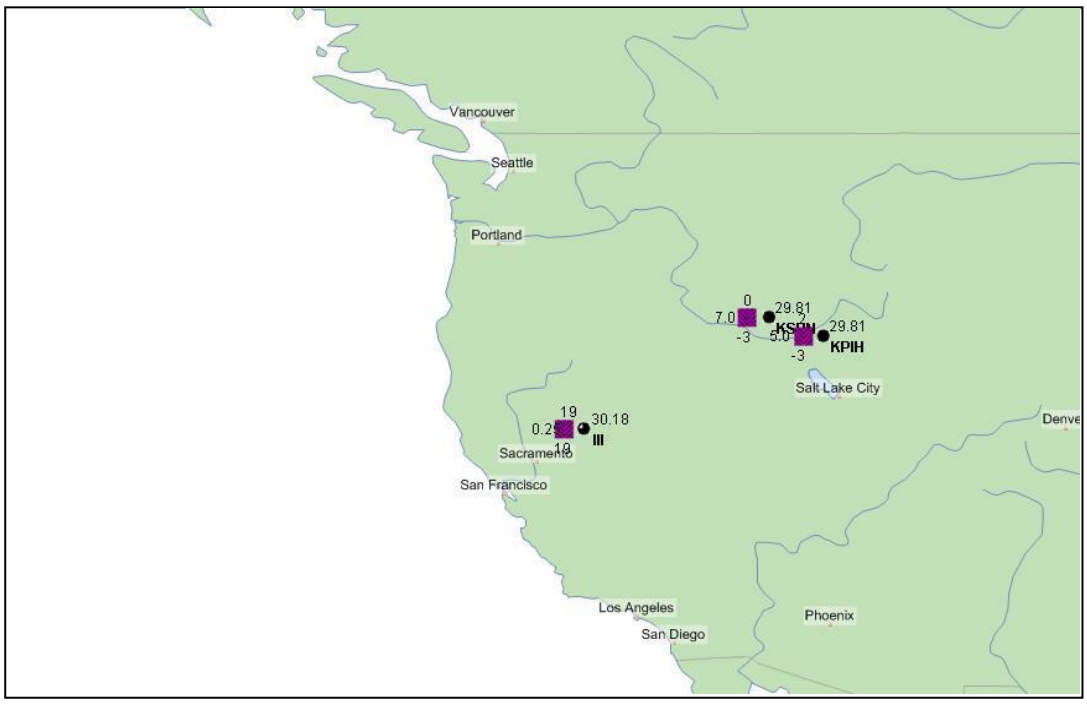
CHALMERS



Weather Client

An Evaluation of the SLD Symbolization and Colouring Standard for Weather Data

Master of Science Thesis in the Programme Software Engineering and Technology

NICOLE ANDERSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, 2012

Weather Client
An Evaluation of the SLD Symbolization and Colouring Standard for Weather Data

Nicole Andersson

Examiner: Christian Berger

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: A map containing METAR data, visualized using SLD/SE and Carmenta Engine.

Department of Computer Science and Engineering
Göteborg, Sweden June 2012

Abstract

Visualization of weather data on the Weather Information Exchange Model (WXXM) in geospatial systems using the visualization and colouring standard Styled Layer Descriptor (SLD)/Symbology Encoding (SE) is a rather unexplored area. In this thesis, a system named the Weather Client has been developed as a proof-of-concept prototype to evaluate said standards.

Issues found during the implementation served as a ground for the evaluation and they were analysed in depth using a modified root cause analysis method. Each issue was evaluated with regard to functionality, usability, maintainability and efficiency.

The result is a number of improvement proposals for the SLD/SE standard, an evaluation of the root cause analysis method as well as recommendations to Carmenta AB.

The result of the evaluation indicated that the SLD/SE standard show great potential for describing weather data in geospatial systems. However, there are a number of issues that need to be resolved and further investigation is recommended.

Keywords: Styled Layer Descriptor, Symbology Encoding, Weather Information Exchange Model, METAR, weather visualization, map rendering, geospatial systems, root cause analysis, 5 Whys analysis method.

Acknowledgements

I would like to thank Carmenta AB for giving me the opportunity to write this thesis. Their knowledge and expertise in the field were of great help. I would also like to thank my supervisor Lars Pareto for all his help and support during the thesis work.

Table of Contents

1 Introduction

Today, weather phenomena causes flight delays all over the world and it is not uncommon for those working in the aviation industry to stand unprepared when a blizzard or storm strikes. For *airline companies,* better weather information would enable them to: better calculate flight paths, take precautions against snow-covered landing strips and last but not least, make better fuel calculations. In a study from 2007, it was found that two thirds of all weather related delays could be avoided, by the use of weather information, thus making significant cost savings [1]. This opportunity drives airline companies in different countries to cooperate on weather information exchange as well as visualization, and standards for such.

Noticeable such standards are the Weather Information Exchange Model (WXXM), designed by Eurocontrol in partnership with NNEW, the Styled Layer Descriptor (SLD) standard [2] and the Symbology Encoding (SE) standard [3]. All these standards are XML based data formats, currently under development and the latter two have already been adopted by companies in the geospatial and Air Traffic Management (ATM) domains. WXXM is considered the youngest and most immature standard of the three, and have yet to gain acceptance from the community.

However, the companies exploring WXXM and SLD/SE have recognized risks with these technologies. There is little evidence that implementations using the standards allow for maintainable software, and that the visualization under constraints of the standards meet end users' expectations.

This thesis approaches this problem by an exploratory case study at Carmenta AB, a Swedish company that is one of actors in the geospatial domain. They see business potential in these formats and wish to evaluate WXXM in combination with SLD/SE in order to establish whether or not their product should implement and include support for weather information.

The purpose of this thesis is to evaluate to which degree the SLD/SE applied to weather information on the WXXM format allows usable, maintainable and efficient software. By usable we mean to what extent the system meets the user's requirements. By maintainable we mean the ability to service and maintain the software after its release. By efficient we mean the ability to fully utilize the available resources. Our evaluation consists of a proof of concept prototype and a Root Cause Analysis (RCA) of the problems experienced during development work with the purpose of finding limitations in the standards. Our research questions are:

- RQ1: What limitations of the SLD/SE standard affect functionality?
- RQ2: What limitations of the SLD/SE standard affect usability?
- RQ3: What limitations of the SLD/SE standard affect maintainability?
- RQ4: What limitations of the SLD/SE standard affect efficiency?

The main result of this thesis is a list of improvement proposals for SLD/SE. Other results are recommendations for Carmenta AB and an evaluation of the root cause analysis method.

1.1 Delimitations

The Weather Information Exchange Model (WXXM) will not be evaluated as a standard but as a data source for SLD.

2 Theory

This chapter defines the software qualities used in our evaluation and aviation weather concepts of the software studied. It also describes data formats such as AIXM, WXXM and SLD/SE and the different server types that can be used for distributing weather information.

2.1 Software Quality Requirements used for Evaluation

Central to our research questions are the software qualities defined in ISO9126. In particular: *functionality*, *usability*, *maintainability* and *efficiency*. [4]

The **functionality** of the system is described by what functions it provides. Functionality is determined by how many of the system's essential functions that are implemented. Most functional requirements are boolean, meaning that they are either fulfilled or not. In some cases the system is able to perform a part of a task e.g. displaying one of two things, thus making the requirement partially fulfilled.

Usability is determined by looking at how well the system meets the user's expectations. The system should be attractive in the eyes of the user and it should be easy for the user to understand how the system is supposed to be used, e.g. what tasks the user can perform with the help of the system. The learning curve for the system should be low and the user should easily be able to operate it. Usability is a bit hard to determine, since it is based on personal experience on how one perceives the system.

Maintainability describes the ability to service the software after its release. It is determined by how easy it is for new and old developers of the system to find faults within the software, and apply corrections in a timely manner. Code readability is a factor that highly affects maintainability, since finding errors is hard when one cannot understand what has been written.

Efficiency is described as the ability to fully utilize the resources at hand while providing good response, and processing times. For instance, if some data is to be sent over http-requests then we would not want to send unnecessary data since that will delay the packages and decrease the overall efficiency.

2.2 Aviation Weather Concepts

Forecasts and reports on current aviation weather come in a number of different report types. This thesis will focus on METAR reports and touch on similar report types such as Terminal Aerodrome Forecasts and SPECI.

2.2.1 METAR

A METAR report is an *observation* of the weather conditions surrounding an airport; such reports are typically generated once an hour and come in two different formats, one of which is text based and the other one is described by a graphical composition of the weather information. The text based presentation, as seen in Figure 2.1, is often preferred by pilots, whereas meteorologists rather view the graphical representation, as seen in Figure 2.2, while working on weather forecasts. [5]

```
METAR KSUN 261456Z 07009KT P6SM OVC021 00/M03 A2981 RMK SLP009
```

Figure 2.1. An example of a text based METAR observation from the airport KSUN.

The information available in a METAR report is:

- Altimeter
- Airport Identifier
- Cloud Cover
- Dewpoint
- Present Weather
- Temperature
- Visibility
- Wind barb

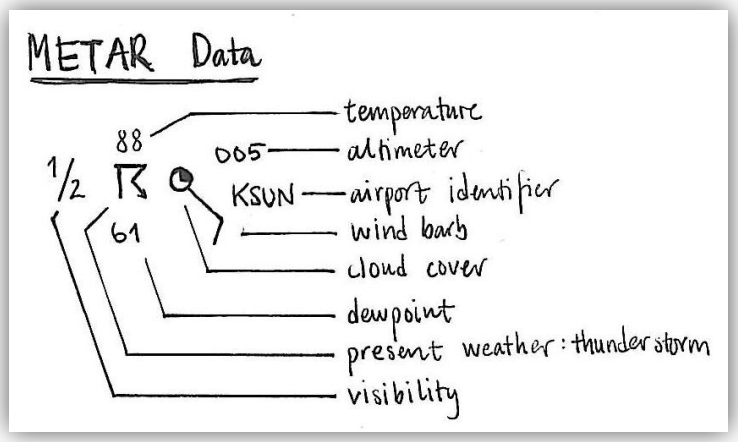


Figure 2.2. An example of a graphical representation of a METAR observation.

Cloud Cover describes how many oktas of the sky that is covered by clouds. If the amount of clouds is 0 oktas, then the sky is clear; if it is 8 oktas, the sky is completely covered by clouds [6]. To differ between human and automated observations, there are two ways in which the METAR report can tell the reader that the sky is clear. The abbreviation "CLR" is used when an automated station reports that there may be clouds above 12 000ft and "SKC" is used when a human report is stating that the sky is completely clear above. Furthermore, the cloud cover symbols are colour coded in order to fit the four flight categories. [5] The different cloud cover symbols that exist are described in Figure 2.3.

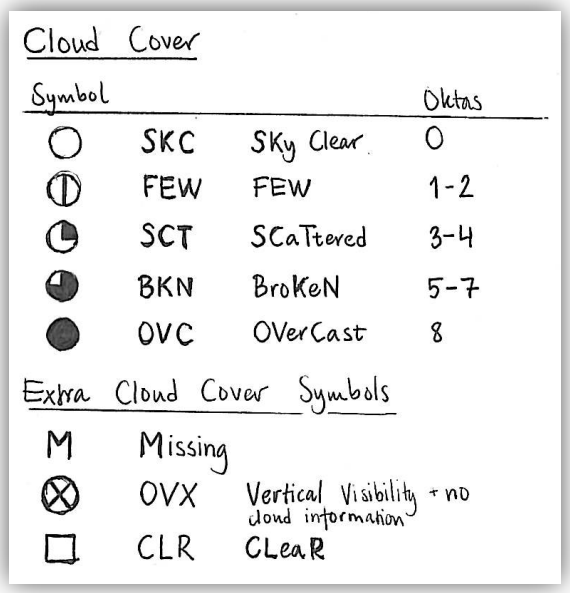


Figure 2.3. A list of symbols that describes cloud cover.

The "Present Weather" symbol describes the currently observed weather at the airport. There are 99 different symbols for describing all possible weather phenomena. [5]

The graphical representation of a wind barb describes both wind direction and velocity. The direction is illustrated by the rotation of the straight line that the barbs are connected to. A barb that is rotated by 90 degrees indicates that the wind is blowing *from* east [7]. The velocity is described by a composition of three possible barbs; the first being a half barb that indicates a velocity of 5kts; the second, a full barb that indicates a velocity of 10kts and a third being a flag that indicates a velocity of 50kts [5], see Figure 2.4.

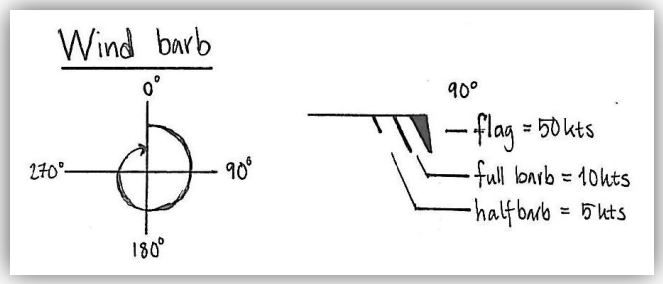


Figure 2.4. Description of velocity and direction of a wind barb.

The METAR standard is regulated in WMO – No. 49, Vol 2 and the regulators are WMO and ICAO. [8]

2.2.2 Terminal Aerodrome Forecast (TAF)

A Terminal Aerodrome Forecast (TAF) contains, to as great extent as possible, the same information as a METAR report with the difference that TAFs describe *forecasts* and METARs describe *observations*. The period of validity for each TAF varies depending on airport. [9]

2.2.3 SPECI

A SPECI report is a special weather report from an airport and it is issued when there has occurred a significant change in weather conditions. The content of a SPECI report is very similar to that of a METAR report. [10]

2.3 Data formats

In this chapter, we will describe data formats such as WXXM, SLD/SE and GML, all of which were used for implementing the Weather Client. AIXM was not used in our implementation but it is similar to WXXM and more research has been done in this area, thus we describe it in short as a reference.

