Implementing Service Model Visualizations

Utilizing Hyperbolic Tree Structures for Visualizing Service Models in Telecommunication Networks.

Andreas Lundgren
Abstract

This paper describes the design, implementation and evaluation of HyperSALmon, a Java™ open source prototype for visualizing service models in telecommunication networks. For efficient browsing and graphical monitoring of service models using SALmon, a service modeling language and a monitoring engine (Leijon et al., 2008), some kind of interactive GUI that implements a visualization of the service model is desired. This is what HyperSALmon is intended to do.

The prototype has been designed in accordance with suggestions derived from a current research report of visualization techniques (Sehlstedt, 2008) appropriate for displaying service model data. In addition to these suggestions domain experts at Data Ductus Nord AB has expressed an urge for implementation of further features, some of their suggestions are deduced from research documents (Leijon et al., 2008; Wallin and Leijon, 2007, 2006), while others have been stated orally in direct relation to the prototype implementation work. The main visualization proposal is to use tree structures. Thus, both traditional tree structures and hyperbolic tree structures have been utilized, where the main navigation is set to occur in the hyperbolic tree view.

In order to contribute further to this report I provide a discussion addressing problems related to the context of implementing a prototype for service model visualization using open source frameworks that meets the requirements set by the service model network architecture, the domain experts and the suggestions in the research report (Sehlstedt, 2008, page 51-52). Finally, I will present drawn conclusions of the attempted prototype implementation, illustrating potential strengths and weaknesses and consequently introduce suggestions for possible improvement and further development.
Contents

1 Background 1
   1.1 Purpose ................................................................. 1
   1.2 Method ................................................................. 1
       1.2.1 Method strategy .................................................. 2
   1.3 Introduction to SALmon ............................................. 2
   1.4 Visualization suggestions .......................................... 2
       1.4.1 Suggestions based on the research report ................... 3
       1.4.2 Additional suggestions from Data Ductus Nord AB .......... 4

2 Visualizing Service Models 4
   2.1 The GSM service model ............................................. 5
   2.2 System requirements ................................................. 6

3 Adopting Open Source Frameworks 6
   3.1 Treebolic - A Hyperbolic Tree Browser .......................... 6
       3.1.1 Treebolic Browser .............................................. 7
       3.1.2 Treebolic Generator ............................................ 7
       3.1.3 Treebolic Application .......................................... 7

4 SALmon 7

5 HyperSALmon - A Service Model Visualization Prototype 8
   5.1 Challenges ............................................................. 9
   5.2 Implementing suggested features ................................... 9
       5.2.1 Suggestion 1: A traditional tree visualizing the technical view of a service model. ........................................... 10
       5.2.2 Suggestion 2: An “Overview + detail” interface to provide instant status overview. .............................................. 11
       5.2.3 Suggestion 3: A search function allowing multiple terms search, and providing own search term suggestions in case of misspellings from the user. ................................................. 12
       5.2.4 Suggestion 4: Clearly visible search hits with, e.g., highlighting in the both the detail and the overview window, and with automatic translation of the detail window to make search hits come in focus. .................. 13
       5.2.5 Suggestion 5: A kind of semantic zooming to accomplish stronger connection between the nodes in the tree and their respective attribute information. ................................................... 14
       5.2.6 Suggestion 6: A sorting function allowing the user to choose between alphabetical order, geographical order, order based on number of open tickets, and order based on downtime. ................................. 14
       5.2.7 Suggestion 7: Greater separation of different levels in the hierarchy by increasing the interspaces (width ways) between them. ......................................................... 15
       5.2.8 Suggestion 8: Presentation of current attribute values in context with past values using sparklines. ................................. 15
       5.2.9 Suggestion 9: A function making it possible to create sub graphs, these could then be placed in a tab menu to facilitate change of graph view. 15
5.2.10 Suggestion 10: Dynamic queries connected to the attributes, to provide more advanced filtering than in the prototypes. ........................................... 16
5.2.11 Suggestion 11: Adaptation of colors to gain better visual clarity, and perhaps more than four different colors to visualize status. ......................... 16
5.2.12 Suggestion 12: A “Time-slider” for browsing/tracking historic events within the service model. ................................................................. 17
5.2.13 Suggestion 13: Navigational filters to help user focus on hot-spots and problems. ................................................................. 18

6 Implementation Experiences .................................................. 18
6.1 Maintenance ................................................................. 19
6.2 Functionality ................................................................. 19
6.3 Usability ................................................................. 20
6.4 GPLv3 issues ................................................................. 20

7 Discussion and Conclusions .................................................. 20
7.1 Future Work and Further Development .................................. 20
  7.1.1 Prototype resource- and project planning ............................. 21
  7.1.2 Discuss and decide whether any methodology should be utilized ... 21
  7.1.3 Deployment of test team and master test plan ......................... 21
  7.1.4 Construction of method stubs or deployment of an operational SALmon API for realistic data provision and visualization ...................... 21
  7.1.5 Producing documentation ........................................... 21

8 Acknowledgements ............................................................ 22

A Prototype source and binaries ............................................. 25
  A.1 Release notes ............................................................ 25
        A.1.1 Disclaimer ...................................................... 25

List of Figures

1 SALmon provider dependency for both class definitions and instance data. ... 3
2 The principal components the GSM model hierarchy. .......................... 5
3 Treebolic architecture. ................................................... 7
4 Treebolic Browser. Example usage; Hyperbolic visualization of hierarchical data on the animal kingdom. .................................................. 8
5 Treebolic Generator. The tree view is also an XML editor. .................. 9
6 Treebolic Application. Initialization; If no data provider has been set you are requested to enter a data provider URL. ........................................ 10
7 HyperSALmon architecture. ............................................. 10
8 HyperSALmon. Early experiment displaying 1,000 hypothetical GSM cells at once. ................................................................. 11
9 HyperSALmon. “Overview + detail” interface. ............................... 12
10 Treebolic. Search engine declaration example. .................................. 13
11 HyperSALmon. Declaration of an alarm search function. ................. 14
12 Example usage of sparklines to monitor attribute values over time; Monitoring attribute “Functioning” which has been reported non functional between 16:35 to 16:55. ................. 16
13 HyperSALmon. Example of color adaption; using more than four different colors for visualizing alarm status. .................... 17

List of Tables

1 Human-Computer interaction design guidelines: Color coding recommendations. 17
1 Background

Service models and telecommunication networks are complex creations containing vast amounts of data. These networks are monitored and managed from Network Operations Centers (NOCs). One of the NOC technician’s main job assignments is to handle long lists of triggered alarms. The number of alarms ranges between 100,000 - 1,000,000 per day in a medium-sized NOC and as new systems and services are introduced the number of alarms are ever-increasing (Wallin and Leijon, 2006). This problem calls for new measures and consequently the industry is starting to change its focus from resource monitoring to service monitoring (Wallin and Leijon, 2006).

