Geographical and Syntactical Integration of Geospatial Data

Master Thesis in Computer Science

Kristian Lunde

December 11, 2005
Halden, Norway

Høgskolen i Østfold
Avdeling for Informasjonsteknologi

Abstract

A map repository is a collection of geospatial data that describes elements of the world. The geospatial data describes features geometrically and semantically. One of the crucial tasks a map repository has to support is the integration of new data, and the updating of already existing features. This thesis deals with the problem of integrating and updating geometrical features in a map repository. The integration is viewed both from a geometrical and a syntactical point of view.

The geometrical integration process section of this thesis contains one theoretical part and one practical part. The theoretical part of the geometrical integration process contains a detailed description of the problems that arise when trying to merge data sets which are different from each other. When the problem has been thoroughly described, one or more solutions to these problems are described. The practical part describes a step by step process of doing a geometrical integration. The practical geometrical integration process is based on the solutions which are sketched in the theoretical section.

The second major element of this thesis is the syntactical integration of geometrical data. Similarly to the geometrical integration, the lazy integration is divided into a theoretical and a practical section. The theoretical part uses a method called lazy integration originally developed for semantical data, as a foundation to build a lazy integration method for geometrical data. The main objective with lazy integration is to preserve the structure of the integrated data set. Using this method, it is possible to store geospatial data that have different structures and information in one GML file. The practical section uses the lazy integration process on two different problem cases.

Both the theoretical and practical part of this thesis contain real world examples to define and describe problems and solutions.

Acknowledgements

I would like to thank Gunnar Misund for valuable guidance and supervision throughout this master thesis. I would also like to thank Linda Kjeldsen, Hilde Steinheim, Mats Lindh, Christer Stenbrenden and Bjørn Håkon Horpestad for a lot of fun when we wrote our master theses. Thanks goes to Morten Granlund for the helpful inputs on writing a master thesis. Kudos goes to Harald Vålerhaugen for his help and guidance on the lazy integration approach.

At last I would like to thank the JUMP team. Without the development of the JUMP workbench much of my work in this thesis would have been much more troublesome.

Prerequisites

The readers of this master thesis should have knowledge similar to a bachelor degree in computer science. It is also preferable with thorough knowledge about geographical information systems. It is preferable that the readers also possess some knowledge on standards as XML [15], GML [17] and XML schemas [31].

Contents

List of Figures

Chapter 1

Introduction

This thesis deals with the integration of geospatial data. This includes both the geometrical and syntactical integration of geospatial data.

Mankind has always used maps to navigate in its surroundings. These maps have been illustrations of the surroundings and displayed important information. A map is a scaled down illustration of the world and our geographical environments. For the common user it is important that the map is displayed correct. However, a map is only correct in a limited period of time. The world and our geographical environments are constantly changing. These changes may be influenced both by nature and by humans, they can also be small or large scaled. Small changes may be a slope failure or an alteration of a road. A large change of the environment is for instance the earthquake that lead to the tsunami incident in the pacific in December 2004. The earthquake moved large islands with several centimeters. The dynamic environments require that maps are updated and corrected from time to time.

A map repository is a collection of geospatial data which describes elements of the world. The geospatial data describes geometrical and semantical features. One of the crucial tasks a map repository have to support is the integration of new data, and the replacement of already existing features. This master thesis deals with the problem of integrating and updating geometrical features in a map repository.

Geometrical integration is the process of integrating geometrical features from one data set into another data set. It is similar to a map conflation method called boundary alignment. Map conflation is a difficult and troublesome problem area, and boundary alignment is the easiest of the conflation methods. Other map conflation [40] methods are out of scope in this thesis. The geometrical integration process integrates a section of data to an already existing data set, this section of features are new and lie adjacent to already existing features. The integration of new data into an already existing data set does not go painlessly. Problems that occur in this process can occur from different reasons. The two main reasons are errors in the structural build up of the file or geometrical errors in one or both of the data sets. These errors result unaligned data sets. Line segments that cross data sets can and most likely will not coincide without any modifications. To solve this problem the data sets have to be aligned along their common borders. This means that the line segments that cross the border of the update area become coherent, and result in one data set, with

continuous line structure. To approach the geometrical integration, JUMP [22] will be used to perform a geometrical integration. There will also be sketched solutions to the problems that are pointed out.

The geometrical integration does the integration of two geometrical data sets, however, it does not say anything about how these data should be integrated syntactically. Syntactical integration deals with the process of integrating several data sets to one data set. The process of integrating new data into an already existing data set syntactical is problematic. It can often result in loss of information or precision. These problems occur when features are converted from their original state into the structure of the new data set. Loss of both information and precision are of course an unwanted side effect of the syntactical integration. To solve this problem a method called lazy integration will be explored, to see if it suites as a method of integrating geometrical data syntactically. Lazy integration has earlier been used as to integrate semantical data into one data set.

The information on the processes of the geometrical and syntactical integration is important, and should be stored together with the integrated features. Such information is called metadata, and is stored together with the geometrical features. Metadata contain information that is relevant to the data set, for instance the date of the integration process, the coverage of the feature, the resolution of the feature, the format of the original data, description and creator. Inconsistency will occur if it is integrated a unknown data source. It is important to know the resolution of the existing data sets, the original sources. This information is decisive to further integration processes. For instance it is not desirable to replace data in the repository with lower resolution. This is why metadata is a vital part in both the geometrical and syntactical integration process. Metadata will be examined and used to develop a structure that contains relevant information for the integration processes.

The scope of this thesis is to explore the problems of geometrical and syntactical integration. Semantical integration is out of scope in this thesis. The purpose is also to sketch solutions to these problems. It is also a goal to explore the possibilities of geometrical integration using the JUMP [22] workbench. At last it aims to develop a lazy integration approach for geometrical data.

1.1 Revision

Revision is the process of confirming the correctness of information and correcting the errors found in that process. The process of revising information is done with information where the correctness of the data is not known. It is also done with the result of a process or product to verify the correctness. A revision process can be performed in small scale, for instance a person which is checking a letter for typing errors and correcting them is performing a revision. A small revision is usually performed by few persons. It can also be performed in large scale; this is usually done with important data. A large scale revision includes several persons, and over a significant time period. A large scale revision will most likely require that the finished result has a low error rate. The NASA space shuttles are objects exposed to large scale revisions. The shuttles need to be as secure as possible to

ensure the safety of both humans, and expensive equipment. The revision of the shuttles has to reveal critical errors that can endanger the mission. The NASA revisions are may be one of the largest forms for correctness checking in the world. In this setting a revision will be less critical but indeed important to the consistency of the map repository. It is important to ensure the correctness of geospatial data that should be integrated in the repository. If erroneous data should enter the repository it would lead to an inconsistent repository. The correctness of any data extracted from the repository would in that case be unknown. This is the main cause why new data should be revised before it is added to the repository.

1.1.1 Incremental Update

An incremental update relate to the integration of new data into an existing system. A repository is a storage system which can store large amounts of data. The repository can be built in several ways, a repository could be built and all information could only be added once; it would not allow entrance of new data. The repository could have to be rebuilt whenever new data was added. The most sensible way to do this however, is by an incremental update anytime new data should be added to the repository. The use of incremental update can be performed in different ways depending on the data structure and information type. An incremental update of a repository enables it to expand at anytime. With this ability the repository is dynamic to fit different needs at different times. One of the key features with an incremental repository is that data can be added when it is collected or available. This means that a repository can start with a small amount of data, and end up with vast amounts of data. The major advantages with a repository with incremental update support, is the ability to expand it at any time.

The Bottom/Up Approach

The bottom up approach is a way of storing data sets and storing the changes done on those data sets. With this strategy the original data sets are stored as they originally are. Whenever changes are applied on one of the data sets, these changes are stored as patches in separate files. These patches only contain the changes done on the data set. A new patch is created every time a new change is done on a data set. A change can apply both for the original data sets and patches which have been applied to the data set earlier. With this approach the original data set is kept in its original state. In order to view the data set in its newest version; the data set has to be loaded and the patches have to be applied to the data set.

The Top/Down Approach

In contrast to the bottom up approach, the top down approach does not store the original data sets as they are. When an update on a data set is performed it is changed to fit the latest change, and a patch is created. The patch contains the changes which are applied to the original data set. This result in a different way to display the data set. The data set is

always updated and correct, but if the original data set should be viewed the patches have to be applied to the data set. This revert the data set to its earlier or original state.

1.1.2 The OneMap Project

Project OneMap [36] is a long term project that work with geographical data. One of the main goals of project OneMap is to contribute to an open GeoWeb. It aims to share geographical information using open formats, developing and distributing tools as open source. The OneMap philosophy is to keep things as simple as possible. At all times there are several ongoing sub projects and master thesis in the OneMap project. During the lifetime of the OneMap project; it has been the source of several publications, presentations and papers. The OneMap project is an associative member of the Open GIS Consortium.

The thought about simplicity also applies to the physical storage of data. The OneMap repository is file based, and stored in tiles. A tile based storage system means that the geographical features are stored in rectangle areas; these tiles are organized with naturally coherent features that cover a limited coverage classes, for instance a coastline class. Today the OneMap repository support GML 2 [17] format. Future development may introduce GML 3 [18] as a standard file format. One of the advantages with GML 3 is the possibility to store topological information on features.

An OneMap Repository [33] article describe a new design of the repository. It discusses through several scenarios the possibilities that one might achieve by doing some changes in the structure of the repository. This article proposes a repository that is able to store historical data. Historical data would enable the repository to revert a data set to an earlier edition if that would be necessary. The repository will in that case support retrieval of historical changes in maps. Historical changes are vital to several systems for instance emergency rescue operations and monitoring of glaciers. Further on the article describe storage of metadata together with geometrical features, with is essential in geometrical integration cases. The article also propose a new view on the storage focus, it is suggested that a feature based focus could give a more satisfying focus than today's tile based focus. The main limitation of a feature based archive is automatically updates of large data sets. With this argument the feature based focus are somewhat doubtful.

1.1.3 The Peer-Review Process

The OneMap [36] repository has chosen a peer-review process as a revision method to ensure the correctness of new data. A peer-review process has its origins from the verification of academically papers and prepublications. In such a process the work of an author is distributed to other persons with expertise in that area. These persons do a validation of the work and either accept it, reject it, or propose some changes or enhancement of the work. This process can be performed in several iterations, which result in a final report or a publication.

The peer-review process used in the OneMap [36] project is much alike the original peer-review process. The major difference is the data which should be verified. In contrast

to the original peer-review process which reviews papers or prepublications, the OneMap peer-review process review geospatial data. The purpose with this peer-review process is to get a correct data set which can be integrated into a map repository. It is required that the geospatial data is without flaws; this makes the peer-review process very useful in that situation. The peer-review process is initiated when a new piece of geospatial data should be added to the map repository. Whenever new data should be added, the geographic area where the data should be submitted is locked. This is done to ensure consistency of the map repository. For instance if two updates over the same geographical area were to be added to the repository simultaneous, this would create uncertainty of which of them that were newest and therefore most correct. If an update of the repository is initiated while another update already is in the peer-review process, the second update will be rejected. The person that initiates the peer-review person is the first person that verifies the data as correct. When this is done, other persons that have a good overview of the area view the data, and either accept it or reject it, or propose corrections to the data. During this process the data might undergo major editing, and might be altered by all persons involved in the peer-review process. After each alteration every involved persons have to agree on the alteration. When agreement is reach on the correctness of the data, the peer-review process is finished, and the data is ready for integration to the map repository. The peer-review process is not part of the repository, but a separate process that relate to the repository.

The peer-review process has been tested [37] in the OneMap project and worked without much trouble. One of the drawbacks with the peer-review process is that it might be very time consuming. The process is also based on human interaction, and it would be troublesome to automate the peer-review process. Nevertheless the peer-review process is by no doubt a very powerful tool, which secures the correctness of the data.

1.1.4 Where is Incremental Update of Geospatial Data Used

The use of incremental update of geospatial data is widely used in various areas. The main users of this approach are grass root projects, volunteer projects, and community projects. Some companies and organizations may also use this approach, but are not the main users of this approach. For example community mapping projects around the world use this approach to store their data, as they collect it manually with their global positioning system (GPS) [41]. Grass root projects that have need of geographical information in their projects are users of this approach. There are also many other volunteer projects with various problems that in some context relate to geographical information. Common for all these groups are that they do not have the resources to collect all geographical data they need at once. This information is built up piece by piece; after a while these pieces form a map of some extent. The possible companies and organizations that use this approach do most likely relate to volunteer work to some degree, or have a open content view on geographical information.

1.2 The Structure of The Thesis

Chapter 2 gives an introduction to methods and topics which are relevant for this thesis. It starts with an introduction of the term lazy integration. Lazy integration is the foundation of the work done in chapter 5. Further on the basic of geometrical integration is introduced. This topic deals with integration of geospatial data into an already existing geospatial file. Syntactical integration is the next topic that is introduced. Under the syntactical integration section, GML [17] is described. GML is a markup language for geographical information. At last the data that is used in the examples and scenarios in this thesis is presented.

Chapter 3 is a brief introduction to other projects that work with similar problem statements. JUMP, OpenJUMP, JCS Conflation Suite and JTS Topology Suite are projects which are described closer in this chapter. Other projects that have worked with geometrical or syntactical integration are also presented.

Chapter 4 is the first of two research chapters and deals with the geometrical integration of data. Two scenarios are described here, and used in illustrations through chapter 4 and 5.

In chapter 5 the syntactical integration and lazy integration are in focus. It outline the problems with syntactical integration and use the lazy integration [38] approach to suggest a solution. Metadata is presented as an important tool in the syntactical integration.

Chapter 6 give a conclusion of the research. It summarizes the key results and discusses the result. It also describes further work that can be done on the area.

Chapter 2

Background

This chapter gives an introduction to the terms and method which this thesis is built upon. The term lazy integration is described; introductions to both geographical and syntactical integration are given. These three topics are the focus of this thesis. At last the data sets used in the cases are described.

2.1 Geometrical Integration

Geometrical integration is the operation of unifying two data sets. A geometrical integration involves different tasks. The two main operations that a geometrical integration consists of is the alignment of line segments from different data sets, and the replacement of a geometrical features. Many of the problems are alike a process called map conflation. This section will give an introduction the common approach a map conflation uses. Some of the methods used in map conflation are easily adapted to the geometrical integration process.

2.1.1 Map conflation

Yuan and Tao [42] define two types of map conflation, horizontal and vertical conflation. Vivid Solutions [22] has defined a third one in their JCS conflation suite [14], internal conflation. The differences between these classifications of conflation are:

Horizontal conflation, this conflation type deals with the problem of removing discrepancies between the boundaries of two data sets. Examples of this kind of conflation could be edge matching of rivers, or roads from two separate data sets or the matching of adjacent boundaries from different data sets. These kinds of operations are also known as boundary alignment.

Vertical conflation, work with discrepancies in data sets that are positioned in the same area. An example of this type of conflation is the removing of discrepancies from two data sets with different resolution, containing the same type of data and over the same area. Another example is the matching of roads or rivers from two data sets over the same area.

Internal conflation, this kind of conflation is only in use on single data sets, and is used in the process of removing overlaps, cleaning of coverages and quality assurance.

The Map Conflation Work Flow

Yuan and Tao [42] define five different steps in the process of conflating geospatial data:

Data pre-processing, this task prepare the data sets for conflation. This process is composed of error checking the data sets, if errors occur these will be fixed if possible. It also checks that both data sets have the same projection, datum and similar coordinates.

Map alignment, this task merge the two data sets together. If the objects in the data sets don't coincide there may be done some transformations to achieve this.

Matching and checking features, this is the most crucial part of the conflation. This is where the actual conflation is carried out. Features from both data sets that correspond are found on background of some criteria's for the conflation process. These criteria's may be adjacency or nearest distance.

Post match processing, at this point the automated conflation is completed, but this is not enough, since conflation is difficult to automate without any human interaction. Post match processing include the human interaction of conflation. Any mismatches and other errors done by the automated conflation has to be corrected by human interaction.

Discrepancy Correction or Information Transferring. This is the final step of a conflation operation; at this point different processes are applied to the new data set. These processes correct coordinate errors of matched points, attribute transfer from both data set to the new data set.

Map Conflation Algorithms

Conflation involve geometric, topological and attribute algorithms, but since this thesis only evolve around geometric integration other algorithms are briefly described here.

There are several different methods of conflating geospatial data, but all of them have one thing in common, they need two data sets with similar geographic coordinates to do the conflation process.

There are two types of quite simple mathematical methods which are used in virtually every conflation algorithm. The Euclidean method is used to get the distance between two vertices.

$$D2 = (X2 - X1)^2 + (Y2 - Y1)^2$$

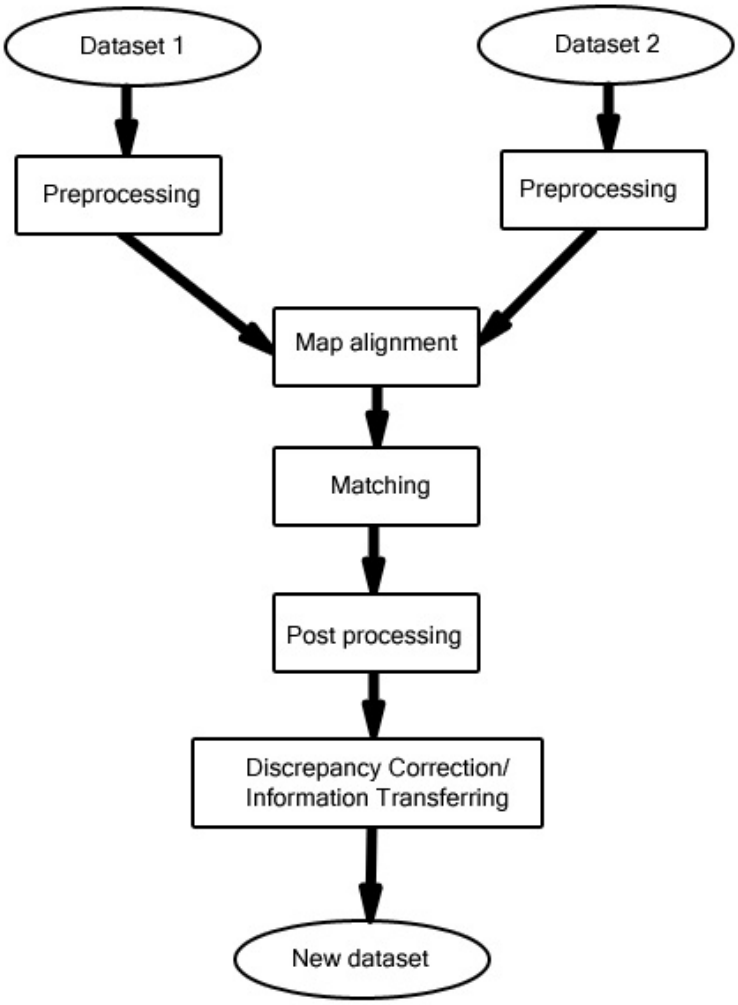

Figure 2.1: Illustration of the workflow in a conflation process.

The other mathematical method is the Hausdorff distance which calculates the distance between linear objects. It determines the largest minimum distance between line A and line B and the largest minimum distance between line B and line A. This is done by moving a "dynamic circle" along one of the lines so that it always touches the other line, the largest radius of the circle is the largest minimum distance from line A to line B or line B to line A. The mathematical formula of this distance is found by:

$$Dh = max(d1, d2)$$

These examples of mathematical methods are example of some of the tasks a conflation process involves. There are needed several other mathematical methods to complete a conflation process, but this is out of scope in this thesis.

Coverage Alignment is a vertical conflation and matches several vertices in data set A with data set B, an example of this alignment is road alignment, see figure 2.2. The JCS Conflation Suite [14] has set some rules for handling this, both of the data sets may have to be adjusted; this adjustment may involve both moving and inserting vertices of data set B. The next step will be to insert vertices in data set A to ensure that it is noded correctly with data set B. When noding there are two options, if two contiguous segments in data set A match a single segment in data set B the way that a vertex exist in data set A but not in data set B. A choice can be made to remove the vertex in data set A by merging the segment or to insert a vertex in data set B by splitting the segment. This choice may depend on the allowance of editing on the relevant data sets.

Boundary Alignment works with the problem of aligning common edges from different data sets. This is a set of algorithms that has some requirements to the data sets.

- The coverages can not be overlapping.
- Each coverage has to be clean. A clean coverage is without any errors. A error in this context is for instance a dangling edge.
- The discrepancies between the data sets have to be small.

The original data sets will not be changed in any way by a boundary alignment algorithm. The output data will consist of a large coverage containing the common edges aligned with no gaps or overlaps, it will be correctly noded. The boundary alignment algorithm needs one parameter, the tolerance distance that each point should have. Figure 2.3 illustrates two data sets that should be merged together. The edge vertices in the data sets do not coincide, this result in a non-coherent line structure. After a boundary alignment process is performed on the data sets these edge vertices coincide, and the line segment are aligned. Figure 2.4 illustrates the result of a correctly performed boundary alignment.

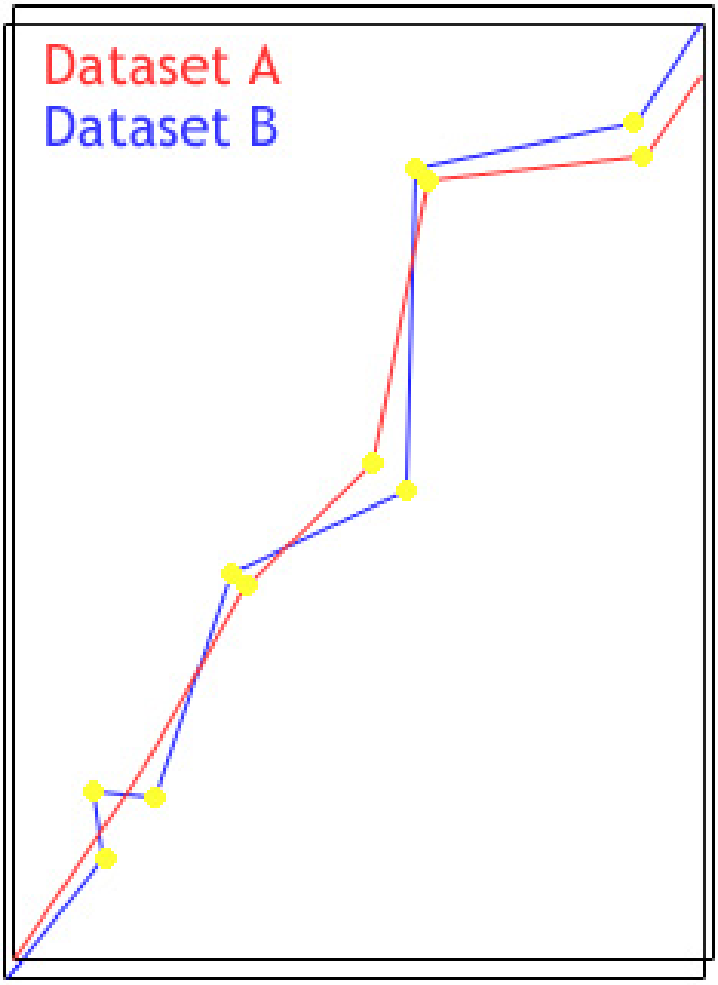

Figure 2.2: Example of coverage alignment

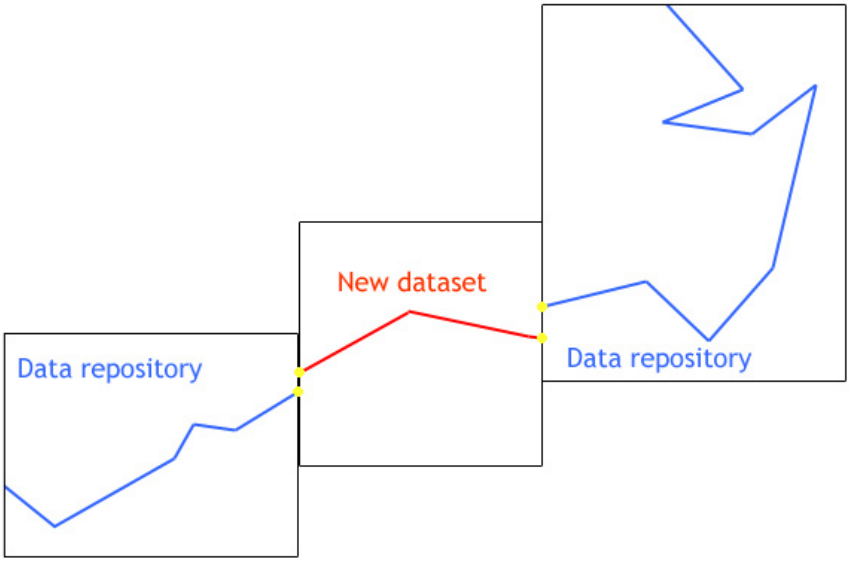

Figure 2.3: Data set before a boundary alignment process is applied on them.