2.3.1 Weather Information Exchange Model (WXXM)

The Weather Information Exchange Model (WXXM) is an XML standard for how weather data should be described. It consists of three parts, at different levels of abstraction:

- **The Weather Conceptual Model (WXCM)** – an abstract, high-level, conceptual model which describes the data content and relations between objects with UML diagrams as well as plain text.
- **The Weather Exchange Model (WXXM)** – the abstract, logical representation of the WXCM using UML diagrams.
- **The Weather Exchange Schema (WXXS)** – an XML Schema defining the coding of the weather data. Geography Mark-up Language (GML) is used here for implementation of geospatial specific aspects of the data, such as coordinates etc. [11]

In this report we will refer to the standard as WXXM and not differ between the different levels of abstraction.

The Weather Information Exchange Model was developed by Eurocontrol in partnership with NNEW [12].

2.3.2 Aeronautical Information Exchange Model (AIXM)

The Aeronautical Information Exchange Model (AIXM) is a data model for aeronautical information. It is very similar to WXXM and can be considered a sibling standard. However, while WXXM focus on how to represent weather data, AIXM focus on aeronautical information instead. Examples of such information are airspace, runways, routes etc. [13]

2.3.3 Styled Layer Descriptor (SLD) and Symbology Encoding (SE)

Together, Styled Layer Descriptor (SLD) [2] and Symbology Encoding (SE) [3] define an XML standard that is used for describing how the visualization of different layers in geospatial systems should look like. The portrayal of, in our case, weather data is done by writing rules describing what object we wish to visualize and what we want it to look like on the map.

Both SLD and SE are defined and provided by Open Geospatial Consortium. In this thesis, we will refer to the whole package as *SLD/SE*.

2.3.4 Geography Mark-up Language (GML)

Geography Mark-up Language is an XML grammar used for expressing geographical features that can be described using points, lines and polygons [14]. Carmenta Engine uses GML coordinates to plot geographic data.

GML is standardized by Open Geospatial Consortium (OGC).

3 Research Methods

This chapter describes the research strategy that was used in this thesis as well as an in-depth description of the root cause analysis method that was developed during the work.

3.1 Research Strategy

The research strategy adopted for this thesis was an *exploratory case study* [15] where we used the implementation of a proof-of-concept prototype in order to investigate the research problem and answer the following research questions:

- RQ1: What limitations of the SLD/SE standard affect functionality?
- RQ2: What limitations of the SLD/SE standard affect usability?
- RQ3: What limitations of the SLD/SE standard affect maintainability?
- RQ4: What limitations of the SLD/SE standard affect efficiency?

3.2 Research Design

The exploratory case study was conducted at Carmenta AB, a Swedish company that is one of actors in the geospatial domain. They would like to investigate the maturity of the SLD/SE and WXXM standards.

Figure 3.1 shows our research design: the tasks and their relationships. The first task in our research design was to determine the software quality requirements for both the prototype and the SLD files that would be used by the client application. The requirements were divided into the four categories: functionality, maintainability, usability and efficiency. More information about these qualities can be found in chapter 2.1.

The next step in our research design was, as can be seen in Figure 3.1, the implementation phase. The implementation itself consisted of; WXXM files containing the data to be visualized, SLD/SE files containing information about the visualization aesthetics and the Weather Client – a prototype for displaying weather data.

The requirements for the Weather Client were evaluated by us in the capacity of developers.

During the implementation phase, a number of issues were found and carefully documented. For each issue, we documented the following information:

- The functional requirements that the issue was related to
- What the result would have been if there were no problem
- Observed effects

After each issue was documented, an evaluation of the software quality requirements was performed. The evaluation consisted of eight questions regarding usability, maintainability and efficiency. Each question was a statement put in a positive manner, e.g. “It was easy to ...”. The answer to the statements was given on a scale ranging from 1 to 10, where 1 indicated *strong disagreement* and 10 indicated *strong agreement*. The evaluation was performed by us in the capacity of developers.

Evaluation of functional requirements was performed separately, not only because these requirements overlap the issues but also because they are boolean and thus cannot be answered with a number on a scale.

After the evaluation of the software quality requirements, a root cause analysis (RCA) was performed for each issue. By performing root cause analyses, we were able find the root cause(s) and learn more about the problems. RCA was also used as a *verification* method in cases where we had suspected that we already had found the root cause(s) but wanted to make sure that those suspicions were correct and that we had not overlooked anything. More information about the root cause analysis used can be found in chapter 3.3.

With the help of the documentation of the issues, software quality requirements evaluations, and root cause analyses we developed possible solutions to the problems. For some problems, there were several solutions while others had only one. The solutions were then evaluated based on the software quality requirements in order to determine if they would be better than the current solutions and in the case where there were several solutions to one problem, which solution would be best. The result was an improvement proposal for the SLD/SE visualization standard.

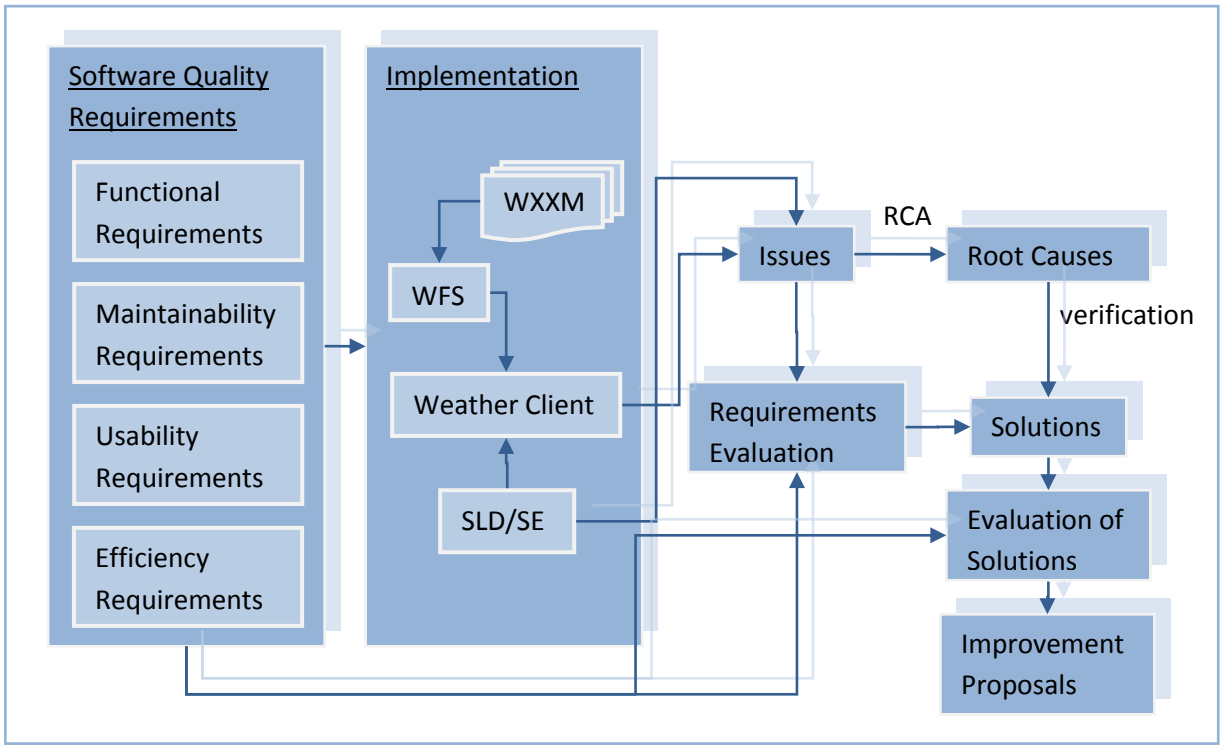


Figure 3.1. A graphical description of the research strategy where each node represents the output of each task.

3.3 Root Cause Analysis

Analysis of the problems that was found during the implementation phase was performed in-depth using a Root Cause Analysis (RCA) method. The aims for using an RCA method was to

- **Quickly** find the cause(s) of a given problem
- **Structure** the investigator's thoughts

- Further analyze the problem in order to
    - Learn more about the problem domain
    - Establish if there were **more** root causes
    - Whether the identified causes indeed were root causes or not

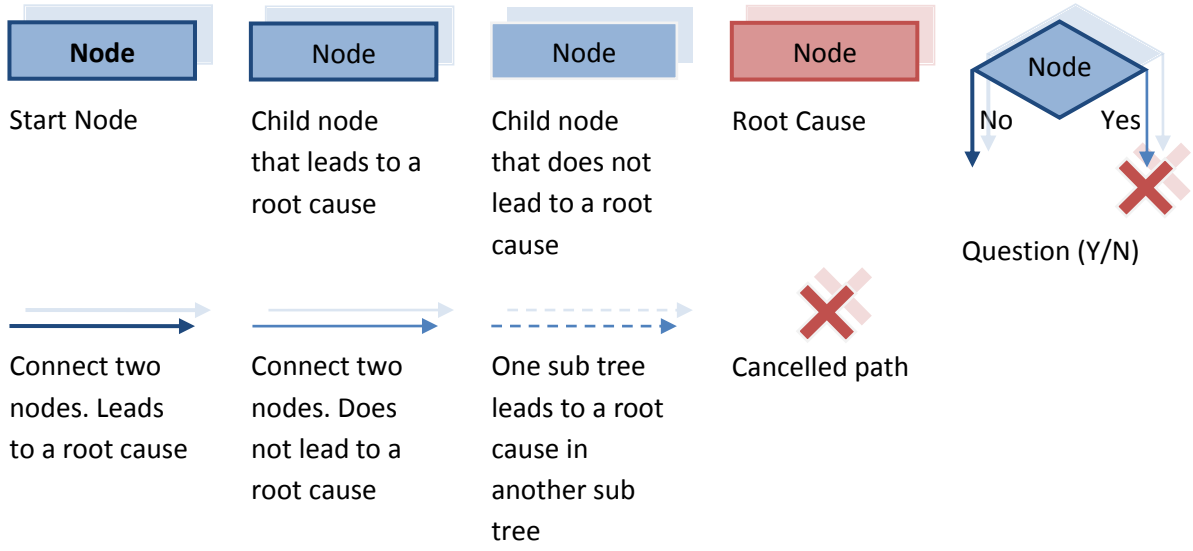
Since most RCA methods have their focus on business goals and accident analysis, finding one that suited this case study was not easy. The method that most resembled what we were looking for was Toyota's "*5 Whys*" analysis method [16]. However, there were strong concerns that this method would not provide us with enough information to solve the problems. We also felt the risk of stopping at symptoms instead of root causes.

In the end, a new root cause analysis method was developed during the project with inspiration taken from the "5 Whys" method. The "*Extended 5 Whys*" takes an *iterative* approach at finding root causes within *software* problems. It enables the investigator to broaden his knowledge about the problem domain and promotes thinking outside the box.

3.3.1 Step-By-Step Description of the Extended 5 Whys Method

1) Find your first question. Finding the question that will elicit the root cause(s) could be a bit tricky so think about possible alternatives and then decide on how to phrase your question regarding the fault you have found. Add this as your start node.
2) Find an answer to your question. Add this node under the first one and connect them with a directed line, from parent to child.
3) Are there any more answers? Add them next to the answer in 2). This should make your graph into a tree.
4) Take your first answer and turn it into a why-question. E.g. the answer "The glass was filled with water" can be turned into "Why was the glass filled with water?". Find the answer(s) and write them down as child node(s). Repeat this for every bottom node until you have found the root cause(s).
5) If you at some point feel like further investigating a node, you may add a new Yes/No question. The "No" track should be to the left and "Yes" to the right.
    - Cross-mark the wrong answer
    - Continue at the correct trail by asking a new question. This question should be related to the parent node of the Y/N question node.
6) If you believe that a trail is leading you astray or that it will not yield any interesting information that will lead you to a root cause, cancel it by cross-marking it.
7) If a trail has been cross-marked, backtrack to the parent node and repeat step 4, 5 or both.
8) If one trail will lead you to the same root cause as found within another sub tree, then indicate this dependency with a dotted, directed line.
9) Mark all nodes leading down to a root cause with a bold border.
10) Paint all lines that lead down to a root cause in a darker colour.
11) Paint all root causes red to further articulate the result.

3.3.2 Symbols and Notations



3.3.3 Issues with 5 Whys and how Extended 5 Whys handles them

The greatest criticism directed towards the "5 Whys" method concerns its inability to analyze the problems to a sufficient depth. Below is a list of reasons why this RCA method should not be used [17]. For each reason, there is a description of what countermeasures our extended version of the 5 Whys analysis method takes.

1) **The risk of stopping at a symptom, rather than at the actual root cause.**[17]

This problem depends on the investigator's skills, knowledge and willingness to perform an in depth analysis. By having a developer as an investigator, one can presume that he will have a greater interest in finding root causes and a responsibility for solving them. The ability to gain knowledge about the problem and problem domain through questions (other than why-questions) promotes thinking outside the box. With greater understanding comes greater ability to reach lower-level root causes.

2) **The investigator lacks knowledge about the problem domain and is not inclined to further broaden his knowledge.**[17]

When the investigator feels like he is missing something or that some part of the problem has not been investigated, he is able to add further questions in that area. The iterative way to find root causes will open up the investigator's mind to new ideas and force him to find out more about the problem domain.

3) **There is no support to help the investigators to ask the right questions.**[17]

This problem is very hard to counter. Sometimes it can be hard to find the right start questions. It is then recommended that one first tries different possibilities and after thinking about them decides which one should be the start node. Also, asking the right questions in the middle of the RCA can be hard and this is something that will only come from experience.

4) The risk of two different teams ending up with different cases for the same problem. [17]

The investigator is able to broaden his knowledge about the problem domain by adding several answers to one question, or adding new information through y/n-questions. This should lead to similar looking trees and same root causes, even if the analysis was performed by different individuals.