It lies in the nature of the service model to change over time as events take place in its dynamic network. Hence, visualizing the service model implies more than displaying a static view of the model, but more so the ability to display its historic states as well. This is necessary, e.g., when service break-down occurs, allowing for “time travels” to track specific break-downs and check whether Service Level Agreement (SLA) conditions are met. The ability to monitor these networks over time and in great detail lends itself to greater definition of more advanced and detailed SLAs. This opens up for increased competition, more complex service offerings, distribution of services and a market for SLAs (Wallin and Leijon, 2007).

The principal for this implementation project and thesis work is Data Ductus Nord AB, a Swedish IT company that focuses mainly areas of system integration, software development, and network management. They have developed a definition of the service model and a dedicated service modeling language and monitoring engine called SALmon (Leijon et al., 2008). This service model definition together with SALmon will serve as foundation for the implementation of a visualization of the service model.

1.1 Purpose

The main purpose of implementing visualization for service models is to provide a tool for efficient visual navigation and monitoring of the GSM service model (described in section 2.1). What is to be considered efficient navigation in this context is derived from the suggested features in a research report by Sehlstedt (2008) and the expressions from Data Ductus Nord AB in section 1.4.2.

Further purposes with this implementation may be experimental and still unknown, as it may later be evident that an implementation of visualization as such can provide purposes for viewing service model in unexpected ways, revealing patterns and scopes of use that are not yet obvious.

1.2 Method

To gain an understanding about the implementation task, the domain of service modeling, benefits of service modeling and service monitoring, some background research was performed. This was done by reading papers, research reports, studying GSM service model data and observing a demonstration of SALmon.

To support criteria for efficient visual navigation and monitoring of the GSM service model the visualization attempt will be implemented in line with the proposed characteristics (described in sections 1.4.1 and 1.4.2) resulting from research on appropriate visualization techniques for this purpose along with suggestions from the principal.
1.2.1 Method strategy

As the full implementation of a service model visualization may be considered a relatively comprehensive task, the work strategy for this thesis project is as follows:

1. **Implement a visualization prototype.**

   The visualization prototype is requested by the principal to be implemented in Java™. Furthermore, it is desirable if the prototype is based on open source software that can provide visualization of hierarchical data in tree structures.

2. **Implement as many of the suggested features as possible.**

   By tweaking, modifying and extending the functionality of the open source software tool the suggested features may be implemented. These features regard desired and required elements of functionality that have been previously identified and expressed by the principal. However, a certain amount of own creative liberty may be acceptable during the implementation process as long as the end result is considered to meet operation. This decision is given by the principal in connection with examinations and communications during the implementation process.

3. **If implementation not possible, a proposed implementation is provided instead.**

   The implementation of the suggested features requires a variable amount of effort in order to be considered fully implemented. Some features can be easily implemented by tweaking existing methods and algorithms, others may even already be implemented in the software while some features require far more attention or may even be considered impossible to implement within this time frame due to lack of resources or other influential circumstances.

   If a suggested feature is considered not possible to implement I will provide a proposition on how this feature could be implemented and what means I value required for the realization of such an implementation.

1.3 Introduction to SALmon

SALmon, a service modeling language and monitoring engine, attempts to define services and SLAs as well as the underlying structure of the service model network (Leijon, Wallin, and Ehnmark, 2008). The modeling language and engine itself is implemented as a prototype under development and lacks a graphical user interface (GUI). The implementation of a suitable GUI is vital to provide effective visualization, navigation and overview of the service model network. Since the description of the service model can be assumed to be dynamic the GUI itself must support visualization for an uncertain and variable amount of pre-defined classes and objects. It is therefore imperative that all class definitions as well as object instances and attribute values can be requested and obtained from the data provider (SALmon). This dependency is illustrated in figure 1.

1.4 Visualization suggestions

A current research report of visualization techniques (Sehlstedt, 2008), appropriate for displaying service model data, presents recommendations on how visualization of service models could be implemented. The report’s main conclusion proclaims that the utilization of tree structures
is advantageous for the purpose of visualizing service models. According to the report’s conclusion the traditional tree was considered the easiest to navigate and the best at displaying structure although it lacks overview and requires a lot of scrolling. In addition, the author’s subjective opinion was that the hyperbolic browser is the visualization, of those evaluated in the report, with the greatest potential as it is not limited to visualizing trees only, in the start view the focus is on the node representing the service, it has a built-in fisheye view, attributes can be visualized in many different ways, etc. (Sehlstedt, 2008) One of the main purposes of visualizing service models should be that it is realized in accordance with expressed suggestions and recommendations from the principal. The report provides a summary of suggestions for a future GUI (Sehlstedt, 2008, page 51-52). In addition to these suggestions Data Ductus Nord AB has expressed an urge for implementation of further or refined features.

1.4.1 Suggestions based on the research report

The research report (Sehlstedt, 2008) suggests that a future GUI should implement the following features:

1. A traditional tree visualizing the technical view of a service model.
2. An “Overview + detail” interface (Hornbæk et al., 2002) to provide instant status overview.
3. A search function allowing multiple terms search, and providing own search term suggestions in case of misspellings from the user.
4. Clearly visible search hits with, e.g., highlighting in the both the detail and the overview window, and with automatic translation of the detail window to make search hits come in focus.
5. A kind of semantic zooming to accomplish stronger connection between the nodes in the tree and their respective attribute information.
6. A sorting function allowing the user to choose between alphabetical order, geographical order, order based on number of open tickets, and order based on downtime.
7. Bigger separation of different levels in the hierarchy by increasing the interspaces (width ways) between them.