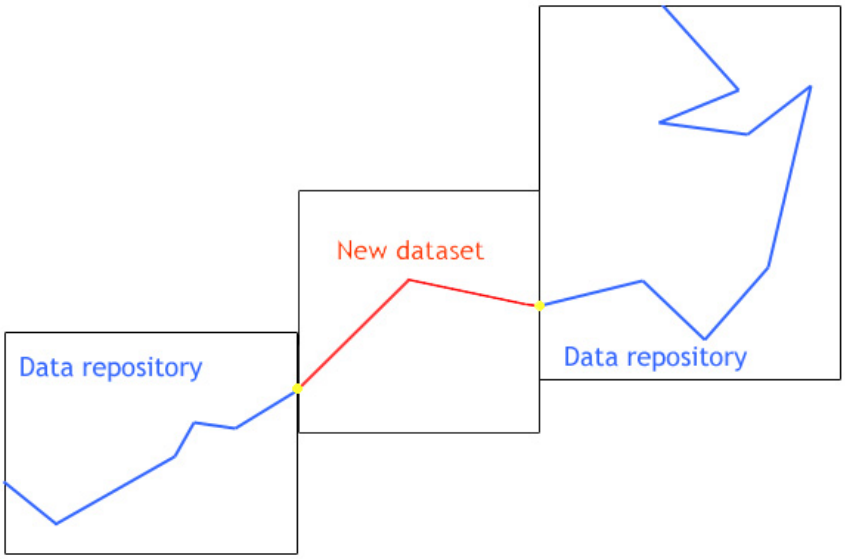

Figure 2.4: Data sets after the boundary alignment were applied.

2.2 Syntactical Integration

In this setting the term syntactical integration denote the adding of geospatial syntactical data to another geospatial syntactical file. In many ways geometrical and syntactical integration are seen as one operation, since the geometrical integration combine two different data set. It can therefore be difficult to part the terms geometrical integration and syntactical integration. The main difference is that the geometrical integration does the geometrical correction and aligning of the data. The syntactical integration does not interfere with the geometrical information at all. Syntactical integration configures the structural anatomy of the GML [17] file. For instance this can be where the geometrical integrated information should be added in the geospatial file. The syntactical integration process also work with the metadata of the geospatial information, for instance when the geometrical and syntactical integration process were performed, by whom it were performed, what is the original source of the geometrical integrated data and so on. This information is as essential as the geometrical information. Even though this information isn't seen as the vital information, is it the framework of the geometrical information. Without a proper syntactical integration process that store vital metadata, will the result be a chaotic structure, with no further possible integration of new data. The study in this thesis will be file based. Which mean that syntactical data will be integrated in another file with different syntactical structure. In this thesis the syntactical integration will be based on the GML [17] file format.

2.2.1 Geography Markup Language

Geography Markup Language (GML) [17] is a dialect of XML [15]. GML instance documents are built in the same way as XML instance documents. GML is written with XML schemas [31] and is tailored to model storage of geographical information. It provides a set of objects used to describe geometrical features, coordinates, coordinate systems, geometry and measurements. GML was initially developed by Clemens Portele but it is now followed up by OpenGIS Consortium (OGC) [3]. GML use the OGC simple feature model [39] to represent geographic primitives. This includes all common feature types as for instance polygon, linestring, point, rectangle and line. The newest version of GML, GML 3 [18] support storage of topological information; earlier versions do not have this feature.

2.3 Lazy Integration

Lazy integration [38] is a strategy used in the OneMap [36] project. Lazy integration is an approach that merges geospatial features into a existing geometrical data set. Common for the features are that they have a different structural syntax than the data set. Earlier such integrations have included change of the structural syntax of the geospatial features. The lazy integration however, does not use this technique. The thought of lazy integration is that new data that are syntactically integrated into a file should be modified as little as possible. It can also be called a non-intrusive integration because it does not affect the geometrically integrated data. To keep the integrated data as close to the original data as

possible will avoid loss of precision, both geometrically and semantically. Using this method also avoid large scale manual work. So far in the OneMap project this integration method has been used to integrate semantic information.

The lazy integration approach defines its own request schema, which encapsulate all other classes. The only purpose of the request schema is to unite the other lazy integration schemas. GML [17] instance documents which use the lazy integration approach are based on the request schema. A schema called utils defines a set of abstract elements. These abstract elements are the foundation of all other elements used in the lazy integration system. The feature collection schema is based on the utils schema and defines the overall structure of a GML instance document based on lazy integration. The elements which are defined in the feature collection schema are subelements of the abstract elements defined in the utils schema. A root element is defined in this schema, it also define a encapsulating element, that surround external integrated features. Integrated features are based on schemas that are extracted from the utils schema. In this context such a schema is called a sub schema. Lazy integration support a lot of sub schemas, and each sub schema defines a natural map layer. The map layers are logical belonging features, for instance coastlines or roads. A sub schema includes external schemas to support other syntactical structures. To integrate a new syntactical structure into the lazy system, the structure has to be added to the correct sub schema/map layer. A new sub schema has to be derived from the utils schema and included by the request schema. Each sub schema can be expanded with new external schemas to fit future needs. With these possibilities the lazy integration approach is a extensible system. Figure 2.5 is a simple illustration of how the lazy integration is used. The GML instance document is based on the lazy integration schemas, the lazy integration use several external schemas to support different syntactical structures. In figure 2.6 the internal structure of the lazy integration approach is described. The Request, FeatureCollection and utils schemas are the core schemas.

2.3.1 Geometrical Integration

In the process of integrating several data sets, the geometrical integration deals with the merging and integration of geometrical features and segments. A geometrical feature is a visual illustration of a real world environmental object. A geometrical feature can consist of points and line segments. The process of performing a geometrical integration can involve modifications of features. For instance some line segments in a feature can be replaced by new line segments, provided by the new data set.

2.3.2 Semantic Integration

Semantical information is information about information, also called metadata. In this setting semantic information is metadata about geometrical information. Metadata describe relevant information about a geometrical feature. Such information can for instance be that a line segment is a road, for example a highway. Another example can be a line segment that defines part of a coastline, without semantical information is it impossible to know what

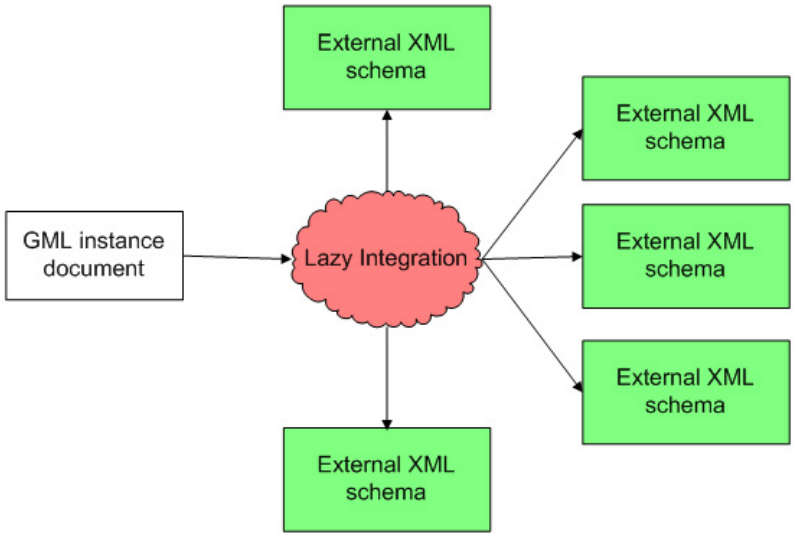

Figure 2.5: The GML instance document use the lazy integration schemas. The lazy integration refer to external schemas.

kind of real world phenomenon a geometrical feature represents. Identification of a real world phenomenon is called classification, and is one of the major problems with semantical information. For example a bridge is classified as an obstacle in nautical charts. Still in a road map a bridge will be classified as a bridge or a road. These problems result in an area with lots of ambiguity. The problem of semantical integration [30] is thoroughly described in a master thesis by Bjørn Håkon Horpestad.

2.3.3 Syntactical Integration

Syntactical integration is the process of merging the structural build up of different data sets. The syntax that defines a geometrical feature is usually built up in a certain way. Trouble arises when data sets built on different rules shall be merged together. Since they have different syntactical structure, is it troublesome to merge the files without modifying the syntactical structure of the integrated data. There are several ways to solve this; one of the approaches is used in this thesis, and is called lazy integration.

2.4 Data Sets

New York City is one of the biggest and most renowned cities in the world. The interest of this city come from the fact that it contains many features typical for a city, like roads, airports, industry and power grids. The geographical placement of the city adds interesting features like islands, rivers and coastlines. The third and most important reason is that

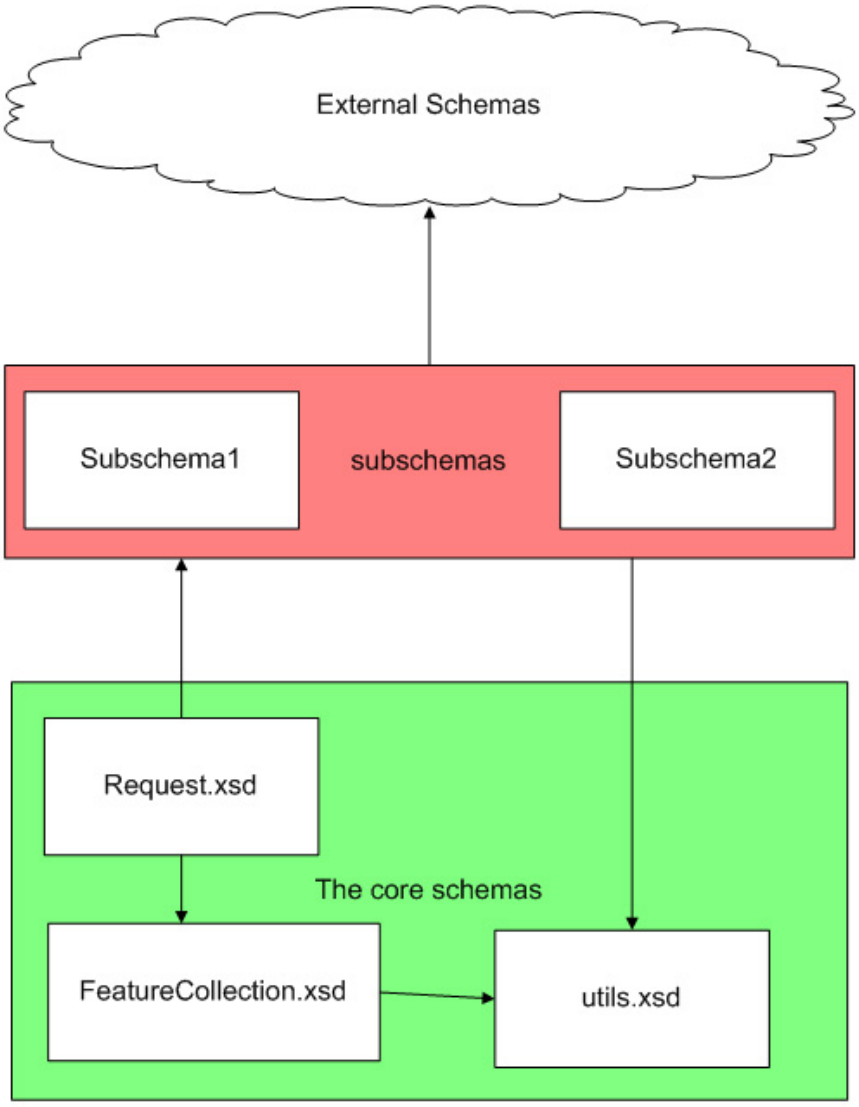

Figure 2.6: The lazy integration structure consists of the core schemas, Request, FeatureCollection and utils. The River and Coastline schema is XML schemas that are subclasses of the utils schema. The sub schemas import external schemas that is needed to represent different GML structures.

there are much data available about the city. VMAP0 (Vector Smart Map Level 0) [11] covers the whole world and provides data everywhere, whereas VMAP1 (Vector Smart Map Level 1) [8] and DNC (Digital Nautical Chart) [12] have available data for certain areas. New York City is one of the places covered by all three data sets. All data sets are based on the Vector Product Format (VPF) [20].

2.4.1 VMAP0 Data

VMAP0 is an updated and improved version of the Digital Chart of the World [32]. It provides worldwide coverage of vector-based geospatial data. VMAP0 includes major road and rail networks, hydrologic drainage systems, utility networks, major airports, elevation contours, coastlines, international boundaries and populated places. The data in VMAP0 is derived from either Operational Navigation Chart (ONC) [10] or Jet Navigation Chart (JTC) [13]. The ONC have a horizontal accuracy of 2040 meters, and the JTC have a horizontal accuracy of 4270 meters. The vertical accuracy on contours are +- 152.4 meters and +- 30 meters on spot elevations.

2.4.2 VMAP1 Data

VMAP1 is divided into 234 geographic zones. At the present time only 55 selected areas are available. The rest of these zones are classified as confidential by the U.S department of defence. VMAP1 is structural similar to VMAP0, and contains all the standard topographic vector data types familiar to GIS users. The VMAP1 data content includes 10 thematic layers, boundaries, coastlines, road, rail and hydrography to mention a few. VMAP1 accuracy can be divided into horizontal and vertical accuracy. VMAP1 product resolution is based on 1:250000 map scale and the data are also divided into four different classes.

2.4.3 DNC data

The Digital Nautical Chart (DNC) [12] is a vector based product designed to provide an up-to-date seamless database of the world. It is produced in the standard VPF [20]. The features are thematically organized into 12 layers or coverages including: Cultural landmarks, Earth Cover, Environmental, Hydrography, Inland Waterways, Land Cover, Limits, Aids to Navigation, Obstructions, Port Facilities, Relief and Data Quality. The main focus of DNC is on coastline, harbour and near coastline/harbour related information. DNC data has consists of 4 types of data sets, each set having different accuracy. These four data sets are Harbour, Approach, Coastal and General. The Harbour data set is most accurate, and the General data set is least accurate. It is worthwhile to note that the DNC Coastal data set is less accurate than the VMAP1 data set. In this paper we will use DNC Harbour and Approach data.

Chapter 3

Related work

This chapter gives a brief survey of some of the similar projects and software packages available.

3.1 JCS Conflation Suite

JCS Conflation Suite [14] from Vivid Solutions is an open source package developed with the Java [27] programming language. The main purpose of the JCS Conflation Suite is to offer a set of geo-spatial conflation operations. These operations include pre-processing of data sets and conflation operations.

The pre-processes of data sets are meant to prepare the data sets for the actual conflation process. If errors occur in the data sets they will affect the result of the conflation process. It is therefore essential to remove all possible errors from the data sets. The JCS Conflation Suite has several methods which prepare a data set for conflation. These methods are called Coverage Cleaning and detect and remove overlaps and gaps automatically.

The two main operations in the JCS Conflation Suite are the boundary alignment and coverage alignment. The boundary alignment method aligns the edge vertices in neighbour tiles with each other. JCS support the boundary alignment method on closed geometries such as polygons and rectangles. The coverage alignment method aligns vertices in data sets which are overlapping each other.

The JCS Conflation Suite includes several other modules, the road matching module; it detects differences in two versions of a road network. This includes attribute transfer, missing sections of a road can be added from one network to another. The precision reduction module reduces the precision of the coordinates in a data set. This can be done either by reducing the number of decimal places or by a given factor. The geometry difference detection module finds differences between two different data sets. The module includes two ways of determining differences, exact matching and matching with tolerance. Exact matching requires that the matches are identical, matching with tolerance give a specified tolerance buffer to each vertices. If vertexes from both data sets are within each others buffer zone, they are seen as exact.

3.2 JTS Topology Suite

The JTS Topology Suite [21] is a Java [27] API implementation of the OpenGIS Simple Features Specification(SFS) [39], and 2D operations. This means that the JTS Topology Suite is an implementation of a spatial model, and is capable of representing geometrical features. These geometrical features include the regular features like Points, MultiPoints, LineStrings, LinearRings, MultiLineStrings, Polygons, MultiPolygons and GeometryCollections. The purpose of JTS Topology Suite is to provide an API used to develop applications which support operations like integration, cleaning, validation and querying of data sets. The 2D spatial operations implementation use binary predicates to compare geometries for overlaps and gaps. The analysis methods include operations like intersection, union and difference. JTS Topology Suite use the Well-Known Text(WKT) file format which is defined in the OpenGIS Simple Features Specification.

This is a powerful package which in cooperation with other applications and/or APIs can be the source of very solid and reliable software. As an example of this the JUMP [22] package, JUMP is built up with the JTS Topology Suite as a foundation package. JUMP is described further in section 3.3.

3.3 JUMP Unified Mapping Platform

JUMP is a workbench to view and manipulate geo-spatial data. JUMP is a collaboration project between Vivid Solutions [9], Refraction Research [6] and British Colombia Ministry of Sustainable Resource Management [1] in Canada. JUMP support both standards such as ESRI's shape format [26] and GML [18] from the Open Geospatial Consortium [3]. The JUMP workbench can create, edit and store geometric data on layer level and in project files. JUMP allows multiple layers, transparency, labelling, geometry collections and colouring of geometry and several other viewing options. JUMP is free, open source and developed with Java [27]. JUMP is composed of three main modules:

- The JUMP Unified Mapping Platform [22] is the framework of the system. This includes the user interface, the JUMP API and the basic structure of the system as Input/output operations and some of the spatial operations.
- The JTS Topology Suite [21] is an OGC [3] complaint 2D manipulating API of spatial data. The JTS [21] is developed by Vivid Solutions [9].
- The JCS Conflation Suite [14] is a conflation API, this conflation suite handle both horizontal and vertical conflation as well as internal conflation. JCS is also developed by Vivid Solutions. See section 3.1 for a more detailed description of the JCS Conflation Suite.

Figure 3.3 show a screenshot of the JUMP workbench. JUMP is plug-in based; this means that third party developers can contribute with new modules/plug-ins which manages different kind of geometric operations.

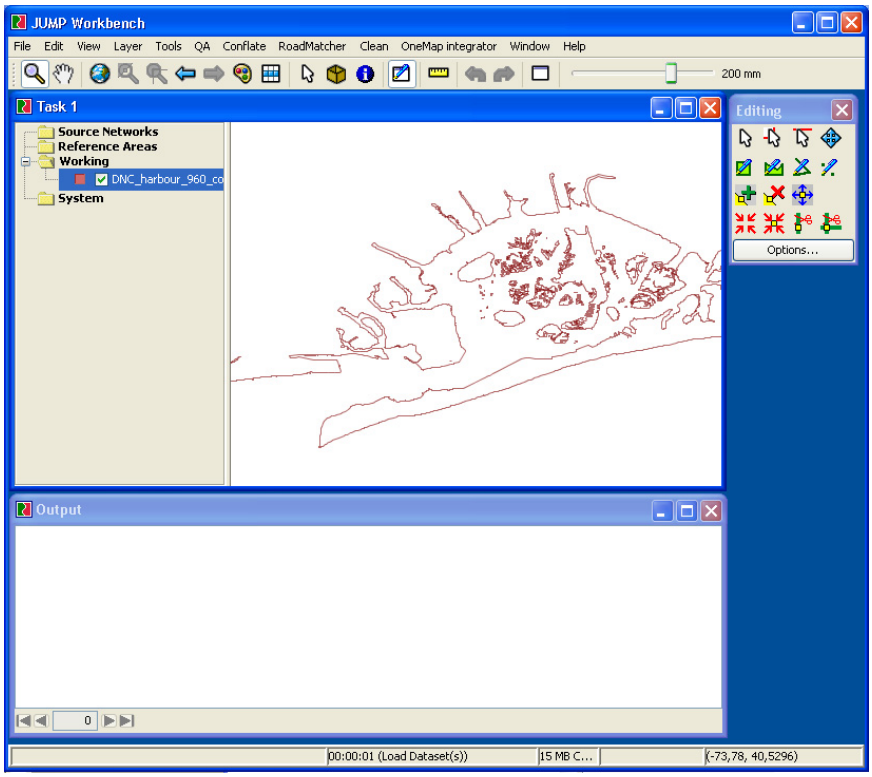

Figure 3.1: JUMP screen shot.

3.4 Open JUMP

The original JUMP [22] project was started by Vivid Solutions [9] and funded by several other sponsors. Shortly after the first version, the JUMP platform grew to be a popular GIS application. However, after a while the funding of the project stopped and Vivid Solutions [9] lowered the activity on the project. At this point the platform was widely used by world wide GIS communities, and seen as one of the best free GIS software platforms. One of the main reasons the JUMP [22] platform grew so popular was the easiness of creating plug-ins, and lots of plug-ins had been created by users of the platform. Two main co-projects were already working with plug-ins and further development of JUMP [22], the JUMP Pilot Project [2] and Project SIGLE [5]. These co-projects coordinated and encouraged the effort of volunteer development of JUMP [22]. These two co-projects formed the OpenJUMP [4] committee, which oversees the direction of a new JUMP core. The core is built up by the already existing JUMP core with improvements developed by the volunteer development community. The OpenJUMP [4] aims to provide a common platform for development efforts, and will help the developer community to avoid compatibility problems and avoid duplicate development of plug-ins. At the time of writing OpenJUMP [4] version 0.1 has been released, which is the first version of OpenJUMP [4] released so far.

3.5 Automatically Annotating and Integrating Spatial Datasets

This article [16] was written at the University of Southern California. It describes an information integration approach to conflate and annotate online geospatial data. They have developed an application which integrates satellite images from Microsoft Terraservice with street information from U.S. Census TIGER/line files, and building information from the web. This is used to identify buildings on satellite images. This application is able to perform an automatically conflation process which identifies roads in satellite images, with a certain error rate. The integration process uses what they call control points, these points are corresponding points in both data sets. These points are used as reference for the alignment of all other points that have to be aligned. The control points can be found using Microsoft Terraservice Landmark Service software, or by analyzing satellite images using vector data. The control point pairs have latitude and longitude values as geospatial reference. These control point pairs are analyzed and erroneous pairs are removed. A method called vector median filter is used to filter out insignificant points. Both the control point pairing and the vector median filter are performed on both data sets. When this is done the conflation process begins, which starts with the alignment of the control point pairs. The alignment of other features are done by performing a delaunay triangulation and piecewise linear rubber sheeting to determine appropriate transformations.

3.6 Integration of Heterogeneous GML Sources

The paper is written by Gunnar Misund and Harald Vålerhaugen and focus on the integration of heterogeneous GML [17] sources. The integration of heterogeneous GML has two main focuses, applications which work with heterogeneous GML and storage systems which can store heterogeneous GML files. The main focuses in the paper are cascading GML analysis, lazy integration and a generic GML browser.

The goal of a cascading GML analysis is to partially automate the creation of templates which are used to load GML files into an application. It also solves the problem with missing schemas or incongruity in a schema. There are developed four methods for retrieving information on a GML instance document.

The first method is called schema analysis. The objective of a schema analysis is to find out how the elements in the schema relate to each other. It also checks to see if the elements are indirectly derived from a GML native type. An application that uses this method can treat elements as their base type. The information collected from the schema analysis is used to create a mapping file; the mapping file is a vocabulary of the schema.

A schema analysis may in some cases fail to create a complete mapping file, reasons for that might be unreachable schemas or inconsistency in the schema. In such cases a structural analysis can be used; a structural analysis analyzes the content of the GML instance document based on the elements. To do this some structural rules provided by GML 2.x should be used. The first rule says that the root element must be directly or transitively descended from gml:AbstractFeatureCollectionType. The second rule is that relations between classes should be represented through associations and properties.

If a schema is unavailable, incomplete or instance documents are not correctly in accordance to the schema, the instance documents have to be parsed and analyzed based on their structure. This is a process called manual analysis and might not always succeed. These situations require human assistance.

The cascading process is a framework for combining the forces of several analyzing methods. It uses the methods mentioned above together, to form a powerful analysis tool. The cascading process was built with Java [27] and used SAX [34] to parse instance documents. The elements from the instance document are mapped into an internal tree-model.

Lazy integration has its origins from this paper; it describe a non-intrusive method to integrate data from several sources. This method was used to integrate semantic information on a feature. This semantic information was collected from several sources. Lazy integration is described in more detail in chapter 2.

The last topic the paper discusses is a generic GML browser. The generic GML browser was developed to test the Cascading process and the lazy integration approach. It transforms GML into Scalable Vector Graphics (SVG) [25]; SVG is used to visualize the geometric constructs and provides easy access to the non-geometric properties of the features.

3.7 Automated Conflation of Florida State Highway Data with Larger Scale County Data

The paper [19] describe the use of ESEA's Conflation System (ECS). ECS is an automated conflation system, it reduces the effort of conflating vector maps. To describe the conflation process they used data from Florida department of transportation; Florida state highway data and larger scale county data to be more precise. The file type of these data were on ESRI's shapefile [26] format. The ECS perform a conflation on two coverages at a time. It identifies one of the coverages as base coverage, which is of highest accuracy. A non-base coverage is the other coverage which is of lesser accuracy than the base coverage. There is not applied any changes or modifications on the base coverage. The non-base coverage is transformed via rubber-sheeting to match the base geometry during the conflation process. An ECS conflation process has three steps:

Node matching is used to create rubber-sheeting transformations and to match node features. To perform a node match, distance, topology and attribute information are used. A node match is only performed in a user specified distance of the coverages. When the node matching is completed, the node pairs are used to bring the non-base coverage into better alignment with the base coverage. This is done using a rubber-sheeting transformation. When the automated node matching is finished, the user can add more node pairs or remove node pairs, if it would be necessary.