5) The risk of only finding only one out of many root causes. [18]

In this extended version, the investigator is able to add several solutions to a question by making the answers into sub-trees. Each trail will be followed in depth and will thus point out one or several root causes.

4 The Weather Client - A Prototype for Evaluating the Usage of SLD/SE

This chapter contains the requirements specification for the Weather Client, consisting of functional requirements that include the design of the Graphical User Interface (GUI), usability, maintainability and efficiency requirements. It also contains a description of the implementation and an evaluation of the software quality requirements.

4.1 Software Quality Requirements

4.1.1 Functional Requirements

The functional requirements for the Weather Client are divided into the seven categories; menu bar, toolbar, Panel: Map, time slider, Panel: Area, Panel: Layers and Panel: METAR. Each category contains a number of functional requirements that are arranged into a hierarchical list. Each element in the list has an associated bullet point in a particular colour that indicates the requirement's priority. Red indicates high priority; orange indicates medium priority and green indicates low priority. Since the focus of the Weather Client is to create a proof of concept prototype for visualizing weather information in geospatial systems using SLD files, only the most basic features have been given a high priority. The time slider, for instance, is a very interesting feature which would be nice to have in a client; however, it is not necessary in order to evaluate SLD/SE and has thus been given a medium priority. The features which have been given a low priority are mostly of a cosmetic nature, meaning that they will enhance the prototype's user experience. However, we can manage well without them because they do not affect the map visualization.

Menu bar

- Exit
- About

Toolbar (for quick buttons)

- Zoom in
- Zoom out
- Move (hand tool)

Panel: Map

- Display a map
- Display legend
- Scale bar
- Altitude
    - Display current altitude
    - Change altitude

Time slider

- Manually change between observations
  - Step with buttons
  - Step by moving the indicator with the mouse
- Playback function
  - Play
  - Pause
  - Stop
  - Fast forward
- Display the date of the selected observation

Panel: Area

- Drop-down menu with predefined areas of the map

Panel: Layers

- Select data source for winds, clouds, jets and CATs
- Select SLD for all of the above
- Toggle the visibility of the weather phenomenon above on and off.

Panel: METAR

- Toggle METAR on/off
- Select a predefined METAR SLD file from the drop-down menu
  - Handle online SLD files
  - Handle local SLD files
- Add new SLD files
- Show all layers that are found within a selected SLD file
- Show the SLD layers that are visible on the map. The visible layers are a subset of the available layers
- Move an available layer to the list of visible layers
- Remove a visible layer from the list of visible layers (will not affect the list of available layers)
- Move a visible layer up in the hierarchy
- Move a visible layer down in the hierarchy
- Display raw METAR for a selected airport

4.1.1.1 Graphical User Interface Design

Inspiration for this GUI has been taken from similar applications such as NOAA's "METARs Java Tool" [5] and Google Earth [19].

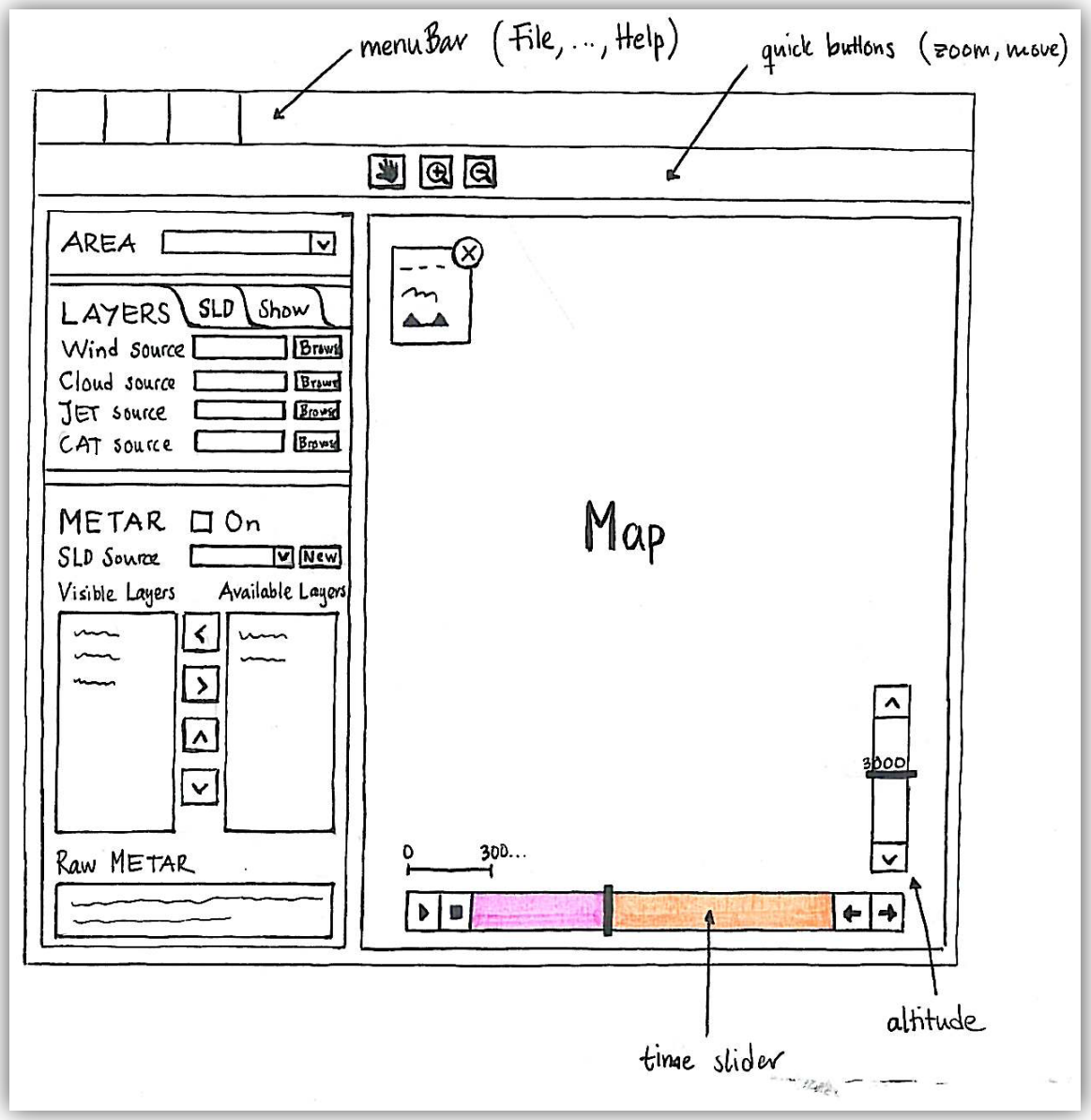


Figure 4.1. The first sketch of the graphical user interface.

Menu bar
Program commands such as “File → Exit” and “Help → About Weather Client” can be found on the menu bar.

Quick buttons
This is where the user will gain quick access to features such as *move*, *zoom in* and *zoom out*.

Map
Default setting for the map is to show a plain map without any aviation specific layers. Default area is North America.

Legend
Displays the meaning of the different symbols used in the map.

Scale bar
The scale bar displays the current scale for the map.

Altitude
The vertical scrollbar to the right is used for selecting an altitude. The number over the horizontal indicator shows the selected altitude for winds, clouds, jet streams etc and is measured in meters over sea level. The *up* and *down* buttons can be used to move the indicator. Default value is set to normal cruising altitude.

Time slider
Using the time slider, the user can browse present and old weather data (and hopefully also forecasts). Default value is *LATEST* because the user will almost always start by viewing the most recent information. Different colours should be used to indicate past, present and future. The user should be able to see weather over time by using the playback function. The playback function has the following features; *play*, *pause* and *stop*.

By using the left and right arrows, the user is able to step from observation to observation. Holding the mouse pointer over the indicator will display the date of the current observation. The indicator can also be moved along the time slider in order to select an observation.

Area
By selecting different alternatives from the drop-down menu, the user can switch between different map views such as Northern America, Europe etc.

Layers
This is a tab-window where the user can choose what layers to view, where each layers will get their data and what SLD to use for styling.

On the first tab, the user can add a source to each layer. When pressing “*Browse...*” the user will be presented with a list of possible sources to choose from.

The second tab enables the user to choose SLDs in the same manner as with the data source.

Checking the “Show All” checkbox on the third tab will display all layers that have both source and SLD attached to them. Unchecking the box will remove the layers from the map. The user is also able to individually select which layers he wants to view.

METAR

In this sub window, the user is able to manage METAR data. The checkbox at the top toggles the METAR layer on and off. The user can choose an SLD source from the drop down menu or he may add a new using the "*New*" button. When an SLD has been chosen, the layers found within the SLD file will be visible in the "*Available Layers*" listbox and those layers that are visible on the map are found in the "*Visible Layers*" listbox. Visible layers are a subset of available layers. The user can use the *left* and *right* buttons to move layers to, respectively from the list of visible layers. The *up* and *down* buttons are used to organize the visible layers. The layer at the bottom of the list will be painted first on the map and the layer in the top of the list will be painted last. This way, one layer is able to overwrite another layer.

At the bottom of the METAR window is a text field where the raw METAR for a selected airport id is shown.

4.1.2 Usability Requirements

Answers to the following statements are given on a scale from 1 to 10, where 1 indicates *strong disagreement* and 10 indicates *strong agreement*.

- It is easy to understand what features the Weather Client supports.
- It is easy to learn how to operate the Weather Client
- The overall layout of the prototype is visually appealing

4.1.3 Maintainability Requirements

Answers to the following statements are given on a scale from 1 to 10, where 1 indicates *strong disagreement* and 10 indicates *strong agreement*.

- A user with a similar background as the author can easily understand the written code
- It is easy to identify faults within the code
- It is easy for a user with a similar background as the author to make changes/updates to the program

4.1.4 Efficiency Requirements

Answers to the following statements are given on a scale from 1 to 10, where 1 indicates *strong disagreement* and 10 indicates *strong agreement*. As a reference, responsiveness of the application should be similar to that of Google Earth [19].

- Zooming in and out of the map does not take an unreasonable amount of time
- Moving the map using the hand tool does not take an unreasonable amount of time
- Toggling on and off layers can be smoothly performed
- The playback function of the time slider runs smoothly at a constant pace

4.2 Implementation

The implementation of the Weather Client is written in C# and a UML diagram that describes the overall structure of the system is available in Figure 4.2. The system can be divided into four different parts; the Weather Client, Carmenta Engine, the configuration file and the SLD/SE files along with their respective providers.

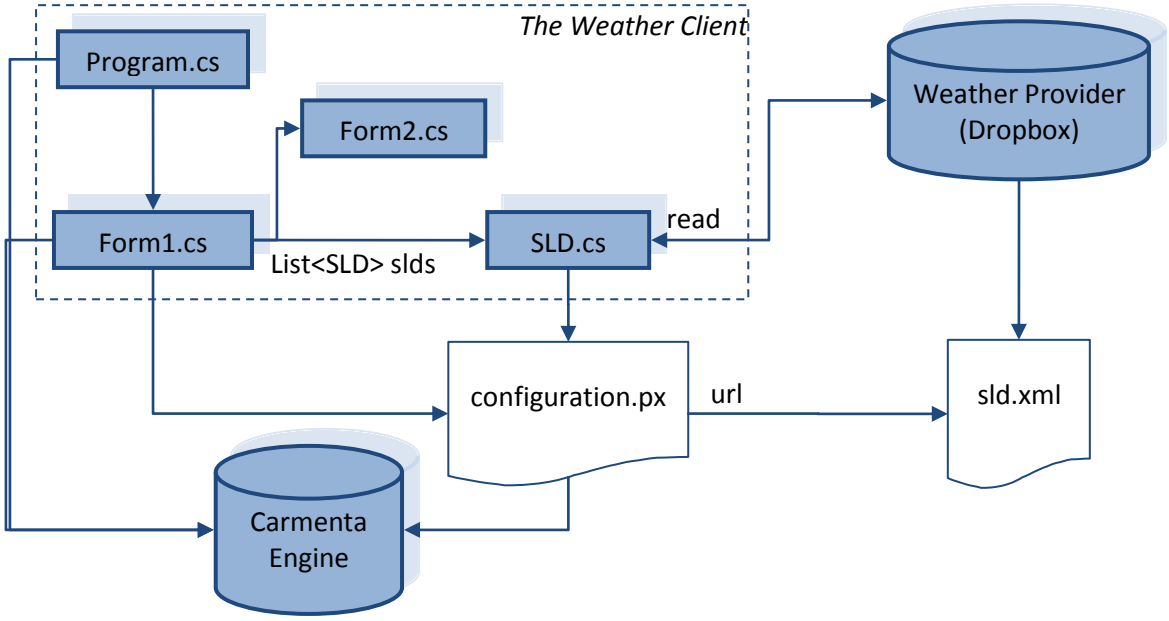


Figure 4.2. A UML diagram describing the composition of the entire system.

4.2.1 The Weather Client

The Weather Client is composed of the following files; Program.cs, Form1.cs, Form2.cs and SLD.cs.

The purpose of *Program.cs* is to make some initializations related to Carmenta Engine as well as running Form1.cs.

Form1.cs is the main body for the Weather Client. All GUI interactions and updates are performed here as well as communication with Carmenta Engine. A reference to the configuration file (configuration.px) needed by Carmenta Engine is found here. Furthermore, Form1 holds a list of all SLD/SE files that are entered into the program.

Each instance of the *SLD* class contains:

- URL – the URL to the SLD/SE file
- available_layers – the layers within the SLD/SE file
- visible_layers – this SLD's layers which are present in the configuration file (.px)

We parse the SLD/SE file in order to determine what layers are available for visualization and those layers are added to the list of available_layers. The SLD class also handles all modification of the configuration file. Such modifications are changing what layers should be visible on the map and in which order they should be visualized (the order of the layers).

Form2.cs is a graphical frame used only for input of new SLD/SE files.

