HOKKAIDO UNIVERSITY

Title	A New Interactive Visualization Framework for Defining Both Standard Charts and Nonstandard Charts Based on Two Tree-structured Schemata
Author(s)	石, 偉
Degree Grantor	北海道大学
Degree Name	博士(情報科学)
Dissertation Number	甲第11748号
Issue Date	2015-03-25
DOI	https://doi.org/10.14943/doctoral.k11748
Doc URL	https://hdl.handle.net/2115/58734
Type	doctoral thesis
File Information	Wei_Shi.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

A New Interactive Visualization Framework for Defining Both Standard Charts and Nonstandard Charts Based on Two Tree-structured Schemata

A thesis submitted for the degree of Doctor of Information Science to the Graduate School of Information Science and Technology of Hokkaido University

March 2015

Wei Shi

Division of Computer Science

Preface

Information visualization is a widely-used technology which supports users to represent data by the visual properties of a set of graphical objects in a visualization chart[23][28]. Normally, people are more sensitive to visual properties (such as colour and size) than to abstract data. By observing a visualization chart or a set of related visualization charts, users may obtain the information which may be difficult to obtain immediately from observing the original data. Especially people can easily obtain change trends of data, relationships between data items, and comparison results among data. A well-designed charts can also reveal information from the original data. Some visualization in special fields can also provide more information than the original data. For example, we can visualize the coordinate values of some places to define a GeoVisualization chart, and use a map as its background. Then from this GeoVisualization chart, users can know not only the positions of these places, but also the surroundings of these places.

In recent years, data measuring and data gathering technologies have made a great progress. This brings not only the rapid increase of gathered data, but also the growing complexity of the data sets. Multivariate data sets and large scale data sets are becoming popular. This situation increases the importance of visualization technology.

Different kinds of visualization charts have different effects. According to the features of data sets and the purposes of chart creators, creators may use various visualization charts. Even for the same data set, differently designed charts may reveal different aspects of the original data. In our paper, the design of how to represent data using graphical objects and their visual properties is called a *visualization type*.

According to the visualization types of visualization charts, we classified all charts into two categories, i.e., standard charts and nonstandard charts. A basic standard chart denotes such a chart composed using four types of basic graphical objects, i.e., "Point", "Line", "Area" and "Interval". Leland Wilkinson introduced them in Table 8.3 of this book "The Grammar of Graphics"[30]. A standard chart

is either a basic standard chart, or a chart composed with more than one standard chart by geometrically arranging them or embedding some of them into another in a nested way. A chart that is not a standard chart as defined above is a nonstandard chart.

One important research topic on information visualization is to propose visualization frameworks or tools for guiding users to create standard charts and/or nonstandard charts. By the visualization systems proposed in the existing visualization researches, users are supported to create charts either by programming or by declaratively specifying charts.

Creating a chart by programming method enables users to create any charts as their desires. But this method requests users' programming skills. This is very difficult for programming novices. Even for the users who have rich programming experiences, they often need to learn new programming languages provided by these visualization systems. Such a learning process is repetitive and time-consuming.

Another kind of visualization systems enables users to declaratively define charts by their logical specifications. By using these visualization systems to define a chart, users need to declare what kind of graphical objects are included in this chart, and to declaratively define the relationships between visual properties of each graphical object and the attributes of the data set. In our framework, we called each graphical object in a chart, which is used to represent data, a *visual object*. Even novice users can easily learn how to use these systems.

However, these systems still have the following three major problems:

1. The power of such a system depends on the size of its library of registered chart defining functions. The developers of such systems need to frequently update this library to satisfy users' requirements. This is a tedious, expensive and never ending endeavour, and user requirements cannot be satisfied immediately.

2. The system developers determine how to parameterize the graphical objects in the system library. Users have to read the instructions in detail before using these systems.

3. Selecting an appropriate graphical object from the library is not easy. Two or more pre-defined graphical objects provided by the system may look similar, because they may use the same geometrical shape as their appearances.

Here we propose a new generic visualization framework based on two tree-structured schemata. It enables users to declaratively define both standard charts

and nonstandard charts without using any pre-defined non-primitive templates. In a visualization chart, visual objects are copies of one or more graphical objects with some modification of some visual properties according to the data. These graphical objects are called *visualization templates* of this chart.

Only by directly and visually manipulating these two tree-structured schemata, users can flexibly define the mapping from data to its visual representation. Tree-structures are widely used in different research fields, so most users can quickly understand and start to use them. Tree structures also can clearly represent the relationships among the attributes of data sets, and the relationships among the different components of a chart as introduced in Section 2.1.

In our framework, the two kinds of tree-structured schemata are respectively named as the data view schema (DVS) and the chart view schema (CVS). They have five main functions:

1. by manipulating DVS, users can define an object-oriented view,
2. generating an object-oriented query for generating data objects from the data retrieved from relational databases,
3. defining how to instantiate the visualization templates for generating visual objects using generated data objects,
4. customizing the background, the legend, and the coordinate system of a visualization chart, and
5. defining composite charts.

The data objects generated in our framework should have the same structure as the visualization template. The system can instantiate the visualization template by these data objects without any further data conversion. Our framework represents each visual property of any graphical object used in a visualization chart as the node of a tree structure that logically represents this chart. This tree structure ensures that users can clearly and easily customize each graphical object, use the graphical objects to define the chart components, and compose a visualization chart using these chart components. By linking some nodes of DVS to nodes of CVS, users can define the visual mapping from data to its visual representation.

This framework provides a template composition method. We developed two function objects. By using these two objects, users can compose a complex visualization template by primitive graphical objects, such as rectangle object and line

object. Users can combine existing graphical objects and define the geometric relationship among the graphical objects using our method without any programming effort.

We also discuss how to apply our framework to support users to define GeoVisualization charts. GeoVisualization charts are a kind of visualization charts which represent the data including location information in each data item. Such kind of charts usually has maps as their backgrounds, and the position of each visual object in these charts is determined by the location information of each data item. Our framework provides four convenient functions for supporting users to rapidly define such GeoVisualization charts. Our framework also wraps the function of ArcGIS[3] system.

The charts defined by our framework are interactive. Besides the common chart interaction operations, our framework also provides five chart modification interactions. These five operations enable users to indirectly modify the two kinds of schemata which defined the chart by directly manipulating the chart.

In this thesis, we have also shown an extended framework. Users can use it to visualize data retrieved from Semantic Web. It aims to convert the "web of documents" to the "web of data". The rapid development of Semantic Web makes it become an important data source. Data in a local database is finite, while data on the Web are infinite. Although the data in Semantic Web are freely accessible, they are still a good complement of the local databases. However, the data in Semantic Web are represented as triples, which is different from the data representation in the relational database. Users are difficult to retrieve desired data from the Semantic Web without any background knowledge. To solve this problem, our framework reuses the exploratory semantic web search method proposed by Piao [16]. Our framework can represent the search result provided by Piao's framework in a Data View Schemata. Then users can easily visualize the retrieved data from Semantic Web, or visualize the integrated data obtained from both local database and Semantic Web.

