to have |
| 5-MAP_VIEWS | | x | | | required |
| 6-MAP_MOREMAPS | | x | | | Nice to have |
| 7-MAP_UNITS | | x | | | Nice to have |
| 8-MAP_CONTROL | x | x | | | Nice to have |
| 9-EDITOR_SCENARIO | x | | | | Nice to have |
| 10-EDITOR_PLAYBACK | x | | | | Nice to have |