Line matching is performed after the user verifies the node matching. The user defines a distance from the line, and defines a region where the matches can be found. A path from the other coverage is used to consider matches if it lies inside the user sat distance. A line match process can also use attribute information to help the matching process.

The feature merging select the desired features and attributes from the non-base coverage and includes them in the base coverage. The features which are merged in can be merged together with already existing features, or inserted if there are no overlaps.

The ECS is not a hundred percent automated, and needs human assistance to perform some of the decision making, still it reduces the amount of human interaction.

Chapter 4

Geometrical Integration

Geometrical integration is the process of integrating geometrical data together. The integration process can be feature based; it is a process where geometrical features from a data set are integrated into another data set. Still the integration process do not have to be feature based, integration of feature members can also be performed. Feature member integration is the integration of elements which is part of a feature into another data set, and another feature. A geometrical integration process does usually involve only two data sets at the time. The main problem with this process is that integrated features and feature member elements do not align without any modifications. This usually creates a non-coherent line structure. Throughout this section the geometrical integration process is explored. During the exploration the main problems with geometrical integration are defined, there are also sketched some theoretical solutions to these problems.

4.1 Geometrical Integration Scenarios

The scenarios described in this section will be used to emphasize problems and solutions later in this chapter, and the next chapter.

4.1.1 Area of Interest

The main focus in this paper will be on the New York area, but it is desirable to geographically define an area of interest. The tiles which are used in VMAP0, VMAP1 and DNC are to large to be used directly as example data. It was therefore required that a self produced tile was made; this tile has the coordinates, upper left X: -74.3, upper left Y: 41, lower left X: -73.7 and lower left Y: 40.4. The tile is seen in figure 4.2, along with the harbour and approach tiles. The original tiles are seen in figure 4.1.

4.1.2 East River, La Guardia Airport

The administration of La Guardia Airport has an ongoing project to secure the airport from possible terror attacks. The project uses a GIS tool to illustrate and detect possible danger

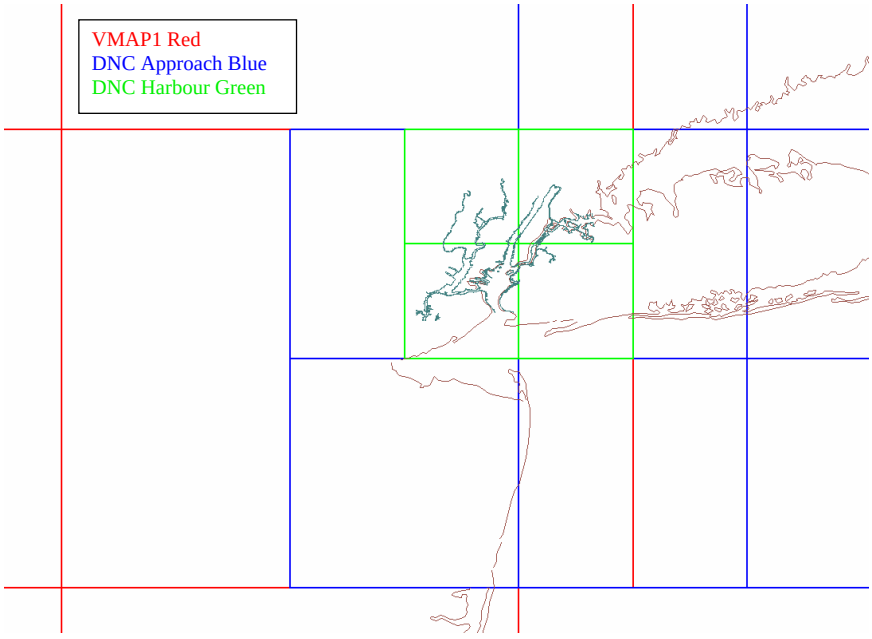

Figure 4.1: The different tile sizes in VMAP1, DNC Approach and DNC Harbour.

areas. Their original data sets are built up by VMAP0 [11] data and TIGER/line [28] data. But the VMAP0 does not give enough detailed information on the coastlines of the area. The coastline of La Guardia Airport is seen as a possible danger area to terror attacks, these coastlines can be reached easily without anyone noticing. It is therefore crucial that the coastline areas that are not secure are revealed. The administration is therefore updating their map repository with VMAP1 [8] data of the area. The process of replacing geospatial data run into two main problems, the line segments from VMAP0 and VMAP1 do not form a coherent line structure. The second problem occurs after the geometrical integration, when the syntactical data from VMAP1 should be integrated in the VMAP0 data set. During the process information from VMAP1 is lost.

4.1.3 Jamaica Bay

The environmental project "Save the World" is worrying about the coastlines of Jamaica Bay. These coastlines have been known for its rich wildlife for decades. During the past years the wildlife started to diminish. One of the main reasons for the wildlife diminishing is the increase of the human population in the area. The area is a natural paradise and therefore attracts a lot of people. The increase in human interference in the area, affect the wildlife. This worries "Save the World" and they have started to build up a map repository to store geospatial information on the wildlife. Their original map repository consist of VMAP0 [11] over the area. VMAP0 does not have enough accuracy so they will

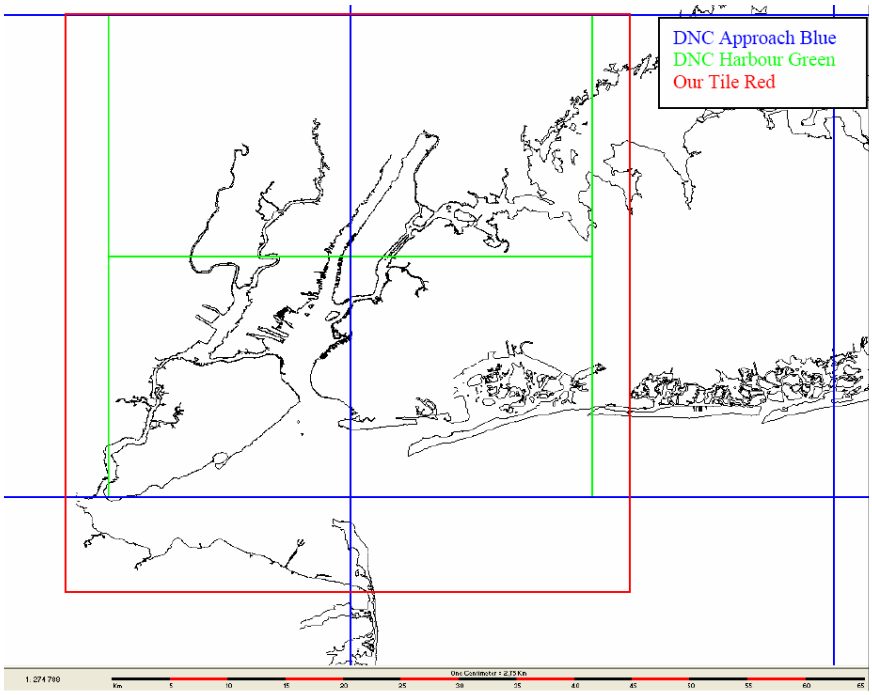

Figure 4.2: The DNC harbour, DNC approach tile, and the tile over the area of interest used in this thesis.

Figure 4.3: A overview of Jamaica Bay in VMAP1.

replace VMAP0 data with DNC [12]. When "Save the World" starts the replacement of data several problems arise. The two data sets do not form a coherent line structure after the replacement of the data. There is also uncertainty in how these line segments should be connected. Another problem is that they do not know if they did the syntactical integration correct, it seems like the syntactical integration of the data sets caused loss of information.

4.2 Cleaning Data Sets

To ensure a correct geometrical integration process it is essential that the data sets which are used in the process are represented correctly. This means that the data sets have to be without any type of geometrical errors. If a geometrical integration process is performed on a set of uncleaned data sets the result will most likely be faulty. There are three main causes of geometrical errors in data sets, dangling edges, gaps and overlaps. Each of these errors will be discussed in detail in this section.

4.2.1 Dangling Edges

Dangling edges are a common error type found in geometrical data sets; a dangling edge is an erroneous gap in an otherwise coherent linestring. The challenge is to avoid these dangling edges or reduce them to a minimum. A dangling edge can sometimes be mistaken with the natural gap in the line segment of a linestring. Throughout this section on dangling edges, obvious examples of dangling edges will be used. The dangling edge problem is not present at the same degree in the VMAP1 and VMAP0 data as it is in DNC data.

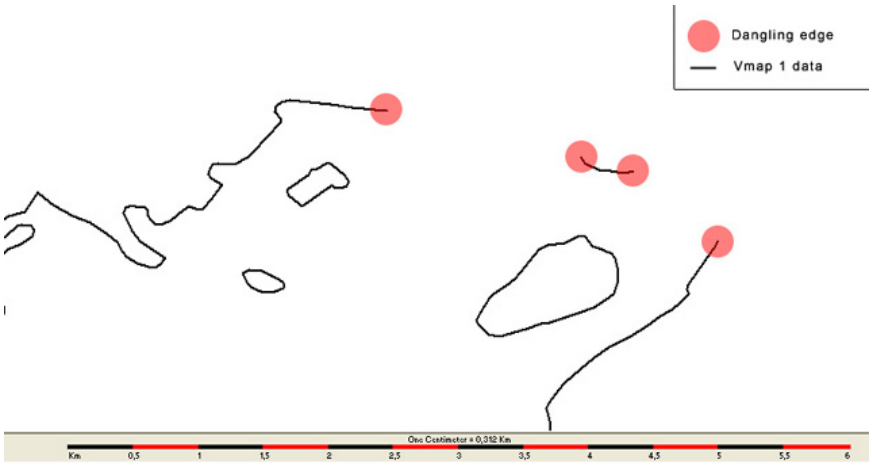

Figure 4.4: A coastline with several dangling edges, the edge vertices are seen as red circles.

Solutions to Dangling Edges

There are several solutions to the dangling edge problem area, and the solutions we sketch here are only two of them. Figure 4.4 show a coastline from the Harbour collection in DNC data. This is the most precise coastline information DNC offer. To use this data further in an integration process it is essential to remove these dangling edges before the integration process.

Manual Removal of Dangling Edges This example of removing dangling edges is theoretical, and will need human assistance if it is implemented. The first step in the process of removing the dangling edges is to identify all edge vertices. In this setting, a edge vertex can be both a start and a end vertex in a linestring. Figure 4.4 illustrate this identification process. Every edge vertex is highlighted with a red circle. Step two is to determine which vertices which are edge vertices. There are two reasons why a vertex can be an edge vertex. Either it is supposed to be an edge vertex and is correctly an edge vertex. The second reason is that the data set contains errors which have turned a connected node into an edge vertex. The nodes in a linestring are highlighted with a yellow circle in figure 4.5. These yellow nodes are marked because they might be used to connect a dangling edge to the linestring. The red circles in figure 4.5 are edge vertices in the data set. Step three is to connect the red vertices either with another red vertex or with one of the yellow vertices which represent a node in a line string. Since this is a manual correction of the edge vertices it is quite obvious that point p1 and point p2 in figure 4.5 should be connected together. Point p3 and p4 should also be connected together. The result of this process are displayed in figure 4.8, it show a connected network, with no dangling edges.

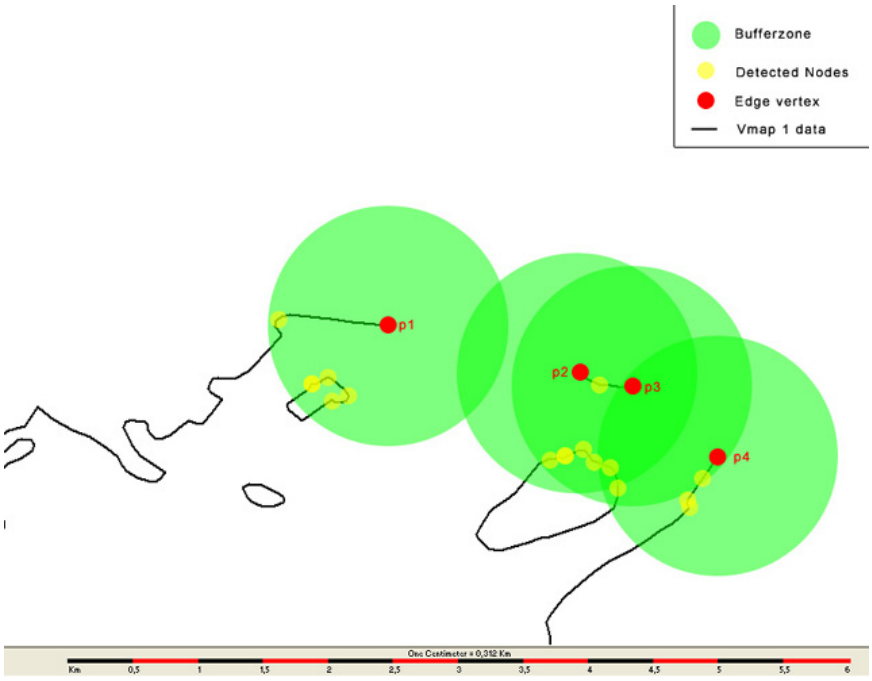

Figure 4.5: Red circle are edge vertices, green circles identify the buffer zone of the edge vertices. The yellow vertices are nodes in line segments that is inside a buffer zone.

Removing Dangling Edges Using Conflation Internal conflation is a method developed to use conflation operations on a single dataset. Internal conflation was first introduced by Vivid Solutions in the technical JCS Conflation Suite report [23]. Since internal conflation use usual conflation operations, the process in an internal conflation is the same as any other conflation process. It must be emphasized that even with conflation operations this process most likely have to be human assisted to some degree. The process might be performed in iterations. Step one is to define a tolerance limit which define the buffer zone around all edge vertices. In step two all edge vertices that have other vertices inside their buffer zone are identified. An optional step three might be to check if the vertices that should to be connected together have the same attribute values. This might prevent that data of different categories are wrongly connected together. The next step is to connect the edge vertex to the vertices in its buffer zone to each other, using a linestring, or by snapping the vertices together to one vertex. Pseudo code of this would look a bit like this:

```
limit = toleranceLimit;
while limit less than maxTolerancelimit {
    detect edge vertices;
    add buffer zone to edge vertices;
    Check for identical attribute information;
    snap or add a linestring between matching vertices;
    increase the limit value
}
```

Figure 4.5 show the first iteration of this process. It shows a green buffer zone around each edge vertex. The red circle is the edge vertex. The yellow circles mark all other vertices inside the buffer zone of each edge vertex. In figure 4.5 the p2 point is in the buffer zone of point p3 and vice versa, this is also the fact for p3 and p4. Using conflation methods p1 and p2 are excluded from this iteration, p1 because it has no edge vertex in its buffer zone. Point p2 is excluded on the basis of position; it faces away from point p3. Since p3 are in the buffer zone of p4 and p4 is in the buffer zone of p3 they are a match. Point p3 and p4 are connected through a new line segment in the linestring. Figure 4.6 show the result of the first iteration, it create a line segment between point p3 and p4. Iteration two is illustrated in figure 4.7. The green circles are the edge vertices, the red circles are the edge vertices buffer zone and the yellow circles are all other vertices in the buffer zones. As seen both point p1 and p2 are in each others buffer zone. Figure 4.8 show the result of this process, line segments has been inserted and all dangling edges has been removed.

4.2.2 Gaps and Overlaps

Gaps and overlaps are problems that can occur in data set with coverages, polygons and rectangles. If these problems had occurred when several data sets were merged together it would have required a coverage alignment procedure to solve it. In this section gaps and overlaps on single data set are discussed. Gaps are areas which are not covered by any features. Occurrences of gaps are only seen in data set that contain coverages of polygons. These gaps may be natural and correct, or they may be the result of a faulty data set.

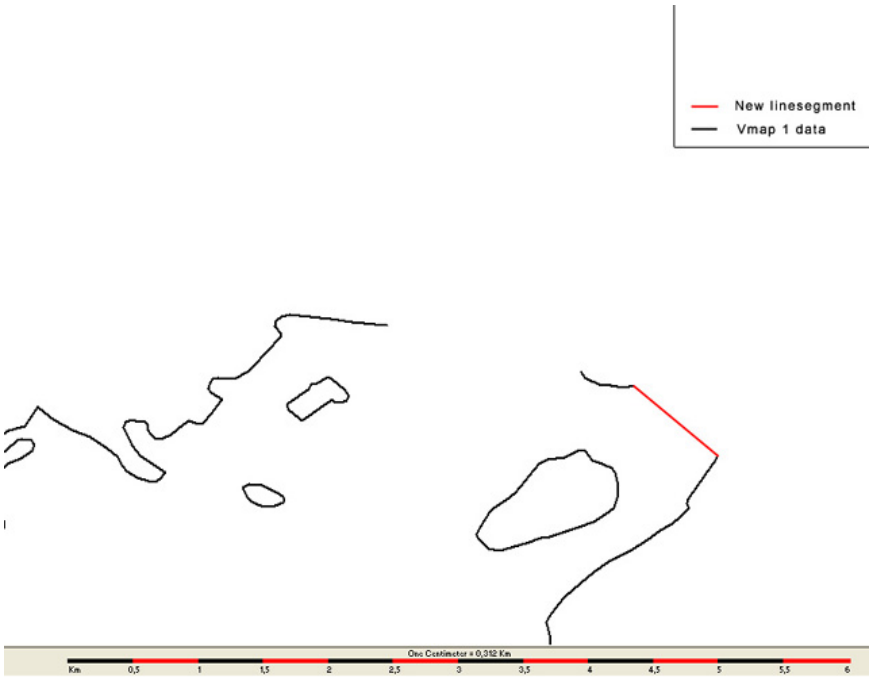

Figure 4.6: The first gap has been removed, seen as the red line segment.

Overlaps can arise when several features cover some of the same areas. In some settings overlaps can be a correct representation, but in most settings overlaps are undesirable and should be removed. It is important to notice that both gaps and overlaps not always are errors in the data set. It should be a hundred percent certainty that the data set is erroneous, before any attempts to remove gaps and overlaps are performed.

To do a correction of gaps or overlaps it is crucial that the user has studied the data set in detail. It can also be useful to check external sources of the data to form several impressions of the data set.

Gaps will in most settings be natural, and should therefore not be removed. If a gap is known as faulty and it is necessary to remove it, it should be done with focus on the knowledge of the gap. The gap should be shared evenly by the adjacent coverages unless the coverages are of different importance. If one of the coverages are of more importance than the other and it has most of the adjacency of the gap, that coverage should be changed and cover the gap.

Removing Overlaps should as mentioned also only be performed in certainty of its incorrectness. If the overlapping coverages are of same importance they should remove the overlap by evenly covering the overlap. The overlapping coverage should reduce its size on the overlapping area. The coverage that is overlapped should reduce half of its size in the

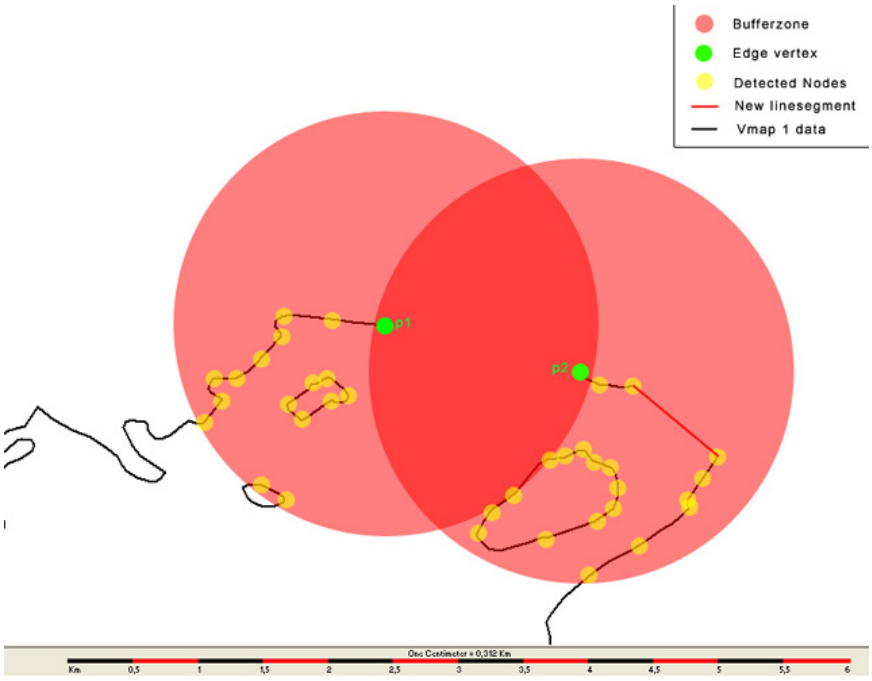

Figure 4.7: The green circles are the edge vertices, red circles are the buffer zone and the yellow circles are nodes in linestrings identified as nodes inside the buffer zones.

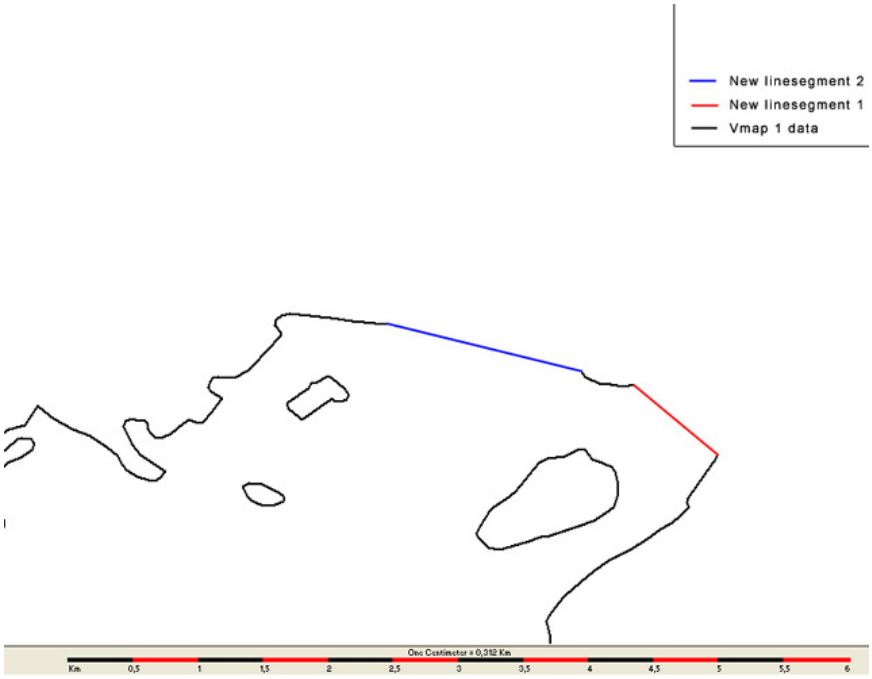

Figure 4.8: The red line is the line inserted in the first iteration, the blue line was inserted in the second iteration.

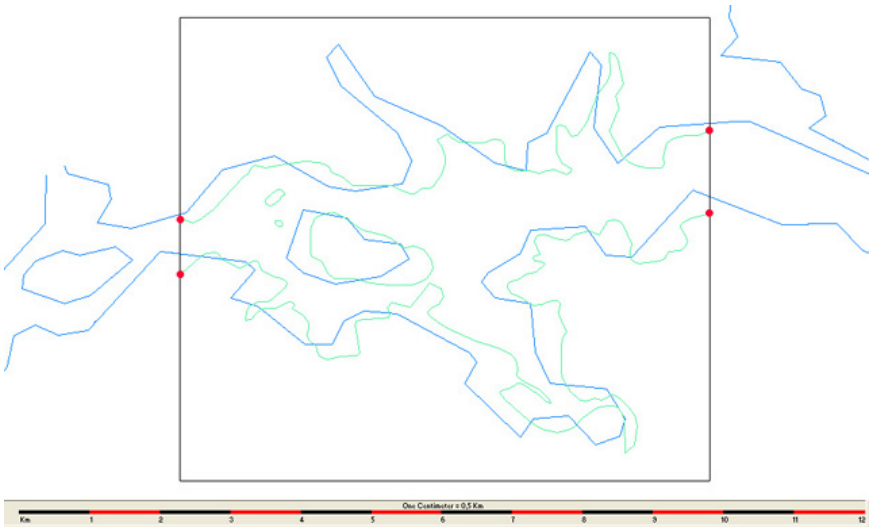

Figure 4.9: The rectangle define the update window, the blue lines are features from VMAP0, the green lines are features from VMAP1. The red circles are edge vertices in the VMAP1 data set.

overlapping area. If one coverage has more importance than the other, this coverage should cover the are that are overlapped, whether that coverage is overlapping or are overlapped.

4.3 Geometrical Integration Problems

This section deals with the problems which can arise during an integration process. After the definition of a geometrical integration problem, one or several methods to solve these problems are presented.