4.2.2 Carmenta Engine

Carmenta Engine is one of Carmenta AB's products [20] and given a configuration file containing data source and styling document, it provided us with a map to display in the Weather Client.

4.2.3 Configuration

In order to receive a map from Carmenta Engine, a configuration file is needed. In our project we made a configuration with two layers which the Weather Client is able to switch between. The first layer is a regular Web Map Service (WMS) layer that is used to visualize a map containing the following layers; Countries, Borders, Lakes, Rivers and Cities – in other words, a regular map.

The second layer in the configuration file is used for visualization of an SLD/SE file. As stated earlier, the Weather Client is able to modify this layer in order to load different SLD/SE files but in our case the default file is an SLD/SE file containing styling for METAR reports.

4.2.4 SLD/SE files and Weather Provider

There was very little WXXM data available but we did manage to get a hold of some METAR reports from one of the participants in the OWS-8 project [21]. However, the server that hosted the reports was not compatible with Carmenta Engine thus leading us to a workaround solution that simulated the use of a weather provider. The solution was to retrieve the METAR reports from the server and store them in a Dropbox [22]. Since the file has a public URL, the Dropbox successfully worked as a weather provider.

4.2.5 The Graphical User Interface

The resulting graphical user interface of the Weather Client's main frame (Form1.cs) is shown in Figure 4.3. The resulting design of Form2.cs can be found in Figure 4.4.

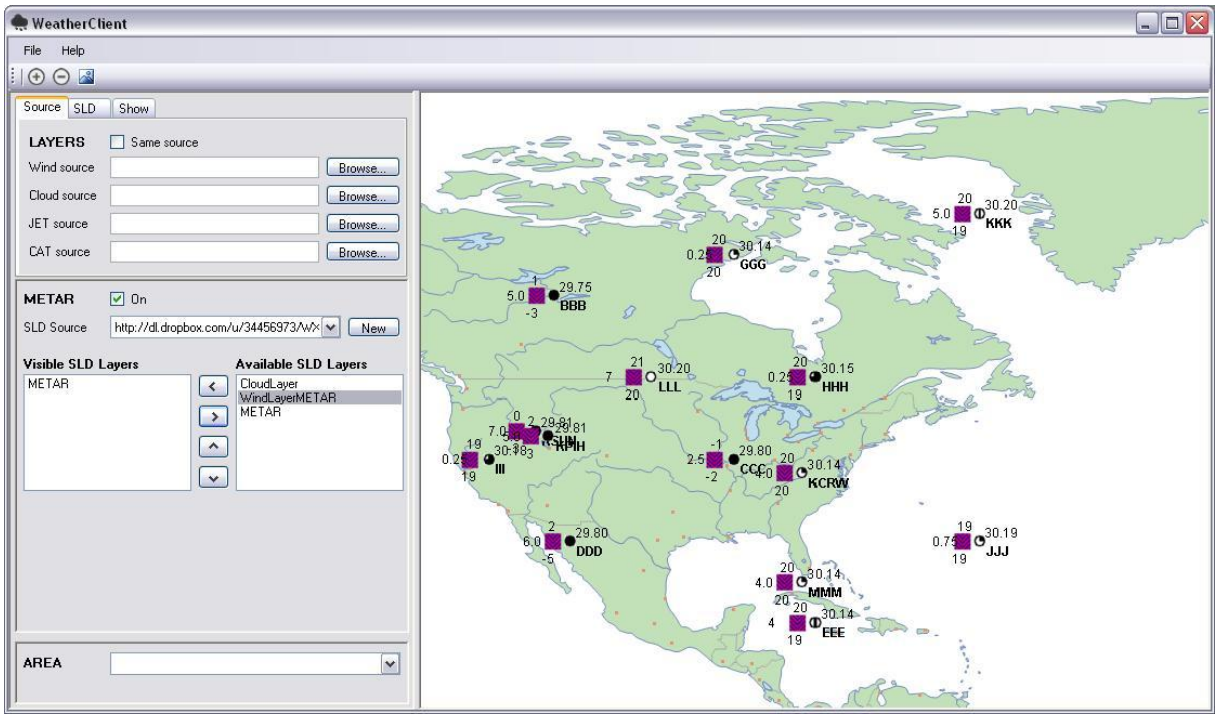


Figure 4.3. The graphical user interface of the Weather Client.

The overall structure of the GUI is very similar to that of the initial design; see Figure 4.1. There are however some changes. The panel "Layers" is still present but currently unused due to lack of weather data to style. The "METAR" panel handles everything that concerns the visualization of our METAR reports. The "Area" panel is currently unused. The area to the right displays the map.

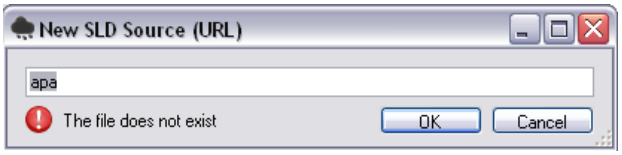


Figure 4.4. The dialog for adding new SLD/SE files. A warning is displayed when the file does not exist.

4.3 Evaluation of the Software Quality Requirements

A list of all the functional requirements and their status (implemented or not) is available in Table 4.1. The main reason why some of the functions were not implemented was that they were considered unimportant for the purpose of the thesis, namely the evaluation of the SLD/SE standard. For instance, the map was vital for the project but legend, scale bar and altitude was not.

The time slider was not implemented due to lack of time as well as not being the focus of the thesis.

The two panels "Area" and "Layers" were given graphical components but their functionality was not implemented. The "Area" panel was considered unnecessary because it was not vital for the success of the project. The "Layers" panel was not fully implemented due to the lack of WXXM data (as stated earlier we were only able to get a number of METAR reports).

The ability to handle local SLD/SE files was not added due to lack of time.

The portrayal of raw METAR data was not added since it was not a part of the map visualization and thus not a vital function.

Table 4.1. Evaluation of the Weather Client's functional requirements.

Category	Functional Requirement	Priority	Implemented (Y/N)
Menu bar	Exit	High	Yes
	About	Medium	No
Toolbar	Zoom in	High	Yes
	Zoom out	High	Yes
	Move (hand tool)	High	Yes
Panel:Map	Display a map	High	Yes
	Display legend	Medium	No
	Scale bar	Low	No
	Altitude – display current altitude	Low	No
	Altitude – Change altitude	Low	No
Time slider	<all requirements>	Medium	No
Panel:Area	Drop-down menu with predefined areas of the map	Medium	Graphical component is present. Functionality is not
Panel:Layers	<all requirements>	High	Graphical component is present. Functionality is not
Panel:METAR	Toggle METAR on/off	High	Yes
	Select a predefined METAR SLD file from the	High	Yes

	drop-down menu – Handle online SLD files		
	Select a predefined METAR SLD file from the drop-down menu – Handle local SLD files	Medium	No
	Add new SLD files	High	Yes
	Show all layers that are found within a selected SDL file	High	Yes
	Show the SLD layers that are visible on the map. The visible layers are a subset of the available layers	High	Yes
	Move an available layer to the list of visible layers	High	Yes
	Remove a visible layer from the list of visible layers (will not affect the list of available layers)	High	Yes
	Move a visible layer up in the hierarchy	High	Yes
	Move a visible layer down in the hierarchy	High	Yes
	Display raw METAR for a selected airport	Medium	No

A list of all the usability, maintainability and efficiency requirements is found in Table 4.2. Since the two panel's "Area" and "Layers" functionality is not implemented yet they are present in the GUI, it is not obvious to the user what features the Weather Client supports. The GUI itself is visually appealing and easy to operate. The written code is structured and there are comments describing all functions, thus facilitating code changes and updates.

The Weather Client does not fulfil the efficiency requirements concerning map updates. Each update takes a considerable amount of time, far greater than that of Google Earth. We were unsuccessful in determining the exact source of the lag but we are fairly confident that the SLD/SE standard is not at fault.

Table 4.2. Evaluation of the Weather Client's usability, maintainability and efficiency requirements

Requirement	Score (1-10)
It is easy to understand what features the Weather Client supports	5
It is easy to learn how to operate the Weather Client	8
The overall layout of the prototype is visually appealing	8
A user with a similar background as the author can easily understand the written code	7
It is easy to identify faults within the code	7
It is easy for a user with a similar background as the author to make changes/updates to the program	7
Zooming in and out of the map does not take an unreasonable amount of time	3-5
Moving the map using the hand tool does not take an unreasonable amount of time	3
Toggling on and off layers can be smoothly performed	4
The playback function of the time slider runs smoothly at a constant pace	Not implemented

5 Visualization of METAR Data Using SLD/SE

This chapter describes the software quality requirements for the SLD/SE-file that was used to visualize METAR data by the Weather Client. It also contains documentations of all the issues encountered during the implementation of the prototype, as well as requirements evaluation and root cause analysis for each issue.

5.1 Software Quality Requirements

The software quality requirements are divided into four categories; functional, usability, maintainability and efficiency requirements. The functional requirements are boolean, which means that they are either fulfilled of not fulfilled. However, the other groups of software quality requirements are of a more subjective nature and thus cannot be considered as boolean. The success rate of each requirement is instead measured on a scale from 1 to 10.

5.1.1 Functional Requirements

In order for the user to be content with the visualization of METAR data, all requirements with *high* priority must be fulfilled. *Medium* ranked requirements are not as necessary as those with a high priority, but fulfilling the medium ranked requirements will greatly increase the appearance and increase the usability of the application. Requirements given a *low* priority do not need to be fulfilled since they concern minor cosmetic details and does not affect the over-all readability for the visualization.

High priority

- Display words/numbers
- Display the correct symbol from a list of symbols describing either cloud cover or weather symbols
- Display wind barbs

Medium priority

- Automatically toggle viewing of the METAR composites on/off depending on map scale. E.g. the user would not like to view all METARs when zoomed out in order to view the entire world. Without proper toggling, the Earth would be covered entirely by METAR data and one would be unable to identify anything.
- View METAR data over time. E.g. Data from one time period should be added to one layer, while data from another point in time should be added to a different layer. This enables the user to switch between layers and observations.
- Add colour to cloud cover symbols

Low priority

- Add colour to words/numbers

5.1.2 Usability Requirements

Answers to the following statements are given on a scale from 1 to 10, where 1 indicates *strong disagreement* and 10 indicates *strong agreement*.

- It was easy to understand how to implement the feature
- The graphics are clear and looks like the description
- It did not take long to implement the feature

5.1.3 Maintainability Requirements

Answers to the following statements are given on a scale from 1 to 10, where 1 indicates *strong disagreement* and 10 indicates *strong agreement*.

- It is easy to identify code segments that contains the feature
- It is easy to identify faults within the code
- A user with a similar background as the author can easily understand the written code
- It is easy for a user with a similar background as the author to make changes/updates to the feature

5.1.4 Efficiency Requirements

Answers to the following statements are given on a scale from 1 to 10, where 1 indicates *strong disagreement* and 10 indicates *strong agreement*.

- There is no unnecessary duplication of data
    - By reducing the size of the XML document, one will also reduce transmission times

5.2 Issues with Styled Layer Descriptor/Symbology Encoding

5.2.1 First Issue – Overwritten Words/Numbers

5.2.1.1 Related Functional Requirements

- Display words/numbers (High priority)
- View METAR data over time (Medium priority)

5.2.1.2 Wanted Result

Functional Requirements: Display the following information in a clear and structured manner.

- Airport identifier
- Altimeter
- Dewpoint temperature
- Visibility
- Temperature

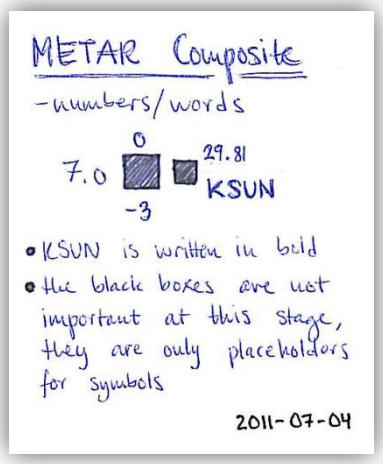


Figure 5.1. Describes how text-based information should be positioned and formatted in METAR composites

The sketch in Figure 5.1 describes how the result should look like for airport KSUN. For a more detailed description of how the graphical representation of a METAR report should look like, see Chapter 2.2.

Software Quality Requirements: Should consort with the previously stated requirements.

5.2.1.3 Observed Effects

Visual Result: As appears in Figure 5.2, the METAR composites for KSUN and KPIH appears to be overwritten by other data. In this example, we can see that KSUN's altimeter, temperature, visibility and dewpoint temperature are affected. It is also evident that for the KSUN airport, the only attribute that appears unaffected by this issue is the airport identifier.

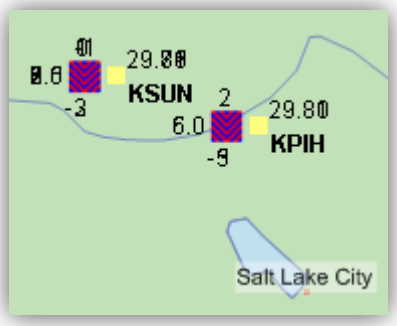


Figure 5.2. Overwritten METAR data

Software Quality Requirements:

Table 5.1. Evaluation of the software quality requirements for "overwritten words/numbers"

Requirement	Score (1-10)
It was easy to understand how to implement the feature	8
The graphics are clear and looks like the description	2
It did not take long to implement the feature	7
It is easy to identify code segments that contains the feature	7
It is easy to identify faults within the code	4
A user with a similar background as the author can easily understand the written code	7
It is easy for a user with a similar background as the author to make changes/updates to the feature	7
There is no unnecessary duplication of data	10

5.2.1.4 Root Cause Analysis

From the root cause analysis diagram in Figure 5.3 we learn that the reason why we suffer from overwritten words/numbers is that the server which provides the METAR reports does not handle time stamps – nor does our implementation. Instead of receiving only one report from each airport at a point in time, we get a number of reports from the same airports but with different time stamps. When the client side server visualizes the reports the result is overwritten information.