Acknowledgments During my work of this thesis, I have been encouraged and guided by many people. I wish to express my appreciation to them.

I would like to thank my supervisor Prof. Yuzuru Tanaka for his guidance, encouragement, tolerance and his friendship. He did not only provide guidance, but also taught me how to perform research and to put everything into a wide range of vision. He also gave me the encouragement to carry out this work every time I was

discouraged. I am grateful to him for the opportunity to work on this research.

Thanks to my colleagues in the Knowledge Media Research Group for their encouragement, discussions and friendship, especially to Hajime Imura and Bin Piao for being good friends and senior associate to me. I was also encouraged and supported by many people.

Finally, special thanks go to my parents for their long patience and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Information visualization is a widely-used technology which supports users to represent data by the visual properties of a set of graphical objects in a visualization chart[23][28]. Normally, people are more sensitive to visual properties (such as colour and size) than to abstract data. By observing a visualization chart or a set of related visualization charts, users may obtain the information which may be difficult to obtain immediately from observing the original data. Especially people can easily obtain change trends of data, relationships between data items, and comparison results among data. A well-designed charts can also reveal information from the original data. Some visualization in special fields can also provide more information than the original data. For example, we can visualize the coordinate values of some places to define a GeoVisualization chart, and use a map as its background. Then from this GeoVisualization chart, users can know not only the positions of these places, but also the surroundings of these places.

In recent years, data measuring and data gathering technologies have made a great progress. This brings not only the rapid increase of gathered data, but also the growing complexity of the data sets. Multivariate data sets and large scale data sets are becoming popular. This situation increases the importance of visualization technology.

Different kinds of visualization charts have different effects. According to the features of data sets and the purposes of chart creators, creators may use various visualization charts. Even for the same data set, differently designed charts may reveal different aspects of the original data. In our paper, the design of how to represent data using graphical objects and their visual properties is called a *visualization type*.

According to the visualization types of visualization charts, we classified all charts into two categories, i.e., standard charts and nonstandard charts. A basic

standard chart denotes such a chart composed using four types of basic graphical objects, i.e., "Point", "Line", "Area" and "Interval". Leland Wilkinson introduced them in Table 8.3 of this book "The Grammar of Graphics"[30]. A standard chart is either a basic standard chart, or a chart composed with more than one standard chart by geometrically arranging them or embedding some of them into another in a nested way. A chart that is not a standard chart as defined above is a nonstandard chart.

One important research topic on information visualization is to propose visualization frameworks or tools for guiding users to create standard charts and/or nonstandard charts. By the visualization systems proposed in the existing visualization researches, users are supported to create charts either by programming or by declaratively specifying charts.

Creating a chart by programming method enables users to create any charts as their desires. But this method requests users' programming skills. This is very difficult for programming novices. Even for the users who have rich programming experiences, they often need to learn new programming languages provided by these visualization systems. Such a learning process is repetitive and time-consuming.

Another kind of visualization systems enables users to declaratively define charts by their logical specifications. By using these visualization systems to define a chart, users need to declare what kind of graphical objects are included in this chart, and to declaratively define the relationships between visual properties of each graphical object and the attributes of the data set. In our framework, we called each graphical object in a chart, which is used to represent data, a *visual object*. Even novice users can easily learn how to use these systems.

However, these systems still have the following three major problems:

1. The power of such a system depends on the size of its library of registered chart defining functions. The developers of such systems need to frequently update this library to satisfy users' requirements. This is a tedious, expensive and never ending endeavour, and user requirements cannot be satisfied immediately.

2. The system developers determine how to parameterize the graphical objects in the system library. Users have to read the instructions in detail before using these systems.

3. Selecting an appropriate graphical object from the library is not easy. Two or more pre-defined graphical objects provided by the system may look similar,

because they may use the same geometrical shape as their appearances.

Here we propose a new generic visualization framework based on two tree-structured schemata. It enables users to declaratively define both standard charts and nonstandard charts without using any pre-defined non-primitive templates. In a visualization chart, visual objects are copies of one or more graphical objects with some modification of some visual properties according to the data. These graphical objects are called *visualization templates* of this chart.

Only by directly and visually manipulating these two tree-structured schemata, users can flexibly define the mapping from data to its visual representation. Tree-structures are widely used in different research fields, so most users can quickly understand and start to use them. Tree structures also can clearly represent the relationships among the attributes of data sets, and the relationships among the different components of a chart as introduced in Section 2.1.

In our framework, the two kinds of tree-structured schemata are respectively named as the data view schema (DVS) and the chart view schema (CVS). They have five main functions:

1. by manipulating DVS, users can define an object-oriented view,
2. generating an object-oriented query for generating data objects from the data retrieved from relational databases,
3. defining how to instantiate the visualization templates for generating visual objects using generated data objects,
4. customizing the background, the legend, and the coordinate system of a visualization chart, and
5. defining composite charts.

The data objects generated in our framework should have the same structure as the visualization template. The system can instantiate the visualization template by these data objects without any further data conversion. Our framework represents each visual property of any graphical object used in a visualization chart as the node of a tree structure that logically represents this chart. This tree structure ensures that users can clearly and easily customize each graphical object, use the graphical objects to define the chart components, and compose a visualization chart using these chart components. By linking some nodes of DVS to nodes of CVS, users can define the visual mapping from data to its visual representation.

This framework provides a template composition method. We developed two function objects. By using these two objects, users can compose a complex visualization template by primitive graphical objects, such as rectangle object and line object. Users can combine existing graphical objects and define the geometric relationship among the graphical objects using our method without any programming effort.

We also discuss how to apply our framework to support users to define GeoVisualization charts. GeoVisualization charts are a kind of visualization charts which represent the data including location information in each data item. Such kind of charts usually has maps as their backgrounds, and the position of each visual object in these charts is determined by the location information of each data item. Our framework provides four convenient functions for supporting users to rapidly define such GeoVisualization charts. Our framework also wraps the function of ArcGIS[3] system.

The charts defined by our framework are interactive. Besides the common chart interaction operations, our framework also provides five chart modification interactions. These five operations enable users to indirectly modify the two kinds of schemata which defined the chart by directly manipulating the chart.

In this thesis, we have also shown an extended framework. Users can use it to visualize data retrieved from Semantic Web. It aims to convert the "web of documents" to the "web of data". The rapid development of Semantic Web makes it become an important data source. Data in a local database is finite, while data on the Web are infinite. Although the data in Semantic Web are freely accessible, they are still a good complement of the local databases. However, the data in Semantic Web are represented as triples, which is different from the data representation in the relational database. Users are difficult to retrieve desired data from the Semantic Web without any background knowledge. To solve this problem, our framework reuses the exploratory semantic web search method proposed by Piao [16]. Our framework can represent the search result provided by Piao's framework in a Data View Schemata. Then users can easily visualize the retrieved data from Semantic Web, or visualize the integrated data obtained from both local database and Semantic Web.

