Project 14: Design och konstruktion av grafeditor

Project Planning Document

Project submitter: Swedish National Defence College

Project Group Aliquid

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

Abstract

Project 14: *Design och konstruktion av grafeditor* is a short-term project to improve the military strategic visualization software STRATMAS on behalf of the Swedish National Defense College. The goals of the project are to implement nodes and a graph editor into the existing system and will be performed by Team Aliquid in the course DD1392 at KTH during 2014.

Version 2.0.0 , Printed 2014-04-01

Table of Contents

Document Change Record

- 2014-01-03: Document created and structure added.
- 2014-02-09: Added basic text or list of todos for all sections except feasibility assessment.
- 2014-02-11: Added project plan stages, description and graphics.
- 2014-02-16: Removing todo’s, portraying correct format of names and completing risk assessments.
- 2014-03-23: Version bump 2.0.0
- 2014-03-26: Added answers to risk template.
- 2014-04-09: Done with risks.

1. Statement of the problem

1.1. Detailed Statement of the Problem

STRATMAS (**STRAT**egic **MA**nagement **S**ystem) is simulation software in several incarnations designed for simulating populations of various scales and the effects of events and agents in said populations. This project concerns the latest version of STRATMAS (V7.5, the eighth variant of STRATMAS), and has two main objectives and one necessary (but secondary) objective. Any later reference to STRATMAS refers to the latest incarnation unless otherwise specified.

Main objectives:

- Enable STRATMAS to simulate graph-based models (road networks, etc.) together with the already present cellular automaton and agent simulator and have them all interact with each other in suitable ways.
- Create a graph editor for describing the new graph-based models.

Secondary objective:

- Make STRATMAS capable of running on newer hardware and software than it was originally designed for. It was last used 2006 so the age is not a huge problem.

Description of the overarching architecture of STRATMAS

STRATMAS is split into three essential parts: a server, a client and TacLan, a language for communication between the first two. The server gets a description of the simulation (a scenario) to run in TacLan and then performs the actual simulation, returning the result (also in TacLan) to all registered clients. The client can edit the starting state of the simulation and what fixed events will happen and at what time during the simulation, as well as display the outcome and current state of the simulation. TacLan is an XML-based markup language to describe all the parts of the simulation.

There is also an evolver, for taking a scenario and finding an optimal configuration based on a few variable parameters and an evolutionary algorithm, as well as a server dispatcher for distributing scenarios to multiple servers depending on their load.

Basic analysis of required work to complete the project

All of these parts are likely to require at least minor changes, with the exception of the server dispatcher that should not need to know anything about the semantic details of what it is passing on.

The server will need the integration of an entirely new type of simulation that needs to interact with those already present. This will require a good understanding of the structure of STRATMAS, more specifically of how simulation is done, to be able to integrate graphs in a (mostly) seamless way. It will also require knowledge of the actual models used in the existing simulation and what models would be useful for the various uses of graphs.

The client will at least need to be able to show the graph-based data, possibly also edit it. It is possible that it would be easiest to create the graph editor as a separate program, but the most useful would probably be to integrate it directly into the existing client. The specifics of this will have to wait until we have more information.

TacLan needs to be able to describe these graphs and their effects, which will at least require additions to the semantics, but possibly also to the syntax. All three of the client, the server and the evolver need to be able to understand these additions/changes.

1.2. Motivation

This project was not our first choice in the course, but it is still intriguing. No one else but Swedish National Defence College would have this kind of problem or solution. It’s a game but real, and we all love games.
At KTH we will see many future work providers, but they are all businesses. Organizations like Swedish National Defence College is invisible at events and corporate exposiciones days, even if they live across the street.
Swedish National Defence College is respected and will of course look great in our future employers’ eyes.

1.3. Goals

What we hope to achieve from this project correlates well with our future careers. Besides the experience from having worked on a large project, we will probably learn about parallelism and simulation in this project which may prove useful later. We will will learn c++ programming, build systems (mainly make, cmake and ant), XML Schema Definition, handling large code bases and working in medium sized teams.

1.4. Skills Baseline

There is some knowledge we all have that can form some sort of baseline. We have all done a fair bit of programming in Java (which the client is written in). Most of us also have experience with version control (specifically Git), which is rather necessary for a project of this size.