4.4 Boundary Alignment of Geometrical Features

The coastlines in the area of La Guardia airport in New York have large differences between the VMAP0 and VMAP1 data sets. The map repository contain VMAP0 data and it is desirable to update this area with more precise information such as VMAP1 data. Figure 4.9 show the differences between VMAP0 and VMAP1 data. The blue lines are VMAP0 data and the green lines are VMAP1 data. The black rectangle mark the area of update, the red circles are the edge vertices in VMAP1. All VMAP0 information will be replaced with VMAP1 data inside the update rectangle.

In figure 4.10 the geometrical integration process on the left side of the update rectangle are dealt with. Figure 4.11 show the geometrical integration process on the right hand side. The blue circles are buffer zones to each of the edge vertices in VMAP1 data. Initially all the edge vertices had a small buffer zone, but if no node or vertex from VMAP0 data are

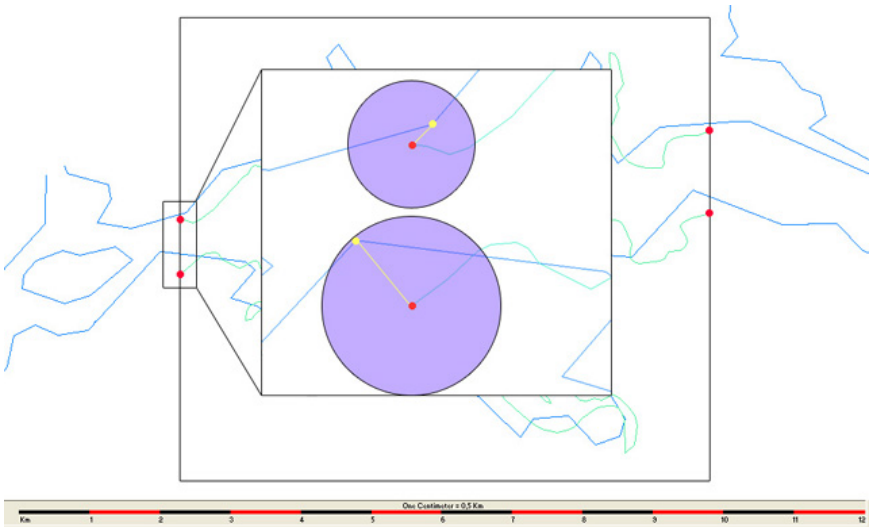

Figure 4.10: The integration process on the left side. It moves the edge vertices from VMAP0 to coincide with the edge vertices in VMAP1.

found inside the buffer zone, the buffer zone size increases. The initial size and the max size of the buffer zone are predefined by the user. This goes also for the increase size of the buffer zone on each iteration. When a node or vertex are found in a buffer zone it create a new vertex in VMAP0 data set, at the exact position of the edge vertex in the VMAP1 data set. This new vertex are connected to the node/vertex that were found in buffer zone. This is seen as yellow lines and circles in figure 4.10 and figure 4.11. This create a graphical continuous coastline in the merging point of VMAP0 and VMAP1.

The result of this geometrical integration process are seen in figure 4.12. The buffer zones and edge vertices are removed an the result is a continuous coastline.

4.4.1 Indecisive Integration

Indecisive integration is a problem state that complicate a regular integration process. The common criterion for these problem statements is that they have several possible ways to carry out the integration process. The integration process has no possibility to know which of the outcome give a correct result. This lead to a maximum fifty percent chance of correctness, and this percentage might be far less than this. Because of this high error percentage it is not sensible to develop a integration method without human interaction. If the problem is easy it is enough that one person monitor and correct the conflation process. If the problem is cumbersome a peer-review process might be suitable. The area of Jamaica Bay on Long Island have several instances which might cause a troublesome integration process. Figure 4.13 show the area of Jamaica Bay with VMAP0(blue) and VMAP1(green) data.

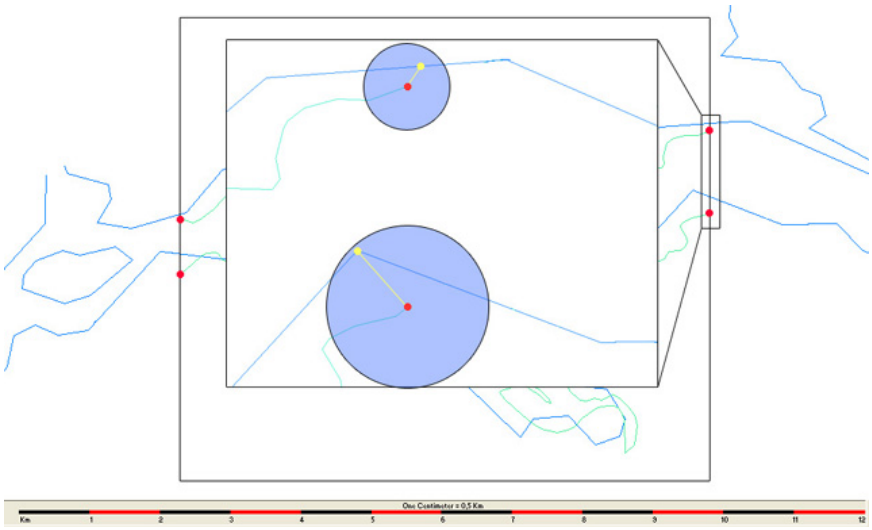

Figure 4.11: The integration process on the right side. It moves the edge vertices from VMAP0 to coincide with the edge vertices in VMAP1.

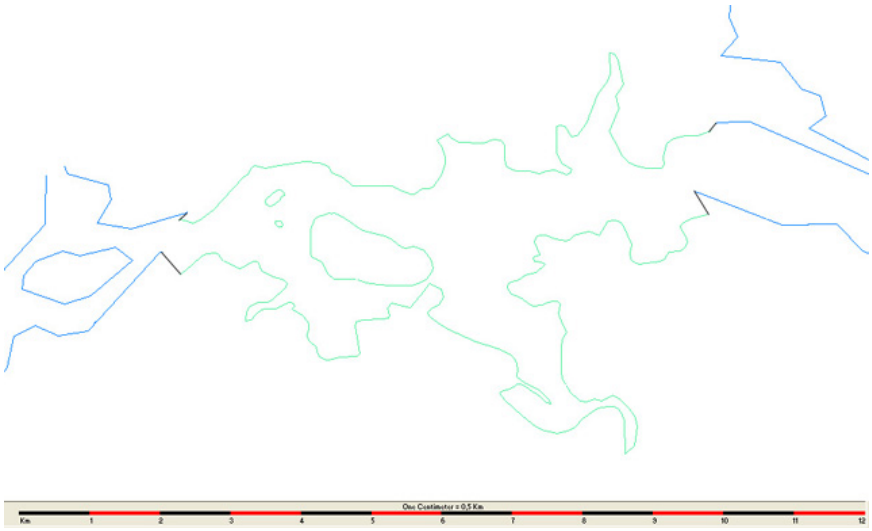

Figure 4.12: The result of the integration process, a continuous coastline.

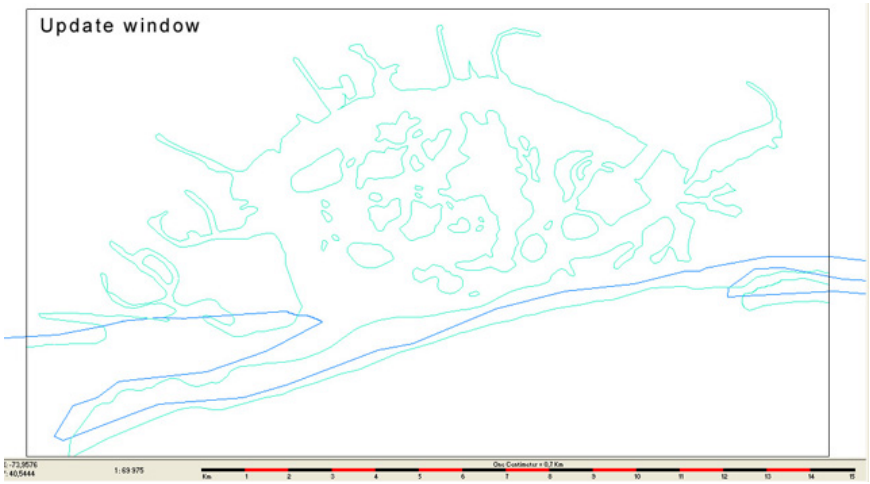

Figure 4.13: Jamaica Bay, VMAP0 is seen as blue lines and VMAP1 is seen as green lines.

Indecisive Merging

One of the problems found in the Jamaica Bay scenario is the indecisive merging problem. This problem arise when a edge vertex find several possible nodes and line segments in its buffer zone. In such a instance it is not possible for a computer based system to determine which of the nodes the edge vertex should connect to. It could in fact just pick the node closest to the edge vertex. The chance of that node being the correct one is more or less fifty percent, depending on how many nodes found in the buffer zone. An example of this problem is found in the Jamaica Bay area, this is seen in figure 4.14 in the area of interest box. Figure 4.14 show the border of the update window and the edge vertices in the new data set. Two of the edge vertices have two different nodes from two different line segments in its buffer zone. It might seem logical to connect to the node that is closest, however in many instances this will prove to be wrong, that include the example in figure 4.15. In our scenario it is quite obvious how these edge vertices should be connected to nodes in the line segments. The upper edge vertex should be connected to the line above and the edge vertex in the middle should also be connected to the line above. Figure 4.16 show the result of this conflation process. Without topological information it will be impossible to automate this process and it will require some human interaction.

The Island Problem

The island problem occurs in the integration process where data sets of different resolution are used. In the data set with high resolution a feature can be defined as an island, in the low resolution data set this island does not exists, but is seen as a peninsula. The problem does not occur in the integration of complete features, but in the integration of line segments from a feature. It can be required to transfer only some line segments from a feature into

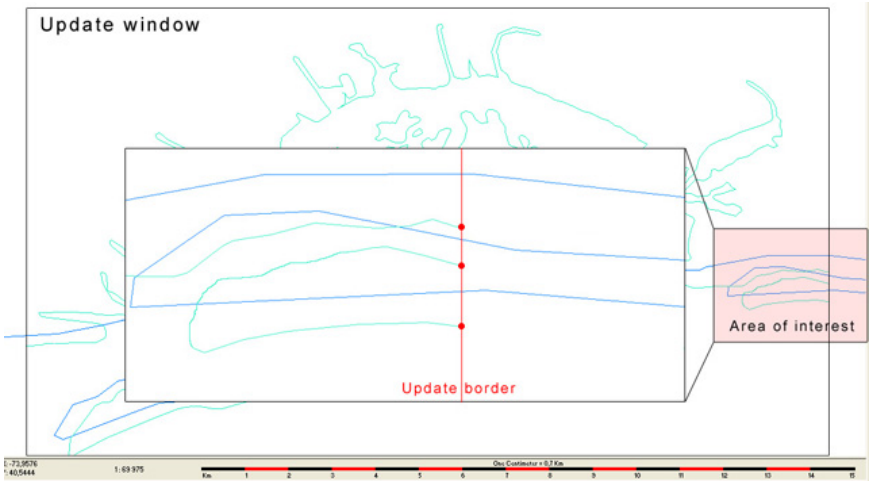

Figure 4.14: The area of interest are zoomed in, the red line is the border of the update window, and the red circles are the edge vertices.

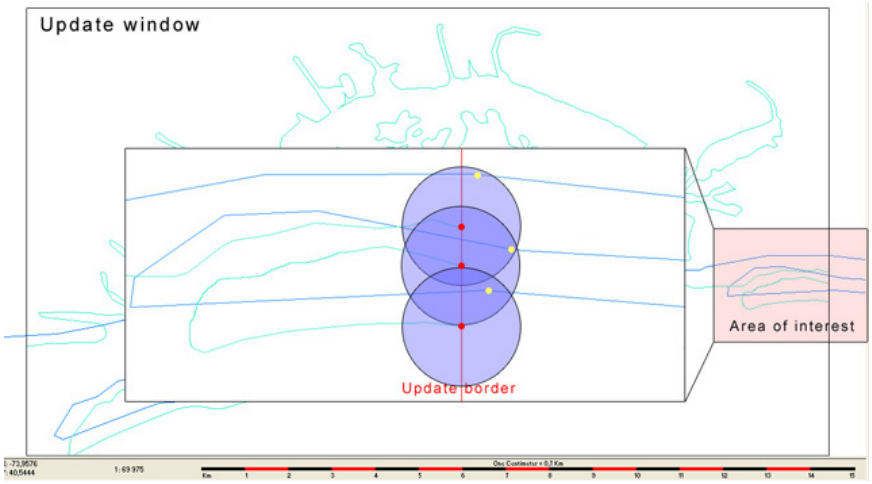

Figure 4.15: Each edge vertex has its own buffer zone(blue circle). Nodes in the line segments from the data repository seen as yellow circles. Notice that two of the edge vertices has two nodes in their bufferzones.

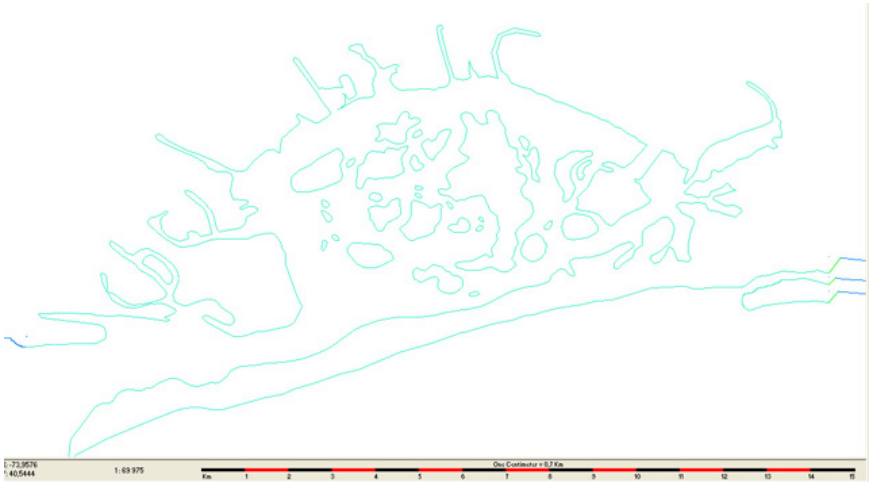

Figure 4.16: The result of a correct integration process.

the target integration data set. The setting creates uncertainty of how these line segments should be handled. It is impossible to know if the new data set contains an island or a peninsula which should be connected to the already existing data set. There are several ways to solve this problem but it requires human interaction, and a possible peer-review process. This peer-review process is needed because the solution might be ambiguous. This problem arises when it unknown if new data should be integrated as an island or connected to the existing data. Jamaica Bay has lots of islands and there are quite large aberration between VMAP1 data and DNC Harbour data in this area. When updating small areas in second scenario this problem is common. One of these occurrences is seen in figure 4.17, the blue line are VMAP1 data and the red lines are DNC Harbour data. Figure 4.17 show the edge vertices in the DNC dataset.

The Regular Integration Method If a regular geometrical integration process is carried out on this kind of problem it will most likely succeed but it will produce a possible erroneous result. If the result is correct or erroneous depends on the new data set which should be integrated. If we use the example in figure 4.17 to do a straight forward conflation process the result will be incorrect. As seen in figure 4.18 the usual buffer zones are applied to all edge vertices, and matching nodes in the existing line segment are found. The next step in the integration process will be to connect the old and new data set together, this is seen in figure 4.19. This result is incorrect, it create in this setting a coastline with an island inside the coastline. As mentioned above the correctness of the use of regular conflation on these problem instances will in many settings produce a positive result.

The Peninsula Method A more secure way to integrate this type of data is to merge a few of the edge vertices and connect the rest of the edge vertices with each other. It is

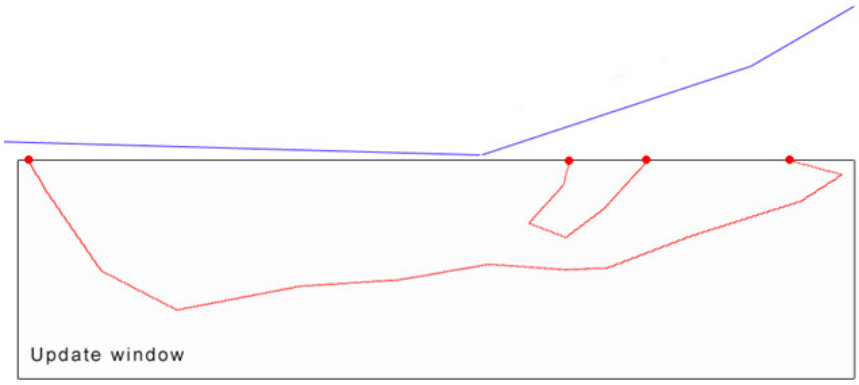

Figure 4.17: The lines inside the update window the new feature member lines which should be integrated to the existing data set (DNC Harbour). The blue lines are line segments from the original data set (VMAP1).

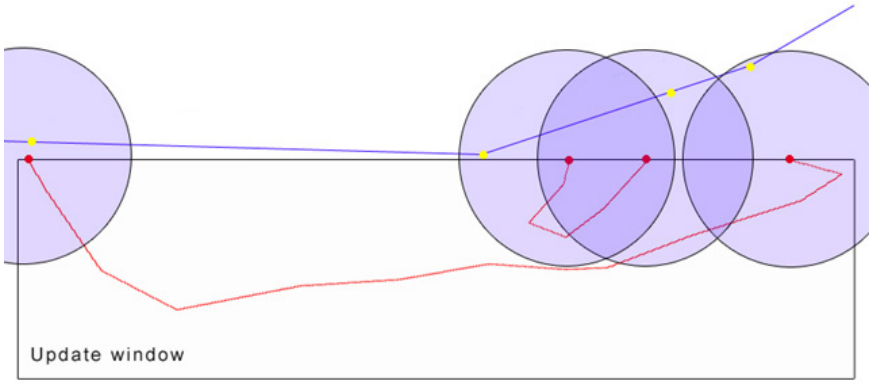

Figure 4.18: This figure use the regular integration method. All edge vertices use their buffer zones to find nodes in the original line segments, and create a line segment between the edge vertex and the nodes.

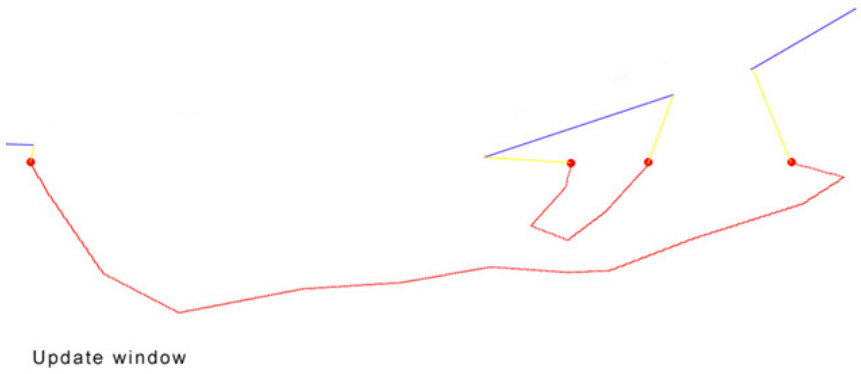

Figure 4.19: The result of a regular integration processes. This create a faulty result, with a island inside the coastline.

still no guarantee that this will create a correct result. Figure 4.20 show the first of three examples using this method of integration. Figure 4.20 merge the two edge vertices to the left with the nodes found in the buffer zone. The two edge vertices that are to the right are connected to each other. This procedure create a peninsula of the new data, connected to the VMAP1 data set, this is seen in figure 4.21. The result of this is correct and will be seen as a sensible way to integrate the DNC data. The next example in figure 4.22 combine the edge vertices in the middle with nodes found in the buffer zone. The second edge vertex is also connected to the edge vertex to its left and the third edge vertex is connected to the edge vertex to its right. This is an incorrect conflation this create a coastline but it has an undefined line segment connected to it, the result make no sense. The result of that process is seen in figure 4.23.

The third way of integration is to connect the two edge vertices to the right to the nodes found in their buffer zone, seen in figure 4.24. The edge vertices to the left are connected to each other and the result are as in the first example a peninsula, figure 4.25. This is actually the same method used as the one in the first example but it uses the edge nodes to the right and not to the left.

The Island Method The final solution and most likely the most secure way to solve this problem is to do a internal integration. With this approach the edge vertices in the integrated line segments are connected to each other, as seen in figure 4.26. This will create a correct line structure since no modifications are done to the existing data set. The result of this approach always creates a island, of the integrated line segments.

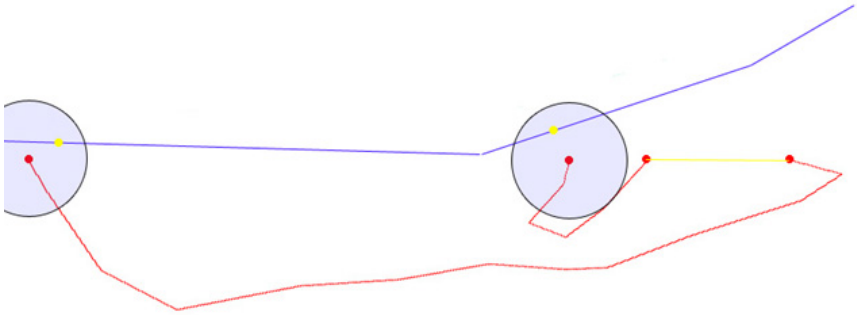

Figure 4.20: The use of the peninsula method using the two edge vertices on the left, and connecting the edge vertices on the right to each other.

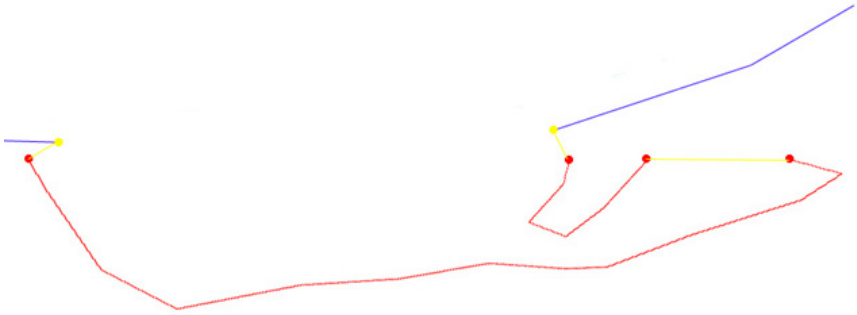

Figure 4.21: The result of a left side peninsula integration This give a correct result.

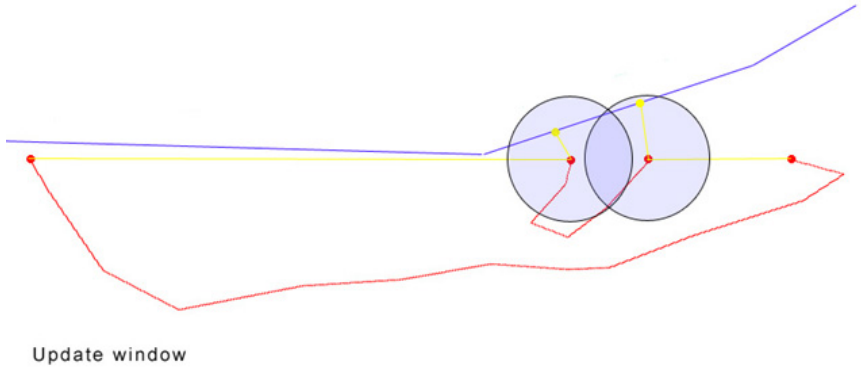

Figure 4.22: The use of a center peninsula method, connect the edge vertices in the center to the VMAP1 line segment.

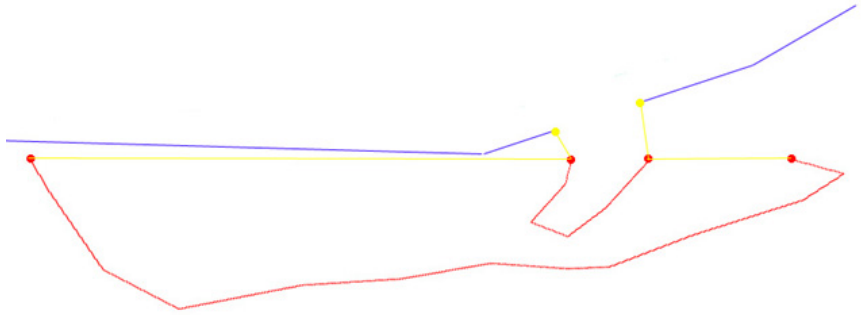

Figure 4.23: The result of a center peninsula method, this is a incorrect result.