1.1 Background and Motivation

1.1.1 Information visualization and visualization systems

Information Visualization is a widely-used technology and an important research field of computer science. This technology aims to guide users to create charts composed by a set of graphical objects. Some visual properties of these graphical objects are used to represent data retrieved from various data sources. Because most people are more sensitive on perceiving visual properties, such as color and size, than abstract numeric and/or non numeric data, it is easier for users to obtain their desired information from charts than from the original data. Under the help of modern visualization technology, users can reveal some information hidden in the abstract data. For example, users can easily know the data change trend from a line chart. In a histogram, users can easily compare the values represented by the visual objects of this chart.

In recent years, data measuring and gathering technology have made a great progress. For example, the popularization of smart mobile phones brings the rapidly increasing information of human activities. The data sets used to store the gathered data are becoming more and more complex. Their sizes and dimensions increase greatly. Large scale data sets and multivariate data sets become popular. Because of these changes of the features of data sets, many new research challenges on information visualization appeared. How to effectively and exactly visualize such kind of complex data sets becomes more and more important.

Many researchers devote themselves on information visualization. After surveying their researches, we found they mainly focus on the following four aspects:

(1) How to design a visualization?

(2) How to guide users to simply visualize data by computers?

(3) How to improve users' chart manipulating experiences?

(4) How to accelerate visualizing processes and chart interacting processes?

The researches of the Aspect (1) discuss how to effectively use graphical objects and their visual properties to represent different data sets. Some visualization types may enable users to more easily obtain their desired information. Some visualization types may suit for data gathered from special fields (e.g., biology or geography), which may have their own common features. The researchers may want to develop

new visualization types for visualizing data sets with the same feature to improve chart readers' comprehensions.

The researches of the Aspect (2) propose new visualization frameworks, methods or systems, which can guide users to define visualization charts. How to simply the visualization processes is considered by researchers. Researchers also want to develop a unified framework for creating all types of charts.

The researches of the Aspect (3) discuss how to make users easily interact with generated charts. The increasing size of a data set will lead to the increasing amount of corresponding visual objects in a visualization chart. How to find one or more special visual items in a chart becomes a problem. Researchers aim to improve such chart exploring experiences by providing necessary operations, such as zoom in/out.

The last researches of the Aspect (4) try to solve the problems happened when the data sets are too large. To visualize such a kind of data sets, the system has to spend more time on drawing visual objects to represent the data. How to speed up this visualization process becomes a new challenge. Meanwhile, how to reduce the lag time happened when users interact with such kind of charts is also considered by researchers.

Our research in this paper focuses on Aspect (2) and Aspect (3). For explaining our research clearly, we define the record visualization templates and aggregate visualization templates. We also classify all charts into standard charts and non-standard charts. Furthermore, we divided all conventional visualization systems into two categories. I will explain these definitions and classifications in the following parts of this section.

Record templates and aggregate templates

In our framework, the visualization type of a chart is determined by its visualization template. Each visual object is an instantiated copy of the visualization template. When creating a chart, our system will copy visualization templates and modify some visual properties of the copies of the visualization templates. The values of the visual properties of each visual object are instantiated by the associated attribute values of the corresponding data object.

Using the copies of the visualization templates to represent data, there are two common situations:

1. one copy of the visualization template represents only one tuple in a data table;

2. one copy of the visualization template represents more than one tuple in a

data table.

According to these two different situations, we classify all visualization templates into two categories, i.e., record templates and aggregate templates.

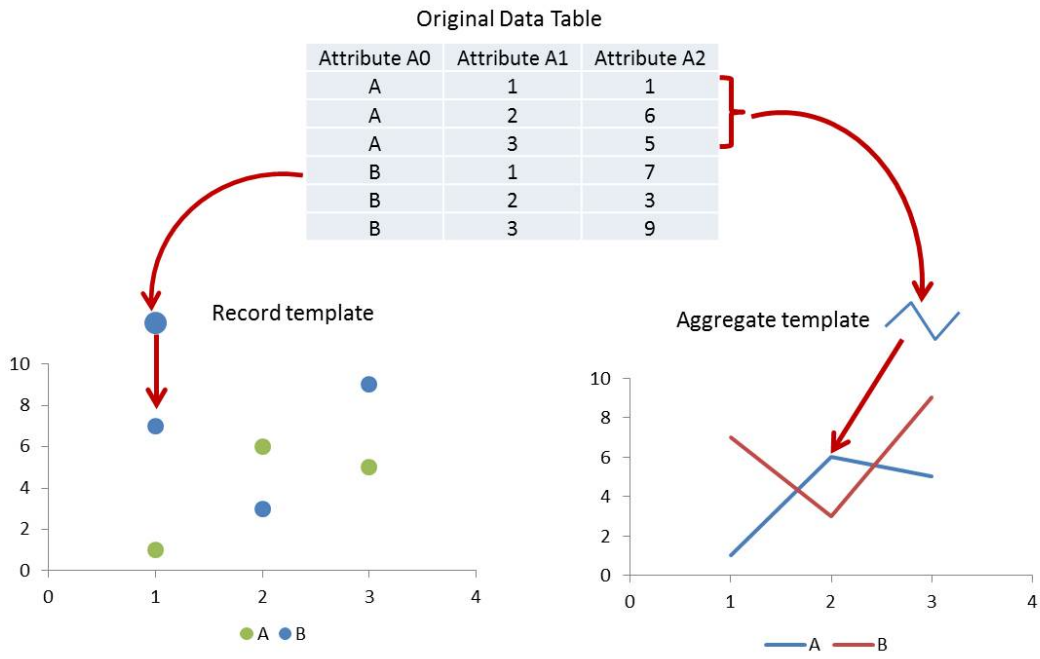


Figure 1.1: A record template and an aggregate template

Figure 1.1 shows two simple examples of using these two kinds of templates to separately define a discrete chart and a continuous chart. Record templates are usually used to create discrete charts, and aggregate templates are usually used to create continuous charts.

Standard charts and nonstandard charts

In this thesis, we define two different types of charts, standard charts and non-standard charts. A standard chart may be a basic standard chart or a composite standard chart. A basic standard chart denotes such a chart composed using four types of basic graphical objects, i.e., “Point”, “Line”, “Area” and “Interval”. Leland Wilkinson introduced them in Table 8.3 of the book “The Grammar of Graphics” [30]. ”Point” and ”Line” are the graphical objects we are familiar with. According to the definition in this book, an “Area” is a graphical object containing all points in the region between a coordinate axis and a line graphical object representing

some function f(x). Figure 1.2 (a) is an example chart defined by an "Area". An "Interval" is a graphical object with two ends. One end is used to denote a single value through its location, and the other end is anchored at some point (usually representing a zero value). Figure 1.2 (b) is an example chart defined using "Intervals". A standard chart is either a basic standard chart, or a chart composed with more than one standard chart by geometrically arranging them or embedding some of them into another in a nested way.

Users may use a complex graphical object as a circle with no more than four parameters, i.e., the x and y coordinates of its origin, its size, and its color to represent each database tuple with no more than four attributes. Such kind of charts are also standard charts, and the graphical object is treated as a "Point".