Some of us have experience in C++ (which the server is written in) and all of us have written at least a little C. We probably need more people to learn C++, but several of us have expressed a wish to do so. It will mostly be done by discussing with those that already know it and reading the existing code base as well as research on the Internet.

We have one or two who have experience with GUI-related work beyond the basics, but we are rather confident that we can come up with a good concept and then learn what we need to realize it. The implementation of this would require knowledge of JOGL as well as AWT/Swing.

Most of us have used XML in some form or other, but our experience with XSD is limited. To learn this we will most likely read up on the existing implementation of TacLan as well as research on the Internet.

Some of us are good at writing and the formalia of work around a project, while most of us are not. The group is primarily put together based on programming skill and the ability to reason about logical and structural problems. But we have at least 2 members who can write well written text.

Some of us have experience with working in larger groups of programmers; primarily scrum or agile project planning of different degrees. Communicating this knowledge should be easy.

We all lack experience with simulation theory but have plenty of experience with making games as well as algorithms. Most of our learning here will come from understanding the existing simulation and talking to the customer about the what and how of simulating graph-based data, as well as research dependent on that conversation.

2. Background to the Problem

2.1. Commercial background

This section is not applicable because this project is not developing a commercial product.

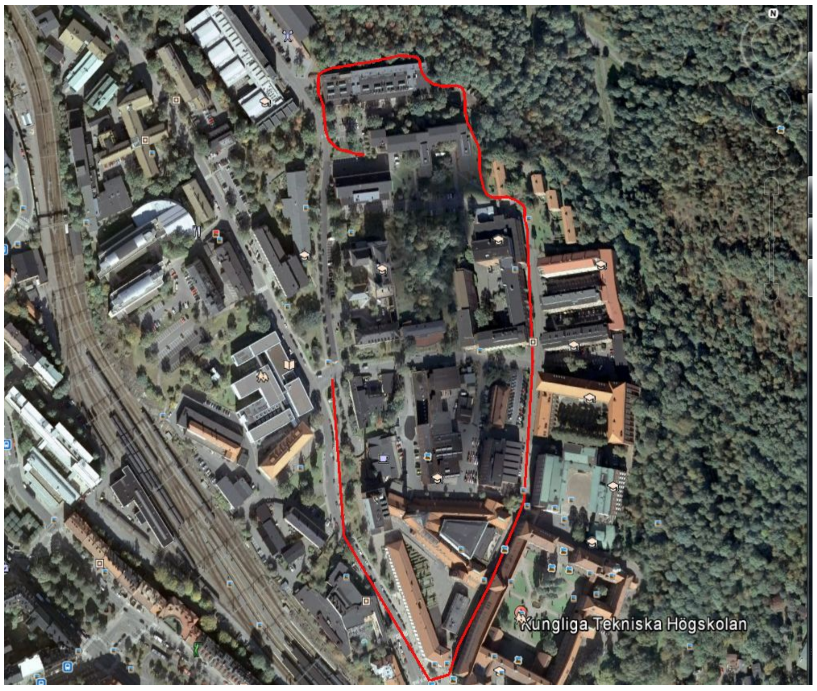
2.2. Scientific Background

Since we have two problems, firstly a graph-editor and secondly a way to simulate data that can be represented by a graph, the background can be split neatly into two categories.

There are plenty of graph-editors, so for this part of the problem we have plenty of sources for research. At least one major feature of most graph-editors is likely not applicable to this project: most graph-editors can distribute nodes automatically to make the graph easy to comprehend, but our graphs are required to have a direct mapping to the underlying map, making that sort of feature useful only for visualization, and possibly not needed.