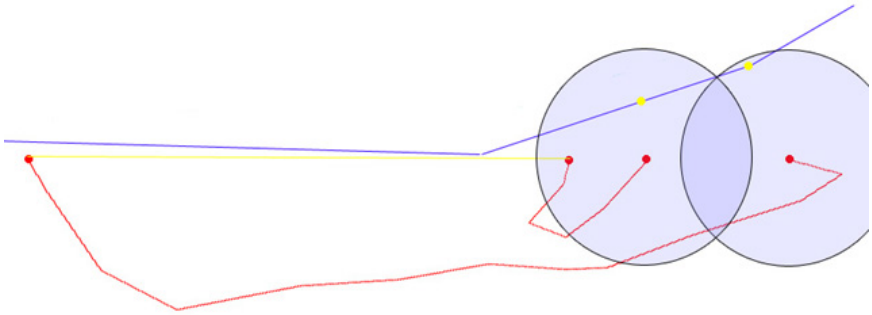

Figure 4.24: The use of a right side peninsula integration, connect the edge vertices to the right to the line segment.

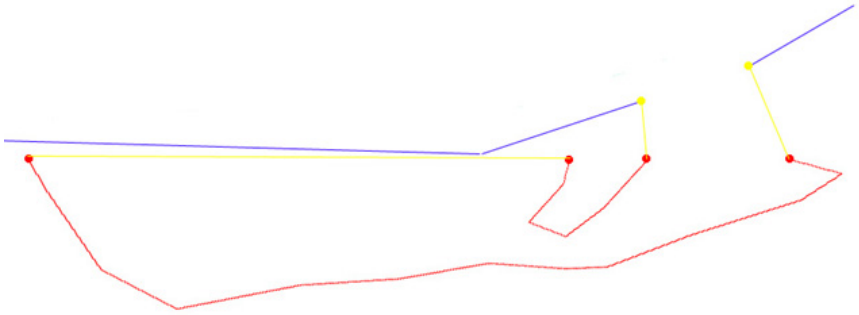

Figure 4.25: The result of a right hand peninsula integration, this give a correct result.

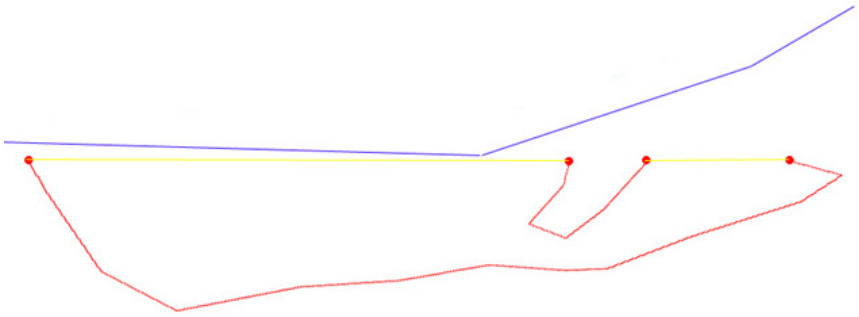

Figure 4.26: The result of a island method, this do not interfere with the existing data set. It use a dangling edge process on the new data set and create a island of the line segments in the new data set.

4.4.2 Integration of Closed Geometries

A closed geometry is geometry with no edge vertices, also known as a face. An example of this is an island or a reef. The problem occurs when the integration of a new face intersects or touches existing line segments. The new data are more accurate than existing data in the repository, and is the main cause of the intersection. But there can be other factors that cause this situation. These factors can be differences of the datum [35] or projection [7] of the data sets. This will create errors in the repository and can not be accepted. This problem is illustrated in figure 4.27. There are no edge vertices in the new data set, it is also presumed that the new data set has higher quality than the existing data set so to manipulate the new data set is not an option.

To solve this problem one has to consider the existing data set and the update window that holds the new geometry. Since the new data set should not be touched, we will find the nodes in the existing dataset closest to the update window. When these nodes are found one or two of the corners of the update windows are found. If one or two corners should be used depends on the positioning of the line segments node close to the update window. When these nodes and corners are found, it connects the nodes to the corners and changes the direction of the line segment. Figure 4.28 show the update window with highlighted nodes in yellow and window corners as blue circles. In figure 4.29 the new lines are marked in blue, notice that the line follow the edge of the update window. The result of this process is seen in figure 4.30. The old coastline that intersected with the new data set is replaced with parts of the update window border.

This will in many settings give a positive result, but some precautions have to be made. In some instances the feature in the update window will lie adjacent to the border of the update window. In such an instance a extension to the method is needed. It needs a margin on the outside of the update window. This margin does not need to be large, just big enough

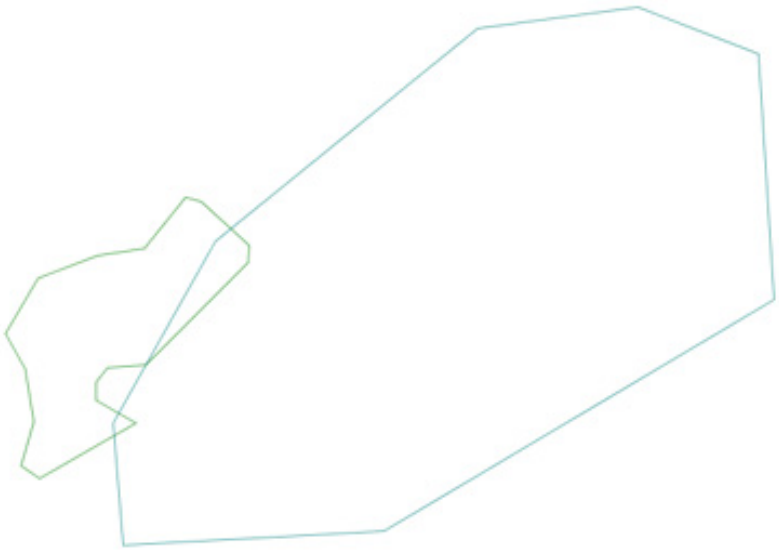

Figure 4.27: The large island is VMAP1 data and the small island is DNC Harbour data. The DNC island intersect the VMAP1 island.

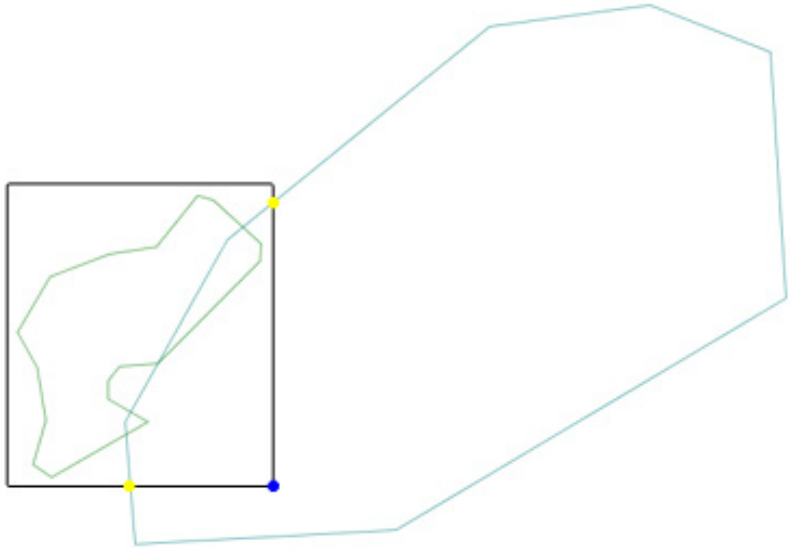

Figure 4.28: Finding the nodes that cross the update window, seen as yellow circles. The corner of the update window is highlighted as a blue circle.

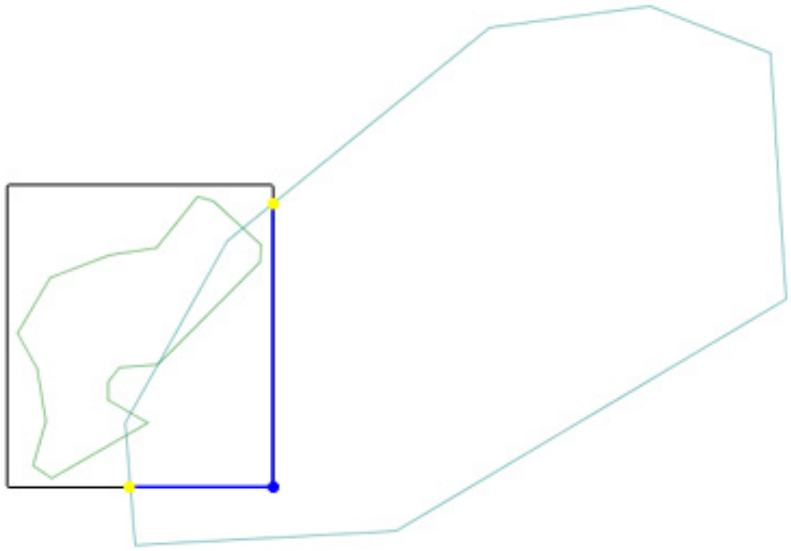

Figure 4.29: When the nodes are found it connect the nodes along the border of the update window.

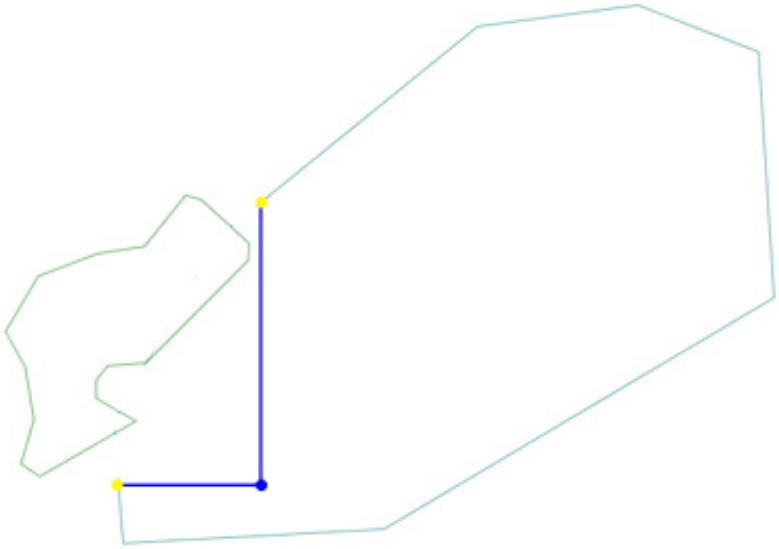

Figure 4.30: The result of the face integration process, notice that the new data have not been touched.

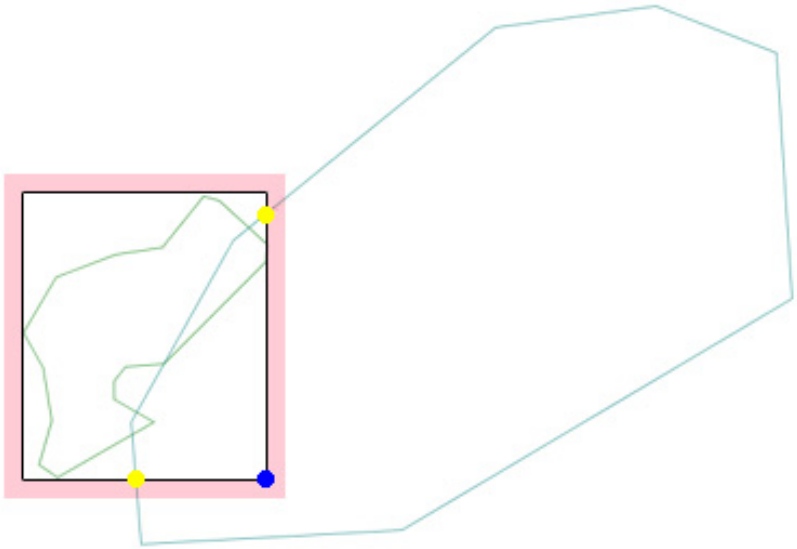

Figure 4.31: The update window use a margin to create a gap between the line segments in the new and old data set.

to separate the new feature from the existing feature. The use of margin around the update window is illustrated in figure 4.31. The line segments that are found near the border of the update window will be moved to the margin of the update window. The new direction of the line segment follow this margin, see figure 4.32. This will result in a integration of the new data set, with a guaranteed gap between the line segments in the old data set and new data set. Figure 4.33 show how the result of the use of margins.

4.5 The Geometrical Integration Process

A geometrical integration process are a complex procedure, where many special problems can arise. These problems can be simple or they can be very complex. The complex problems will in most settings require human interaction, and are not possible to solve automatically by a computer. As seen earlier in the problems described the La Guardia airport scenario illustrates a rather simple problem, and the Jamaica Bay scenario illustrates a difficult problem. To highlight these scenarios the JUMP [22] platform has been used to perform these geometrical integration processes. It is the JUMP 1.2 alpha version that has been used in these examples. Earlier versions of JUMP [22] do not have the splitting of line segments tool and can therefore not be used for this purpose.

The examples and processes used in this section only represent the first iteration in a peer-review process. It will describe the actual integration of new data, and which steps are taken to integrate it in the existing repository. Further in this the peer-review process

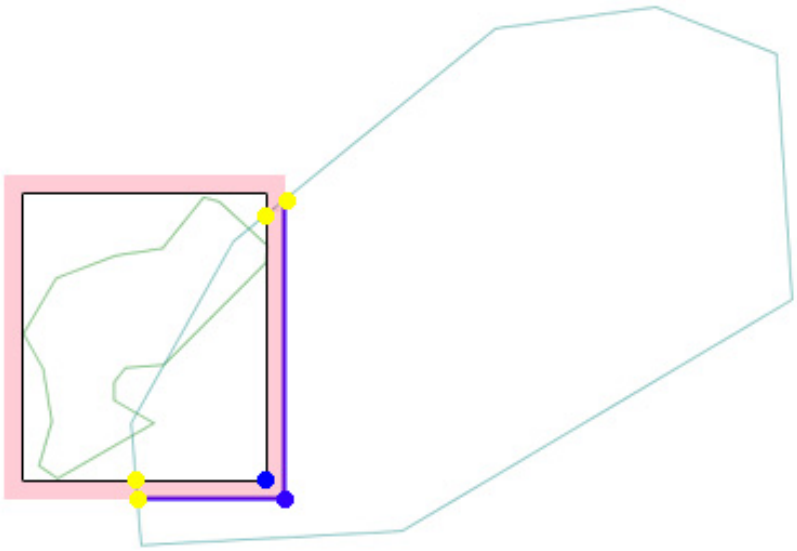

Figure 4.32: When a margin is used the intersection point of the update window and the line segment from the existing dataset is moved to the border of the margin.

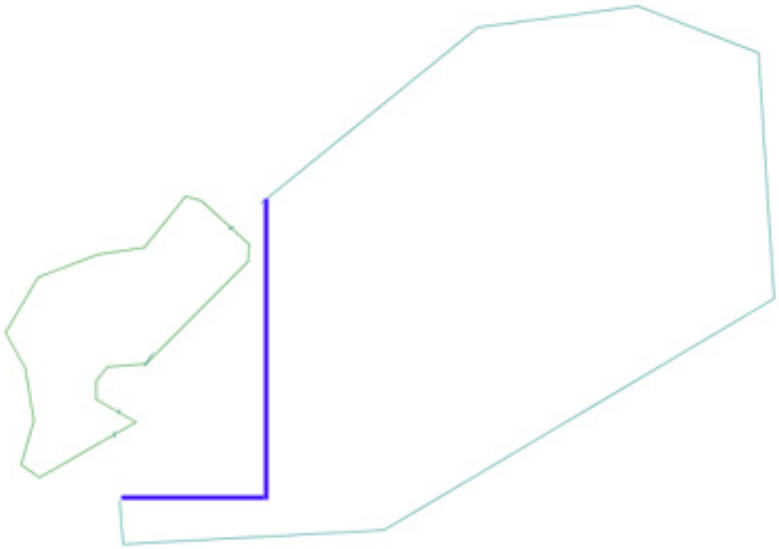

Figure 4.33: The result of use of margin guarantee gap between the line segments in old and new data set.

the correctness and acceptance of the data would be in focus.

4.5.1 Geometrical Integration With JUMP

This section will deal with the manual geometrical integration process on the scenarios in section 4.1.

La Guardia Airport Scenario

The first scenario in section 4.1 describe the coastal area of La Guardia airport. The VMAP1 data is of highest accuracy and should not be modified, the VMAP0 data can and will be modified to match the edges vertices in the VMAP1 data set. It is now time to do the actual geometrical integration on the area. When both data sets are loaded into JUMP [22] as layers, they have to be set as editable layers. Editable layers means that it is possible to do changes on the layers. The next step is to define the area of replacement, the area where VMAP0 data should be replaced with VMAP1 data. This is done by drawing a fence, this fence is seen in figure 4.34 as the blue rectangle. This figure 4.34 also show VMAP0 data as green lines and VMAP1 data as red lines. Now the geometrical integration area is determined and it is time to retrieve the line segments in VMAP1 that lie inside the fence. To do this, the VMAP0 layer has to be disabled; when it is disabled it is not visible. Then by right clicking inside the fence and choose the "select features in fence" all vertices inside the fence are chosen. This method also selects some vertices that lie outside the fence, but these will be removed later. When the vertices are selected, it is time to right click again and chose "copy selected items". Then create a new layer and paste the copied vertices in the new layer. At this point we have no more use of the VMAP1 layer, so it is deselected/not visible. In the new layer that were created the line segments that are crossing the border of the fence are removed, and the edge vertices are positioned exactly on the border of the fence, see figure 4.35. Now the new data from VMAP1 are ready to be integrated in the VMAP0 data set, but before that the VMAP0 data set has to be prepared for the integration. This requires that only the VMAP0 data set and the fence is visible/selected. The line segments inside the fence are highlighted and divided from the continuous line segment using the line splitting tool. The line segments inside the fence are deleted, and the edge vertices in the outside border of the fence are aligned exactly to the border. The result of this is seen on figure 4.36. At this point it is time to merge the VMAP0 layer with the selection of the VMAP1 layer. This is done by copying all features in the new layer that were created earlier in the process, and pasting them into the VMAP0 layer as seen in figure 4.37. Nevertheless this does not complete the geometrical integration process. As seen in figure 4.37 the line segments from VMAP1 does not align with the line segments in the VMAP0 layer. The next step aligns the edge vertices in VMAP0 layer with the VMAP1 edge vertices. To do this the "move vertex" tool are used, and the edge vertices in the VMAP0 layer are moved to align with the vertices in VMAP1. See figure 4.38. The next step is to save the VMAP0 layer as a new data set, the finished result of this new data set is seen in figure 4.39.

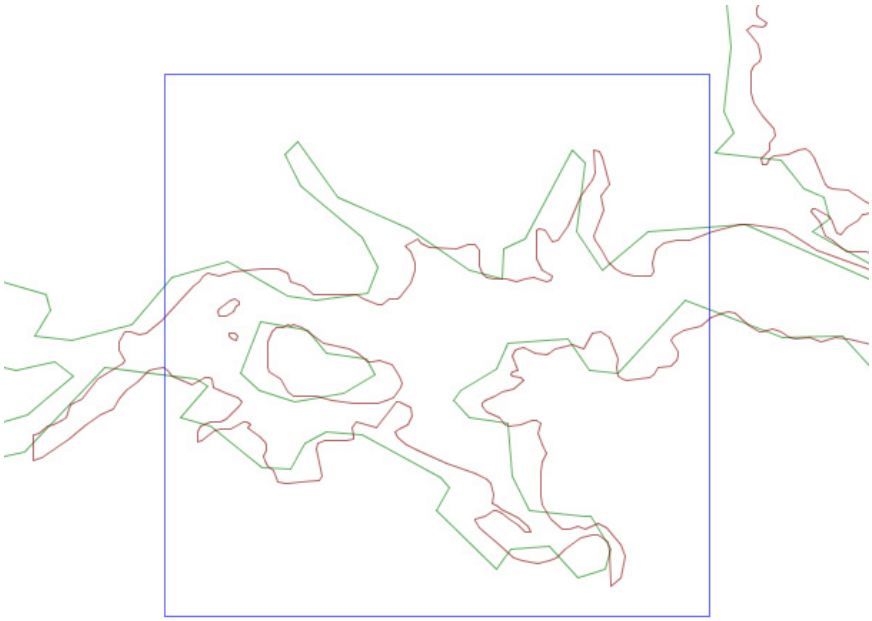

Figure 4.34: The first step in the integration process with the use of the JUMP workbench. The Fence seen as the blue rectangle. VMAP0 data as green lines and VMAP1 data as red lines.

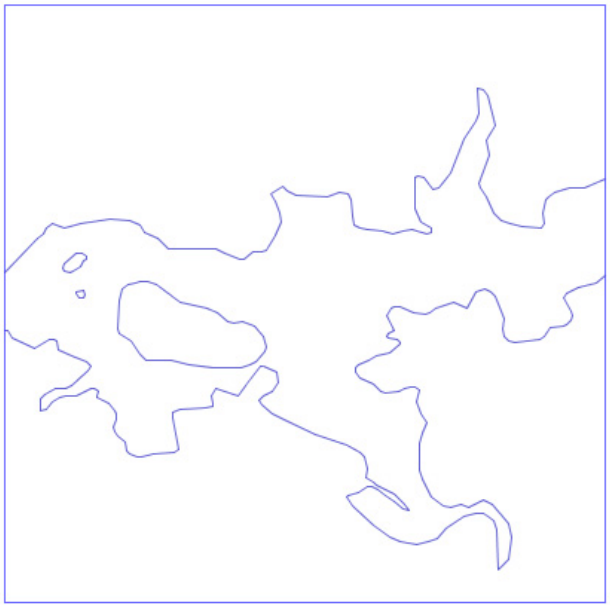

Figure 4.35: The second step in the integration process. The selection of VMAP1 data inside the fence.

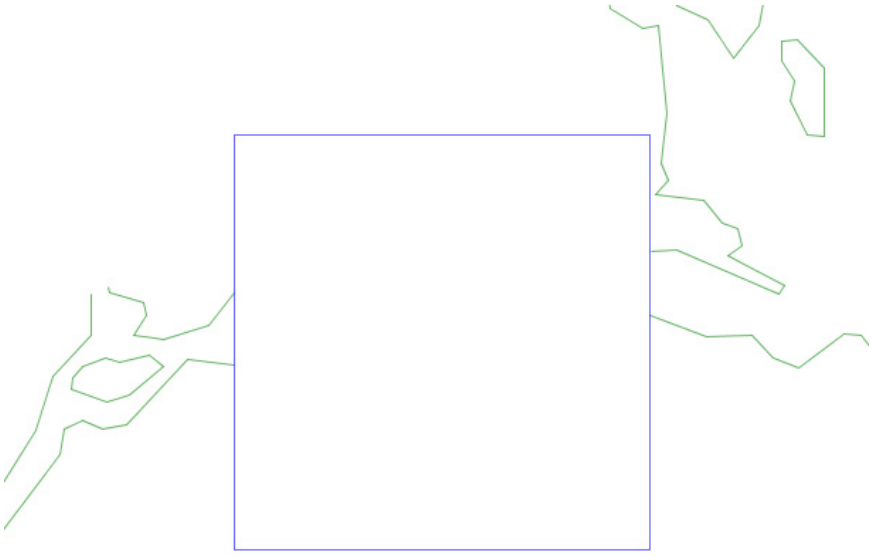

Figure 4.36: The third step of the integration process. The result of removal of VMAP0 data inside the fence/integration area.

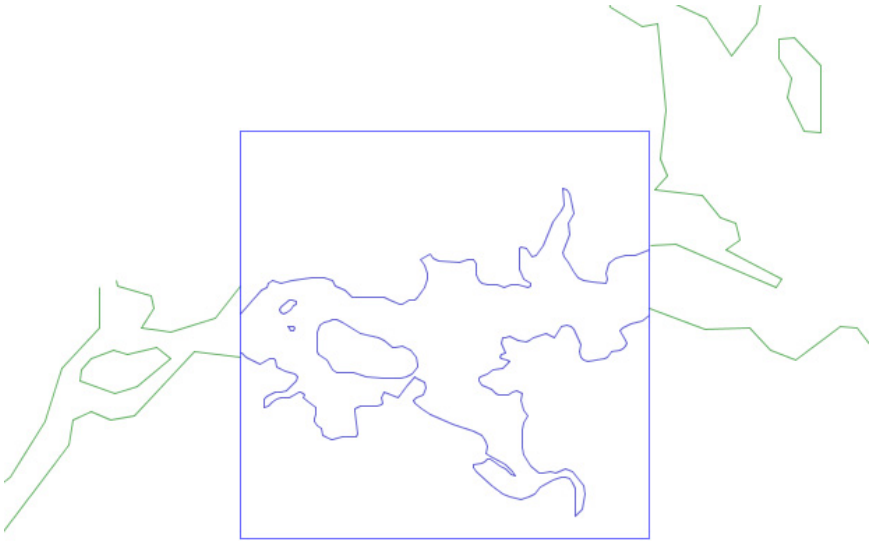

Figure 4.37: The fourth step of the integration process. VMAP1 data is integrated with the VMAP0 data, edge vertices are not aligned.