8. Presentation of current attribute values in context with past values using sparklines.

9. A function making it possible to create sub graphs, these could then be placed in a tab menu to facilitate change of graph view.

10. Dynamic queries connected to the attributes, to provide more advanced filtering than in the prototypes.

11. Adaptation of colors to gain better visual clarity, and perhaps more than four different colors to visualize status.

1.4.2 Additional suggestions from Data Ductus Nord AB

12. A “Time-slider” for browsing/tracking historic events within the service model.

13. Navigational filters to help user focus on hot-spots\(^1\) and problems.

In this thesis work I will investigate the possibility to implement these suggestions in practice, propose a Java\(^\text{TM}\) prototype for this purpose and discuss complications that arise in the implementation context.

2 Visualizing Service Models

It is possible to visualize the service model, but this visualization requires to be expressed through multiple views in order to cover all the relevant aspects of the service. Wallin and Leijon (2007) enumerates four views that enable visualization of a multi purpose service model through a Quality of Service (QoS) perspective:

- a \textit{technical view}
- a \textit{marketing view}
- a \textit{customer-care view}
- an \textit{end-customer view}

All these views are to be regarded when attempting to visualize service models for telecommunication networks. But since each view is targeted to a specific user group, some of the views may deserve more attention than the others depending on what user group the intended end users belongs to. Data Ductus Nord AB lists four categories of proposed end-users of the application:

- NOC technicians
- Product owners\(^2\)
- Customer service

\(^1\)A region of high or special activity within the service model.
\(^2\)Owners of products that consume telecommunication services.
• Customers

We can deduce from this list of proposed end-users that each one of them seems to fit into any one, or many, of the four QoS views. Generally, it can be concluded that the NOC technicians would benefit from viewing the service model through a technical view, the product owners might prefer a marketing view, the customer service a customer-care view and customers preferably assimilate the visualization through an end-customer view.

This thesis project is based mainly on the findings derived from the research report by Sehlstedt (2008) which is directed towards the technical aspects of the service model visualization. Consequently, the technical view is the main focal point for this implementation work. The technical view of the service model aims to display hardware status in a comprehensive manner, as the NOC technicians and field engineers highest interest is to know what to fix next (Wallin and Leijon, 2007).

2.1 The GSM service model

In order to implement a visualization of the service model one needs to have an idea of the service model purpose and how its architecture is composed. The description of the service model utilized herein is based on a hypothetical definition of the GSM service model developed and presented by Leijon, Wallin, and Ehnmark (2008). The GSM service can be basically described as an object-oriented model consisting of a GSM service, areas and cells which contain a number of Base Station Controllers (BSCs) and Mobile Switching Centers (MSCs) which are interconnected with several transceivers (TRXs) in a complex transmission network. The model structure is strictly hierarchical with the GSM service as root, areas and cells as branches and BSC, MSC and TRX components as leaves. Figure 2 shows a simplified illustration of the principal components in the GSM model and how they are hierarchically related to each other.

![Figure 2: The principal components the GSM model hierarchy.](image)

Each component or object in the GSM service model has a number of attributes that are important for describing the status of the model component and its elements. The service operators rely on these attribute values when diagnosing problems encountered in the model. It is therefore essential that these attributes, values and status indications are clear, tangible and claims a central role in the visualization.
2.2 System requirements

A visualization of service models requires that the hosting system can process and render visual representations of large data flows simultaneously. Service providers have large infrastructure and service portfolios. There can be several million cells, edge devices, areas and customers (Leijon et al., 2008). This places high demands on the hosting system’s hardware to deliver a visualization that is smooth and satisfactory. The visualization is requested to be implemented in the Java™ programming language. Although there are other, more hardware near programming languages that provide better support for graphics acceleration than what Java™ offers (Sangappa et al., 2002). However, these programming languages usually involves more complex coding and are platform dependent, which in this case is not desirable when implementing a visualization that should be available to as many potential stakeholders and actors as possible (Tyma, 1998).

3 Adopting Open Source Frameworks

One advantage of choosing to implement the visualization of a popular programming language such as Java™ (Atkinson et al., 1996) is that the resource of active open source projects on the Internet is monumental. For example, SourceForge.net, which is one of the largest resources for open source applications on Internet, provides about 176000 open source projects. Finding an appropriate open source framework for service model visualization implies, as suggested by the research report (Sehlstedt, 2008), identifying a software tool that aims to visualize information in the form of tree structures. This task should be considered perilous as it infers making sure that the framework measures up to expectations and that program code can be easily modified to meet the requirements set for the final application (Howison and Crowston, 2004).

There were several applications that provided hyperbolic rendering of hierarchical data on the Internet, on SourceForge.net there were, at the time of this thesis project, about 20 libraries or applications that offered this capability in various extents. Among the most promising were Hypergraph, HyperTree Studio, Treebolic and Hyperbolic Tree Java Library.

3.1 Treebolic - A Hyperbolic Tree Browser

After a series of short pilot tests, investigations and evaluations the Treebolic application suite (Bou, 2009) proved itself as a promising candidate for visualizing service models. This conclusion was based on the documented performance Treebolic delivered, the author’s ambitions and recent code contributions and the generally good prospects for the implementation of a visualization tool for service models that were appreciated by myself empirically and with approval by domain experts at Data Ductus Nord AB.

The Treebolic suite consists of three applications: the Treebolic Browser, the Treebolic Generator and the Treebolic Application. The Treebolic engine core requires to be fed by a data provider in order to visualize the provided data. In Treebolic version 2.0.3 a number of data providers are supported: XML-DOM, XML-XSLT, graph (spanning-tree), SQL, XSLT freemind, XSLT gxl and DOT. Since the project is open source it opens up for possibilities to develop custom interfaces for providing virtually any kind of hierarchical data.