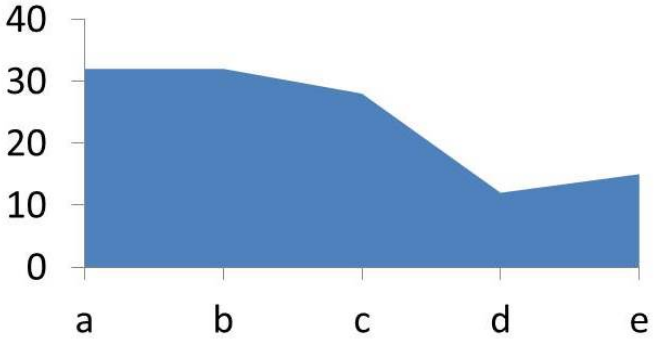
A chart that is not a standard chart as defined above is a nonstandard chart. Users may use a more complex graphical object such as a dinosaur or a face shown in Figure 1.2 (c) as a graphical record template to represent each database tuple. Such a complex graphical record template has more than four properties to represent the same number of tuple attributes. For example, users can create a nonstandard chart using the different relative location of each dinosaur's head or the different nose shape of each face to represent different values of the fifth attribute. In this way, users can use these complex objects to define some nonstandard charts. For supporting users to define such a kind of nonstandard charts, the system needs to be able to register a corresponding graphical object in the template library.

There is another kind of nonstandard charts, each of which uses one graphical object to represent more than one tuples of a data table. The flow map is such a kind of nonstandard charts. The graphical objects used in such charts may be simple. For example, polygonal graphical objects are often used in flow maps. The definition of such charts requires further data processing. The creation of this kind of nonstandard chart is not yet well supported by existing visualization systems.

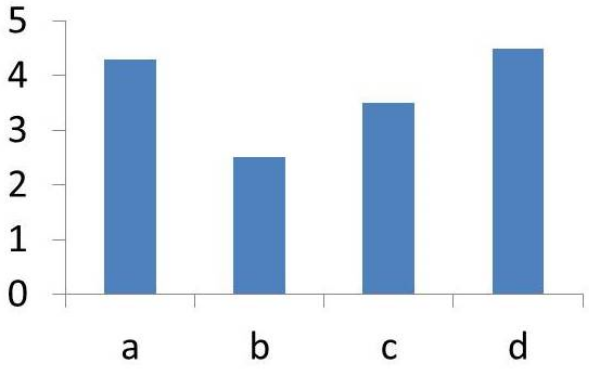
There may be some other types of nonstandard charts which are not classified either of these two types mentioned above. Such nonstandard charts are beyond the scope of our discussion in this paper.

Two categories of visualization systems

According to the chart defining methods, we can divide conventional into two categories. The visualization systems in the first category enable users to define charts by programming. The visualization systems in the second category enable users to define charts by their logical specification.



(a) A chart created by an "Area" object



(b) A chart created by four "Interval" objects

(c)Two special graphical objects

Figure 1.2: Two charts which are respectively defined using the "Area" object and "Interval" objects, and two special graphical objects

The visualization systems in the first category usually provide a new programming language for users to define their desired charts. Mathematica [17] is one of this kind of visualization system. Using such a visualization system, users are requested for the programming skill. This is a not an easy task for casual users because it requires significant programming skills. It is very difficult for programming novices. Even for the users who have programming experiences, they may also have to spend time to learn a new language.

For solving the above problem, researchers develop visualization systems in the second category. Such a system provides a library of graphical objects. Users can use a declare method to specify the components of a chart. Although using the declare method, users can specify how to use visual properties of graphical objects to represent data. Protovis [2] and ggplot2 [29] are such kinds of visualization systems.

However, the systems in the second category still have three major problems.

First, the power of such systems depends on the sizes of their libraries of registered graphical objects. The developers of each visualization system pre-defined the graphical objects in each library. End-users only can select one and use it as the visualization template of a chart. The developers of such systems need to frequently update the library of graphical objects to satisfy users' requirements. This is a tedious, expensive and never ending endeavour, and users' requirements cannot be satisfied immediately. Even a user has professional skill on the system development, he also can not define new graphical objects by himself. This problem limited end-users to define custom-designed charts.

Second, the system developers determine how to parameterize these graphical objects. When a system developer designs a graphical object and register it to the library, he need specify what visual properties of this graphical object can be used to represent data, and what kind of data structure are requested by the defined object. The parameterization of different graphical objects may be different. When users want to use the graphical object to define a chart, they must specify the parameters of the used graphical object correctly. In some situations, users have to read the instructions in detail before using these systems. The instructions may also make users confused.

Third, selecting an appropriate graphical object from the library is not easy. Users usually select a graphical object which will be used as a visualization template according to their appearances. However, two graphical objects with the same appearances may used for defining different charts when they are parame-

terized by different methods. For example, ggplot2, provides two graphical objects, "geom_histogram" and "geom_bar", which are both defined as rectangles, but used in different cases. Such similarity may confuse end users.

These problems usually happen when users want to define nonstandard charts by these systems.

1.1.2 The chart of Napoleon's Russian campaign

By defining various complex charts, developers can demonstrate the power of a visualization system. In these complex charts, there is a famous one called Napoleon's Russian campaign in 1812(Figure 1.3). Our research also starts from this chart.

This chart was published in 1869 by Charles Minard. This chart has two sub-charts. One is a flow map, which describes the campaign paths of three Napoleon's troops, as well as the incremental casualties, drawn on a simple map. This flow map is a nonstandard chart, because each path has different and changing widths to represent the number of survivors at different locations. If each path had a fixed width, it is classified as a standard chart (line chart). The other sub-chart is a line chart, which is drawn below the flow map to show the temperature decrease as the main army retreated from Moscow. This chart should be read from right to left.

Edward Tufte said it is "the best statistical graphic ever drawn"[26] because Charles Minard's smart design. The flow chart represents a data set with five attributes. The line chart as a complement of this flow chart helps readers understand why the amount of survivors decreased so quickly.

This chart attracted many researchers in information visualization fields. Michael Friendly surveyed the research studies on this chart, and published the result on his website [7]. From this survey, we can know the researches about this chart focus on three issues:

1. how to extend this chart to effectively represent more information;
2. how to enable users to more easily obtain the information; and
3. how to use modern visualization tools to automatically construct this chart as well as its extended versions. Solving the first two issues depends on the design of the chart.

Our research also focuses on the issue 3. We found no visualization system can define this chart in a generic method. We aimed to propose a framework which enables users to define both standard and nonstandard chart in a unified method.