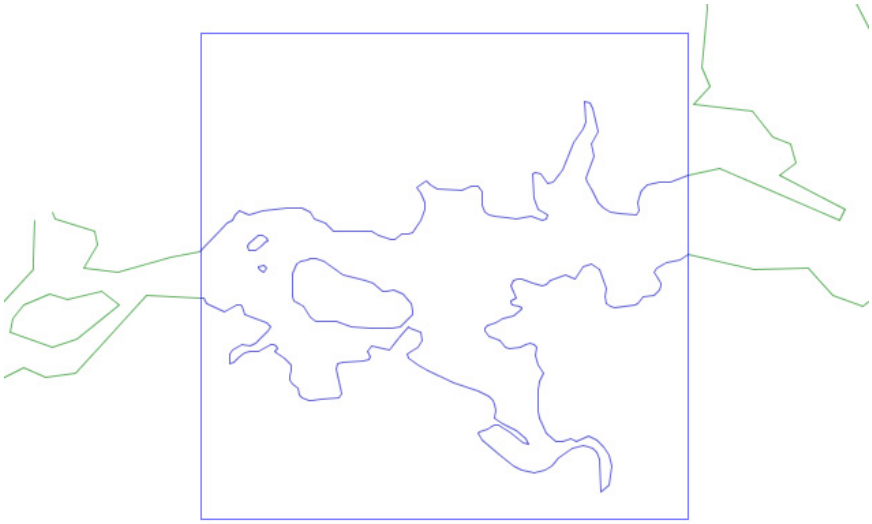

Figure 4.38: The fifth step of the integration process. VMAP1 data integrated with the VMAP0 data, edge vertices are aligned.

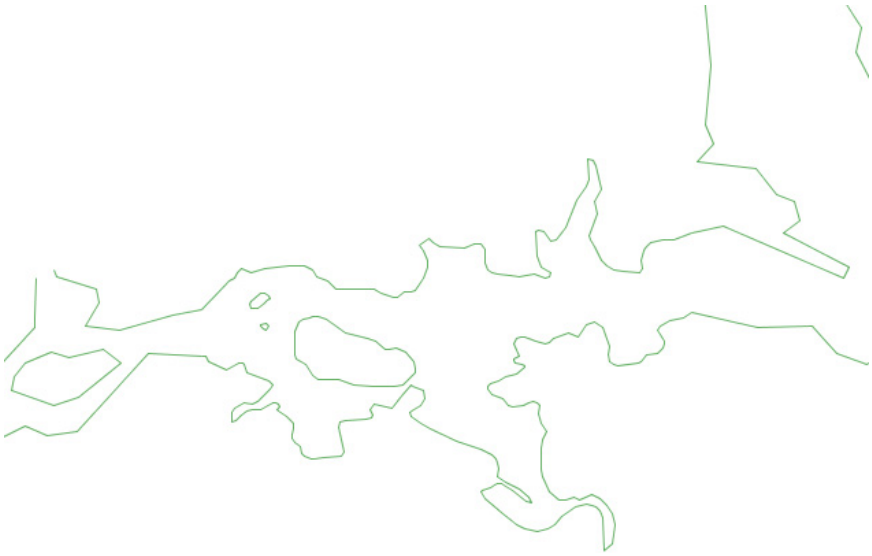

Figure 4.39: The finished result of the integration process using JUMP.

Jamaica Bay Scenario

In the second scenario described in section 4.1 the area of Jamaica Bay is described. In difference to the La Guardia airport scenario one the data set are over this area are built up by VMAP1 data, and should be updated with DNC data. The geometrical integration process will performed in the same way as the first scenario was, this scenario will therefore not be described as accurate as the first scenario. Step one in the process are to load both data set into JUMP and make them editable. Next step is to draw a fence that defines the update area. The DNC data in that area will be extracted and temporary stored in another layer. The original DNC data set is no longer needed and is removed. Features and line segments from the VMAP1 data set that are in the update area are removed, as seen in figure 4.41. The DNC data are now copied from the temporary layer and into the VMAP1 layer. The edge vertices in the VMAP1 data set are aligned with the edge vertices from the DNC data set. The result of this process are seen in figure 4.44 and figure 4.45.

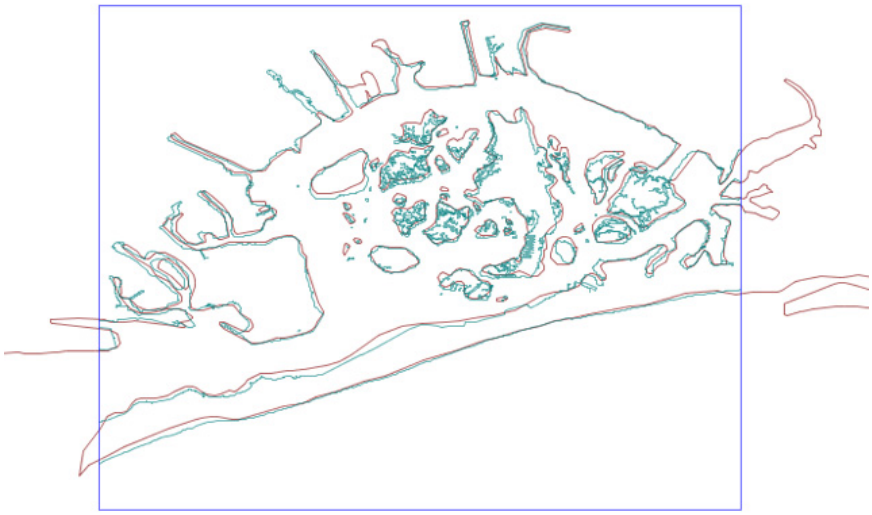

Figure 4.40: The first step of the integration process in the Jamaica Bay area. Data from both VMAP1 and DNC are visible, the blue rectangle define the fence/update window.

Figure 4.41: The second step of the integration process of Jamaica Bay. The DNC data in the update area are extracted from the original data set.

Figure 4.42: The third step of the integration process of Jamaica Bay. Features and line segments in the VMAP1 data set that are in the update window are removed.

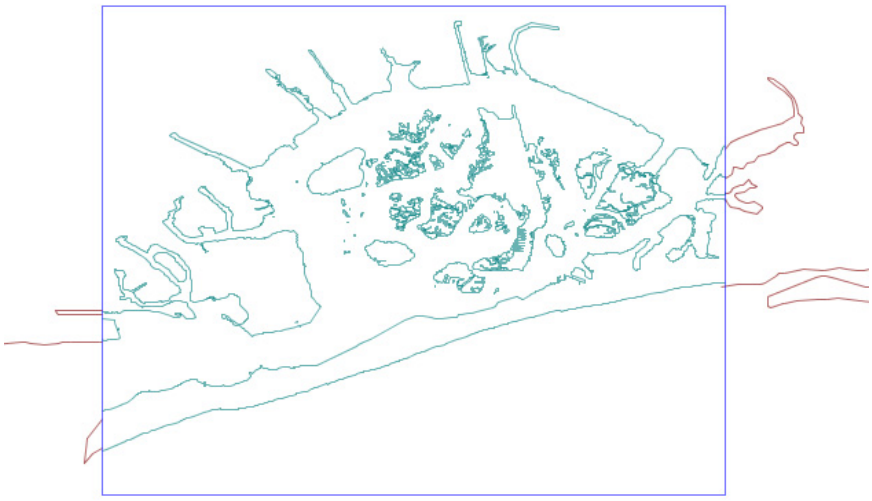

Figure 4.43: The fourth step of the integration process of the Jamaica Bay. DNC data are only inside the update window and VMAP1 data are only on the outside of the update window.

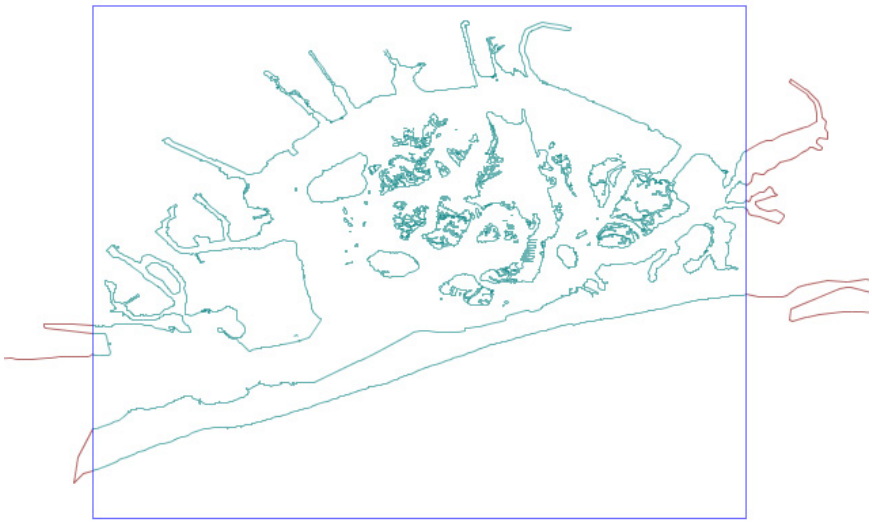

Figure 4.44: The fifth step of the integration of Jamaica Bay. DNC and VMAP1 data are aligned.

Figure 4.45: The result of the integration process of Jamaica Bay. The line segments are aligned and the integration process are finished.

Chapter 5

Syntactical Integration

In section 4 the introduction to geometrical integration was described, in this section syntactical integration will be discussed with regards to the geometrical integration. Semantic integration [38] will not be discussed here. Syntactical integration is the process of merging several geometrical data sets into one file without any loss of information. The main focus will lie on the process of integrating several data sets into one GML [17] file.

The process of doing a geometrical integration process and a syntactical integration process create some contradicting conflicts. The geometrical integration process will merge and align geo-spatial data together, and that is what desirable from that process. In the syntactical integration process the purpose of merging the two geometrical data sets together without loss of information from either of the data sets. A pure geometrical integration process result in aligned data sets, it take no considerations to the syntactical merging process. The geometrical integrated data sets are still saved in separate files, however, this is not desirable. The geometrical integration process can merge the two data sets together, but that create a loss of information. The geometrical integration process do not allow integration of attribute information and other important information as original data, integration date and creator. These are all relevant and very important information that should accompany the geometrical features. If that information is not accounted for, the file would go corrupt after a couple of geometrical integration processes. The term corrupt in this context means that the geometrical information stored in the file would be uncertain. It is impossible to know which geometrical features that have the highest resolution or relevance. This may cause the user of the file to do a geometrical integration process that replace geometrical features of high quality with features of lower quality. This is the main reason of the high relevance of a proper integration process. Still there is one problem that also have to be mentioned. It can be problematic to do the syntactical merging of the geometrical integrated files as a step outside the geometrical integration process. This problem occur if a if the syntactical integration process is performed outside a GIS workbench, with a supporting coordinate system. In such a case the geometrical features coordinates are uncertain. These problems might create erroneous data sets, with noncontinuous line segments in the merging point of the two data sets.

Throughout this chapter some of the problems and with syntactical integration are

highlighted. Further on a meta data model is described, this model is essential to a successful geometrical integration process. The last a geometrical integration method is sketched, this method is known as the lazy integration process [38].

5.1 The Multi Source Polygon Problem

The representation of polygons in GML [17] use the Polygon tag. A polygon have an outer and inner boundary, these boundaries can not cross each other. The coordinates in these boundaries use the tag LinearRing, this tag is a container for coordinates. A characteristic for a linear ring is that the last coordinate have to coincide with the first coordinate in the linear ring.

The core of the polygon problem is that a linear ring can only consist of one set of coordinates. This complicates the conflation process in instances where polygons should be built up by data from various data sets. This complications occur when you have two sets of coordinates, one from the original data set and one from the data set that should be integrated.

The Use of Multi Line Strings to Represent Polygons

The simplest and most straight forward way to solve this problem is to reject the use of the polygon tag and use a multi linestring to represent the polygon. JUMP [22] use this method to represent polygons that have linestrings from various data sources. However this method do not guarantee that the polygon is closed, multi linestrings do not require a closed geometry. If by some reason a error occur this would leave the polygon open and still validate the result as correct. An example of the use of multi linestrings are seen in figure 5.1. As seen the first coordinate in the first linestring is identical to the last coordinate in the last linestring.

```
<gml:MultiLineString>
    <lineStringMember>
        <LineString>
          <coordinates>0.0,0.0 2.0,0.1 2.1,1.22</coordinates>
        </LineString>
    </lineStringMember>
    <lineStringMember>
        <LineString>
          <coordinates>2.1,1.22 3.0,0.345</coordinates>
        </LineString>
    </lineStringMember>
    <lineStringMember>
        <LineString>
          <coordinates>3.0,0.345 0.0,0.0</coordinates>
        </LineString>
    </lineStringMember>
</gml:MultiLineString>
```

Multi Source Polygons Based on a Defined XML Schema

Another way to solve the problem is to develop and use a XML [15] schema customized for this problem. This require an own XML schema [31], and might not be the optimal way to do this, but it ensure that a valid GML file contains closed polygons. The result of this approach are seen below. The XML schema for figure 5.1 is seen in figure 5.1. The polygon XML schema are based on the GML [17] standard and the Dublin Core [29] standard. The core of the new XML schema is that it allows gml:coord elements to lie inside a defined element called Dataset. Using this method the polygon can be built up by coordinates from various data sets. The Dublin Core are used to tag information on the geometrical features. Both original features and integrated features. This method guarantee that the first and last coordinate are equals, in difference to the method mentioned in section 5.1. The drawback with this method is clearly that it require a own XML schema to be accepted as valid XML.

```
<gml:Polygon>
  <outerBoundaryIs>
    <LinearRing>
      <Dataset>
        <dc:coverage>DNC Harbour 177960</dc:coverage>
        <gml:coord>
          <gml:X>0.0</gml:X><gml:Y>0.0</gml:Y>
        </gml:coord>
        <gml:coord>
          <gml:X>0.2323</gml:X><gml:Y>0.97</gml:Y>
        </gml:coord>
      </Dataset>
      <Dataset>
        <dc:coverage>Vmap0</dc:coverage>
        <gml:coord>
          <gml:X>0.2323</gml:X><gml:Y>0.97</gml:Y>
        </gml:coord>
        <gml:coord>
          <gml:X>1.0</gml:X><gml:Y>1.024233</gml:Y>
        </gml:coord>
      </Dataset>
      <Dataset>
        <dc:coverage>Vmap1 NY</dc:coverage>
        <gml:coord>
          <gml:X>1.0</gml:X><gml:Y>1.024233</gml:Y>
        </gml:coord>
        <gml:coord>
          <gml:X>0.0</gml:X><gml:Y>0.0</gml:Y>
        </gml:coord>
      </Dataset>
      </LinearRing>
  </outerBoundaryIs>
</gml:Polygon>
```

```
<?xml version="1.0" encoding="UTF-8"?> <xs:schema
xmlns="no:hiof:onemap:appschema:polygonschema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink"
targetNamespace="no:hiof:onemap:appschema:polygonschema"
elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:import namespace="http://www.opengis.net/gml"
             schemaLocation="feature.xsd"/>

  <xs:import namespace="http://www.opengis.net/gml"
             schemaLocation="geometry.xsd"/>

  <xs:import namespace="http://purl.org/dc/elements/1.1/"
             schemaLocation="simpledc20021212.xsd"/>

  <xs:element name="FeatureCollection" type="FeatureCollectionType"/>
  <xs:element name="Feature" type="FeatureType"
              substitutionGroup="gml:_Feature"/>

  <xs:complexType name="FeatureCollectionType">
    <xs:complexContent>
      <xs:extension base="gml:AbstractFeatureCollectionType"/>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="FeatureType">
    <xs:complexContent>
      <xs:extension base="gml:AbstractFeatureType">
        <xs:sequence>
          <xs:choice>
            <xs:element ref="gml:Polygon">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Dataset">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element ref="gml:outerBoundaryIs">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element ref="dc:coverage"/>
                              <xs:element ref="gml:coord">
                                <xs:complexType>
                                  <xs:sequence>
                                    <xs:element ref="gml:coord">
```

```
                                        <xs:complexType>
                                          <xs:sequence>
                                            <xs:element ref="gml:X"/>
                                            <xs:element ref="gml:Y"/>
                                          </xs:sequence>
                                        </xs:complexType>
                                      </xs:element>
                                    </xs:sequence>
                                  </xs:complexType>
                                </xs:element>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element ref="gml:LineString"/>
                    <xs:element ref="gml:Point"/>
                  </xs:choice>
                </xs:sequence>
              </xs:extension>
            </xs:complexContent>
          </xs:complexType>
        </xs:schema>
```

5.2 Metadata

In this context the metadata is information that tell something about the geometrical and syntactical integration process. This will typically be information that says where the data are from, what kind of format it were, when the geometrical and syntactical integration process were performed and so on. The relevance of some metadata will depend on the usage and requirements of the geometrical data. The metadata elements can of course be modified to fit the needs of the data repository. This section will define metadata that is relevant for the integration of data to the OneMap [36] repository [33]. However these metadata elements seems to be relevant in most settings where a geometrical and syntactical integration process has been performed.

The data set used in through the geometrical integration process has been GML [17], so the specification language are already set. The GML [17] do not have any elements to represent metadata in the form defined here. This require that other resources are used in combination with GML [17] to represent the metadata. The Dublin Core [24] [29] are a initiative for standardization of metadata. Dublin Core [24] [29] are well known and widely accepted initiative. Dublin Core [24] [29] has all the metadata elements needed

in the geometrical and syntactical integration process, this means that it cover the needs completely.

Metadata can be connected to all parts of geometrical information, if it is a geometrical feature, a part of a geometrical feature or a complete set of geometries, for instance a GML [17] file. Depending of the type of geometrical information that are merged the metadata will vary. A geometrical feature will have different metadata information than a line segment will have. Each of the metadata elements required by the OneMap [36] repository [33] are described below:

- Identifier, all features and elements that are integrated into a GML [17] file have to get a unique identifier.
- Date, When was the feature added to the file. This is of historical relevance, and can be used to perform rollbacks of erroneous data. It is also useful to determine a future conflation process, regarding the newest data.
- Format, the original data type has to be known, for instance it is of great relevance if a feature are of Vmap0 [11] or Vmap1 [8] resolution. Required field.
- Creator, this is the name of the person that performed the geometrical and syntactical integration process. A required field if it is of importance to know who performed the integration process. This field can also be used to identify the institution of firm that have done the integration process.
- Description, a description of the geometrical feature, if it has any special characteristics. This is a quite open element where the the user can write what he seems relevant. This field should be optional.
- Coverage, identify the area the it cover or the area the feature is in. This field should be optional.
- Title, A optional field that give the feature a name. If a feature has a special name, this can be a polygon that represent a island, for instance Staten Island. The title field would in that instance contain "Staten Island".
- Replaces, a optional field that are used if the geometrical integration process replace a feature with a new one. This can for instance happen with islands. This field require that historical information are stored in the repository in some way. If not this field are obsolete because without a historical repository the replaced feature would be lost.
- Is Part Of, this element would only be used if the metadata information are on segment level. That means that for instance all line segments in a Polygon have metadata information from the integration process. The is part of field will give information on which feature the line segment are a part of.

The first time a integration process is being performed, a metadata problem occur. Since the file have not been integrated with anything before, it do not contain metadata as it is defined here. This cause a indeterminable data set, because the new data that are merged into the file will have information. This metadata contain information on the integration date, and its original data set. The original data do not have any of these elements, though some of them is irrelevant for the data. The question in such a context is, how should the metadata be added to the original data set? There are two possible approaches to this problem:

Metadata header , this approach add a header of metadata at the beginning of the file. This metadata header will contain metadata which are valid for all features that do not have their own metadata elements. for all features that do not have its own metadata. The drawback with this method is that the original features can not have unique identifiers and names. This may not be essential in some settings, but important in other settings. An example of a metadata header can be seen in below.

```
<FeatureCollectionInformation>
  <dc:identifier>OneMap_coastline_T05</dc:identifier>
  <dc:date>2005-05-06</dc:date>
  <dc:replaces>NewYork_coastlines_2004_05_24.xml</dc:replaces>
  <dc:creator>Kristian Lunde</dc:creator>
  <dc:description>
    Information on what kind of new information this file
    contain that the previous file did not have. A general
    description of the data that this file contain.
  </dc:description>
  <dc:coverage>New York area</dc:coverage>
</FeatureCollectionInformation>
```

Feature based metadata , is a approach where metadata information are distributed to all original features. This open the possibility to give each feature a name and a unique identifier. This method is required if segments of a feature should contain metadata. For instance a line segment that contain the metadata field "is part of" require this approach. If each feature require a special identifier and name, this may create some manual work to edit and give each feature the correct name and identifier. As default the identifers and names should be automatically generated. Due to redundant storage of information this approach will produce files with larger size than with the metadata header approach. An example of feature based metadata storage are seen below.

```
<gml:featureMember>
  <VMAPFeature>
    <faccDescription></faccDescription>

    <FeatureInformation>
      <dc:identifier>OneMap_coastline_T05_L54823</dc:identifier>
```

```
      <dc:replaces>
        OneMap_coastline_segment_345,
        OneMap_coastline_segment_346,
        OneMap_coastline_segment347
      </dc:replaces>
      <dc:date>2005-05-06</dc:date>
      <dc:creator>Kristian Lunde</dc:creator>
      <dc:title>Coastline</dc:title>
      <dc:description>
        Coastline in the area of La Guardia bay.
      </dc:description>
      <dc:format>DNC Harbour 1707960</dc:format>
      <dc:coverage>Jamaica Bay</dc:coverage>
    </FeatureInformation>

    <gml:lineStringProperty>
      <gml:LineString>
        <gml:LineString>
              <gml:coordinates>
                ....
              </gml:coordinates>
            </gml:LineString>
      </gml:LineString>
    </gml:lineStringProperty>
  </VMAPFeature>
</gml:featureMember>
```

Both these approaches support metadata information on integrated feature segments. The latter approach also enable metadata support on features from the original data set. This is a clear advantage when dealing with problems like integration of polygons with line segments from several data sets. A example of feature segment metadata are seen below.

```
<gml:LineString>
  <LineStringInformation>
    <dc:identifier>OnMap_coastline_segment_463</dc:identifier>
    <dc:isPartOf>OneMap_coastline_T05_L54823</dc:isPartOf>
    <dc:date>2005-05-06</dc:date>
    <dc:creator>Kristian Lunde</dc:creator>
    <dc:title>Coastline</dc:title>
    <dc:description>
      Coastline segment in the area of La Guardia bay.
    </dc:description>
    <dc:format>DNC Harbour 1707960</dc:format>
    <dc:coverage>Jamaica Bay</dc:coverage>
  </LineStringInformation>
  <gml:coordinates>
    ....
   </gml:coordinates>
</gml:LineString>
```

To select one of these method should be based on the requirements of the data set, its usage and a personal preference. The first method give files that are smaller than the second method, but do it do not hold specific feature based information. In the OneMap [36] project the second approach are most preferable and are used in the integration process.

5.3 Lazy Integration

Lazy integration is a method already used in the OneMap [36] project, on semantical integration of data [38]. This method should also be able to handle integration of geometrical information. When lazy integration were used on semantic integration it were used to integrate semantic information together with already existing semantic data. This were done by using special purpose XML [15] schemas. The main objective with lazy integration is to preserve the structure of the integrated information. Using this method a GML [17] file will be able to handle storage of geodata that have different structure and information. The purpose of storing the integrated data unmodified is to reduce the risk of corrupting and destroying the geospatial information. If the method should have modified the data the risk of unintentionally changing and creating errors in the information are present. Without changing the structure no errors can occur in the syntactical integration, and the integrated information is kept in its original structure.

5.3.1 The XML Schemas

In the process of developing a lazy integration approach for geometrical information, is it natural to use the original lazy integration approach as foundation. To do this the geometrical integration will use the same schema structure as the original lazy integration approach. The structure will consist of three super schemas that hold the superior GML structure. Then there will be several schemas that extend the structure of the super schemas, and add specialized structure. These classes deal with the actual merging process, and refer to other external XML schemas that should be used to validate a integrated file. The external XML schemas are used to define some part of a GML file which consist of integrated information. To support the original lazy integration of semantic information, the schemas used here are only expanded as a larger structure. This enable the structure to store both semantic and geometric integrated information. In figure 5.1 the schemas structure of the lazy integration approach is illustrated.

The core schemas of the lazy integration is divided into three. These schemas are described below:

Request.xsd is the schema that is used as the default namespace in a GML file which is built on a lazy integration approach. The main purpose with this schema is that it link all the other schemas together, both the super schemas and the sub schemas.