---

3Number of projects listed on SourceForge.net, accessed June 4, 2009.
3.1.1 Treebolic Browser

The Treebolic Browser is an application that hosts the Treebolic engine and links it to various data providers. “Unlike a web-hosted applet, it is not subject to security limitations. The widget core works regardless of the data-feeding mechanism (or provider)” (Bou, 2009). One such provider is a standard XML\(^4\) module that reads XML files. Figure 4 demonstrates how XML data is rendered and browsed as a hyperbolic tree in Treebolic Browser.

3.1.2 Treebolic Generator

The Treebolic Generator is an application that allows the XML description to be generated. It offers four different views of the hierarchical data; a traditional tree view, a hyperbolic tree view, an XML-tree view, and an XML-text view which all are represented as tabs that you can switch between to enable a desired view of the data at any given time. Tree data can be generated and edited by point-and-click and form fill-in actions in the (technical) tree view which also poses as GUI-editor, as shown in figure 5.

3.1.3 Treebolic Application

The Treebolic Application provides a basis to tailor your own application for data visualization. It consists of a clean Treebolic widget powered by the Treebolic engine with support for displaying hierarchical data from a provider visualized as hyperbolic trees, see figure 6. With relatively little effort it is possible to introduce features and application-specific modules from both the Browser and the Generator, such as menus, toolbars, and status bars.

4 SALmon

SALmon is the proposed data provider that can describe the service model (Leijon et al., 2008). A prototype implementation of the full language is under development (Leijon et al., 2009), it is therefore (at the time of this thesis project) not yet possible to perform visualization experiments

\(^4\)Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Definition courtesy of W3.org.
on live data provided by the SALmon engine. Through an Application Programming Interface (API), service model data can be requested and retrieved. The retrieved data is then subject to be visualized in a suggested graphical manifestation. The development of an API for SALmon has been initiated but deserves more attention as it is not yet operational.

With SALmon and its proposed API still undergoing development it cannot be accountable as data provider for the visualization at the time of this thesis project. However, it is possible to implement a visualization of a hypothetical service model composed by static data. This allows for further investigation on how to overcome other challenges that are known to the process of visualizing service models such as system requirements, proper exploitation of the available resources, the volume of service models to be processed, implementing suggested features of the GUI and so forth.

5 HyperSALmon - A Service Model Visualization Prototype

HyperSALmon is the proposed software prototype assembled for the purpose of this thesis project. The prototype intends to provide hyperbolic visualization of GSM service model data provided by the SALmon engine, thus the name; HyperSALmon. The prototype harvests the Treebolic engine for visualizing the service model extended with domain-specific tools to incorporate the implementation of desirable functions and features such as advanced search, sorting, time-event slider etc. HyperSALmon is, at this time of writing, far from being full-fledged. But it has been proven capable of visualizing the service model using tree structures through a technical view, although in some respects, hypothetical, and that the implementation of the its
suggested functionality is feasible. The prototype architecture is illustrated in figure 7.

5.1 Challenges

The main challenges of implementing a visualization of the service model derives from the stated system requirements in section 2.2. Furthermore, providing successful implementation of desired functionalities, visibility and other quality criteria is just as important and challenging. The visualized service model must be navigable, interpretable and accurate. This implies that a balance of available resources and required features is taken into account in the implementation process.

Generally, the prototype must be able to process, update and visualize large amounts of data in real-time. To achieve this it is suggested to implement a navigational filter that allows a compromise between the available visualization resources (screen space, system memory, CPU-power etc.) and essential focus areas of the service model. Attempting to display too much data at once will only clutter the overview making navigation almost impossible, this was evident in an experiment carried out at an early stage in the prototype development phase, see figure 8.

5.2 Implementing suggested features

I will now discuss the desirable features of the visualization, derived from the report (Sehlstedt, 2008, page 51-52) and the domain experts at Data Ductus Nord AB, and how they are reflected in the prototype implementation. In conjunction to this I will explain how features were implemented or if a feature for some reason not could be fully implemented and, if possible, propose alternative suggestions on how such implementation could be realized.
Figure 6: Treebolic Application. Initialization; If no data provider has been set you are requested to enter a data provider URL.

Figure 7: HyperSALmon architecture.

Note: I would like to clarify that the main reason why many of the features was not fully implemented is that the implementation work do claim a lot of time with respect to that it is performed by a single person and within the time frame set for this bachelor thesis work. Another influential factor was that it, at the time, was not possible to provide the prototype with live data from SALmon. Instead, snapshots of experimental GSM data was used, as the prototype was developed in a sandbox\(^5\).

5.2.1 Suggestion 1: A traditional tree visualizing the technical view of a service model.

Problem description: As seen in figure 5, Treebolic Generator supports displaying hierarchical data in multiple views. The tree-view editor and the XML-tree view are two views that visualize the hierarchical data as a traditional tree structure.

Proposed implementation: By introducing tabbed views in HyperSALmon it would be possible to display the actual data simultaneously in multiple views. The suggestion implies

\(^5\)An isolated area where a program can be executed with a restricted portion of the resources available. Definition courtesy of Wiktionary.org
that data should be visualized in a technical-traditional tree and hyperbolic tree view. This suggestion can be achieved by taking advantage of the Treebolic Generator source code for this purpose.

5.2.2 Suggestion 2: An “Overview + detail” interface to provide instant status overview.

Problem description: The main purpose of Treebolic is to provide hyperbolic rendering of hierarchical data (Bou, 2009). This rendering implies to give a good overview and navigation of the data. However, the hyperbolic rendering alone does not provide specific detail about the data, thus a status bar provides a detailed interface describing selected parts of the tree. An overview + detail interface is described as a component consisting of two windows, one window providing the overview and the other window providing the detail (Hornbæk et al., 2002).