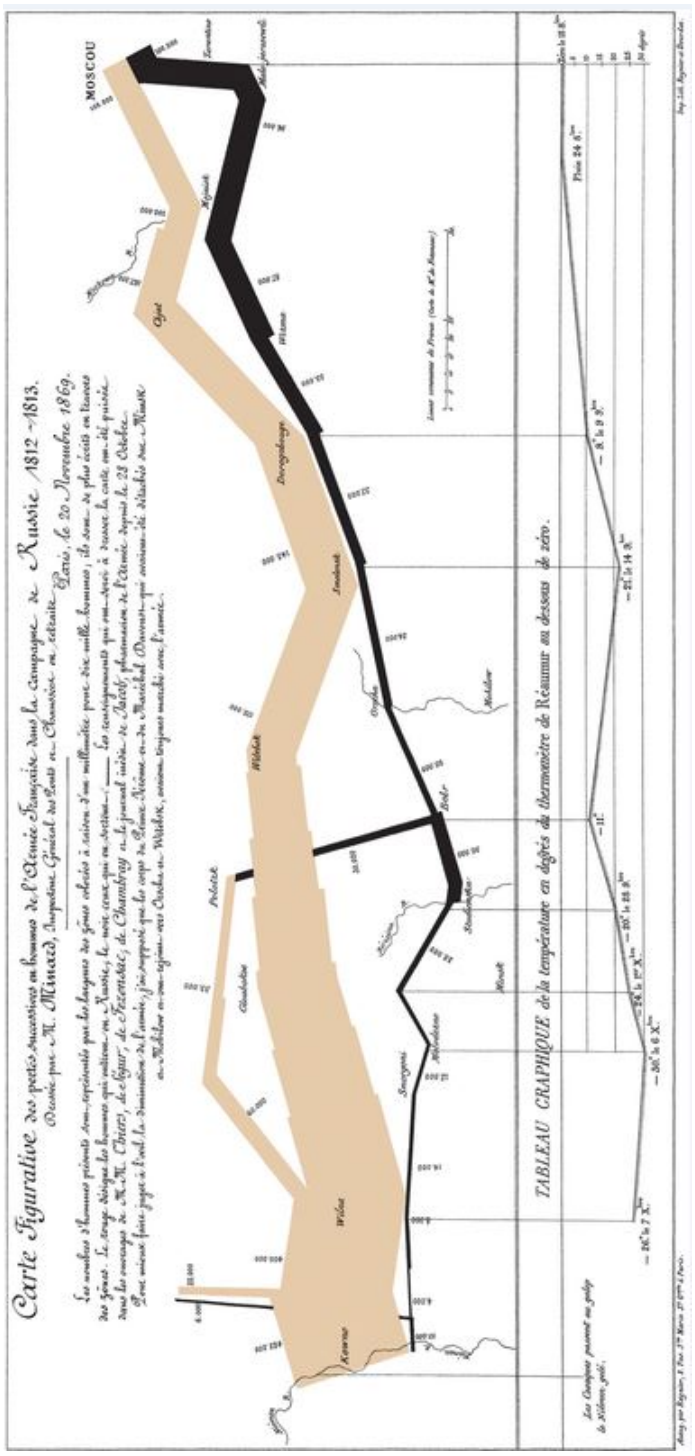


Figure 1.3: The chart of Napoleon's Russia campaign published in [26]

We also recreate and extend this chart to demonstrate the power of our proposed framework.

1.1.3 Semantic Web

For most existing visualization systems, they only consider about visualizing data from local databases or files. However, as the development of the web technology, more and more data is stored on the web. Some researchers have started to consider how to visualize data retrieved from the web. Our framework also aims to enable users to visualize data from different data sources. One important source on the web is Semantic Web.

Semantic Web is a technology that aims to convert the "web of documents" to the "web of data". It provides a framework, named resource description framework (RDF), for describing Semantic Web resources. RDF not only describes what resources are stored in the Semantic Web database, but also describes the relationship among these resources. Semantic Web also provides necessary technology standards, which can help users exactly find out their interesting information.

Because of the advantages of Semantic Web, it is becoming a very important data sources. Many Semantic Web databases are published on the web, such as DBpedia, FOAF, SIOC, and etc., for users to use freely. The amount of such web databases still increase rapidly. In 2007, the amount of the web databases is about 200, and then in 2014, the amount becomes over 1000. Figure 1.4 is the linking open data cloud diagram showing the current situation of the databases on Semantic Web. Each circle in this figure is a database and the links show the relationship among databases.

But the data store in Semantic Web is in a different format with the data in a relational database. Data in Semantic Web is formatted as triples. To retrieve data from Semantic Web, a query language, called SPARQL, is necessary. But there are three main problems need to solve when using SPARQL to retrieve resources from Semantic Web:

1. The grammar of SPARQL is different from SQL, so users have to learn the grammar and to remember the commands;

2. Using the SPARQL to define a query, the background knowledge of Semantic Web is necessary;

3. The function of SPARQL is weak than the traditional query languages, such as SQL. Some advanced data-retrieving functions are unavailable in SPARQL.