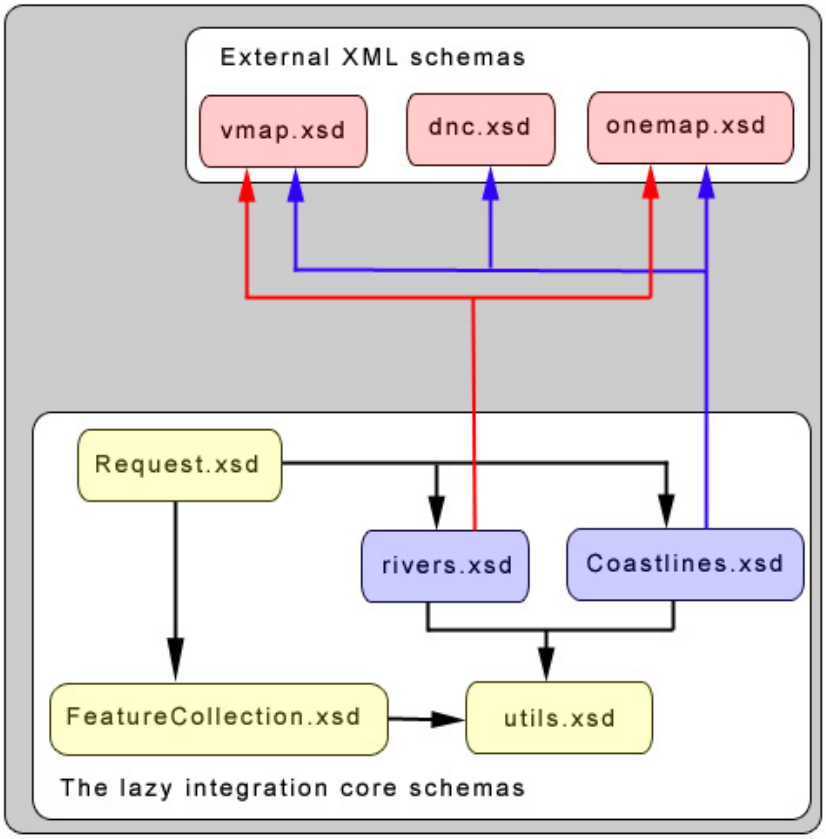

Figure 5.1: The lazy integration structure consist of the core schemas, Request, FeatureCollection and utils.The rivers and Coastlines schema is XML schemas that are subclasses of the utils schema. The rivers and Coastlines schemas import external schemas that is needed to represent different GML structures.

FeatureCollection.xsd define the main XML elements. The FeatureCollection element, which is the root element in the lazy GML file. Other sub root elements are also defined in this schema, as the featureMember element. The featureMember element is the root element of a geometrical feature. The FeatureCollection schema is based on the utils schema.

utils.xsd is top schema, and define the overall structure of the lazy integration structure. This schema define abstract elements that both the featurecollection schema and the sub schemas extends.

Definition XML schemas, is XML schemas that is sub classed from the utils schema. In this thesis the focus will lie on a coastline schema. This schema integrate XML schemas that represent VMAP0 [11], VMAP1 [8] and DNC [12] data. These external schemas are imported as they are and are not changed at all.

Changes Applied to The XML Schemas

The XML schemas that were developed by Harald Vålerhaugen and Gunnar Misund [38], is almost fully adaptable for a geometrical approach. Nevertheless some minor changes had to be done to fit the needs that a geometrical integration requires. In this section there will only be small examples of the changes that were applied to the original lazy integration schemas; the complete schemas is found in Appendix B.2.

Request.xsd The changes done on this schema were merely noticeable. Since the schema is used directly by the GML instance documents it has to include all other needed schemas. The result were that some of the include statements had to be rewrote to load the right schemas.

```
<include schemaLocation="FeatureCollection.xsd"/>
<include schemaLocation="Coastline.xsd"/>
<include schemaLocation="River.xsd"/>
```

All of these schemas was originally included in the Request schema. But since some of the schemas that were included originally were superfluous they were removed. The schemas that are included now is listed above. Notice that the River schema only is used as an example schema and are not actually of any importance in this setting.

FeatureCollection.xsd, the only change that was applied in this schema was the adding of metadata information. In the type definition of the FeatureMember class a metadata element was added. This metadata element was called FeatureInformation and contained Dublin Core [29] elements.

```
<complexType name="FeatureMemberType">
  <complexContent>
    <extension base="one:FeatureAssociationBaseType">
```

```
      <sequence>
        <!-- Metadata information on a Feature -->
        <element name="FeatureInformation">
          <complexType>
            <sequence>
              <element ref="dc:identifier"/>
              <element ref="dc:title" minOccurs="0"/>
              <element ref="dc:date"/>
              <element ref="dc:creator"/>
              <element ref="dc:format"/>
              <element ref="dc:coverage" minOccurs="0"/>
              <element ref="dc:description" minOccurs="0"/>
            </sequence>
          </complexType>
        </element>
        <element ref="one:_AbstractIntegratedFeature"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

utils.xsd, in order to keep the lazy integration schemas as close to the original schemas as possible; the changes tried to be minimal. The main change the utils schema was the adding of metadata information on feature fragments. A feature fragment is for instance a line segment that together with other line segments form a feature.

```
<complexType name="AbstractIntegratedFeatureType">
  <complexContent>
    <extension base="one:AbstractFeatureCollectionBaseType">
      <sequence>
        <!-- Metadata information on feature fragment level-->
        <element name="FeatureFragmentInformation" minOccurs="0">
          <complexType>
            <sequence>
              <element ref="dc:identifier"/>
              <element ref="dc:title" minOccurs="0"/>
              <element ref="dc:date"/>
              <element ref="dc:creator"/>
              <element ref="dc:format"/>
              <element ref="dc:coverage" minOccurs="0"/>
              <element ref="dc:description" minOccurs="0"/>
            </sequence>
          </complexType>
        </element>
        <element ref="one:_abstractFeatureFragment"
                 maxOccurs="unbounded"/>
      </sequence>
    </extension>
```

```
  </complexContent>
</complexType>
```

The metadata element is called FeatureFragmentInformation and contain information that were described in the Metadata section 5.2. Since not all feature fragments will have metadata information, this element is optional.

Coastline.xsd was the schema that had to undergo the largest changes. Since the other core schemas already supported most of the geometrical integration, this schema had to define what kind of features that were allowed to be integrated. Since most of this thesis has dealt with coastlines it was natural that coastlines were used in this section as well. The coastline schema did exist in the original lazy integration system, but it did not support integration of geometrical features from sources as VMAP0, VMAP1 and DNC. Even though this was the schema that underwent the largest changes, these changes consisted of adding support for the data formats mentioned above.

```
<complexType name="IntegratedCoastLineType">
  <complexContent>
    <extension base="one:AbstractFeatureCollectionBaseType">
      <sequence>
        <element ref="one:coastFragment" maxOccurs="unbounded"/>
        <element ref="vmap:VMAPFeature" maxOccurs="unbounded"/>
        <element ref="dnc:Feature" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

VMAP0 and VMAP1 is described by the same schema B.2.6, this result in a element that is used to refer to features of both data sets.

River.xsd rivers has not been the focus in this thesis. The river schema is therefore only included to illustrate that the lazy integration approach can include several schema classes. No changes were done to the river schema since it only were mentioned for illustration reasons.

5.3.2 Using Lazy Integration

In the previous sections the structure and technical build up of geometrical lazy integration were discussed. In this section the focus lie on the use of geometrical lazy integration. This means that we will see how the actual XML instance documents are built up. To do this the two examples used throughout this thesis will be used. As in chapter 4 the point of the process is to integrate a data set of different build up/resoultion into an existing data set of the same area. That process used the JUMP workbench to merge the two data sets together. There is no available workbenches to perform a lazy integration process. So the integration

process done in this section are based on manual labor. The integration processes shown here are not meant be a real life lazy integration implementations, but rather examples.

A lazy integration approach can be connected to a geometrical integration process in two different ways:

- As a integrated part of the geometrical integration process.
- As a separate process that is performed after a geometrical integration process.

There are no obvious method that is better that the other, but a lazy integration process that runs separate from a geometrical integration process have some requirements. These requirements are related to the geometrical integration process. It such a case the geometrical integration process should not merge the files together. All other processes can be performed but not the merging process. Since the lazy integration process require separate files it is essential that the geometrical integration process do not merge the files. The step by step guide to lazy integration seen in section 5.3.2 applies for both of the methods mentioned above. The only difference are how and when these steps are integrated with the geometrical integration process.

Using Lazy Integration Step by Step

The lazy integration approach can be summarized into a few steps. These steps define a standard procedure that starts with two different sets of features and end up with a merged data set.

- When a lazy integration process are initiated the first thing that has to be done is to define which data set that should be the base data set, and which that should be integrated into the base data set. If the lazy integration process is an internal process in the geometrical integration process, will this step already be covered. However if the lazy integration process is carried out as an external process, the user have to define the relevance of the data sets.
- The root element of the base data set is replaced with a new root element. This element define a feature collection and contain all other features and elements in the XML instance document. This start element contain all namespaces needed, this is regular XML practice. In the example schemas defined in this thesis this element is named featurecollection.
- The next step in the process to integrate all features from the integration data set to the base data set. This process is an iterative process that writes all features into the base data set one by one. There are several XML elements which are attached to a integrated feature. All integrated elements are inserted into a element which surrounds the feature. This element contain two child elements, a feature information element which contain metadata about the lazy integration process and the integrated data. The second element is a element that contain the integrated feature. Throughout this this thesis this element will be called IntegratedCoast.

When these steps are carried out, the lazy integration process is completed.

La Guardia Airport Scenario

In chapter 4 the La Guardia airport scenario were subject to a geometrical integration process. This section illustrate the lazy integration process on the La Guardia airport scenario. Still the VMAP0 data set is the base data set and the VMAP1 data set contain features which should be integrated into this.

The first operation performed on the two data sets are the replacement root element in the VMAP0 data set. The existing root element is replaced with an element that contain a common interface to all lazy integration schemas and namespaces. The original root element in VMAP0 is seen below:

```
<VMAPCollection
 xmlns="no:hiof:onemap:gml:appschema:vmapschema"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="no:hiof:onemap:gml:appschema:vmapschema
 http://www.ia-stud.hiof.no/~bjornhho/digmap/gmlEx/vmapschema.xsd">
 :
 :
 :
 </VMAPCollection>
```

The original root element in the VMAP0 data set is replaced with the element seen below.

```
<one:FeatureCollection
 xmlns:one="http://onemap.org"
 xmlns:onen="http://www.onemap.net"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:vmap="http://www.onemap.net/vmap"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.gisline.no"
 xsi:schemaLocation="http://www.onemap.org Request.xsd"
 fid="ID000004">

:
:
:

</one:FeatureCollection>
```

The bounding box from the VMAP0 data set is kept intact, since the integration process do affect that. The original VMAP0 features are not modified at all. There are two reasons to why the features are not touched. Firstly the goal of the lazy integration process is to

store the data as close to its original origin it is not modified. Secondly there are no reason to modify these features, they have not been modified.

The next step in the process is to integrate the features from the VMAP1 data set. Before a feature is integrated a featureMember element is written. This element surround the integrated feature. Then a metadata structure is written, and at last the feature is integrated into a IntegratedCoast element.

```
<one:featureMember>
    <FeatureInformation>
        metadata
    </FeatureInformation>
    <one:IntegratedCoast>
        Integrated feature
    </one:IntegratedCoast>
</one:featureMember>
```

In the example above the structure of a integrated feature element is described. The featureMember element is the first element that define that this feature has been integrated into the existing data set.The metadata is defined secondly and at last the feature is defined.

The featureMember element contain metadata about the integration process and the feature integrated. The metadata information is added in a FeatureInformation element. The metadata information is collected with a combination of automated and human interaction. Some of the metadata elements are clearly collected automatically as date of integration process and identifier. Manually collected metadata elements are obviously elements as name of the person performing the integration process. A complete valid lazy integration structure for a integrated VMAP1 feature is seen below.

```
<one:featureMember>
    <FeatureInformation>
            <dc:identifier>onemap_10102</dc:identifier>
            <dc:title>Coastline</dc:title>
            <dc:date>2005-06-20</dc:date>
            <dc:creator>Kristian Lunde</dc:creator>
            <dc:format>VMAP 0</dc:format>
    </FeatureInformation>

    <one:IntegratedCoast>
        <one:layerDescription>Coastline</one:layerDescription>
        <vmap:VMAPFeature>
            Integrated feature data.
        </vmap:VMAPFeature>
    </one:IntegratedCoast>
</one:featureMember>
```

When the iterative process of integrating features from the Vmap1 data set into the Vmap0 data set, the process is completed. In the example seen in B.3.1 a section of the VMAP0 file after the integration process is seen.

Jamaica Bay Scenario

The second scenario that has been used in this thesis is the Jamaica Bay scenario. As in the La Guardia Airport scenario 5.3.2 the lazy integration approach will be the same.

The first thing that is done is the replacement of the root element in the VMAP1 data set. The new root element is seen below, notice that an extra namespace is added, the DNC namespace, in the La Guardia Scenario 5.3.2 this namespace was obsolete since no DNC data were present.

```
<one:FeatureCollection
 xmlns:one="http://onemap.org"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:vmap="http://www.onemap.net/vmap"
 xmlns:dnc="http://www.onemap.net/dnc"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.gisline.no"
 xsi:schemaLocation="http://www.onemap.org Request.xsd"
 fid="ID000004">

:
:
:

</one:FeatureCollection>
```

The original VMAP1 features are not touched, they are not modified at all. The last process is to add all new features from the DNC data set to the VMAP1 data set. All the DNC features are added into a corresponding featureMember element and metadata are attached to each featureMember element. When the featureMember element and the metadata are written the DNC feature is inserted into a one:IntegratedCoast element. This element is the actual rapper element for the integrated feature. Below is the structure of the integrated DNC feature described. As seen the familiar featureMember element is defining the start of a integrated feature. The one:IntegratedCoast element is defining the start of the actual feature data. The feature is identifed as a DNC feature and therefore inserted into a dnc:Feature element.

```
<one:featureMember>
    <FeatureInformation>
        <dc:identifier>unique feature id</dc:identifier>
        <dc:title>name of feature</dc:title>
        <dc:date>Integration date</dc:date>
        <dc:creator>
           Name of the person performing the integration process
        </dc:creator>
        <dc:format>Feature original format.</dc:format>
    </FeatureInformation>
    <one:IntegratedCoast>
```

```
        <one:layerDescription>Coastline</one:layerDescription>
        <dnc:Feature>
            Integrated feature data
        </dnc:Feature>

    </one:IntegratedCoast>
</one:featureMember>
```

This is done in an iterative process. Figure B.3.2 show a piece of the Jamaica Bay file where a lazy integration process has been applied.

Chapter 6

Discussion and Conclusion

This chapter is a summarize this thesis, it give a discussion of the work done, the future work on the topic is described. At last a conclusion of the thesis is given.

6.1 Discussion

The result of this thesis is a theoretical description of geometrical and syntactical integration, together with the real life examples. This form a foundation for further study and development on the area.

There are several areas which could have been improved in this thesis. One of the major improvements would have been the connection between the geometrical and syntactical. They might seem to detached from each other. A improvement would have been that these processes were coupled, and described with a better relation to each other. In the geometrical integration process the cleaning preparation of a data set is thoroughly described, this should have been an implicit assumption that the data set were clean and correctly displayed. This would have given more time to study the problems around geometrical integration.

One of the problems which occurred in the research of the geometrical integration process was to find a proper tool. The JUMP [22] workbench was a superb tool, however the JCS [14] plugin which enabled modifications of line segments did not support splitting and cutting of line segments. This was solved with the help of the original JUMP crew, who were kind enough to send me the alpha version of JCS 1.1.0 which contained this feature. Through the writing of this thesis a lot of problems has occurred, but most of them did were solved one way or the other.

Future work on the area should include several of the tasks described below. An advantage which would confirm the processes defined in this thesis are real life test of the sketched problem solutions. Future work should test the described problem solutions on several real life data sets to guarantee the correctness.

The amount of time available when writing this thesis did not allow time to develop a geometrical and syntactical integration tool. This should be one of the main focuses of

future work on this area. A tool to ease the merging of data sets and add meta data would be a valuable resource to a map repository administrator. Such a tool would have the ability to do both the geometrical and syntactical integration process during and end up with one result, the integrated data correctly integrated both geometrically and syntactically. It would also be preferable to do develop a lazy integration tool, which gave the ability to easy develop lazy integration schemas which supported different sets of XML schemas [31].

An interesting extension to this thesis would be the study of other conflation methods with regards on integration of geometrical data into a map repository.

6.2 Conclusion

Throughout this thesis both the geometrical and syntactical integration of geospatial data have proven to be cumbersome problems. It has also been seen that such processes require human interaction to obtain a correct result. However in spite of these limitations the goal of this thesis has been achieved, to uncover the geometrical and syntactical integration problems and sketch a solution to them. The geometrical integration process has proven that its largest limitation is its requirement for human interaction. But if we account for that limitation, a geometrical integration is fully feasible. From a syntactical integration point of view the lazy integration is a powerful tool, which enable the creation of multi source GML [17] files. In both a geometrical and syntactical integration the peer-review process is seen as a valuable tool to ensure the correctness of the integration process, and the integrated data.

Bibliography

[1] British Columbia Ministry of Substainable Resource Management. URL: http://www.gov.bc.ca/bvprd/bc/channel.do?action=ministry&channelID=-8393&navId=NAV_ID_province.

[2] JUMP Pilot Project. URL: http://jump-pilot.sourceforge.net/index.php.

[3] Open Geospatial Consortium, Inc. (OGC). URL: http://www.opengis.org.

[4] Open JUMP. URL: http://www.openJUMP.org.

[5] Project SIGLE. URL: http://www.projet-sigle.org/.

[6] Refraction Research Inc. URL: http://www.refractions.net/.

[7] Understanding Map Projections. Technical report.

[8] Vector Smart Map Level 1. Technical Report MIL-PRF-89033.

[9] Vivid Solutions. URL: http://vividsolutions.com.

[10] Digital Chart of the World (DCW). Technical Report MIL-D-89009, April 1992.

[11] Vector Smart Map Level 0. Technical Report MIL-PRF-89039, 1995.

[12] Digital Nautical Chart. Technical Report MIL-PRF-89023, 1997.

[13] Open Memorandum Concerning NGA's Federal Register Announcement To Withdraw Aeronautical Products (Maps, Charts and Associated Data) from Public Access, January 2005.

[14] Dave Blasby, Martin Davis, Djun Kim, and Paul Ramsey. Gis conflation using open source tools.

[15] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Franïcis Yergeau. *Extensible Markup Language (XML)*. World Wide Web Consortium (W3C), February 2004.

[16] Ching-Chien Chen, Snehal Thakkar, Craig Knoblock, and Cyrus Shahabi. Automatically annotating and integrating spatial datasets. June 2003.

[17] Simon Cox, Adrian Cuthbert, Richard Martell, Paul Daisey, and Ron Lake. *OpenGIS Geography Markup Language Implementation Specification*. Open GIS Consortium, Inc., September 2002.

[18] Simon Cox, Paul Daisey, Ron Lake, Clemens Portele, and Arliss Whiteside. *OpenGIS Geography Markup Language Implementation Specification*. Open GIS Consortium, Inc., January 2003.

[19] Stanley L. Dallal. Automated conflation of florida state highway data with larger scale county data.

[20] David M. Danko. The Vector Product Format, An Overview. Technical report.

[21] Martin Davids and Jon Aquino. *JTS Topology Suite Technical Specifications*. The Jump Project, October 2003.

[22] Martin Davids and Jon Aquino. *JUMP the Unified Mapping Platform Workbench User's Guide*. The Jump Project, November 2003.

[23] Martin Davis. *JCS Conflation Suite Technical Report*. The Jump Project.

[24] Dublin Core Metadata Initiative. *Dublin Core XML Schemas*.

[25] David Duce, Ivan Herman, and Bob Hopgood. Svg tutorial. 2002.

[26] Environmental Systems Research Institute, Inc. (ESRI). *ESRI Shapefile Technical Description*, 1998.

[27] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha, editors. *The Java Language Specification*. Addison-Wesley Professional, second edition, 2000.

[28] Robert Heiztman. Building a manifold base map using tiger/line data.

[29] Diane Hillmann. *Using Dublin core*. Dublin Core Metadata Initiative.

[30] Bjørn Håkon Horpestad. Semantic Integration of Geospatial Data Using Feature Type Hierarchies. Master's thesis, Østfold University College, November 2005.

[31] Anthony M. Futrell Jr. The W3C XML Schema. 2001.

[32] Sindre Langaas. Completeness of the Digital Chart of the World (DCW) database. Technical Report 2/1995, 1995.

[33] Mats Lindh, Bjørn Håkon Horpestad, and Kristian Lunde. The Onemap Repository. 2005.

[34] Kalyani Mandapaka. A Simple API for XML. 2002.

[35] Dennis Milbert. A Tutorial On Datums, March 2005.

[36] Gunnar Misund. One Map. *HOIT*, January 2002.

[37] Gunnar Misund, Henning Kristiansen, and Mats Lindh. Distributed GML Management with SVG Tools. *HOIT*, 2002.

[38] Gunnar Misund and Harald Vålerhaugen. Integration of heterogeneous gml sources. 2004.

[39] Open GIS Consortium. *OpenGis Simple Features Specification for SQL*, May 1999.

[40] Alan Saalfeld. Conflation: Automated map compilation. *International Journal of Geographical Information Systems*, 1988.

[41] S. K. Upadhyaya, G. S. Pettygrove, J.W. Oliveira, and B. R. Jahn. An Introduction - Global Positioning System. Technical report.

[42] Shuxin Yuan and Chuang Tao. Development of conflation components. June 1999.

Appendix A

List of Terms

API - Application Programming Interface.

DNC - Digital Nautical Charts.

ECS - ESEA Conflation System.

GIS - Geographical Information System.

GML - Geography Markup Language.

GPS - Global Positioning System.

JAVA - Programming language from Sun Microsystems.

JCS - JCS Conflation Suite.

JTS - JTS Topology Suite.

JUMP - JUMP Unified Mapping Platform.

OGC - OpenGIS Consortium.

SFS - Simple Feature Specification.

SVG - Scalable Vector Graphics.

URL - Uniform Resource Locator.

VMAP - Vector Smart Map.

VPF - Vector Product Format.

WKT - Well-Known Text.

XML - eXtensible Markup Language.

XSD - XML Schema Definition.