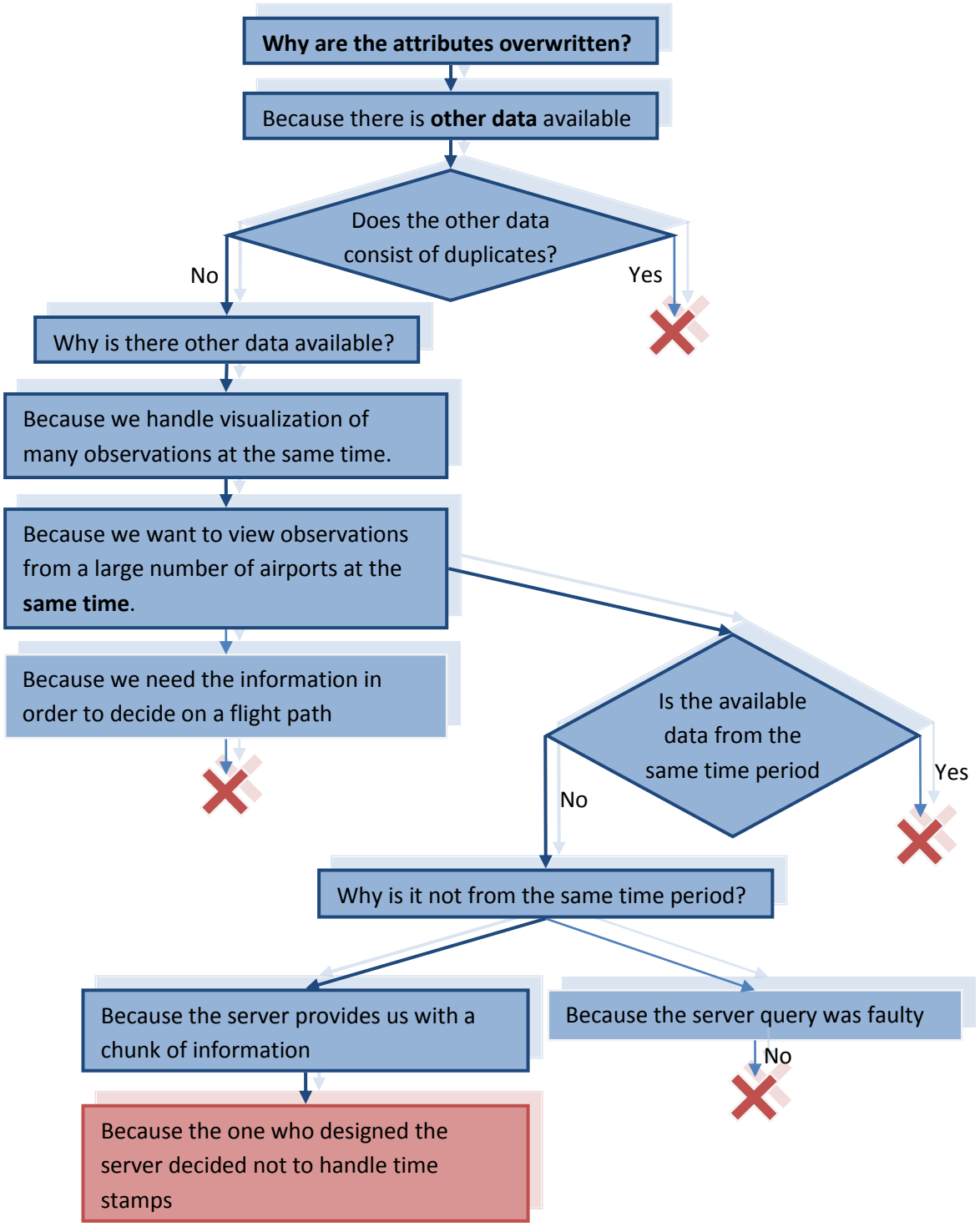


Figure 5.3. The root cause analysis diagram for the issue "overwritten words/numbers".

5.2.2 Second Issue – Mapping of Different Values onto Different Symbols

5.2.2.1 Related Functional Requirements

- Display the correct symbol from a list of symbols describing either cloud cover or weather symbols (High priority)

5.2.2.2 Wanted Result

Functional Requirements: Depending on the data content, a matching symbol should be used for each observation in order to describe the current circumstances in weather and cloud cover. For instance, if one airport's cloud cover is set to "overcast" then this should be depicted by a round, coloured circle but if that attribute is set to "clear" the centre of the circle should be left clean. A list of all cloud cover symbols is found in Figure 2.3 on page 10.

Software Quality Requirements: The written code must satisfy the previously specified software quality requirements.

5.2.2.3 Observed Effects

Visual Result: The result does to some extent consort with the visual requirements. As can be seen in Figure 5.4, cloud cover is visualized and the weather is not. Instead of displaying a symbol that describes the observed weather, a "dummy point" is used. The reason why we use a dummy point instead of implementing all the weather symbols is because there are 99 symbols and implementing all of them would be too time consuming. A full implementation of the cloud cover is considered enough to test the concept.

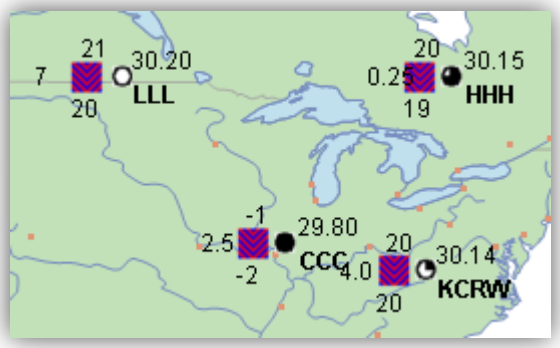


Figure 5.4. METAR observations for four airports. Cloud cover is depicted as a circle and the observed weather is depicted by a dummy point.

Implementation Result: Each symbol must be added as a separate rule with an attached filter as an if-clause. In order to visualize all possible cloud cover symbols, the user need to write seven rules with similar content as the example below.

The tag <Filter> is used to identify what should be visualized. In our case we want the cloud cover attribute found within a WXXM-file to be equal to "CLEAR". The XML path to the attribute is given within a <PropertyName> tag.

Min- and Max-scale denominator denotes during what level of zoom the object should be visible. The path to the object itself, along with various styling parameters is found within a <PointSymbolizer> tag. In our case, we have decided to let the cloud cover symbol be the centre of the METAR composite and thus we have no displacement. The only styling parameter used for cloud cover is <Size> which is set to 10px.

Example. Code for visualizing the symbol for clear cloud cover.

```
<se:Rule>
   <ogc:Filter>
      <ogc:PropertyIsLike wildcard="*" singleChar="#" escapeChar="!">
         <ogc:PropertyName>avwx:SurfaceReport/
                  avwx:aerodromeWxObservation/om:Observation/om:result/
                  avwx:cloudCondition/wx:CloudCondition/wx:cloudAmount
         </ogc:PropertyName>
         <ogc:Literal>CLEAR</ogc:Literal>
      </ogc:PropertyIsLike>
   </ogc:Filter>
   <sld:MinScaleDenominator>1000.0</sld:MinScaleDenominator>
   <sld:MaxScaleDenominator>16000.0</sld:MaxScaleDenominator>
   <se:PointSymbolizer>
      <!-- This is an "anchor point" and image for "Cloud Cover" -->
      <se:Graphic>
         <se:ExternalGraphic>
            <se:OnlineResource xlink:type="simple"
                           xlink:href="http://dl.dropbox.com/u/34456973/
                           CloudCover/SKC.PNG"/>
            <se:Format>image/png</se:Format>
         </se:ExternalGraphic>
         <se:Size>10</se:Size>
      </se:Graphic>
   </se:PointSymbolizer>
</se:Rule>
```

Quality Requirements:

Table 5.2. Evaluation of the software quality requirements for "mapping of values…"

Requirement	Score (1-10)
It was easy to understand how to implement the feature	4
The graphics are clear and looks like the description	10
It did not take long to implement the feature	1
It is easy to identify code segments that contains the feature	3
It is easy to identify faults within the code	4
A user with a similar background as the author can easily understand the written code	5
It is easy for a user with a similar background as the author to make changes/updates to the feature	6
There is no unnecessary duplication of data	3

5.2.2.4 Root Cause Analysis

From the root cause analysis in Figure 5.5, we learn that the implementation time was long due to a steep learning curve caused by a, to us, unusual way of conduct. Furthermore we learn that the root cause is lack of support in the SLD/SE standard for duplicate-free mapping of symbols. The standard also lacks the ability to group items together.

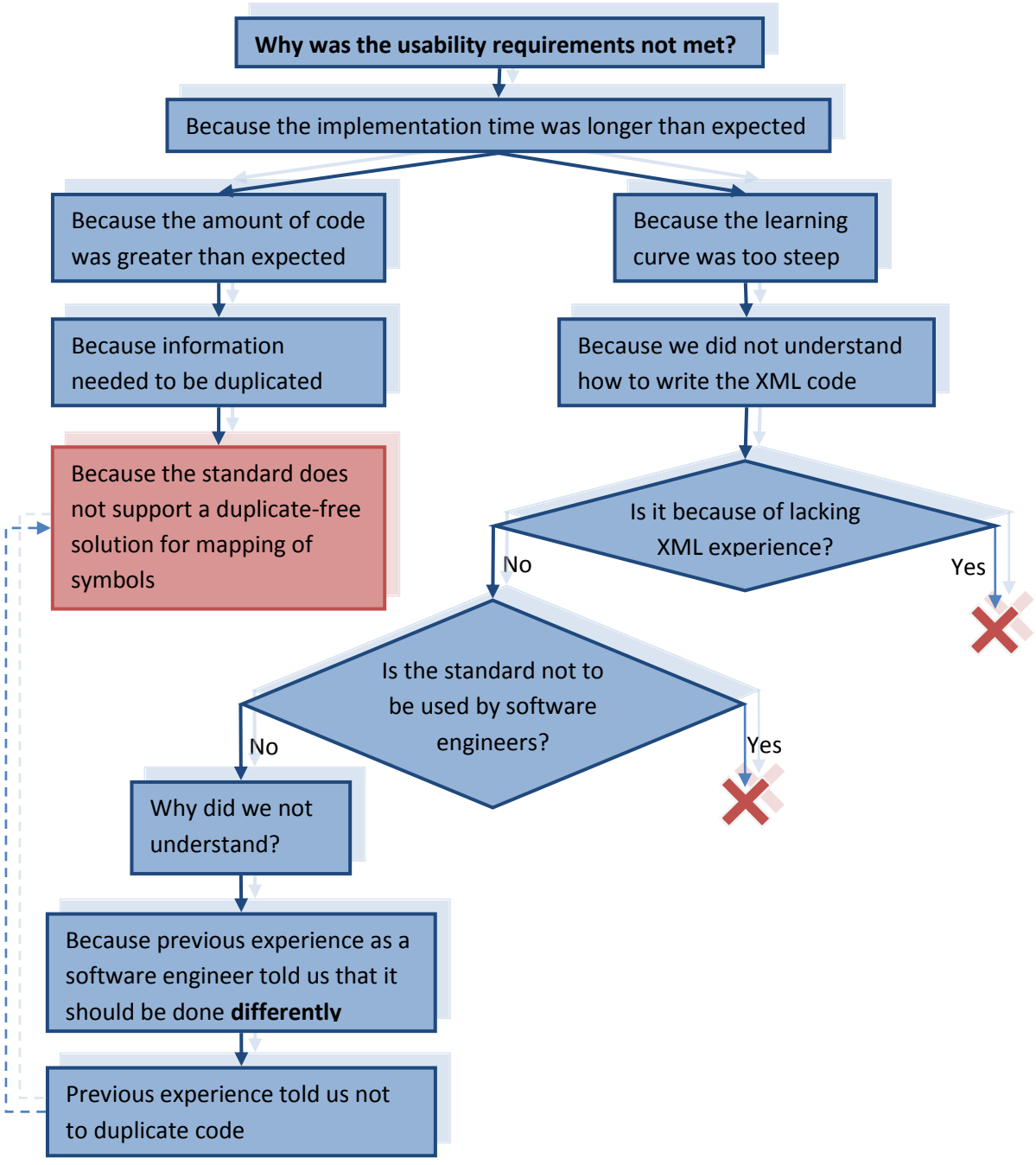


Figure 5.5. The root cause analysis diagram for the issue "mapping of different symbols onto different values".

5.2.3 Third Issue – Rules within Rules

5.2.3.1 Related Functional Requirements

- Display words/numbers (High priority)
- Display the correct symbol from a list of symbols (High priority)

5.2.3.2 Wanted Result

Functional Requirements: There may be some cases when it is important to use two or more rules for determining if an item should be visualized or not. We must therefore be able to add multiple rules for an item.

Software Quality Requirements: Should comply with previously stated requirements.

5.2.3.3 Observed Effects

Visual Result: None

Implementation Result: It is possible to achieve the desired result where point P_1 is visualized if and only if both rules R_0 and R_1 are true. This is done by adding R_1 as an and-clause to R_0, as can be seen in the example below.

The example does not show any object-specific details like for example, how it should be styled. The only tag that is relevant in this case is the <Filter> tag which is used to identify the object and when we want to visualize it. The two rules R_0 and R_1 are given using a <PropertyIsLike> tag and they can be found within an <And> tag, which is used to indicate that both rules must be true for the filter to evaluate to true. In our example, the first rule R_0 states that the input file should contain METAR data and the second rule R_1 states that the object we want to style is cloud condition set to "CLEAR". In other words, the object will be visualized if and only if the input file contains METAR data and there exist an observation(s) containing cloud cover=CLEAR.