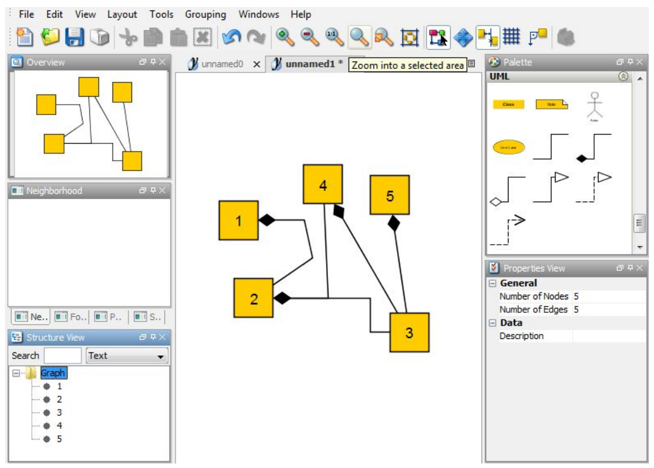
Google Earth path editor

Google Earth has a “graph”-editor to let users create paths. This tool is very similar to what we want to build, with the obvious exception that it is very limited in the type of graphs you can create. it lets you draw and edit the path on top of a map, giving the user the ability to draw graphs on top of the map will probably be a good idea considering the fact that they are supposed to represent roads, power grids and other infrastructure.



You create a path in google earths path tool the following way: by clicking anywhere on the map a new node will be created connected to the leaf node of the path. Clicking on a node and dragging it around lets you adjust the position of the node. It has one additional interesting feature that lets you add nodes and edges by freehand drawing with your mouse. This is a feature we should consider reproducing in order to enable easy creation of curve-looking branches.

yEd graph editor

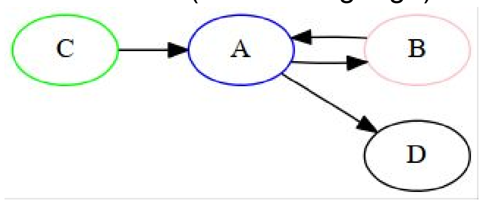
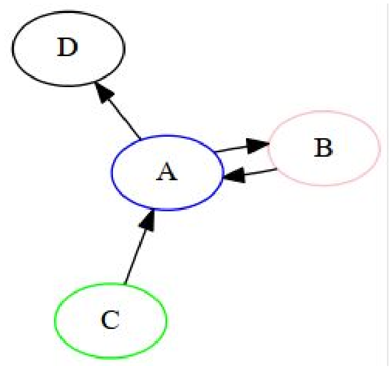


Properties View
General
Number of Nodes 5
Number of Edges 5
Data
Description

yEd is a graph editor with some interesting features we should consider. Every node and edge can have properties that you can edit in the properties view. The simulation will probably need more data than just the position of the nodes. Creating a similar feature would enable a user to create and edit this data - e.g. data to give the simulation information about what graphs are systems of roads and what graphs are power grids, or tell about different conditions in different parts of a road. Another interesting feature is the palette, it lets you create nodes and edges with predefined properties. This saves you from having to set all the properties of every node manually.

GraphViz graph visualization software

GraphViz is set of tools for visualizing graphs presented in a text-based language called “dot”. This usually means turning a graph in dot-file representation into an image of some format (e.g. gif, svg, pdf, etc.). GraphViz doesn’t include any extensive GUI, but there are many external graph GUI programs that use GraphViz. As such GraphViz is not interesting for this project in the GUI aspect, but rather in how it stores and handles graphs. It is possible STRATMAS could use GraphViz (or any of its external APIs) as a library and perhaps reuse the dot language for storing graph data.
As shown by the example, dot is very simple and could be used store a lot of data easily. Also note how unknown node- and edge-attributes are ignored by the visualization tools, allowing all kinds of (text) data to be stored. This type of language may be easier for the average user to write than the current xml-files in STRATMAS. Although we will write an GUI, creating very large datasets in a GUI may be tedious a simpler format like this might help in such case.

Simple example graph in dot.	Corresponding visualization in png imageformat.
`// c/c++ style comments` `// direction graph named myGraph.` `digraph "myGraph" {` `rankdir="LR" // nodes left to right` `// some nodes` `A [description="This is node A.",` `color="blue"]` `B [color="pink"]` `// some edges` `A -> B [dist=42] // dist gets ignored` `B -> A [dist=3]` `// more` `C [color="green"]` `C -> A // no attributes!` `D; // no quotes!` `A -> D` `}`	With the “dot” (not the language) “filter”:  With the “neato” filter: 

```
// c/c++ style comments

// direction graph named myGraph.
digraph "myGraph" {

        rankdir="LR" // nodes left to
right

        // some nodes
        A [description="This is node
A.",
                color="blue"]
        B [color="pink"]

        // some edges
        A -> B [dist=42] // dist gets
ignored
        B -> A [dist=3]

        // more

        C [color="green"]
        C -> A // no attributes!

        D; // no quotes!
        A -> D
}
```

The second part of the problem, the simulation of graph-represented models, also has some precedence.