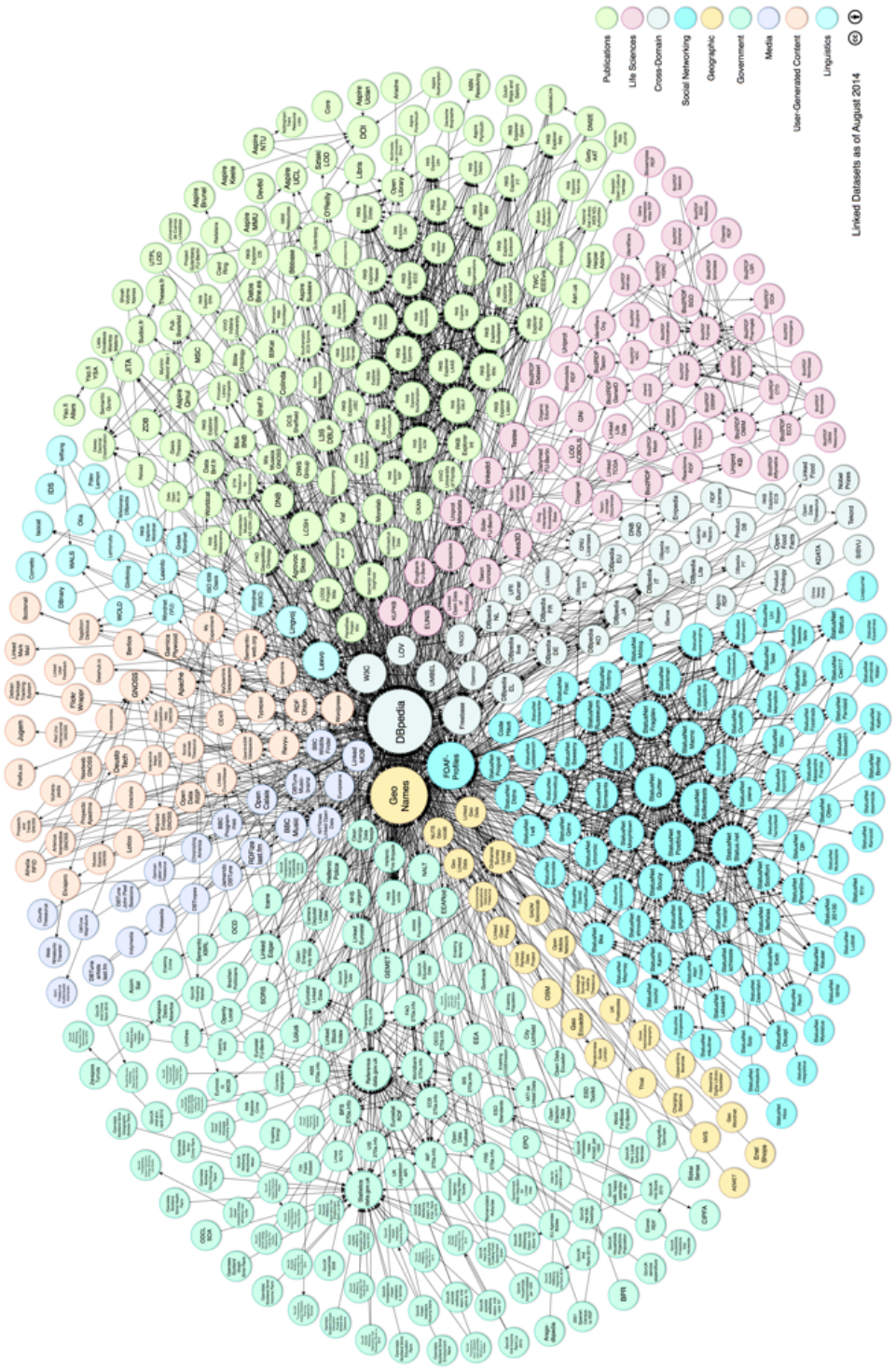


Figure 1.4: The linking open data cloud diagram from [15]

Furthermore, only a few researches discuss how to visualize data retrieved from Semantic Web. Some Semantic Web search systems provide simple visualizations of retrieved data, but it can not satisfy users' various visualization requests.

Although Semantic Web becomes more and more popular and important, users will not publish their private to Semantic Web. Users usually desire to use the data from Semantic Web as a complement of the local databases. Then we also need to consider how to integrate data from Semantic Web and local database, and how to visualize the integrated data. The research on this topic is not sufficient.

1.2 Contributions of This Thesis

In this paper, we propose a new interactive visualization framework for solving the problems mentioned in Section 1.1.

We propose a new generic visualization framework. It includes an intermediate model composed of two kinds of tree-structured schemata. Users can declaratively define both standard charts and nonstandard charts by visually manipulating this intermediate model. One schema is called the Data View Schema (DVS), which is used to define an object-oriented view of a database, to generate queries for retrieving data, and to generate data objects in a specific structure using the retrieved data. The other schema is called the Chart View Schema (CVS), which is used to represent the structure of the chart components in a chart, and the structures of the graphical objects which are used to define chart components. By manipulating such a CVS, users can customize a chart.

The DVS and the CVS have five main functions:

1. by manipulating DVS, users can define an object-oriented view of a relational database,

2. generating an object-oriented query for generating data objects from the data retrieved from a relational database,

3. defining how to instantiate the visualization templates for generating visual objects using generated data objects,

4. customizing the background, the legend, and the coordinate system of a visualization chart, and

5. defining composite charts.

Our framework represents each attribute of a data set as a tree node in the DVS. Comparing to other database query languages, such as SQL, our framework enables users to define hierarchical relationship among these attributes. The structure of a DVS also represents the relationship among the attributes. By manipulating the DVS, users can visually define an object-oriented view of the relational database, and generate an object-oriented query by NHibernate technology. Using this query, our framework can retrieve a set of data objects in the specific structure from the database. The structure of these data objects should match to the structure of the visualization template. Then, the system can directly use these data objects to instantiate the visualization template for generating visual objects without any further conversion.

Our framework represents each visual property of any graphical object used in a visualization chart as the node of a CVS. This tree structure ensures that users can clearly and easily define the DVS to match the structure of the visualization template. Users also can manipulate the CVS to define (1) the appearance of each graphical object, (2) how to use the graphical objects as the chart components, and (3) how to compose a visualization chart using these chart components.

Our framework also allows users to compose a complex visualization template to define their desired charts. To realize this function, our framework provides two special functional objects, i.e., a Transformer and a Shadow. Users can combine more than one primitive graphical object together and define the geometric relationship among these objects by using Transformers and Shadows. Then users can use the composite graphical object as a record template to define complex charts without any programming effort.

In order to support the GeoVisualization chart definition, we also provide five special functions in our framework. Our framework supports to automatically convert a geographical name into the x and y coordinate values through a web service. Our framework can automatically generate the mapping from the position information in data objects to the position properties of visual objects. We also simplify the process of specifying a map as the chart background, and wrap the function of ArcGIS[3].

In this framework, besides of the common operations for interacting with a chart, we also provide a set of new operations for supporting users to directly manipulate the generated charts to indirectly modify the DVS and the CVS.

We have also extended our visualization framework for supporting users to visualize the data retrieved Semantic Web. Users can directly access the Semantic

Web and exploratorily search their desired information through the method proposed Piao[16]. According to the search results, our framework will automatically generate a corresponding DVS. Users can manipulate both the DVS representing the data from Semantic Web resources and the DVS representing the data from a local database in a unified method. Users can also define a join operation between these two DVSs to integrate the data from Semantic Web and the local database. Users can use the method provided by our framework to visually define both standard charts and nonstandard charts using the data from the Semantic Web, from a local database, or the integrated data from these two different data sources.

1.3 Structure of the Thesis

The remainder of this thesis is structured as follows:

In Chapter 2, we introduce our visualization framework. We define an intermediate model, composed of two kinds of tree-structured schemata. Then we show the details of how to use this intermediate model to define a chart. We also introduce the template composition method which supports users to create composite visualization templates by primitive graphical objects. Last, we explain how to apply our framework to define GeoVisualization charts. In chapter 3, we describe the interaction operations provided in our framework. By directly manipulating the generated charts using these operations, users can indirectly modify the intermediate model. In chapter 4, we explain how to extend our visualization framework to visualize data retrieved from Semantic Web. In chapter 5, we introduce a system developed based on our proposed framework using the NHibernate technology [6] and Webble World technology [11]. In chapter 6, we introduce other visualization systems. We compare our framework with them and show the comparison result in this chapter.

Finally, we will make some concluding remarks in chapter 7.