Example. Code for using two rules to visualize the cloud cover symbol "CLEAR".

```
<se:Rule>
   <ogc:Filter>
      <ogc:And>
         <ogc:PropertyIsLike wildcard="*" singleChar="#" escapeChar="!">
            <ogc:PropertyName>avwx:SurfaceReport/gml32:name
            </ogc:PropertyName>
            <ogc:Literal>METAR*</ogc:Literal>
         </ogc:PropertyIsLike>
         <ogc:PropertyIsLike wildcard="*" singleChar="#" escapeChar="!">
            <ogc:PropertyName>avwx:SurfaceReport/
                    avwx:aerodromeWxObservation/om:Observation/om:result/
                    avwx:cloudCondition/wx:CloudCondition/wx:cloudAmount
            </ogc:PropertyName>
            <ogc:Literal>CLEAR</ogc:Literal>
         </ogc:PropertyIsLike>
      </ogc:And>
   </ogc:Filter>
   ...
<se:Rule>
```

Software Quality Result:

Table 5.3. Evaluation of the software quality requirements for "rules within rules"

Requirement	Score (1-10)
It was easy to understand how to implement the feature	6
The graphics are clear and looks like the description	-
It did not take long to implement the feature	6
It is easy to identify code segments that contains the feature	9
It is easy to identify faults within the code	4
A user with a similar background as the author can easily understand the written code	6
It is easy for a user with a similar background as the author to make changes/updates to the feature	7
There is no unnecessary duplication of data	3

The software quality requirement concerning not duplicating information is not met in the case where two or more items have one rule in common.

Let us say that we have two points, P_1 and P_2 that we want to visualize. There are three rules, R_0, R_1 and R_2. Point P_1 will be visualized if and only if R_0 and R_1 are true. P_2 will be visualized if and only if R_0 and R_2 are true. It would be beneficial to use a hierarchical ordering of rules in order to reduce redundancy. Figure 5.6 illustrates the scenario where we currently have redundancy.

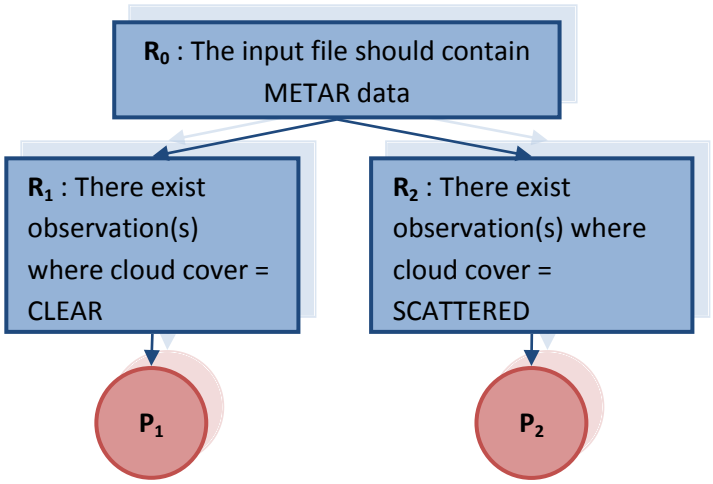


Figure 5.6. Hierarchical ordering of rules for cloud cover visualization.

5.2.3.4 Root Cause Analysis

From the root cause analysis diagram in Figure 5.7 we learn that the SLD/SE standard does not support the use of rules within rules.

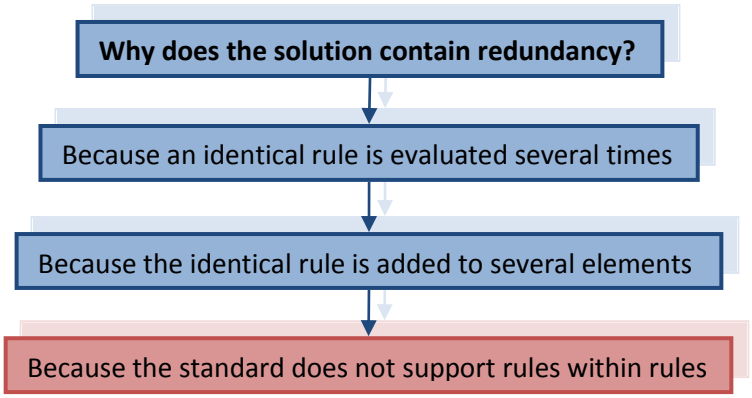


Figure 5.7. The root cause analysis diagram for the issue "rules within rules".

5.2.4 Fourth Issue – Scale Dependent Auto Toggling

5.2.4.1 Related Functional Requirements

- Automatically toggle viewing of the METAR composites on/off depending on map scale

5.2.4.2 Wanted Result

Visual Result: METAR composites should only be visible when the map's scale is appropriate. Exactly what is appropriate is hard to determine since we are unable to test this feature, but there exist a distance where METAR composites should be removed from the map in order not to risk having the map overflowing by composites.

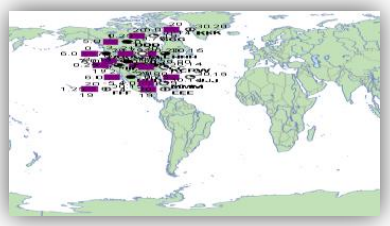


Figure 5.8. METAR visualization without scale dependent auto toggling

5.2.4.3 Observed Result

Visual Result: When zooming out, the METAR composites are not toggled of and that leads to a map covered by composites to the extent where it is hard to distinguish any other features, see Figure 5.8.

Implementation Result: The feature is implemented using the Min- and Max Scale Denominator tags. The METAR composites should only be visible within these under and upper limits. [3]

Example. Implementation of scale dependent auto toggling.

```
<se:Rule>
   <ogc:Filter>  ...  </ogc:Filter>
      ...
   <sld:MinScaleDenominator>1000.0</sld:MinScaleDenominator>
   <sld:MaxScaleDenominator>16000.0</sld:MaxScaleDenominator>
      ...
   <se:PointSymbolizer>  ...  </se:PointSymbolizer>
</se:Rule>
```

Software Quality Requirements

Table 5.4. Evaluation of the software quality requirements for "scale dependent auto toggling"

Requirement	Score (1-10)
It was easy to understand how to implement the feature	9
The graphics are clear and looks like the description	-
It did not take long to implement the feature	9
It is easy to identify code segments that contains the feature	9
It is easy to identify faults within the code	6
A user with a similar background as the author can easily understand the written code	8
It is easy for a user with a similar background as the author to make changes/updates to the feature	8
There is no unnecessary duplication of data	10

5.2.4.4 Root Cause Analysis

There is no root cause analysis diagram for this problem since we easily determined that it is implementation dependent. This was confirmed by an employee at Carmenta AB.

5.3 Evaluation of Functional Requirements

A list of the functional requirements and their implementation status is found in Table 5.5.

We managed to display all METAR information that should be visualized using text as well as images without rotation. The visualization of wind bards is more advanced and we were unable to achieve a satisfying result. Furthermore, we did not add additional colours to the cloud cover symbols. The only way to add more colours was to add more rules and we decided not to spend time on such implementation.

Wind bars were not implemented because of two problems. The first was Carmenta Engine's inability to perform rotation and displacement in the order that the user had specified in the SLD/SE-file. Due to this implementation dependent problem we were unable to display the correct direction of the wind. The second problem was the barbs attached to the straight line. A new wind barb was needed for each 5kts making this problem similar to *mapping of values*.

Scale dependent auto-toggling was implemented; however, it was not supported by Carmenta Engine.

Since we did not implement any kind of time slider we did not fulfil the requirement of viewing METAR data over time.

Colouring of text was not done due to lack of time and low priority.

Table 5.5. Implementation status of the functional requirements for the SLD/SE file

Priority	**Requirement**	**Fulfilled** (Yes/No/ To Some Extent)
High	Display words/numbers	Yes
High	Display the correct symbol from a list of symbols describing either cloud cover or weather symbols	Yes
High	Display wind barbs	No
Medium	Automatically toggle viewing of the METAR composites on/off depending on map scale	To some extent
Medium	View METAR data over time	No
Medium	Add colour to cloud cover symbols	No
Low	Add colour to words/numbers	No

6 Solutions and Evaluations

6.1 Solutions to Issue: Overwritten Words/Numbers

As we saw from the root cause analysis in Figure 5.3, the problem with overwritten numbers stems from lack of time stamp support. Time stamps are not handled by our client or Carmenta Engine and there is currently no support for a server query that retrieves METAR information based on time stamps. The SLD file itself, is powerless against this type of problem since the SLD/SE standard lacks the needed flexibility. With an SLD file, the designer is able to define how objects should look like; for instance, object alpha should be drawn as a green box that is rotated by 90 degrees and object beta should be drawn as a black dot. However, defining *when* objects should be drawn is not something that the standard has control over or even should be able to control. With that said, we can conclude that there are two places where time stamps can be handled. One option is to place the time stamp solution at the provider side (a server which provides weather observations) and the other option is to embed the solution in the consumer side server application.

6.1.1 Provider Side Solution

A provider side solution would be to support queries based on time stamps. For instance, one should be able to make queries such as “Give us the latest METAR observations for North America” and “Give us the METAR observations for North America that were valid 2011-xx-xx:06:00 thru 2011-xx-xx:12:00”.

Provider Side Solution – Advantages and Disadvantages

+ More precise queries reduce the amount of information that is being sent and unnecessary transmissions are avoided. However,
– Consumer side solutions that implement some switching-between-observations feature will have to perform more queries. This could put a strain on the weather observations provider in terms of the total number of queries it has to process.
+ Time stamps are handled centrally which would reduce the total amount of data processing in the entire system (counting both provider and consumer side).

6.1.2 Consumer Side Solution

A consumer side solution for time stamp management would receive a *bulk* of weather information from the server and needs to *filter* out the unwanted data. The remaining data should then be added to a layer and visualized. In some cases one may want to flick through several observations, e.g. all the weather observations during a certain day or week. The consumer side solution could then split up the observations and divide them into layers depending on their time stamps. Figure 6.1 shows a possible scenario where the consumer side server divides the observations and the consumer side client application is able to easily switch between layers by toggling them on and off.

Consumer Side Solution – Advantages and Disadvantages

– The consumer side does not want to display unnecessary/unwanted data.
– Unnecessary time is spent on retrieval of unwanted data.
– The filtering process takes up resources that could be spend elsewhere.

– Inefficient for the system as a whole.

\+ A consumer side solution decreases the workload for the weather observations provider. This could be desired in a system where there are many users and the server is having problem handling all the requests.

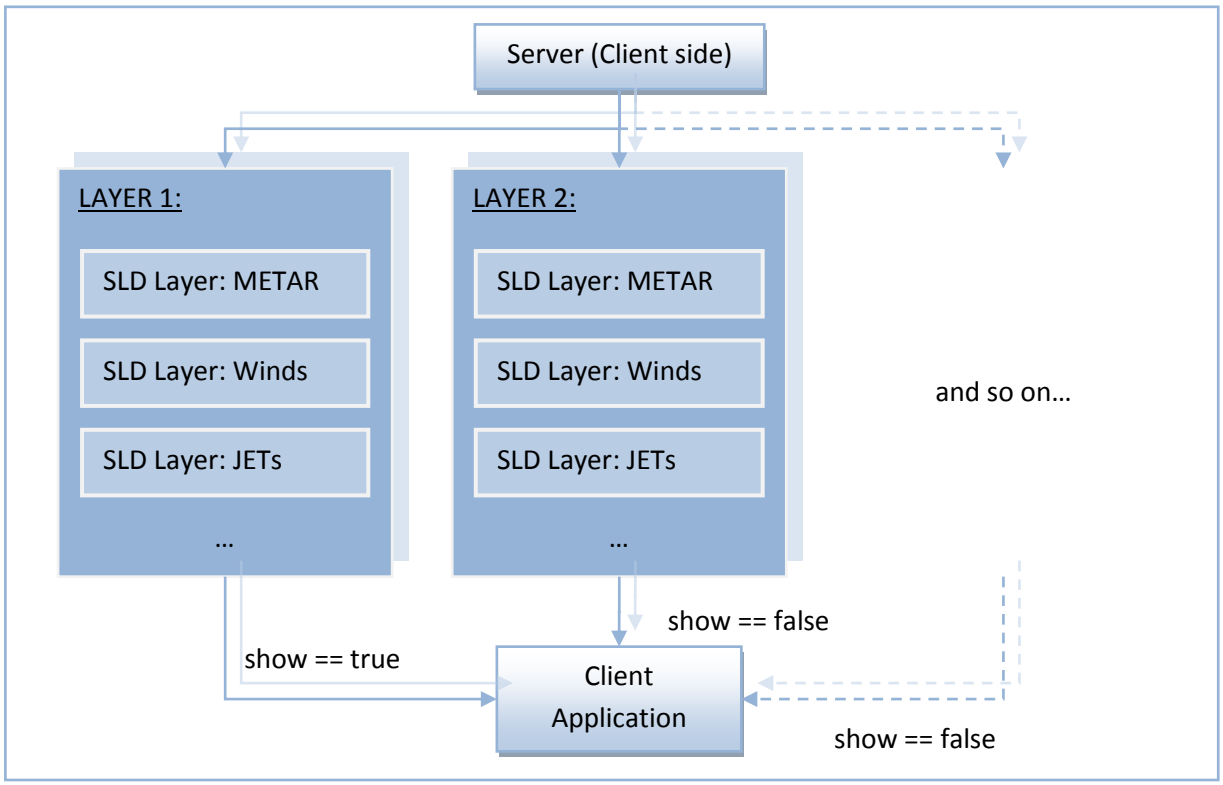


Figure 6.1. A client side solution to the time stamp problem where observations are divided into different layers depending on their time stamps.

6.1.3 Evaluation

Since there is no current solution to compare the provider and consumer side solution with, they will only be compared to each other. The aspects we looked at are maintainability and efficiency. Usability is not considered since there is no actual implementation to look at and thus we are unable to evaluate users' expectations.