LincityNG

Lincity is an open source city simulation game. Given its subject matter it has a need to simulate various forms of infrastructure. The map is based on a grid, where some cells are marked as transport of various kinds. Resources then spread along these cells according to different formulas for different types of transport and different types of resources. The result is basically a cellular automaton with different types of cells. This is at the very least an interesting approach to consider for us, as STRATMAS already contains a system for cellular automata which might be extendable in this way.

The required interplay between the existing simulation in STRATMAS and the new graph-system means we need to research cellular automata and agent-based simulation as well, and come up with a good way to mix them. According to the customer, there seems to have been no real tests of the accuracy of STRATMAS. This makes the minimal requirement of our job easier as there is no rigorous test of correctness we need to perform or pass, but it makes it harder to determine whether our simulations are accurate enough to be useful. In addition, us attempting to perform such a test would require information we do not possess as well as time we will likely not have.

We may have to weigh the simplicity of the simulator against the simplicity of use. If the simulator becomes too abstract it may become difficult for users to interact with it directly.

2.3. Technical Background

The version of STRATMAS that we received was written to work in a variety of platforms. Since the client is written in Java it runs on most platforms. The server is written in C++ and forced to work on some different platforms (Linux, Windows and Mac OS X) with different architectures (x86, x86_64 and PowerPC). We got the client working almost immediately, but the server will need some reworking.

The portability of the server is accomplished by using portable libraries (mainly Boost and Xerces C). These dependencies and their use is the main part that needs updating. Besides just updating to newer versions, some features can be reimplemented in these libraries. For example networking was implemented mostly manually in STRATMAS, but can now be done with boost, which should improve portability.

Platforms used by group members are GNU/Linux, OS X and Windows. We will likely prioritize platforms in that order (with the aim to support all of them). Since none of us owns a PowerPC, we will likely drop support for that platform, unless the customer specifically requests it and we get a machine to test on.

We will try to reuse as much as possible from the version of STRATMAS we received. Rewriting everything is not doable within the given timeframe of the project. As such we will keep using Java for the client and C++ for the server. This should also help reuse portability and parallelism in the project (this version of STRATMAS was done as a degree project by students at KTH Center for High-Performance Computing).

Both client and server uses different lexers to generate parsers. This will be useful as we will likely need to extend the format of the data that is parsed. Not having to set it up from scratch should save us time.

As for version control; we have put the project in a public repository on github (https://github.com/sootn/aliquid-stratmas), after getting permission from the owner to do so. The original STRATMAS didn't show any traces of having been organized with version control software.

3. Risk Assessment

We decided to use the Taxonomy-based Risk Identification template from Carnegie Mellon University that consists of 194 questions (plus a number of sub questions) that serves as a tool to identify risks that might affect the project. We had a risk identification meeting with the majority of the core members where we went through all the questions that related to our project.
Please remember that some risks affect each other in some way, mainly by lowering work morale through additional work that needs to be done.
The top 12 risks that we identified during that meeting are listed below in estimated falling order of expected negative effect. Each risk is presented using a description, an approximate probability that it will affect the project, the estimated effect the risk would have on the project should it affect it, as well as a course of action (COA) which is what we have decided to do about it.

1. The massive size and complexity of the existing system might cause difficulty in understanding and implementation of our solution.
    - Probability - Guaranteed, as this is an inherent property of the project.
    - Effect - High, it slows down our work-progress to a crawl and that in turn turns members off from even working on it at all, affecting work morale.
    - COA - Accepting it, there is no way to use the existing codebase for our project without looking at the back-end of it. The only alternative would be to re-implement the entire thing, which there is not enough time for.

2. Low work-morale can cause members to not contribute to the project.
    - Probability - Very High, as we have observed this behaviour extensively thus far.
    - Effect - High, having an effectively lower workforce greatly increases the workload on each active member.
    - COA - Continue the attempts to atleast get the members to the meetings, where we also regularly review the work we've done since the last meeting. If too many members are unable to contribute to the project in the following weeks we will consider dropping the server-side implementation of our solution.