Appendix B

Source code

B.1 JUMP Templates

B.1.1 GML Input Templates

Vmap Input Template

```
<?xml version="1.0" encoding="utf-8" ?> <JCSGMLInputTemplate>
  <CollectionElement>VMAPCollection</CollectionElement>
  <FeatureElement>VMAPFeature</FeatureElement>
  <GeometryElement>gml:lineStringProperty</GeometryElement>
  <ColumnDefinitions>
      <column>
        <name>faccDescription</name>
        <type>STRING</type>
      <valueelement elementname="faccDescription"/>
      <valuelocation position="body"/>
      </column>
  </ColumnDefinitions>
</JCSGMLInputTemplate>
```

DNC Input Template

```
<?xml version="1.0" encoding="utf-8" ?> <JCSGMLInputTemplate>
  <CollectionElement>FeatureCollection</CollectionElement>
  <FeatureElement>Feature</FeatureElement>
  <GeometryElement>gml:lineStringProperty</GeometryElement>
  <ColumnDefinitions>
      <column>
        <name>facc_description</name>
        <type>STRING</type>
      <valueelement elementname="facc_description"/>
      <valuelocation position="body"/>
```

```
        </column>
    </ColumnDefinitions>
</JCSGMLInputTemplate>
```

B.1.2 GML Output Templates

Vmap Output Template

```
<?xml version="1.0" encoding="UTF-8"?> <VMAPCollection
xmlns="no:hiof:onemap:gml:appschema:vmapschema"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="no:hiof:onemap:gml:appschema:vmapschema
http://www.ia-stud.hiof.no/~bjornhho/digmap/gmlEx/vmapschema.xsd">
<%FEATURE%>
<gml:featureMember>
  <VMAPFeature>
    <faccDescription>
        <%=COLUMN faccDescription%></faccDescription>
    <gml:lineStringProperty>
      <gml:LineString>
        <%=GEOMETRY%>
      </gml:LineString>
    </gml:lineStringProperty>
  </VMAPFeature>
</gml:featureMember>
<%ENDFEATURE%>
</VMAPCollection>
```

DNC Output Template

```
<?xml version="1.0" encoding="UTF-8"?> <FeatureCollection
xmlns="no:hiof:onemap:gml:appschema:dncschema"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="no:hiof:onemap:gml:appschema:newDNCschema
dncschema.xsd">
<%FEATURE%>
<gml:featureMember>
  <Feature>
    <facc_description>
        <%=COLUMN facc_description%></facc_description>
    <gml:lineStringProperty>
      <gml:LineString>
```

```
        <%=GEOMETRY%>
      </gml:LineString>
    </gml:lineStringProperty>
  </Feature>
</gml:featureMember>
<%ENDFEATURE%>
</FeatureCollection>
```

B.2 Lazy Integration Schemas

B.2.1 Request.xsd

```
<?xml version="1.0" encoding="UTF-8"?> <schema
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:one="http://onemap.org" targetNamespace="http://onemap.org"
elementFormDefault="qualified" version="0.9">
  <annotation>
    <appinfo>Map.xsd v0.9 2004-03</appinfo>
    <documentation xml:lang="en">
Top level schema for OneMap integrated featurecollections
</documentation>
  </annotation>
  <include schemaLocation="FeatureCollection.xsd"/>
  <include schemaLocation="River.xsd"/>
  <include schemaLocation="Coastline.xsd"/>
</schema>
```

B.2.2 FeatureCollection.xsd

```
<?xml version="1.0" encoding="UTF-8"?> <schema
xmlns:one="http://onemap.org"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://onemap.org"
elementFormDefault="qualified"
version="0.9">
  <annotation>
    <appinfo>Map.xsd v0.9 2004-03</appinfo>
    <documentation xml:lang="en">
Top level schema for OneMap integrated featurecollections
</documentation>
  </annotation>
```

```
  <!-- import constructs from the GML Feature and Geometry schemas -->
  <import namespace="http://www.opengis.net/gml"
          schemaLocation="./gml/feature.xsd"/>
  <import namespace="http://purl.org/dc/elements/1.1/"
          schemaLocation="simpledc20021212.xsd"/>
  <include schemaLocation="utils.xsd"/>
  <!-- ========================================================
global element declarations REMEMBER SUBSTITUTION GROUP ON MAP!
========================================================== -->
  <element name="FeatureCollection" type="one:FeatureCollectionType"
           substitutionGroup="gml:_FeatureCollection"/>
  <element name="featureMember" type="one:FeatureMemberType"
           substitutionGroup="gml:featureMember"/>
  <!-- ========================================================
type definitions for Map  model
========================================================== -->
  <complexType name="FeatureCollectionType">
    <complexContent>
      <extension base="one:AbstractFeatureCollectionBaseType">
        <sequence>
          <element ref="gml:boundedBy"/>
          <element ref="one:featureMember" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="FeatureMemberType">
    <complexContent>
      <extension base="one:FeatureAssociationBaseType">
        <sequence>
          <!--  Metadata information on a Feature -->
          <element name="FeatureInformation">
            <complexType>
              <sequence>
                <element ref="dc:identifier"/>
                <element ref="dc:title" minOccurs="0"/>
                <element ref="dc:date"/>
                <element ref="dc:creator"/>
                <element ref="dc:format"/>
                <element ref="dc:coverage" minOccurs="0"/>
                <element ref="dc:description" minOccurs="0"/>
              </sequence>
            </complexType>
          </element>
          <element ref="one:_AbstractIntegratedFeature"/>
        </sequence>
      </extension>
    </complexContent>
```

```
  </complexType>
</schema>
```

B.2.3 utils.xsd

```
<?xml version="1.0" encoding="UTF-8"?> <schema
xmlns:one="http://onemap.org"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://onemap.org" elementFormDefault="qualified"
version="0.9">
  <annotation>
    <documentation xml:lang="en">
Top level schema for OneMap integrated featurecollections
</documentation>
  </annotation>
  <!-- import constructs from the GML Feature and Geometry schemas -->
  <import namespace="http://www.opengis.net/gml"
          schemaLocation="./gml/feature.xsd"/>
  <import namespace="http://purl.org/dc/elements/1.1/"
          schemaLocation="simpledc20021212.xsd"/>
  <!-- ========================================================
global element declarations
========================================================= -->
  <element name="AbstractFeatureCollectionBase"
           type="one:AbstractFeatureCollectionBaseType"/>
  <element name="FeatureAssociationBase"
           type="one:FeatureAssociationBaseType"/>
  <element name="_AbstractIntegratedFeature"
           type="one:AbstractIntegratedFeatureType"
           abstract="true" substitutionGroup="gml:_Feature"/>
  <element name="_abstractFeatureFragment"
           type="gml:FeatureAssociationType"
           abstract="true" substitutionGroup="gml:featureMember"/>
  <!-- ========================================================
type definitions for Map  model
========================================================= -->
<complexType name="AbstractFeatureCollectionBaseType">
  <complexContent>
    <restriction base="gml:AbstractFeatureCollectionType">
      <sequence>
        <element ref="gml:description" minOccurs="0"/>
        <element ref="gml:name" minOccurs="0"/>
        <!-- the first two elements are used by the
```

```
        gml2svg-transformation, to turn on/off visibility of layers -->
        <element name="layerDescription" minOccurs="0">
          <simpleType>
            <restriction base="xs:string">
              <pattern value="[a-bA-B]{5-20}"/>
            </restriction>
          </simpleType>
        </element>
      </sequence>
      <attribute name="fid" type="ID" use="optional"/>
    </restriction>
  </complexContent>
</complexType>

  <complexType name="FeatureAssociationBaseType">
    <complexContent>
      <restriction base="gml:FeatureAssociationType">
        <attributeGroup ref="xlink:simpleLink"/>
        <attribute ref="gml:remoteSchema" use="optional"/>
      </restriction>
    </complexContent>
  </complexType>

  <complexType name="AbstractIntegratedFeatureType">
    <complexContent>
      <extension base="one:AbstractFeatureCollectionBaseType">
        <sequence>
          <!--  Metadata information on feature fragment level-->
          <element name="FeatureFragmentInformation" minOccurs="0">
            <complexType>
              <sequence>
                <element ref="dc:identifier"/>
                <element ref="dc:title" minOccurs="0"/>
                <element ref="dc:date"/>
                <element ref="dc:creator"/>
                <element ref="dc:format"/>
                <element ref="dc:coverage" minOccurs="0"/>
                <element ref="dc:description" minOccurs="0"/>
              </sequence>
            </complexType>
          </element>
          <element ref="one:_abstractFeatureFragment"
                   maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

B.2.4 Coastline.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
xmlns:onen="http://www.onemap.net"
 xmlns:one="http://onemap.org"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:vmap="http://www.onemap.net/vmap"
 xmlns:dnc="http://www.onemap.net/dnc"
 xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://onemap.org"
 elementFormDefault="qualified" version="0.9">
    <annotation>
        <appinfo>Map.xsd v0.9 2004-03</appinfo>
        <documentation xml:lang="en">

        </documentation>
    </annotation>
    <!-- import constructs from the GML Feature and Geometry schemas -->
    <include schemaLocation="utils.xsd"/>
    <!--<import namespace="http://www.opengis.net/examples"
                schemaLocation="./GML2specex/city.xsd"/>-->
    <!-- <import namespace="http://www.onemap.net"
                schemaLocation="../onemap/get_feature.xsd"/> -->
    <import namespace="http://www.onemap.net"
            schemaLocation="onemap/vmap.xsd"/>
    <import namespace="http://www.onemap.net"
            schemaLocation="onemap/dnc.xsd"/>
    <!-- ========================================================
global element declarations
========================================================= -->
    <element name="IntegratedCoast" type="one:IntegratedCoastLineType"
             substitutionGroup="one:_AbstractIntegratedFeature"/>
    <element name="coastFragment" type="one:CoastFragmentType"
             substitutionGroup="one:_abstractFeatureFragment"/>
    <!-- ========================================================
type definitions for Map  model
========================================================= -->
    <complexType name="IntegratedCoastLineType">
        <complexContent>
            <extension base="one:AbstractFeatureCollectionBaseType">
                <sequence>
                    <element ref="one:coastFragment"
                             maxOccurs="unbounded"/>
                    <element ref="vmap:VMAPFeature"
                             maxOccurs="unbounded"/>
                    <element ref="dnc:Feature"
```

```
                                maxOccurs="unbounded"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="CoastFragmentType">
        <complexContent>
            <extension base="one:FeatureAssociationBaseType">
                <choice>
                    <element ref="onen:ResponseFeature"/>
                </choice>
            </extension>
        </complexContent>
    </complexType>
</schema>
```

B.2.5 river.xsd

```
<?xml version="1.0" encoding="UTF-8"?> <schema
targetNamespace="http://onemap.org"
 xmlns:one="http://onemap.org"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:ex="http://www.opengis.net/examples"
 xmlns="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" version="0.9">
    <annotation>
        <appinfo>River.xsd</appinfo>
        <documentation xml:lang="en">
        Schema for integrating rivers from heterogeneous GML documents.
        </documentation>
    </annotation>
    <!-- import constructs from the GML Feature and Geometry schemas -->
    <include schemaLocation="utils.xsd"/>
    <!-- ========================================================
global element declarations
========================================================= -->
    <element name="IntegratedRiver" type="one:IntegratedRiverType"
             substitutionGroup="one:_AbstractIntegratedFeature"/>
    <element name="riverFragment" type="one:RiverFragmenType"
             substitutionGroup="one:_abstractFeatureFragment"/>
    <!-- ========================================================
type definitions for Map  model
========================================================= -->
    <complexType name="IntegratedRiverType">
        <complexContent>
            <extension base="one:AbstractFeatureCollectionBaseType">
```

```
                <sequence>
                    <element ref="one:riverFragment"
                             maxOccurs="unbounded"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="RiverFragmenType">
        <complexContent>
            <extension base="one:FeatureAssociationBaseType">
                <choice>
                    <element ref="ex:River"/>
                    <element ref="osgb:BoundaryLine"/>
                </choice>
            </extension>
        </complexContent>
    </complexType>
</schema>
```

B.2.6 vmap.xsd

Even though the vmap schema is an external schema, it is included there. Hopefully it will give the reader a more complete overview of how the lazy integration use external schemas.

```
<?xml version="1.0" encoding="UTF-8"?> <xs:schema
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.onemap.net"
targetNamespace="http://www.onemap.net"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace="http://www.opengis.net/gml"
             schemaLocation="../gml/feature.xsd"/>
  <!--Global declarations -->
  <xs:element name="VMAPCollection" type="VMAPCollectionType"/>
  <xs:element name="vmapMember" type="VMAPFeatureType"
              substitutionGroup="gml:featureMember"/>
  <xs:element name="VMAPFeature" type="VMAPFeatureType"
              substitutionGroup="_VMAPFeature"/>
  <xs:element name="_VMAPFeature" type="gml:AbstractFeatureType"
              abstract="true" substitutionGroup="gml:_Feature"/>
  <!--Definition -->
  <xs:complexType name="VMAPCollectionType">
    <xs:complexContent>
      <xs:extension base="gml:AbstractFeatureCollectionType">
        <xs:attribute name="lastupdated"
                      type="xs:dateTime" use="optional"/>
      </xs:extension>
```

```
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="VMAPFeatureMemberType">
  <xs:complexContent>
    <xs:restriction base="gml:FeatureAssociationType">
      <xs:sequence minOccurs="0">
        <xs:element ref="_VMAPFeature"/>
      </xs:sequence>
      <xs:attributeGroup ref="xlink:simpleLink"/>
      <xs:attribute ref="gml:remoteSchema" use="optional"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="VMAPFeatureType">
  <xs:complexContent>
    <xs:extension base="gml:AbstractFeatureType">
      <xs:sequence>
        <xs:element name="facc_description"
                    type="xs:string" minOccurs="0"/>
        <xs:element name="name" type="xs:string"
                    minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="highest_z-value"
                    type="xs:int" minOccurs="0"/>
        <xs:element name="hydrological_category"
                    type="xs:int" minOccurs="0"/>
        <xs:element name="usage" type="xs:string"
                                 minOccurs="0"/>
        <xs:element name="existence_category"
                    type="xs:int" minOccurs="0"/>
        <xs:element name="route_intended_use"
                    type="xs:int" minOccurs="0"/>
        <xs:choice>
          <xs:element ref="gml:Polygon" minOccurs="0"/>
          <xs:element ref="gml:Point" minOccurs="0"/>
          <xs:element ref="gml:LineString" minOccurs="0"/>
        </xs:choice>
        <!--Facc description is the description
            belonging to a facc_code. -->
        <!--Elevation -->
        <!--Hydrography -->
        <!--Transportation -->
        <!--Airport usage, example military. -->
        <!--Road and Railroad existence categories,
            wether they are in use or not. -->
      </xs:sequence>
    </xs:extension>
```

```
    </xs:complexContent>
  </xs:complexType>

</xs:schema>
```

B.2.7 dnc.xsd

The DNC [12] schema is also a external schema as the vmap schema is.

```
<?xml version="1.0" encoding="UTF-8"?> <xs:schema
xmlns="http://www.onemap.net"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink"
targetNamespace="http://www.onemap.net"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace="http://www.opengis.net/gml"
             schemaLocation="../gml/feature.xsd"/>
  <xs:element name="FeatureCollection"
              type="FeatureCollectionType"/>
  <xs:element name="Feature" type="FeatureType"
              substitutionGroup="gml:_Feature"/>
  <xs:complexType name="FeatureCollectionType">
    <xs:complexContent>
      <xs:extension base="gml:AbstractFeatureCollectionType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="FeatureType">
    <xs:complexContent>
      <xs:extension base="gml:AbstractFeatureType">
        <xs:sequence>
          <xs:choice>
            <xs:element ref="gml:Point"/>
            <xs:element ref="gml:Polygon"/>
            <xs:element ref="gml:LineString"/>
          </xs:choice>
          <!-- General.-->
          <xs:element name="facc_description"
                      type="xs:string" minOccurs="0"/>
          <xs:element name="hydrological_category"
                      type="xs:integer" minOccurs="0"/>
          <!-- Rivers -->
          <xs:element name="accuracy_category" minOccurs="0">
            <xs:simpleType>
              <xs:restriction base="xs:integer">
                <xs:minExclusive value="0"/>
                <xs:maxExclusive value="3"/>
              </xs:restriction>
```

```
            </xs:simpleType>
          </xs:element>
          <xs:element name="depth_curve_or_value_high"
                      type="xs:integer" minOccurs="0"/>
          <xs:element name="depth_curve_or_contour_value_low"
                      type="xs:integer" minOccurs="0"/>
          <!-- Lakes -->
          <xs:element name="associated_hydrographic_category"
                      type="xs:integer" minOccurs="0"/>
          <xs:element name="shoreline_type_category"
                      type="xs:integer" minOccurs="0"/>
          <xs:element name="required_port_access"
                      type="xs:integer" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

B.3 Lazy Integration Result Data

In the sections below result data from lazy integration processes are illustrated. The result data are generated from the scenarios which have been used in this thesis.

B.3.1 La Guardia Airport

Below is a section of the La Guardia Airport data seen after a lazy integration has been applied to it.

```
<?xml version="1.0" encoding="UTF-8"?> <one:FeatureCollection
xmlns:one="http://onemap.org" xmlns:onen="http://www.onemap.net"
xmlns:gml="http://www.opengis.net/gml"
xmlns:vmap="http://www.onemap.net/vmap"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.gisline.no"
xsi:schemaLocation="http://www.onemap.org Request.xsd"
fid="ID000004">

<!-- original VMAP0 feature --> <gml:featureMember>
    <vmap:VMAPFeature>
        <faccDescription>Coastline/Shoreline</faccDescription>
        <gml:lineStringProperty>
            <gml:LineString>
                        <gml:MultiLineString>
                    <gml:lineStringMember>
                    <gml:LineString>
```

```
                        <gml:coordinates>
                             -64.7100067139,32.3751564026
                           -64.7094039917,32.3797187805
                           -64.7069244385,32.3833503723
                           -64.6998443604,32.3873977661
                           -64.6890029907,32.3893470764
                           -64.6802902222,32.3938331604
                           -64.6738510132,32.3901138306
                           -64.6760864258,32.3851127625
                           -64.6872024536,32.3824768066
                           -64.6939544678,32.383228302
                           -64.6947937012,32.3807220459
                           -64.6926727295,32.3766021729
                           -64.7030563354,32.3750915527
                           -64.7116241455,32.365146637
                           -64.7132797241,32.3657875061
                           -64.7100067139,32.3751564026
                        </gml:coordinates>
                      </gml:LineString>
                      </gml:lineStringMember>
                      <gml:lineStringMember>
                      <gml:LineString>
                        <gml:coordinates>
                             -64.7100067139,32.3751564026
                           -64.7094039917,32.3797187805
                           -64.7069244385,32.3833503723
                           -64.6998443604,32.3873977661
                           -64.6890029907,32.3893470764
                           -64.6802902222,32.3938331604
                           -64.6738510132,32.3901138306
                           -64.6760864258,32.3851127625
                           -64.6872024536,32.3824768066
                           -64.6939544678,32.383228302
                           -64.6947937012,32.3807220459
                           -64.6926727295,32.3766021729
                           -64.7030563354,32.3750915527
                           -64.7116241455,32.365146637
                           -64.7132797241,32.3657875061
                           -64.7100067139,32.3751564026
                        </gml:coordinates>
                      </gml:LineString>
                      </gml:lineStringMember>
                    </gml:MultiLineString>

              </gml:LineString>
          </gml:lineStringProperty>
      </vmap:VMAPFeature>
</gml:featureMember>
```

```
<!-- Integrated VMAP1 feature --> <one:featureMember>
    <FeatureInformation>
            <dc:identifier>onemap_30213</dc:identifier>
            <dc:title>Coastline</dc:title>
            <dc:date>2005-10-16</dc:date>
            <dc:creator>Kristian Lunde</dc:creator>
            <dc:format>VMAP1</dc:format>
    </FeatureInformation>

    <one:IntegratedCoast>
            <one:layerDescription>Coastline</one:layerDescription>
            <vmap:VMAPFeature>
                <faccDescription>Coastline/Shoreline</faccDescription>
                <gml:lineStringProperty>
                    <gml:LineString>
                                        <gml:MultiLineString>
                            <gml:lineStringMember>
                            <gml:LineString>
                              <gml:coordinates>
                                 -73.8968658447,40.7962417603
                                 -73.8972244263,40.7967185974
                                 -73.8981781006,40.7966651917
                                 -73.898147583,40.7962608337
                                 -73.8977355957,40.7957649231
                                 -73.8971252441,40.7957305908
                                 -73.8968658447,40.7962417603
                              </gml:coordinates>
                            </gml:LineString>
                            </gml:lineStringMember>
                            <gml:lineStringMember>
                            <gml:LineString>
                              <gml:coordinates>
                                 -73.8968658447,40.7962417603
                                 -73.8972244263,40.7967185974
                                 -73.8981781006,40.7966651917
                                 -73.898147583,40.7962608337
                                 -73.8977355957,40.7957649231
                                 -73.8971252441,40.7957305908
                                 -73.8968658447,40.7962417603
                              </gml:coordinates>
                            </gml:LineString>
                            </gml:lineStringMember>
                            <gml:lineStringMember>
                            <gml:LineString>
                              <gml:coordinates>
                                 -73.8968658447,40.7962417603
                                 -73.8972244263,40.7967185974
```

```
                                    -73.8981781006,40.7966651917
                                    -73.898147583,40.7962608337
                                    -73.8977355957,40.7957649231
                                    -73.8971252441,40.7957305908
                                    -73.8968658447,40.7962417603
                                </gml:coordinates>
                            </gml:LineString>
                            </gml:lineStringMember>
                        </gml:MultiLineString>
                    </gml:LineString>
                </gml:lineStringProperty>
            </vmap:VMAPFeature>
        </one:IntegratedFeature>
</one:featureMember>

: : : </one:FeatureCollection>
```

B.3.2 The Jamaica Bay Scenario

The result of a lazy integration process applied on the Jamaica Bay scenario is seen below.

```
<?xml version="1.0" encoding="UTF-8"?> <one:FeatureCollection
xmlns:one="http://onemap.org" xmlns:onen="http://www.onemap.net"
xmlns:gml="http://www.opengis.net/gml"
xmlns:vmap="http://www.onemap.net/vmap"
xmlns:dnc="http://www.onemap.net/dnc"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.gisline.no"
xsi:schemaLocation="http://www.onemap.org Request.xsd"
fid="ID000004"> : : <gml:featureMember>
    <vmap:VMAPFeature>
        <faccDescription>Coastline/Shoreline</faccDescription>
        <gml:lineStringProperty>
            <gml:LineString>
                <gml:MultiLineString>
                  <gml:lineStringMember>
                  <gml:LineString>
                    <gml:coordinates>
                        -73.7175979614,40.585521698
                      -73.7107696533,40.5842208862
                      -73.7011947632,40.5831871033
                      -73.6952896118,40.5835456848
                      -73.6881027222,40.5829353333
                      -73.6842956543,40.5833702087
                      -73.677444458,40.5832633972
                      -73.6642074585,40.5831489563
                      -73.6579666138,40.5827484131
```

```
                        -73.6532516479,40.5829124451
                        -73.6480789185,40.5828132629
                        -73.6415252686,40.583278656
                        -73.6334075928,40.5844688416
                    </gml:coordinates>
                </gml:LineString>
                </gml:lineStringMember>
                <gml:lineStringMember>
                <gml:LineString>
                    <gml:coordinates>
                         -73.7175979614,40.585521698
                        -73.7107696533,40.5842208862
                        -73.7011947632,40.5831871033
                        -73.6952896118,40.5835456848
                        -73.6881027222,40.5829353333
                        -73.6842956543,40.5833702087
                        -73.677444458,40.5832633972
                        -73.6642074585,40.5831489563
                        -73.6579666138,40.5827484131
                        -73.6532516479,40.5829124451
                        -73.6480789185,40.5828132629
                        -73.6415252686,40.583278656
                        -73.6334075928,40.5844688416
                    </gml:coordinates>
                </gml:LineString>
                </gml:lineStringMember>
            </gml:MultiLineString>

        </gml:LineString>
    </gml:lineStringProperty>
</vmap:VMAPFeature>
</gml:featureMember> <one:featureMember>
<FeatureInformation>
        <dc:identifier>onemap_10102</dc:identifier>
        <dc:title>Coastline</dc:title>
        <dc:date>2005-10-16</dc:date>
        <dc:creator>Kristian Lunde</dc:creator>
        <dc:format>DNC harbor 960</dc:format>
</FeatureInformation>
<one:IntegratedCoast>
        <one:layerDescription>Coastline</one:layerDescription>
        <dnc:Feature>
            <faccDescription></faccDescription>
            <gml:lineStringProperty>
                <gml:LineString>
                            <gml:MultiLineString>
                        <gml:lineStringMember>
                        <gml:LineString>
```

```
                                <gml:coordinates>
                                    -73.8879013062,40.578163147
                                  -73.8877334595,40.5781784058
                                  -73.887512207,40.5782012939
                                  -73.8870773315,40.5782852173
                                  -73.8864898682,40.5783233643
                                </gml:coordinates>
                              </gml:LineString>
                              </gml:lineStringMember>
                              <gml:lineStringMember>
                              <gml:LineString>
                                <gml:coordinates>
                                    -73.8879013062,40.578163147
                                  -73.8877334595,40.5781784058
                                  -73.887512207,40.5782012939
                                  -73.8870773315,40.5782852173
                                  -73.8864898682,40.5783233643
                                </gml:coordinates>
                              </gml:LineString>
                              </gml:lineStringMember>
                            </gml:MultiLineString>

                        </gml:LineString>
                    </gml:lineStringProperty>
                </dnc:Feature>
            </one:IntegatedCoast>
</one:featureMember> <one:featureMember>
    <FeatureInformation>
            <dc:identifier>onemap_10103</dc:identifier>
            <dc:title>Coastline</dc:title>
            <dc:date>2005-10-16</dc:date>
            <dc:creator>Kristian Lunde</dc:creator>
            <dc:format>DNC harbor 960</dc:format>
    </FeatureInformation>
    <one:IntegratedCoast>
            <one:layerDescription>Coastline</one:layerDescription>
            <dnc:Feature>
                <faccDescription></faccDescription>
                <gml:lineStringProperty>
                    <gml:LineString>
                                        <gml:MultiLineString>
                              <gml:lineStringMember>
                              <gml:LineString>
                                <gml:coordinates>
                                    -73.8032531738,40.6142539978
                                  -73.8031082153,40.6142845154
                                  -73.8029632568,40.6144447327
                                  -73.8028411865,40.614692688
```

```
                -73.8027496338,40.6149635315
                -73.8025360107,40.6153182983
                -73.8023223877,40.6155319214
                -73.802154541,40.6158180237
                -73.8018035889,40.6165351868
                -73.8016357422,40.6168899536
                -73.801651001,40.6171417236
                -73.8015594482,40.6172904968
                -73.8015670776,40.6174926758
                -73.8014907837,40.6176490784
                -73.8013916016,40.6178588867
                -73.8013381958,40.6180343628
                -73.8013458252,40.6184043884
                -73.8014297485,40.6186790466
                -73.8015289307,40.6190299988
                -73.8015899658,40.6193847656
                -73.8016052246,40.6194381714
              </gml:coordinates>
            </gml:LineString>
            </gml:lineStringMember>
            <gml:lineStringMember>
            <gml:LineString>
              <gml:coordinates>
                  -73.8032531738,40.6142539978
                -73.8031082153,40.6142845154
                -73.8029632568,40.6144447327
                -73.8028411865,40.614692688
                -73.8027496338,40.6149635315
                -73.8025360107,40.6153182983
                -73.8023223877,40.6155319214
                -73.802154541,40.6158180237
                -73.8018035889,40.6165351868
                -73.8016357422,40.6168899536
                -73.801651001,40.6171417236
                -73.8015594482,40.6172904968
                -73.8015670776,40.6174926758
                -73.8014907837,40.6176490784
                -73.8013916016,40.6178588867
                -73.8013381958,40.6180343628
                -73.8013458252,40.6184043884
                -73.8014297485,40.6186790466
                -73.8015289307,40.6190299988
                -73.8015899658,40.6193847656
                -73.8016052246,40.6194381714
              </gml:coordinates>
            </gml:LineString>
            </gml:lineStringMember>
          </gml:MultiLineString>
```

```
                        </gml:LineString>
                    </gml:lineStringProperty>
                </dnc:Feature>
            </one:IntegratedCoast>
</one:featureMember>
```