But what do the terms “overview” and “detail” really implicate? In order to explain the terms Hornbæk et al. (2002) utilizes the concept of information space. The overview window should offer a comprehensive display of the information space whereas the detail window should provide details of the information space. The abstract concept of “information space” is affected by numerous definitions depending on its context of use, but generally it can be described as the “entire amount of information available on a particular subject”\(^6\). In the context of service modeling, the information space might refer to the entire amount of information available on a service.

Proposed implementation: HyperSALmon taps the power of Treebolic’s hyperbolic view and status bar to provide an overview + detail interface as shown in figure 9. Instant status

---

\(^6\)Definition courtesy of BusinessDictionary.com
overview is achieved by clicking on a node in the hyperbolic view. The view then animates to the selected node and its attributes are displayed in the status bar. If a specific node is out of scope the user can click-and-drag to navigate within the tree, alternatively use the HyperSALmon search engine to locate the node.

![HyperSALmon interface](image)

Figure 9: HyperSALmon. “Overview + detail” interface.

5.2.3 Suggestion 3: A search function allowing multiple terms search, and providing own search term suggestions in case of misspellings from the user.

**Problem description:** The suggestion 3 (this) and suggestion 4 (below) are interconnected as they relate to functionality in respect of the prototype’s search engine. Treebolic comes with a simple search engine. The search is executed from a context menu that is called when a user right-click on a specific node in the hyperbolic tree view and only affects the selected node and its children. Search terms are entered in a separate status bar form. This search engine only supports case-sensitive single search terms, searching only one of four pre-defined String variables (label, content, id, link) at a time and only returns the first encountered match or no match at all. If a match is made, the status bar displays in what node the match occurred and the hyperbolic view is automatically animated to focus on that node. The search engine can be customized to some extent as it by default needs to be declared within a tooltip section of the provided XML data file. Figure 10 shows a declaration of a search-tool that will return the node where the first match of the provided search term is found. The match-mode “includes” means searching for a match anywhere within the “content” String, match-scope is set to only search the “content” variable and the match-target “%e” is an anchor reference variable for the provided search term (which will be retrieved from the search-term field).

Declaring a search-tool that will return the node where the first match of the provided search term is found. The match-mode “includes” means searching for a match anywhere within the “content” String, match-scope is set to only search the “content” variable and the match-target “%e” is an anchor reference variable for the provided search term (which will be retrieved from the search-term field).

**Proposed implementation:** Adapting the Treebolic search engine to a service model search engine requires a number of modifications to be made. One fundamental change meant that the declaration of node objects was rewritten to take service model-specific variables into account i.e., make service model attributes significant to the search engine. Furthermore, searching must be maintained until all the nodes within the scope of search have been searched and any search results must be stored in an array. Support for multiple search terms can be implemented
by connecting and matching multiple search criterias with each other. This is necessary when more complex problems needs to be located for example, if the operator wants to locate all non functional cells in a specific area where the longest time to repair has exceed the limits of the current SLA. Shneiderman and Plaisant (2005, chap. 14.2) suggests that interfaces for simple search and more advanced search, i.e. including multiple search terms, should be kept separate or multilayered.

In order to provide search term suggestions in case of user misspellings it is required that some kind of dictionary or word database is linked to the search input form. Whenever a user commits a search an algorithm compares the search terms to similar words in the dictionary or database. If similar words are returned they will be presented to the user as suggestions in case there was a misspelling. An alternative way of doing this could be to connect a key-listener to the search input forms. Whenever a character is entered or removed from a search form an algorithm runs a comparison between the entered word (or part of word) and the dictionary or database. If matches are made they will be presented to the user for example, in a pull-down menu under the search input form.

Shneiderman et al. (1998) claims that the ideal search interface is comprehensible, predictable, and controllable, and that many of the current text-search interfaces are unnecessarily complex and characterized by obscure key features. Google search is an example of a successful search engine that provides search suggestions, simple and advanced search implemented in an exemplary way. In February 2009 Google search was the highest ranked search engine in the U.S. accounting for 63.5 percent of all conducted search queries during the period (Nielsen-Online, 2009). However, searching service models and searching text on the Internet differs in various respects and cannot be equated to each other. But designing the prototype’s search engine by, wherever useful, exploit some similar elements from perhaps the world’s most used search engine can have a positive effect on users who will feel familiar with certain elements and search features. Incorporating elements of “best practice”, standards and proven key success factors in own designs is a common phenomenon on the market and popular approach reported by researchers (Schneider and von Hunnius, 2003).

5.2.4 Suggestion 4: Clearly visible search hits with, e.g., highlighting in the both the detail and the overview window, and with automatic translation of the detail window to make search hits come in focus.

Proposed implementation: Once the search has been executed the results should be presented in a lucid manner. This could be achieved, for example by presenting all the nodes where search matches were made, in a status bar as a text list of hyperlinks. Clicking on a hyperlink would
then animate the hyperbolic view so that the specific node is brought to focus. Figure 11 shows how an alarm search function is declared. The search engine is here adapted to accept the “alarm” match-scope. When the alarm search is executed on a specific node it will return any matching node that has a status variable with the value “0”. A node within the GSM model, which can be represented as a service, area, cell etc. is considered to have a status to be either up or down, hence represented by the values “1” or “0”.

```xml
<tools>
 <menu>
   <menuitem action="search" match-mode="includes" match-scope="alarm" match-target="0">
     <label>
       Search "%l" recursively for alarms
     </label>
   </menuitem>
 </menu>
</tools>
```

Figure 11: HyperSALmon. Declaration of an alarm search function.

### 5.2.5 Suggestion 5: A kind of semantic zooming to accomplish stronger connection between the nodes in the tree and their respective attribute information.

**Problem description:** Hornbæk et al. (2002) explains the concept of “semantic zooming” which may occur when observing a geographic area or map where the same area on the map might be shown with different features and amounts of detail depending on the scale.

**Proposed implementation:** The hyperbolic visualization of the service model provides a “fish-eye” view of the data, focusing on a small portion of the data at any time while the rest of the data is located in the peripheral regions of the screen until a change of focus is enforced. As shown in figure 8, attempting to visualize too much data on screen at a given time makes the hyperbolic view almost impossible to navigate in. This phenomenon may also occur when the user chooses to zoom out, bringing the nodes closer together in the view.