3. Some members might not learn how to use git/github or other tools we use.
    - Probability - Medium, there has been a variance among those not familiar with the tools so far.
    - Effect - High, as a member not using these tools is unable to contribute to the project (besides documents).
    - COA - Having an experienced member of the team showing the ropes has proven successful so far, so we will continue offering that help to those still not able to contribute.

4. Some simulation-based requirements might prove too difficult to implement.

    - Probability - Medium, as we've got some pretty smart and skilled people on the team, but simulation is not something we've worked with before.
    - Effect - Medium, the solution might not become feature-complete, but it'll serve it's purpose to some extent.
    - COA - Prioritizing the different server-side simulation features so that the most important ones (ex. pathfinding) are finished before we implement the more nice-to-have (ex. water pollution) ones.

5. The complete lack of unit-testing in the existing code may cause us to introduce unexpected behaviour in the program.
    - Probability - High, as the size and complexity of the program makes it very hard for us to track what the effects of our additions to the existing code will be.
    - Effect - Medium, our use of version control will help us out here a lot, as we can track which individual change introduced the behaviour.
    - COA - We will **not** create unit-tests for the entire existing code-base, as there is not enough time.

6. Our lack of a change control process for internal interfaces might cause some problems later down the development line.
    - Probability - Medium, as we have not put our Architectural Design Document to the test for actual implementation yet.
    - Effect - Medium, previously working code could break by changing somewhere else and an inability to control this would make it take long to find the issue. Although our use of version control will mitigate this.
    - COA - When we have actually put the architectural design to the test we will decide on whether to adopt a change control process.

7. We lack mechanisms to control changes in the product which might cause bad code to slip into the program.
    - Probability - Medium, although what counts as "bad code" is debatable.
    - Effect - Medium, as this would require redoing work that was supposed to be finished. We do however have an "as long as it works" attitude at the moment. Also due to using git, most bad code is easy to rollback.
    - COA - Monitoring the state and decide on it later if we deem it necessary.

8. The product could be next-to impossible to test outside of the existing scenarios.
    - Probability - Almost guaranteed, unless we manage to magically create a full-featured scenario editor that *isn't* a project in and of itself to use.
    - Effect - Low, as we believe that we will be able to modify the existing scenarios sufficiently should we succeed in building the graph editor.
    - COA - Ignore it, because of the low effect, and of the massive amount of work that would be required otherwise.

9. Our development process lacks a strict definition and because of that will maybe not be followed by some members.
    - Probability - Medium, although because it's loosely defined you need to wander off a bit further than you normally would.
    - Effect - Low, someone might implement the wrong thing, but those who doesn't know what do usually don't do it unless they're specifically told to.
    - COA - Monitor it for now. We may decide in a future meeting to define the process in a strict manner if we see the need. For now, formality may hamper our progress.

10. Some requirement validation methods are not well-defined and may cause confusion on whether a feature is complete or not.
    - Probability - Low, as we ourselves decide that and our meetings are frequent that information should be easily accessible.
    - Effect - Low, since a this mostly concerns the GUI, which is hard to test formally, but easy informally.
    - COA - Wait and see if course administration or the customer complains about it, otherwise ignore it.

11. Our lack of quality control could lower the end quality of the product.
    - Probability - Medium, we have already decided against fixing the existing code, which is what quality control would've mostly pointed to.
    - Effect - Low, this was never intended to be high-quality simulation tool anyway.
    - COA - Ignore it, we need to spend our time implementing our solution, and the quality has already vastly improved since we received the code.

12. Customer might read something else out of the requirements.
    - Probability - Low, we have a pretty good idea of what he has in mind, and he knows he can't get all the features that he wants.
    - Effect - Very low, some level of surprise from the customer, either positive or negative. We don't expect him to be upset about something minor, and the main requirements are pretty clear.
    - COA - Ignore it, as he has stated multiple times and has made it clear that he wants us to have a lot of freedom.