In terms of maintainability, a provider side solution is preferred since maintenance would be performed at one place and not at each consumer.

A provider side implementation would, in theory, eliminate transmissions of unnecessary and unwanted data. As a result, data processing for the entire system would be greatly reduced. The same does not go for a consumer side solution, leaving the provider side solution the more efficient one.

When comparing the two solutions we can conclude that the best way to handle time stamps would be to implement a solution at the weather observations provider(s). The only time when one could consider a consumer side implementation is when one wishes to reduce the total number of queries made to the weather observations provider's server. However, this is not something we believe is a common request.

6.2 Solutions to Issue: Mapping of Different Values onto Different Symbols

We learned from the software quality evaluation in chapter 5.2.2.3 that the current implementation, where each symbol must be described by a separate rule, does not meet some of the usability, maintainability and efficiency requirements because;

- The implementation is too time consuming
- Readability is poor
- There is redundancy

The root cause analysis in Figure 5.5 tells us that it is the SLD/SE standard that is at fault since it lacks support for duplicate-free mapping of symbols. Furthermore, we concluded that the reason for the steep learning curve was because the implementation was done in a, for us, unfamiliar manner. As software developers, we would have wanted a different implementation with more emphasis on logical ordering of the symbols. Items that in some way belong together should also be grouped in order to improve readability. When visualizing METAR data, one would like to separate cloud cover symbols from the weather symbols and also illustrate that a symbol belongs to one of these two groups. We propose a solution which we have named *Grouping of Objects*.

To address the problem of redundancy we have devised a solution called *Styling of Multiple Objects*.

6.2.1 Solution: Grouping of Objects

There is currently no built in support in the SLD/SE standard for organizing objects into groups. Figure 6.2 shows what easily could happen to an SLD/SE file if the writer is not careful when adding objects. The blue boxes belong to one type of objects, for example weather symbols; and the red boxes belong to another type of objects, for example cloud cover symbols. If the writer is not careful when organizing his items, he may decrease readability severely and thus making the document harder to maintain. Figure 6.3 illustrates a structured, logical ordering of objects. Such ordering will, when used, increase readability and thus also maintainability.

Current Solution:

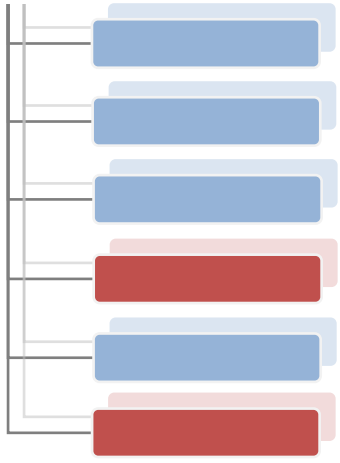


Figure 6.2. No logical ordering of elements, thus leaving blue and red objects intermixed.

Wanted Solution:

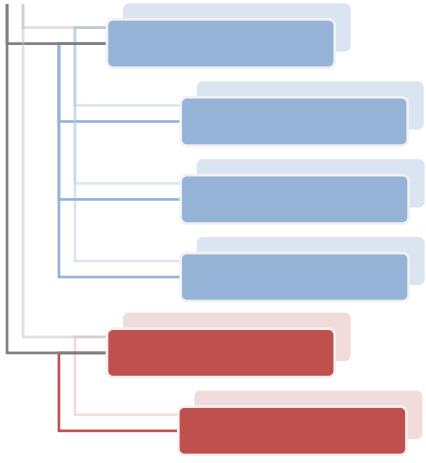


Figure 6.3. Logical ordering of elements, blue and red objects are kept separately.

Preferably, the grouping of objects should resemble some structure which programmers are already familiar with. Such structure would be an if-else statement or a switch-case statement.

We decided on a structure that resembles that of a switch-case statement, see pseudo code in Figure 6.4. An example of how an implementation could look like is found in the example below.

```
switch (cloud cover) {
    case CLEAR:
        symbol = ….clear.png
    case SCATTERED:
        …
}
```

Figure 6.4. Pseudo code for a switch-case statement

Example. How an implementation of *Grouping of Objects* could look like.

```
<!-- "Cloud Cover" images -->
<se:Rule>
   <ogc:Filter>
      <ogc:PropertyIsSwitchCase wildcard="*" singleChar="#" escapeChar="!">
         <ogc:PropertyName>avwx:SurfaceReport/
                  avwx:aerodromeWxObservation/om:Observation/om:result/
                  avwx:cloudCondition/wx:CloudCondition/wx:cloudAmount
         </ogc:PropertyName>
         <ogc:Case>
            <ogc:Literal>CLEAR</ogc:Literal>
            <se:PointSymbolizer>
               <se:Graphic>
                  <se:ExternalGraphic>
                     <se:OnlineResource xlink:type="simple"
                           xlink:href="http://dl.dropbox.com/u/34456973/
                           CloudCover/SKC.PNG"/>
                     <se:Format>image/png</se:Format>
                  </se:ExternalGraphic>
                  <se:Size>10</se:Size>
               </se:Graphic>
            </se:PointSymbolizer>
         </ogc:Case>
         <ogc:Case>
            <ogc:Literal>FEW</ogc:Literal>
            ...
         </ogc:Case>
         <ogc:Case>
            <ogc:Literal>SCATTERED</ogc:Literal>
            ...
         </ogc:Case>
         <ogc:Case>
            <ogc:Literal>BROKEN</ogc:Literal>
            ...
         </ogc:Case>
         <ogc:Case>
            <ogc:Literal>OVERCAST</ogc:Literal>
            ...
         </ogc:Case>
      </ogc:PropertyIsSwitchCase>
   </ogc:Filter>
   <sld:MinScaleDenominator>1000.0</sld:MinScaleDenominator>
   <sld:MaxScaleDenominator>16000.0</sld:MaxScaleDenominator>
</se:Rule>
```

One <Rule> tag is used for each group of objects. In the example above, we show how cloud cover objects could be visualized.

Inside the <Filter> is a <PropertyIsLikeSwitchCase> statement which contains the <PropertyName> as well as a number of <Case> statements. The <PropertyName> works as the switch which should be compared with the <Literal>s inside the <Case>s. Each <Case> contains a <PointSymbolizer> that describes the visualization of the object.

6.2.2 Solution: Styling of Multiple Objects

As can be seen in the example in chapter 6.2.1, the solution enables one to group objects together and each object may have their own individual styling. While this may sometimes be a desired feature, there exist cases where one would like apply styling to several objects. For instance, in the case of visualizing cloud cover, all cloud cover objects have the same styling and this gives us an SLD/SE-file which contains redundancy. In addition to containing the same styling (the same size) there is a significant overhead for just describing something as simple as the size. The pseudo code in Figure 6.5 describes how an implementation of "Styling of Multiple Objects" could look like. The styling is applied to every object within the rule.

```
Rule {
    switch (cloud cover) {
        case CLEAR:
            symbol = ….clear.png
        case SCATTERED:
            …
    } <styling>
}
```

Figure 6.5. Pseudo code for describing styling of multiple objects.

6.2.3 Evaluation

When evaluating the two solutions described above, we will not compare them to each other since they act as a complement to each other. The aspects we look at are *redundancy*, *code readability* and *implementation time*.

The solution "Styling of Multiple Objects" addresses the problem of redundancy by removing the actual redundancy and also the unnecessary overhead which is a result of said redundancy. Even the solution "Grouping of Objects" reduce redundancy in the shape of the overhead that is present in the current solution, where each object is represented by one rule containing almost the same information.

Implementing the two solutions would greatly increase readability since it will become more obvious what objects belong together and that some objects have the same styling.

Since the two solutions reduce redundancy, also the implementation time is decreased. Furthermore, the fact that this solution appears more natural to software developers suggests that learning curve and implementation time would decrease.

6.3 Solution to Issue: Rules within Rules

From the software quality evaluation in chapter 5.2.3.3, we learned that in order to improve the quality, we need a solution that is free from duplicates, takes little time to implement and is easy to understand. One such solution utilizes hierarchical ordering of elements, see Figure 6.6. We also learned that the SLD/SE standard does not support the use of hierarchical rules, see the root cause analysis diagram in Figure 5.7.

```
IF (R0 AND R1) THEN {
    //Do α
}
IF (R0 AND R2) THEN {
    //Do β
}
```

Figure 6.7. Pseudo code that describes the current implementation.

```
IF (R0) THEN {
    IF (R1) THEN
        //Do α
    IF (R2) THEN
        //Do β
}
```

Figure 6.6. Pseudo code that describes a hierarchical solution

As can be seen in Figure 6.7, the current solution can utilize several rules to visualize and object. However, when there are many objects with at least one rule in common the current solution becomes very verbose and suffers from redundancy as well as reduced readability. We have devised two strategies for dealing with these issues. The first is one is named the *hierarchical solution* and the second one is named *Rule Definition.*

6.3.1 The Hierarchical Solution

The hierarchical solution approaches the problem by grouping objects together under an overall rule which the objects have in common. An example of how an implementation of this solution could look like is found below.

Example. How an implementation of the hierarchical solution could look like:

```
<se:Rule>
   <ogc:Filter>
      <!-- Filter for METAR -->
      <ogc:PropertyIsLike wildcard="*" singleChar="#" escapeChar="!">
         <ogc:PropertyName>avwx:SurfaceReport/gml32:name</ogc:PropertyName>
         <ogc:Literal>METAR*</ogc:Literal>
      </ogc:PropertyIsLike>
   </ogc:Filter>
   <se:Rule>
      <ogc:Filter>
         <!-- Filter for Cloud Cover=CLEAR -->
         <ogc:PropertyIsLike wildcard="*" singleChar="#" escapeChar="!">
            <ogc:PropertyName>avwx:SurfaceReport/
                    avwx:aerodromeWxObservation/om:Observation/om:result/
                    avwx:cloudCondition/wx:CloudCondition/wx:cloudAmount
            </ogc:PropertyName>
            <ogc:Literal>CLEAR</ogc:Literal>
         </ogc:PropertyIsLike>
      </ogc:Filter>
      <se:PointSymbolizer>
         ...
      </se:PointSymbolizer>
   </se:Rule>
```

```
   <se:Rule>
      ...
   </se:Rule>
   ...
</se:Rule>
```

The outer rule contains a <Filter> tag that states that the input file must contain METAR data for the objects to be visualized. Furthermore there is one rule for each object we want to style, each with its own <Filter> attached. In the example above, we see a filter for cloud cover = CLEAR.

6.3.2 The Rule Definition Solution

The other strategy that we have come up with is named *Rule Definition*. It is a solution that takes a different approach at avoiding duplications. By defining rules at the top of the document, the user should be able to use them later by writing references. The following is an example of how an implementation of rule definitions could look like.

Example. How an implementation of Rule Definition could look like.

```
<se:RuleDefinition>
   <!--Rule Definition: METAR -->
   <sld:RuleName>METAR_RULE</sld:RuleName>
   <ogc:Filter>
      <ogc:PropertyIsLike wildcard="*" singleChar="#" escapeChar="!">
         <ogc:PropertyName>avwx:SurfaceReport/gml32:name</ogc:PropertyName>
         <ogc:Literal>METAR*</ogc:Literal>
      </ogc:PropertyIsLike>
   </ogc:Filter>
</se:RuleDefinition>

<se:Rule>
<!-- Usage of the defined METAR rule -->
   <ogc:Filter>
      <ogc:And>
         <sld:RuleName>METAR_RULE</sld:RuleName>
         <ogc:PropertyIsLike wildcard="*" singleChar="#" escapeChar="!">
            <ogc:PropertyName>avwx:SurfaceReport/
                    avwx:aerodromeWxObservation/om:Observation/om:result/
                    avwx:cloudCondition/wx:CloudCondition/wx:cloudAmount
            </ogc:PropertyName>
            <ogc:Literal>CLEAR</ogc:Literal>
         </ogc:PropertyIsLike>
      </ogc:And>
   </ogc:Filter>
   ...
</se:Rule>
```

The tag <RuleDefinition> is used to define rules at the beginning of the SLD/SE file. It should contain a unique name which can be used to reference the rule later in the file. It also contains a <Filter> like the regular rules do.

In the example above we have taken the current implementation of multiple rules and substituted the original METAR rule for a reference to the previously defined METAR_RULE.

6.3.3 Evaluation of the possible solutions

The hierarchical and rule definition solutions are compared to both each other and the current solution. The different aspects of usability, maintainability and efficiency that we have looked at are; *number of calculations*, *readability* and *efficiency*.

Number of Calculations

The number of calculations that are needed is implementation dependent.

Look at Figure 6.7 and Figure 6.6 for reference. Consider the following scenario; the rules R_0 is false, thus making it unnecessary to also evaluate R_1 and/or R_2. Even if the current implementation takes this into account and breaks when finding a rule (R_0) to be false, it will still try to evaluate the next set of rules (R_0 AND R_2) thus leading to unnecessary calculations being performed.

Readability

In the current implementation, readability is good but it lacks one thing; namely the ability to logically group items together. For instance, say that we want to do α and β *only* when we have received METAR data. Visualizing the information that α and β have in common through hierarchical rules would increase readability to some extent and in the long run, improve maintainability. The Rule Definition solution will not be able to illustrate clearly that two elements have one rule in common; however, by letting the user separate rule definition and usage, readability would be enhanced. A combination of the two solutions would achieve a high level of readability.

Efficiency

In order to improve efficiency we need to look at the number of lines of code in the two solutions. Reducing the number of lines of code will reduce what needs to be retrieved from the server, thus reducing transfer times. If the user needs to write less code, the implementation will take less time and thus improving the time aspect of usability.

When counting lines of code, what we are counting is number of tags (both start and end tags) since one tag mostly is the same as one line. The only exception is when both start and end tags are found on the same line.