I propose that semantic zooming could be implemented so that when a user zooms out enough, bringing nodes too close to be able to obtain necessary detail, related nodes could be represented by a single node at that specific level of zoom. For instance, when zooming out and nodes are brought closer to each other, the grouped nodes that pass a certain threshold for how close they are allowed to be each other in the view, are collapsed together as a single node. If the user keeps zooming out these nodes, passing yet another threshold for level of detail, collapsing the clustered nodes as a single node. Respectively when zooming in, the collapsed nodes expand when passing the thresholds for levels of detail. This may also improve visual performance and frame rates in dense and voluminous service models as the hyperbolic tree will have less nodes to be visually rendered and refreshed at specific zoom levels.

### 5.2.6 Suggestion 6: A sorting function allowing the user to choose between alphabetical order, geographical order, order based on number of open tickets, and order based on downtime.

**Problem description:** The suggested implementation of a sorting function is applicable on several operations carried out by the prototype, and perhaps most justified as enhanced func-
tionality of the search engine. The suggestion implies that a sought quantity is to be ordered according to specific attribute values and then presented to the user in an appropriate manner.

**Proposed implementation:** The sorting function can be applied to a list of returned search matches, rearranging the results in ascending or descending order with respect to a certain attribute value. This could be realized by, in the list or table of returned search matches, clicking on a column heading to sort the data, switching between ascending or descending order every time the user clicks on a column heading. Numeric values are sorted from highest to lowest value or vice versa whereas alphabetic values are sorted from A-Z and Z-A respectively. Whenever multiple identical values are returned, for example boolean variables displaying status “1” or "0", sorting can be further refined by clicking on any other desired column header. For instance, sorting all nodes/components with status "0" with respect to their longest down-time (the time in which the node/component has been out of service).

Domain experts at Data Ductus Nord AB have expressed a need to provide the user a setup menu or “wizard\(^7\)”-like feature to be launched at the application initialization and on request anytime during execution. This interface will act as a filter allowing the user to select desired scope and focus on hot-spots or problems within the service model depending on geographic areas, number of open tickets, longest open cases etc. I propose that these lists should also be sortable and searchable in order to aid the user in a swift setup of a desired focus and navigation outset of the service model browser. This is discussed further in suggestion 13 (section 5.2.13).

5.2.7 **Suggestion 7:** Greater separation of different levels in the hierarchy by increasing the interspaces (width ways) between them.

**Problem description:** To be able to easily recognize the steps between different levels in the hierarchical data of the service model, it is suggested to increase the interspaces between them.

**Proposed implementation:** The interspaces between hierarchical data levels in the hyperbolic view are by default quite wide. In case the interspaces still feels too short, a tool to increase or decrease the interspaces to the desired length is available.

5.2.8 **Suggestion 8:** Presentation of current attribute values in context with past values using sparklines.

**Problem description:** This suggestion highlights the importance of being able to monitor the attribute values over time using sparklines (line graphs) for visualization, as discussed in detail in the report (Sehlstedt, 2008, section 5.4).

**Proposed implementation:** By selecting, dragging and releasing attribute values from the status bar a graph tab or window pane a sparkline graph could be generated. Once the sparklines have been generated the user can either leave the window to produce a graph over time using the time of generation as initial value and current value as end value. More advanced features may include sliding the initial value further backwards in time to increase the time-span of the graph and perhaps the option of highlighting extreme values and outliers.

5.2.9 **Suggestion 9:** A function making it possible to create sub graphs, these could then be placed in a tab menu to facilitate change of graph view.

**Problem description:** As with suggestion 8, this feature allows the user to monitor selected and separate elements of the service model by locking them on screen as a tab menu or such.

---

\(^7\)A wizard is a user interface element where the user is presented with a sequence of dialog boxes. Definition courtesy of Wikipedia.
Even though the user changes focus or scope of the main tree view, the locked sub graph is always available by switching to its dedicated tab.

**Proposed implementation:** By selecting a node within the hyperbolic view and from a context menu, which is retrieved by right-clicking on the node, the user gets the choice of generating a sub graph of the selected node and its children with the selected node as root node in the sub graph.

I propose creating sub graphs from the context menu since the left mouse button is reserved for panning and dragging the visual scope in the hyperbolic view. Alternatively, to be able to use the left mouse a key combination such as pressing ALT + left click could generate a sub graph tab as mentioned.

### 5.2.10 Suggestion 10: Dynamic queries connected to the attributes, to provide more advanced filtering than in the prototypes.

**Problem description:** Dynamic queries are an interaction technique that allows the user to filter out the components that are undesired in the service model at the moment. For instance, if the user would like to only emphasize the nodes that have 5-10 open tickets or nodes where a certain attribute value is reported. It is suggested that components that has to be displayed not to disrupt the structure, but otherwise would be filtered out, ”should be treated the same way as inaccessible menu choices so that it becomes obvious which components that belong to the chosen interval” (Sehlstedt, 2008, page 51).

**Proposed implementation:** By implementing a filtering menu the user could have the choice of filtering out various components by using check-boxes for pre-defined popular filters or by providing custom terms in a text form field.

### 5.2.11 Suggestion 11: Adaptation of colors to gain better visual clarity, and perhaps more than four different colors to visualize status.

**Problem description:** Color coding is a powerful and useful aid in conveying status and classification of an entity. If used properly it facilitates the understanding of the entity status and classification, but conversely, an improper use may invoke misunderstanding, confusion and distraction to the user. Thus, the definition of a promoting color code schema requires consideration of some general guidelines (Sehlstedt, 2008, section 5.2).
Proposed implementation: The prototype visualizes the status and classification of the service model components through a simple color code schema (table 1) based on recommendations for Human-Computer interaction design guidelines (Brown, 1998). One of the main issues to consider is that problem areas are clearly visible and correctly interpreted. The suggestion is to determine whether more than four different colors can be successfully used to visualize status. As recommended in table 1, color codes for alarms ranges between red, orange, yellow and green. For determining node classes correctly no more than four colors is recommended (Burnette, 1985). Node classification colors consists of blue (GSM cell) and gray (GSM cell children).