4. Project Plan

Deliverable	Draft / Prototype	Deadline 1 / Feature freeze	Deadline 2 / Release	People	Timeslice	Skills
PPD	2014-02-12	2014-02-19	2014-04-08	ALL	Medium	Risk assessment, planning, information gathering
ADD	2014-03-25	2014-04-08	-	ALL	Medium	
URD	2014-02-28	2014-03-05	2014-04-24	ALL	Medium	Social
Server development environment	2014-03-01	2014-04-24	2014-05-13	Anton, Frans (Windows), Daniil, Viktor (Mac)	Medium	Bash, CMake
Simulation	2014-04-01	2014-04-24	2014-05-13	TBD	Large	C++, Boost, graphteory?
Server communicatio n	2014-04-14	2014-04-24	2014-05-13	TBD	Small	XML, Xerces
Client development environment	2014-03-01	2014-04-24	2014-05-13	Andreas, Martin, Sean, Johannes	Medium	Ant, Eclipse
GUI	2014-04-01	2014-04-24	2014-05-13	TBD	Large	Java, AWT, JOGL
Client communicatio n	2014-04-14	2014-04-24	2014-05-13	TBD	Small	JAXP
TacLan definition	2014-04-01	2014-04-24	2014-05-13	Viktor	Small	XML, XSD

4.1. Stages

4.1.1. PPD

By 2014-02-12 all parts of the PPD-1 will be complete and we will mark it as a draft. After that we will make tweaks and smaller changes until deadline at 2014-02-19.

4.1.2. Development environment

Before any work can begin with the client or the server we need to get all developers to understand the existing code base and its structure. Upgrade dependencies to modern versions, read documentation and code to get an overview of the code structure and rewrite build-scripts for modern systems.

4.1.3. Server

The server and client code are separate and have different requirements and different personnel. Therefore we need separate deadlines for different deliverables. Implementing the logic on the server side early is important for the TacLan and client deliverables, and an early prototype would be great.

4.1.3.1. Simulation

Before major development of the application can commence, the server developers need to be able to test the code and view resulting effects. This requires that the “development environment” part of the client is completed. This stage will be time consuming. There is a lot of new logic to write and a lot of existing code to integrate with.

4.1.3.2. Communication

Some time after the work on modifying the simulation has started, we can start extending the communication part of the server. The required protocol changes need to be well defined to begin updating the TacLan serialization on both the server and client in parallel.

4.1.4. Client

To be able to test the server code, everyone that works on the code needs a working client. The first deadline for the client should be getting it to run properly on all developer machines. But we can start to fiddle, refresh and read the code right off the bat.

4.1.4.1. GUI

The GUI is not dependent on anything other than the client development environment. Do we need to change a lot of existing code or can we simply add another window for the graph editing? With the large amounts of prototyping and experimentation necessary in this stage it will require a lot of time.

4.1.4.2. Communication

The required protocol changes need to be well defined to begin updating the TacLan serialization on both the server and client in parallel.

4.1.5. TacLan

TacLan, the protocol between the server and the client needs support for our changes. Before serialization of the graphs on the client can commence we need a clear understanding of how the protocol works and what information the server needs. A working server prototype would give great confidence to complete the protocol code on the client.

4.1.5.1. Definition

Defining the protocol changes for the XML will require knowledge that we do not possess at the time of writing. A large part of this stage will require reading up on XSD (XML Schema Definition) and asking to the customer if they have anything to say about it. This stage is crucial for the completion of both the client and the server. It will require extensive documentation.