Chapter 2

A New Information Visualization Framework

2.1 A Generic Visualization Framework Based on Tree-structured Schemata

2.1.1 Outline of our framework

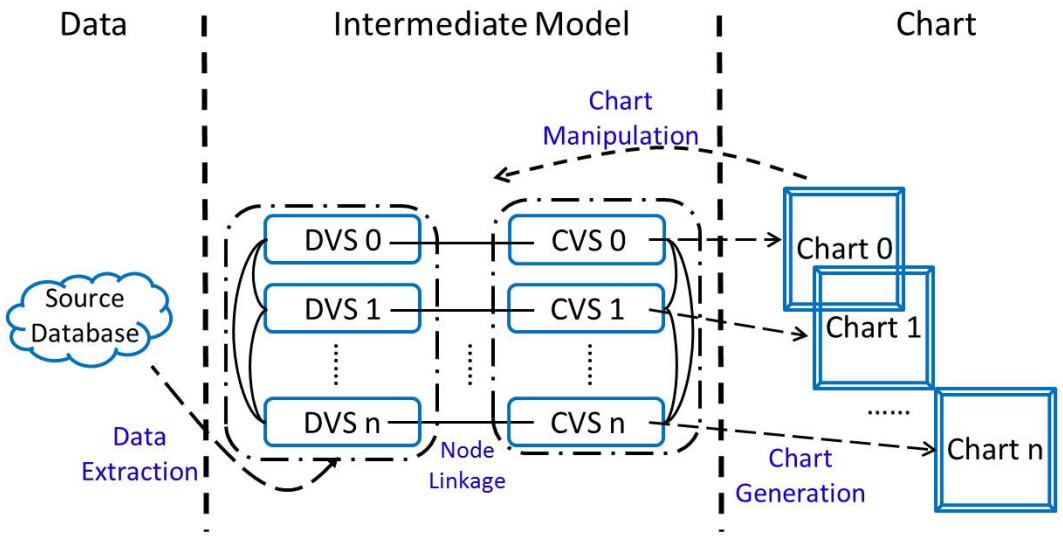
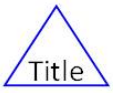


Figure 2.1: The outline of our visualization framework

The main task of each visualization tool is to define the mapping from the source data to a visualization chart. As shown in Figure 2.1, our framework uses tree-

Table 2.1: Nodes for composing DVSs and CVSs

Node Appearance	Schema	Node Name	Function
	DVS	View Node	It denotes the database view that is defined by this DVS. Users can change the title to name the defined view. Normally it is automatically named as View_i.
	CVS	Chart Node	It represents a chart. Its title indicates the chart name. Normally it is automatically named as Chart_i.
	DVS	Attribute Node	It denotes an attribute of the original data set or a derived attribute. Its title indicates the name of this attribute.
	CVS	Property Node	It indicates a property or a property set of some graphical object.
	DVS	Function Node	It is used in defining a derived attribute by applying a computing function to some attribute values. Its title indicates which function it represents.
	CVS	Chart Component Node	It denotes a chart component. Its title indicates which component it denotes. We define four chart-component nodes, i.e., Template node, Coordinate-System node, Legend node, and Background node.
	DVS	Constant Node	It denotes a constant or a parameter value that is used in defining a derived attribute. Its title indicates a constant value or a parameter name.
	CVS	Graphical Object Node	It is a child node of a chart component node. Its title is the name of the graphical object that is used in defining a chart component.
	DVS	Quantification Node	It is a child of some attribute node or of the view node. It stores a condition for quantifying the value of the corresponding attribute, or a set of data-object IDs for specifying which data objects are retrieved from the source database.

structured schemata as the intermediate model to define this mapping. We call these two schemata the Data View Schema (DVS) and the Chart View Schema (CVS). [18] Table2.1 introduces all types of nodes used in these two tree-structured schemata. As shown in Table 2.1, the nodes with the same appearance have two different functions separately in the DVS and CVS. For example, the filled circle represents the "View Node" in the DVS, and represents the "Chart Node" in the CVS. To manipulate DVSs and CVSs, we provide a set of tree manipulations. We show these operations in Table 2.2. Except the operations "Group-By" and "Ungroup-By", these operations are used in most tree structures. We believe users can easily understand them and quickly know how to use them to manipulate DVSs and CVSs. In Section 2.1.4, we will show the details of the operations "Group-By" and "Ungroup-By".

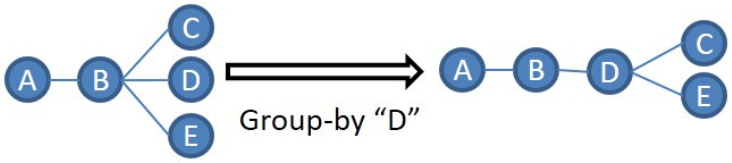
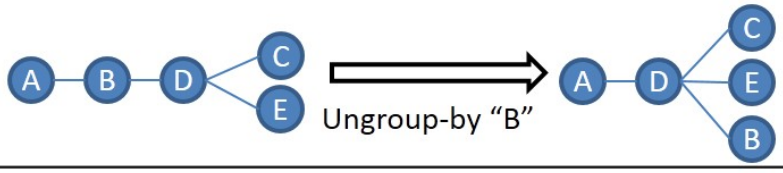
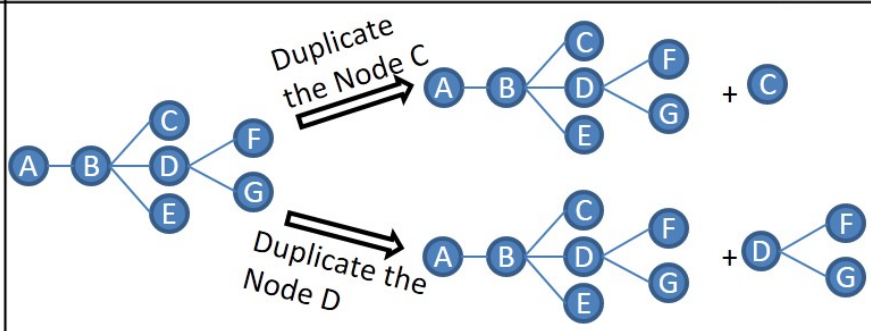
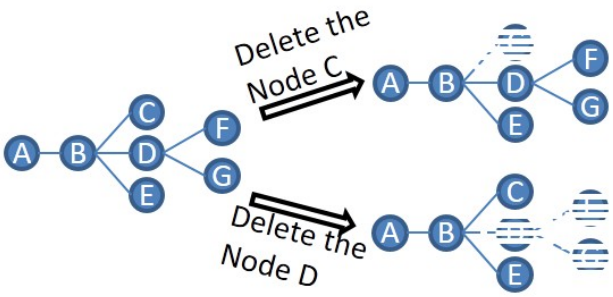
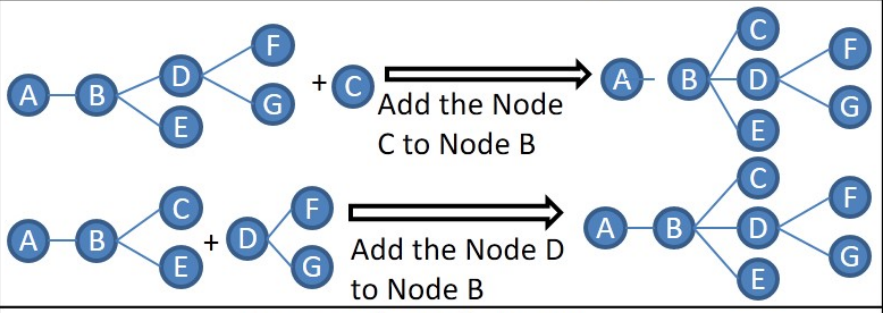
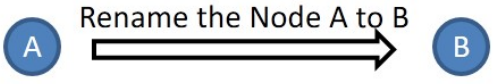
Each DVS is used to retrieve and process appropriate relational data from the source database, and to obtain a set of data objects with several attributes including derived ones defined from original attributes. The structure of the DVS determines the common structure of these data objects.