Table 1: Human-Computer interaction design guidelines: Color coding recommendations.

<table>
<thead>
<tr>
<th>Color</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>White/Black</td>
<td>Base color</td>
</tr>
<tr>
<td>Red</td>
<td>Alarms or errors, Stop</td>
</tr>
<tr>
<td>Yellow</td>
<td>Warnings or data that may require attention</td>
</tr>
<tr>
<td>Green</td>
<td>Normal or OK, Go, Base color if white is too bright</td>
</tr>
<tr>
<td>Saturated Blue</td>
<td>De-emphasis, Shading, Not for critical data</td>
</tr>
<tr>
<td>Desaturated Blue</td>
<td>Cold temperature or water</td>
</tr>
<tr>
<td>Pink (Magenta)</td>
<td>Secondary alarm color, differentiating data</td>
</tr>
<tr>
<td>Turquoise (Cyan)</td>
<td>Differentiating data types</td>
</tr>
<tr>
<td>Other colors</td>
<td>Differentiating data types</td>
</tr>
</tbody>
</table>

Figure 13: HyperSALmon. Example of color adaption; using more than four different colors for visualizing alarm status.

5.2.12 Suggestion 12: A “Time-slider” for browsing/tracking historic events within the service model.

Problem description: Monitoring of the service model requires the ability to track and view both current and historic events. Data Ductus Nord AB proposes the implementation of a “time-slider” that allows the user to slide back and forth in time for selected parts of the model.
Proposed implementation: The Proposed implementation for the implementation of this feature is to provide a time-slider from a context menu whenever the user right-clicks on a node within the hyperbolic view. The time-slider is visible but disabled until the user toggles a check-box that enables the time-slider component. Once the user has enabled the time-slider, the background color changes slightly, conveying that the user is now browsing historic events. Since the time-slider only affects the selected node and its children, unaffected nodes will be filtered out, in the same manner as described in suggestion 10, to clarify that these are not concerned by the time-slider feature.

Dragging the time-slider back and forth generates requests to SALmon for historic data for that particular point in time on the selected node and its children. The range of the time scale of the time-slider component should be a dynamic value that can be redefined by the user. The time-slider is disabled and reset to display current time when the user uncheck the check-box or clicks anywhere outside the context menu.

5.2.13 Suggestion 13: Navigational filters to help user focus on hot-spots and problems.

Problem description: This suggestion is partly discussed in suggestion 10, but this suggestion may imply a wider scope of use.

Proposed implementation: Navigational filtering entails that the user is aided in navigating within service models through filtration. When this feature is requested the user may provide some filter parameters describing where to navigate and what key elements to bring into focus. These parameters should include at least a geographic location but could be complemented by additional parameters to narrow the scope. Additional parameters could be the number of open tickets, cases that have been open for longer than a specific amount of time etc.

This feature has been expressed as especially motivated to be utilized in the prototype’s start-up sequence, allowing the user to select origin of browsing and elements of focus. But it could also be necessary each time the user wants to relocate themselves by traveling large distances in the hyperbolic view of the service model.

For example, the user may want to bring up all GSM areas in the geographic location of ”North Carolina”. This request should retrieve GSM services for all GSM areas located in the geographic location of North Carolina and present them on screen. Depending on the granularity of the filters and functionality of the SALmon API commands, it may be possible to specify locations that cover much smaller geographical areas, such as cities, place names and the like.

6 Implementation Experiences

HyperSALmon is capable of visualizing a hypothetical service model through a technical view. This has been achieved by using cached service model data that have been parsed into XML for visualization by the prototype. Since service model data is dynamic, i.e. subject to change over time, a visualization of static XML data is not sufficient when implementing visualization for service models. In order to determine whether the prototype is capable of visualizing the intended dynamic service model provided by SALmon, an operational API for SALmon and service model database should be deployed and connected to the prototype. As mentioned, this implementation is under construction (Leijon et al., 2009). Alternatively, to simulate the behavior of the SALmon API, method stubs could be written for this purpose. However, writing method stubs requires detail knowledge, insight and access to the SALmon language and engine. SALmon is described both briefly and in detail in various documents Leijon et al. (2008,
2009); Wallin and Leijon (2007). Commencement of writing method stubs was not initiated due to resource constraints and scope of this bachelor’s thesis.

6.1 Maintenance

The application of HyperSALmon is not yet explicitly determined but can be considered to be broad since it is implemented in a platform independent language. It can be offered to end-users as cloud service\(^8\) on a powerful server. An enterprise solution as such means maintenance only needs to be focused on one instance of the service. Enterprise solutions are powerful when they deliver as intended, but flawless enterprise solutions are considered to be utopias and when they collapse the consequences could be costly (Davenport, 1998). This would also be the case if HyperSALmon is to be run on client computers as a Java\(^\text{TM}\) applet or web start application, but since the hardware of the hosting system will vary (since every instance of the application will be executed locally) the visualization may differ and produce an unsatisfactory experience.

If HyperSALmon is to be downloaded by end-users and run as a stand-alone client application it will present difficulties in terms of maintaining all the downloaded instances of the prototype. If patches and bug fixes are produced it could either mean that the users must manually download these from a hosting server and apply them. Another suggestion may be to implement an updating feature that is integrated in HyperSALmon which automatically searches, downloads and applies these while the application is running.

6.2 Functionality

The implementation of the suggested features as described in section 5.2 is in progress. Some of the features have been implemented partly and some has not yet been initiated due to time constraints. Some of the features require that method stubs or a SALmon API to be operationable. This list briefly summarizes the functionality implemented so far:

- Basic hyperbolic visualization of a static and hypothetic GSM model.
- Menu bar added (File, Options, About).
- Displaying icons properly on nodes.
- Node properties/status bar panel added.
- Toolbar added.
- Time-slider added.
- Time-slider check box added (controls whether time slider enabled and resets time slider to current time if deselected).
- If user closes context menu and has checked the time-slider check box, time-slider is automatically deactivated.
- Simple search/alarm location functionality supported (one search term, 0 - 1 hit).
- Support for node each attribute to be stored as unique data types instead of being represented as a single String variable.