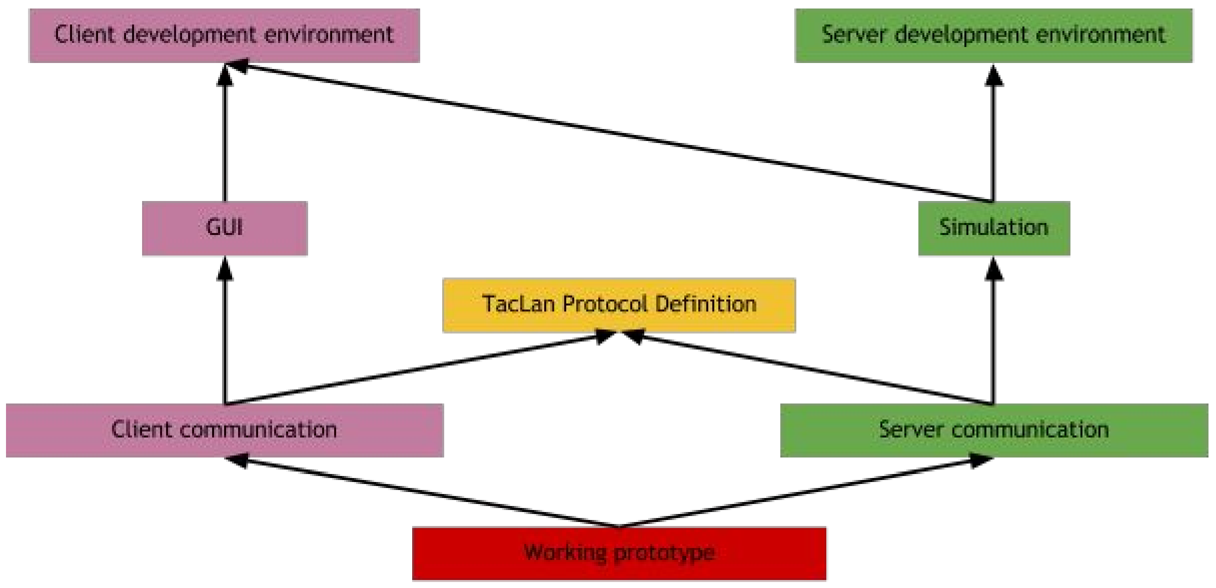

Client development environment
Server development environment
GUI
Simulation
TacLan Protocol Definition
Client communication
Server communication
Working prototype

Fig: Stage dependencies

5. Feasibility Assessment

The feasibility of the development environment stage is high as most of those changes have been completed during the two weeks of writing this document. GUI changes should be fairly straightforward (but time consuming) as the features are nothing groundbreaking and we can take ideas and hints from how other graph editors do UI.
The largest uncertainty is the simulation as we currently do not know how much existing code has to be changed to implement the new features and what subtleties lie hidden within.
TacLan is a well defined protocol that seems to have been built with future additions in mind and the only major challenge lies in making sure that both server and client respond correctly which can be tricky.
In conclusion, we deem it feasible to lay a good groundwork for, and implement one or several, graph-based systems into the existing framework.

References

1. http://www.graphviz.org/About.php, GraphViz, 2014-02-13, retrieved 19:59
2. http://www.thecornwallisgroup.org/pdf/CXII_2007_22_AERW&SAC.pdf, The Strategic Management System (STRATMAS®): The Afghanistan Study 2003 and Exercise Iraq Future 2005
3. https://github.com/sootn/aliquid-stratmas, GitHub repo 2014
4. https://drive.google.com/folderview?id=0B03ikQPlxp--RFZyeUJMTXJ6c3M&usp=sharing , repo for this document and others.
5. http://www.google.com/earth/, Google Earth
6. http://www.yworks.com/en/products_yed_about.html, yEd
7. http://lincity-ng.berlios.de/wiki/index.php/LinCity_Engine, LinCity
8. https://www.kth.se/social/upload/530e47e0f276543ac29235fc/CMU-SEI-93-TR-6.pdf, CMU risk analysis