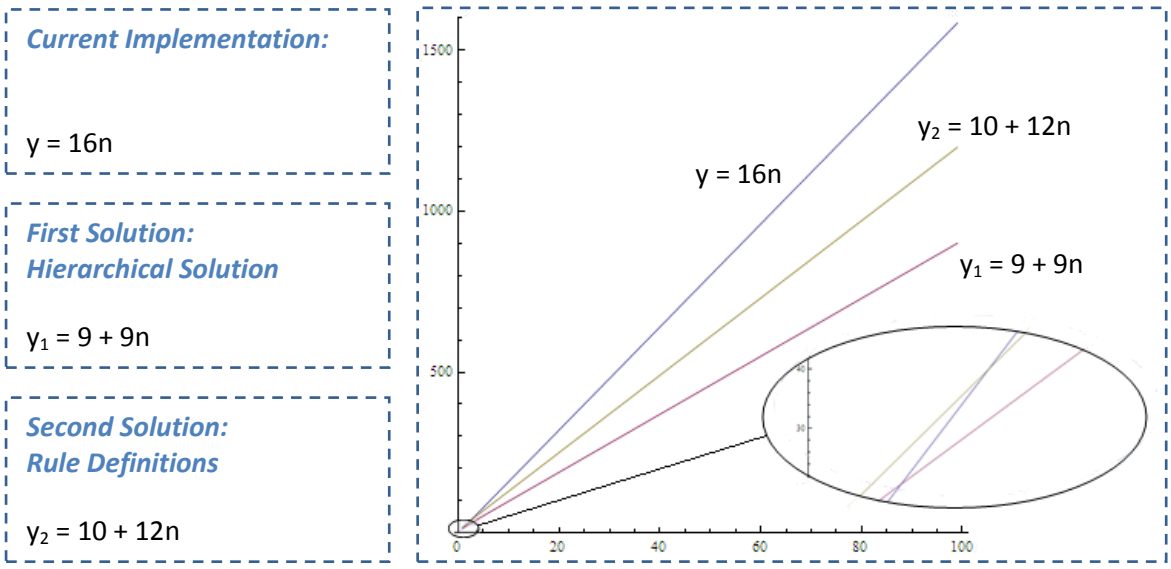


Figure 6.8. Visualization of n elements with one rule in common and one rule for each element.

Figure 6.8 describes the complexity of the different solutions in terms of number of lines of code. Each solution is described as a function based on the estimated lines of code the solution will produce. Judging from the graph it is evident that the hierarchical solution is the most successful in reducing the number of lines of code. For more exact numbers on how the different solutions perform, see Table 6.1.

Table 6.1. Complexity of each solution in terms of number of Lines of Code (LOC).

Number of Elements	Current Solution (LOC)	Hierarchical Solution (LOC)	Rule Definitions (LOC)
O(1)	**16**	18	22
O(2)	32	**27**	34
O(7)	112	**72**	94
O(99)	1584	**900**	1198

The comparison between the three solutions in Table 6.1 shows that the current solution is preferable only for single elements. In all other cases, the hierarchical will require less lines of code to implement. For reference, there are seven different cloud cover symbols and ninety-nine different weather symbols.

6.4 Solution to Issue: Scale Dependent Auto Toggling

As stated in Chapter 5.2.4, the SLD/SE standard does have support for this issue; however, Carmenta Engine does not. Since this is an implementation dependent issue and we are not familiar with the implementation of Carmenta Engine, we cannot and will not be giving any solution(s) to this problem.

7 Improvement Proposals

We propose that the provider side solution described in chapter 6.1.1 should be implemented. Since Carmenta AB develops consumer side solutions we urge them to bring this to the table in the geospatial and ATM domain.

Regarding the issue of mapping objects we propose that the solution “Grouping of Objects” described in chapter 6.2.1, or a similar solution, should be implemented by the SLD/SE standard. We also propose that the second solution “Styling of Multiple Objects” described in 6.2.2 should be implemented as a complement in order to address all aspects of the problem.

To handle the problem with rules within rules, we propose that the SLD/SE standard implements the hierarchical solution described in chapter 6.3.1. Implementation the “rule definition” found in 6.3.2 should act as a complement to the hierarchical solution, however this is not vital.

8 Discussion

In this chapter, we will discuss the research design and give our thoughts on the root cause analysis method. Furthermore, we will discuss the Weather Client as a proof-of-concept prototype and the use of SLD/SE in combination with WXXM in geospatial systems. We will also present the answers to our research questions.

8.1 The Research Design as a Way of Evaluating the SLD/SE Standard

We are very satisfied with the research design that was used in the thesis. It was thorough, reliable and as an end result, helped us to produce an evaluation of the SLD/SE standard as well as improvement proposals to said standard.

The Root Cause Analysis method described in chapter 3.3 was developed during the project where feedback from using it was used to further develop the method. We believe the resulting method was a very useful tool while evaluating the SLD/SE standard. While root cause analysis methods usually focus on business goals and their users are mostly project managers, this method is more useful to software developers. When stuck with a difficult task, it enables the developer to systematically go through the possible sources for the problem. It is easy to learn how to use it and you feel more and more confident for each root cause analysis diagram that you draw.

8.2 The Weather Client as a Proof-of-Concept Prototype

We are fairly satisfied with our implementation of the Weather Client. It was able to provide us with functionality needed for proving that the concept worked, as well as acting as a test tool for the SLD/SE-file. In terms of fulfilling the software quality requirements it performed quite well. See chapter 4.3 for more detailed information. The only requirements that it did not fulfil were a number of functional requirements, as well as the efficiency requirement concerning lag when performing any map actions such as moving the map or zooming in and out. As long as we did not use an SLD/SE-file for styling but only used a simple map showing land, water and cities, the Weather Client ran smoothly. As soon as we enabled METAR data each map update took a considerable amount of time. A couple of changes to the configuration file improved the situation to some extent but not fully. We never found the cause of the problem but we suspected that further investigation of the configuration file, or a look at the Carmenta Engine implementation could lead us to the cause of the problem.

One of the functional requirements that were not implemented was the time slider, which would have been a very interesting and enjoyable feature. When researching weather information visualization we came across a number of clients that had implemented a time slider. However, most of them were terribly slow when running and it would seem as though designing a time slider is a problematic task. We quite early learned that building a time slider with SLD/SE, WXXM and Carmenta Engine would provided a number of interesting programming challenges when deciding what would be the best approach to build a successful implementation. We also realized that, unfortunately, it would take too much time in comparison with how much it would contribute to our analysis of the SLD/SE standard. When looking into the problem with overwritten words we gained some insight into what was necessary in order to implement a time slider, namely time stamp management. Figure 6.1 describes a rough sketch of what a time slider implementation could look like.

Apart from the time slider, we find it regrettable that we were unable to find more weather information on the WXXM data format. The only way of testing the SLD/SE standard was to use METAR reports and we would have wanted to look at other type of reports. For instance, styling of JET streams would have been interesting and would probably have resulted in a more thorough investigation of the SLD/SE standard.

8.3 SLD/SE as a Visualization Standard for Geospatial Systems

During our implementation of the Weather Client it showed that the SLD/SE standard had not been developed in tandem with WXXM. A great number of faults were found and in retrospect we can say that we had problems of varying degree with each and every functional requirement. The problems can be divided into three categories; *implementation dependent*, *data source provider* and the *SLD/SE standard*. Solutions to implementation dependent problems would not be very interesting for the geospatial and ATM community and thus they were not developed.

We only found one problem related to the data source provider and that was the lack of time stamps. Even though this does not directly relate to the SLD/SE standard, it does so indirectly because a system without time stamp management is virtually useless. In our improvement proposal we believe that each weather provider should implement some time stamp management solution to accommodate the needs of the consumers. This was supported by our supervisor, Daniel Tagesson at Carmenta.

In the third category, consisting of problems that are related to the SLD/SE standard, we found the following issues. These issues were used to answer our research questions.

- Mapping of Different Values onto Different Objects
- Rules within Rules

During the OWS-8 project [21], participants evaluating SLD/SE together with AIXM noticed similar problems as we did. They too were concerned that the lack of else-if structure would make rules hard to write and inefficient to evaluate [23]. Instead of using an else-if structure, we proposed a switch-case structure.

8.3.1 Research Question 1: What limitations of the SLD/SE standard affect functionality?

There are no limitations that affect functionality.

8.3.2 Research Question 2: What limitations of the SLD/SE standard affect usability?

The following limitations of the SLD/SE standard affect usability:

- The lack of grouping of objects affect understandability and implementation time
- The lack of rules within rules affect implementation time

8.3.3 Research Question 3: What limitations of the SLD/SE standard affect maintainability?

The following limitations of the SLD/SE standard affect maintainability:

- The lack of grouping of objects affect code readability
- The lack of rules within rules affect code readability

8.3.4 Research Question 4: What limitations of the SLD/SE standard affect efficiency?

From these issues we found the following limitations of the SLD/SE standard that affect efficiency:

- The lack of grouping of objects affect code redundancy
- The lack of rules within rules affect code redundancy
- The lack of styling of multiple objects affect code redundancy

9 Conclusions

The Weather Information Exchange Model (WXXM) is a standard which has not been adopted by many actors in the ATM industry thus making styling of said data rather difficult. While there are no limitations in the Styled Layer Descriptor (SLD)/Symbology Encoding (SE) standard that affect functionality there is a significant number of limitations within the SLD/SE standard that affects usability, maintainability and efficiency.

The research design used in this thesis was successful in evaluating the SLD/SE standard in combination with WXXM. The root cause analysis method that was developed during the implementation phase was a helpful tool while analysing issues found during the work.

9.1 Recommendations

SLD/SE in combination with WXXM has potential but

- It should implement the suggested improvement proposals
- There is a need for more testing to ensure that the two standards to work well together

WXXM has potential but in order to develop it further, it needs more attention from the entire business (weather providers, weather data consumers and developers of WXXM)

If Carmenta AB would like to make use of WXXM in their systems, they should take some part in the development of WXXM and SLD/SE to ensure that the standards reach their potential.

10 References

[20] Carmenta AB, “Carmenta Engine - Products,” [Online]. Available: http://www.carmenta.com/products/carmenta-engine. [Accessed 11 October 2011].

[6] Danmarks Meteorologiske Institut, “Aerodrome Weather Report - METAR and SPECI decode,” 2005. [Online]. Available: http://www.dmi.dk/dmi/koder.pdf. [Accessed 20 October 2011].

[22] Dropbox, [Online]. Available: https://www.dropbox.com/. [Accessed 27 March 2012].

[13] Eurocontrol - AIXM, “Eurocontrol - Aeronautical Information Exchange,” [Online]. Available: http://www.aixm.aero/public/subsite_homepage/homepage.html. [Accessed 16 November 2011].

[11] Eurocontrol, “WXXM 1.1 Primer,” 2010.

[1] FAA, “Report of the Weather-ATM Integration Working Group,” 3 October 2007. [Online]. Available: http://www.jpdo.gov/library/FAA_REDAC_Report.pdf. [Accessed 10 November 2011].

[19] Google, “Google Earth,” [Online]. Available: http://www.google.com/earth/index.html. [Accessed 15 October 2011].

[10] Hong Kong Observatory, “Decoding Aviation Weather Report (METAR/SPECI),” 15 August 2011. [Online]. Available: http://www.hko.gov.hk/aviat/decode_metar_e.htm. [Accessed 16 December 2011].

[4] ISO/IEC 9126-1:2001(E), *Software Engineering - Product Quality - Part 1: Quality Model,* ISO/IEC, 2001.

[18] J. Bennett, “5 Whys Method,” [Online]. Available: http://www.iso9001consultant.com.au/5-whys.html. [Accessed 20 November 2011].

[23] Luciad NV, “OGC Change Request: Composite Rule with support for 'Else If' and nested objects,” 2011-09-15.

[12] NNewWiki, “WXXM - NNEW Dissemination - UCAR Wiki,” 22 February 2011. [Online]. Available: https://wiki.ucar.edu/display/NNEWD/WXXM. [Accessed 15 November 2011].

[5] NOAA, “ADDS - METARs,” [Online]. Available: http://aviationweather.gov/adds/metars/. [Accessed 15 October 2011].

[7] NOAA, “NWS JetStream,” [Online]. Available: http://www.srh.weather.gov/srh/jetstream/synoptic/wxmaps.htm. [Accessed 12 November 2011].

[17] O. Serrat, “Asian Development Bank,” 30 February 2009. [Online]. Available: http://www.asiandevbank.org/Documents/Information/Knowledge-Solutions/The-Five-Whys-Technique.pdf. [Accessed 13 October 2011].

[21] Open Geospatial Consortium, “OGC Web Services, Phase 8 (OWS-8) - OGC(R),” [Online]. Available: http://www.opengeospatial.org/projects/initiatives/ows-8. [Accessed 27 March 2012].

[14] Open Geospatial Consortium, Inc., “Geography Markup Language - OGC(R),” [Online]. Available: http://www.opengeospatial.org/standards/gml. [Accessed 27 January 2012].

[2] Open Geospatial Consortium, Inc., “Styled Layer Descriptor profile of the Web Map Service Implementation Specification [ref. OGC 05-078r4],” Open Geospatial Consortium, Inc., 2007-06-29.

[3] Open Geospatial Consortium, Inc., “Symbology Encoding Implementation Specification [ref. OGC 05-077r4],” Open Geospatial Consortium, Inc., 2006-07-21.

[16] P. G. Preuss, School leader's guide to root cause analysis: using data to dissolve problems, Larchmont, NY 10538: Eye on Education, 2003.

[15] R. K. Yin, Case Study Research - Design and Methods, United States of America: Sage Publications Inc., 2003.

[9] SMHI, “Flyg - Professionella tjänster - SMHI,” 13 May 2009. [Online]. Available: http://www.smhi.se/Produkter-och-tjanster/professionella-tjanster/flyg/taf-och-metar-1.2393. [Accessed 25 November 2011].

[8] World Meteorological Organization, “WMO-No. 49, Vol II: Meteorological Service for International Air Navigation,” Secretariat of the World Meteorological Organization, Geneva – Switzerland, 2004.