\(^8\) A service in which dynamically scalable and often virtualized resources are provided as a service over the Internet. Definition courtesy of Wikipedia.
6.3 Usability

The usability of HyperSALmon has not been investigated enough to be able to draw any clear conclusions as no usability tests have been performed. As the evolution of the proposed functions and the SALmon API progresses, more and more meaningful opportunities to perform usability tests should arise. So far, any observations made of the prototype’s usability are more or less of empirical nature. Testing (low- and high level) and quality assurance of the prototype has not yet been performed in any orderly and notable sense.

6.4 GPLv3 issues

HyperSALmon is based on Treebolic, which is released under a GPLv3 and thus considered to be free software”. Software released under the GPLv3 means that it will be free software and stay free software, no matter who changes or distributes the program (Smith, 2007). Consequently, HyperSALmon must be kept open source and available to other developers. This should not be considered as any threat to the SALmon license since the GPLv3 only affects the visualization prototype, as SALmon is a separate application that only acts as data provider for HyperSALmon.

7 Discussion and Conclusions

This thesis project should be considered both as a feasibility study and as a proposition for the further development of a full-scale service model visualization prototype. Based on the effects demonstrated in this report I claim that the Treebolic engine is sufficient for visualizing hypothetical service model data and that its GUI can be extended to support desired features, as I believe these are matters depending on reasonable time and resource inflictions. I have also presented proposals on how suggested features in a future GUI could be implemented.

One of the main drawbacks with Treebolic is the lack of documentation and community activity support. Although I consider it to be the most promising open source framework for this purpose, the time it took to familiarize myself with the code architecture was significant with respect to the time frame set for this thesis work.

In retrospect, the amplitude of the tasks involved in this thesis work would benefit from efforts by more than one student.

7.1 Future Work and Further Development

The future work and evolution of the prototype needs to be discussed, planned and further supported by domain expertise and workforce as the full-scale implementation of a service model visualization may be considered a demanding task in terms of detail domain knowledge and resources.

I propose the following high priority issues for further development of the prototype (This list may serve as a basis for further discussion and identification of more issues of development):

---

7.1.1 Prototype resource- and project planning

Available resources for further implementation work should be reviewed and a project group deployed. A rough project plan should be composed, since strict and detailed project planning is known to impede creativity (Blomberg, 2003).

7.1.2 Discuss and decide whether any methodology should be utilized

Composing a project plan is one thing, but making sure that the plan is appreciated, followed and that the project evolves in a desired direction requires methods, leadership and determination. There are several methodologies that can be supportive in this matter.

7.1.3 Deployment of test team and master test plan

Quality assurance should be considered. For this issue I propose the utilization of testing. One approach that may be favorable in this issue is the Test Management approach (TMap) (Pol et al., 2002).

7.1.4 Construction of method stubs or deployment of an operational SALmon API for realistic data provision and visualization

It is important for the further development of the prototype that it is provided data in realistic volumes, types and through method calls that will be compliant with the SALmon API.

7.1.5 Producing documentation

Since Treebolic does not come with any comprehensive documentation, except for the Java™ documents, there is a need for producing documentation of the HyperSALmon components and modules (i.e. it’s architecture). It is important that the involved project workers share the same mental model of the prototype. This is usually mandatory in project collaboration and principal when utilizing a methodology.
8 Acknowledgements

First of all, I would like to thank Data Ductus Nord AB for offering me this opportunity to perform this bachelor’s thesis. Thanks to Urban Lundmark, Chief executive officer at Data Ductus Nord AB, who answered and forwarded my request to do this bachelor thesis work at their company. Thanks to Stefan Wallin, Monitoring expert and my supervisor at Data Ductus Nord AB, who set up a workspace for me, explained the problems, introduced me to \LaTeX and helped me focus on what is essential. Thanks to Ulrik Forsgren, Senior programmer and SALmon API developer, for demonstrating SALmon. Thanks to André Jonsson for introducing Das Keyboard II\textsuperscript{10} which I have gratefully borrowed during 8 weeks. Thanks to Göran Landgren, my mentor at Umeå University, who helped me formulate my problems, provided literature, shared his thoughts and ideas about the project and proof-read my reports. Thanks to Torbjörn Nordström, examiner at Umeå University. Thanks to Bernard Bou for his open source application suite; Treebolic. I would also like to thank everyone else at Data Ductus Nord AB that has shared their opinions, interest and suggestions related to my work. Finally, I want to express a special thanks to Mathias Gyllengahm, Senior programmer at Data Ductus Nord AB, who voluntarily dedicated a lot of his time helping me overcome major obstacles, shared his thoughts and programming expertise, and to my family and friends for encouraging and supporting me throughout my work with this bachelor’s thesis.

\textsuperscript{10}“The blank keyboard that clicks”
References


A Prototype source and binaries

For the latest build of HyperSALmon please visit http://66.7.200.105/~fellownu/hs/

A.1 Release notes

Prototype download provided with reservations for URL misspellings, denial of service and other unintended but influential circumstances that are out of the author’s control. Download URL may change or be removed without notice.

If you experience problems, send an email to svph0609@informatik.umu.se.

HyperSALmon requires Java 6™ or later to run. Attempting to run the prototype with any previous version of Java™ will fail. Download Java™ from http://java.sun.com/.

HyperSALmon by Andreas Lundgren is a service model visualization prototype, based on the Treebolic application copyrighted © by Bernard Bou.

Free use of this software programme is hereby granted provided you accept the terms of the GPL3 (GNU General Public License).

The bits of code written by Christophe Bouthier are under the MIT OpenSource agreement.

The parts of code written Vladimir Bulatov come with the following copyright notice:

Copyright 1996 by Vladimir Bulatov V.Bulatov@ic.ac.uk. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

A.1.1 Disclaimer

NO WARRANTIES. THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.