Appendix

Risk Analysis

Below are our answers to the CMU risk analysis template. Because there are a lot of questions, only the answers are shown here. Cross reference with the template [8] for best result.

```
A {
        1 {
                a {
                        1 - yes
                        2 - no
                }
                b {
```

```
        3 - no
        4 - no
        5 - no
        6 - no  //RISK
    }
    c {
        7 - yes, yes
    }
    d {
        8 - yes //limited resources, but he seems to be okay with it
        9 - unknown, no process //RISK
        10 - method varies, some are well defined, others not
    }
    e {
        11 - yes
        11.a - 1-GRAPH_ROADS#3, 2-GRAPH_LINES#2
        11.b a potentially big change to how agents "plan" their movement, an entirely new system that must integrate with the existing ones, a new editor that must integrate with the rest of the client interface
    }
    f {
        12 - no
        12.a - yes // t.ex. xsd
    }
    g {
        13 - yes
        14 - yes
    }
}
2 {
    a {
        15 - no
        15.a - no (not applicable)
        16 - analysis and prototyping
    }
    b {
        17 - yes //RISK
        18 - yes //RISK
        18.a - graph.taclan
    }
    c {
        19 - no
        20 - yes
        20.a - no
        21 - no
    }
    d {
        22 - no
        23 - no
    }
    e {
        24 - no
        25 - no
        26 - yes
    }
    f {
        27 - no
    }
    g {
        28 - yes
        28.a - customization
        29 - not applicable
        30 - not applicable
    }
}
3 {
    a {
        31 - yes //RISK
        32 - yes
    }
    b {
        33 - no
        34 - no
        35 - no
        36 - yes
    }
    c {
        37 - yes
        38 - no
        39 - no
        40 - yes
        41 - yes
        41.a - yes
        42 - no
        42.a - yes
        43 - not applicable
```

```
                44 - not applicable
        }
}
4 {
        a {
                45 - yes
                46 - yes //RISK
                47 - yes
                48 - yes
                48.a - yes
        }
        b {
                49 - yes
                50 - yes
                50.a - no
                51 - yes
                52 - yes
                53 - no
                54 - no
                55 - not applicable
        }
        c not applicable
}
5 {
        a {
                61 - no
                62 - no
                63 - yes
        }
        b {
                64 - no
                65 - no
        }
        c {
                66 - no
                67 - no
        }
        d {
                68 - no
                69 - no
                70 - yes
        }
        e {
                71 - yes //RISK
            }
            f {
                72 - yes
                73 - yes
                74 - yes
                75 - not applicable
                76 - yes
            }
        }
    }
    B {
        1 {
            a {
                77 - no
                78 - no
            }
            b {
                79 - no //RISK
                80 - yes //RISK
            }
            c {
                81 - no //RISK
                82 - yes
                83 - not applicable
            }
            d {
                84 - no
            }
            e {
                85 - no //RISK
                86 - no
                87 - no
                88 - no
                89 - yes
                90 - yes
                91 - not applicable
            }
        }
        2 {
            a {
                92 - yes
                93 - yes
        }
        b {
                94 - yes
        }
        c {
                95 - yes
                96 - yes
        }
        d {
                97 - yes
        }
        e {
                98 - yes
        }
        f {
                99 - no //RISK
                100 - yes
                101 - irrelevant
        }
        g {
                102 - no
        }
}
3 {
        a {
                103 - yes //RISK
                103.a - no
                104 - yes
                105 - yes
                106 - yes
                106.a - when time is short
                107 - no //RISK
        }
        b {
                108 - no //RISK
                109 - yes
                110 - yes
        }
        c {
                111 - yes
        }
        d {
                112 - yes
                113 - no
                114 - yes
                115 - yes
                116 - no
        }
}
4 {
        a {
                117 - no
                118 - yes
                119 - yes
                119.a - yes
        }
        b {
                120 - yes
                120.a - yes
                121 - yes //RISK
                122 - yes
                123 - yes
                124 - no //RISK
                125 - yes
                126 - yes
        }
        c {
                127 - no
                128 - no
        }
        d {
                129 - yes
                130 - yes
                131 - no
                132 - no
        }
}
5 {
        a {
                133 - no //RISK
                134 - yes //RISK
        }
        b {
                135 - yes
                136 - yes
                137 - no
        }
        c {
            138 - yes
            139 - yes
            139.a - yes
            139.b - yes
            140 - yes
            140.a - yes
        }
        d {
            141 - no
            141.a - viability of project. uninteresting problem.
            142 - yes
        }
    }
}
C {
    1 {
        a {
            143 - yes
            144 - yes
            144.a - no
            144.b - no
            145 - no
            146 - no
        }
        b {
            147 - no
            148 - yes
            149 - yes
            150 - yes
            151 - no //RISK
            152 - yes
            153 - no
        }
        c not applicable
        d not applicable
    }
    2  not applicable
    3 {
        a {
            166 - yes
            167 - yes //RISK
```

```
                168 - no
                169 - yes
                170 - no
                171 - yes
                172 - yes
                173 - yes
                174 - realistic
            }
            b not applicable
            c not applicable
            d not applicable
            e {
                188 - no //RISK
                189 - yes
                190 - no
                191 - realistic
            }
            f {
                192 - no
            }
            g {
                193 - no
                194 - no
            }
        }
    }
```