Each CVS is used to define a visualization chart by separately defining each of the four chart components, and how to use them to compose a chart. The four chart components include (1) its visual object set, (2) its coordinate-system, (3) its legend, and (4) its background. Each chart component is either a graphical object or a set of graphical objects. Each CVS has four sub-trees, i.e. the Template subtree, the Coordinate-System subtree, the Legend subtree, and the Background subtree, corresponding to the above mentioned four chart components. Each Graphical-Object node in each of the four subtrees has one or more sets of Property nodes to represent the properties of this graphical object. Users can manipulate these nodes to specify the corresponding chart component, and manipulate these subtrees to compose the resulting visualization chart. Each property node in these subtrees has a default value. Users only need to modify a subset of property nodes to customize a chart.

The relationship between DVS and CVS is defined by setting dashed lines between some of their nodes. Each of these lines defines a mapping from an attribute of some data object to a visual property of some visual object. For example, in Figure 2.2 we link the node "direction" of C2 to the node "Fill" of C5. Corresponding to each value of the attribute "direction", our framework generates a unique color, and specifies a unique color as the value of the property "Fill" of the corresponding visual object.

As shown in Figure 2.1, our framework supports four kinds of linkages among

Table 2.2: Operations for manipulating DVSs and CVSs

Name	Instruction
Group-By	
Ungroup-By	
Duplicate a node	
Delete a node	
Add a node	
Rename a node	

DVSs and CVSs. Besides the linkage between the paired CVS and DVS that have been introduced in the example, users can define:

1. the linkages between two DVSs,
2. the linkages between two CVSs, and
3. the linkages between two sub-trees of a CVS.

The first kind of linkages defines the relation among the visual objects in different charts. The second one defines the relations (relative positions, the chart overlaps or property value bindings) between two charts. The last one defines the relationship between two components of the same chart.

This framework also provides a template composition method for supporting users to define a complex graphical template using primitive graphical objects in the library of graphical objects. Then users can use such newly-defined graphical templates to define charts. This method aims to solve the problem of insufficiently registered graphical objects in the library of our framework.

2.1.2 Visualization process

In Figure 2.2, we show how to use our framework for creating the flow map of the Napoleon's Campaign chart. In this section, we neglect the background of this chart for the simplicity of our discussion.

Step 1: Template Selection

According to the appearance of each path in the resultant chart, we chose a polygon graphical object C4 from our object library C3 (A.3) to define this chart. As described later, this selection is explicitly specified in the CVS. Each visual object is an instantiated copy of the visualization template. The values of some visual properties of each visual object are instantiated by the associated attribute values of the corresponding data object.

Step 2: Automatic Representation of the Template Structure in the CVS

As shown in Figure 2.2, the CVS C5 used to define this chart has two subtrees, the Template subtree and the Geo-Coordinate System subtree, respectively defining the visual objects and the coordinate system.

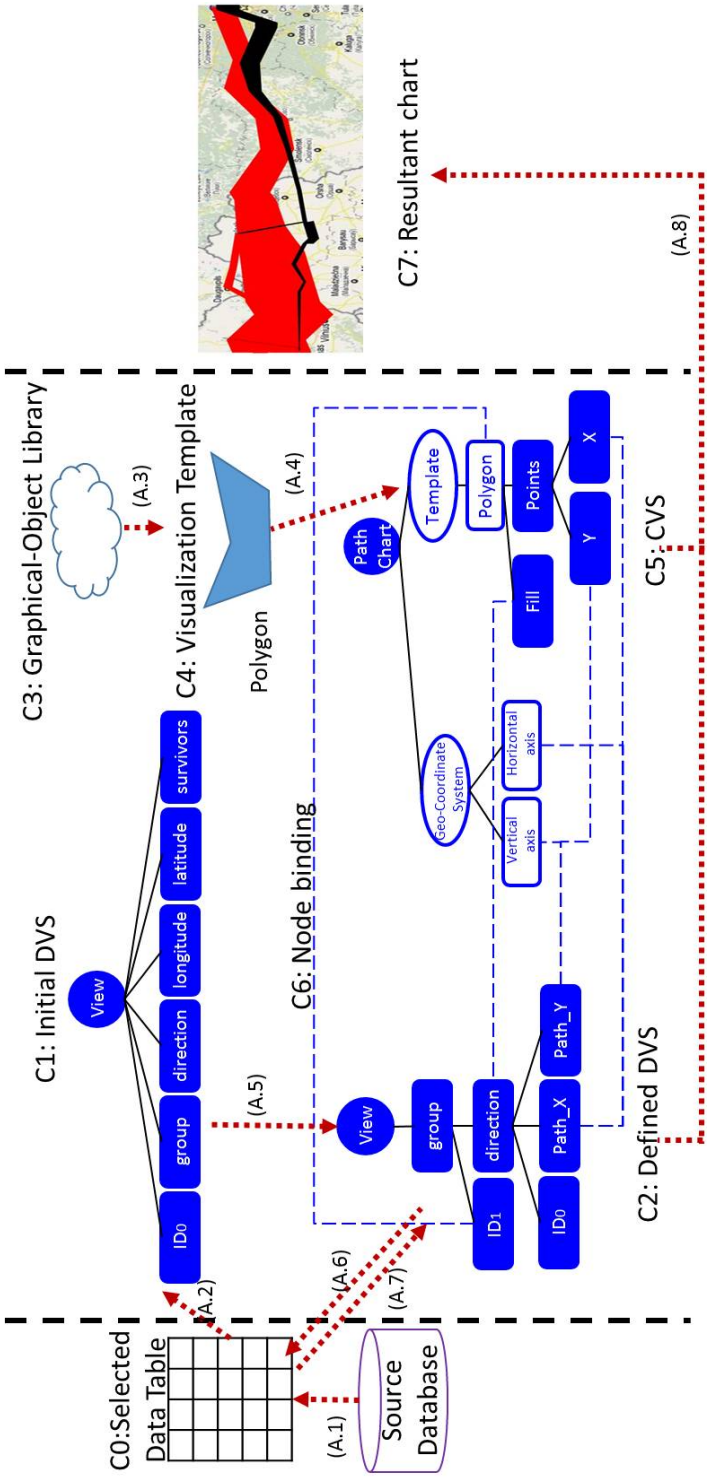


Figure 2.2: Visualization process

After a user selects a graphical object and specifies it as the visualization template, our framework will automatically extract the visual properties of the selected graphical object. Then the system represents this selected graphical object and its visual properties as the Template subtree. The Graphical-Object node indicates the selected graphical object, and each Property node represents one of its visual properties. In this example, this Graphical-Object node is the "Polygon" node (A.4).

Step 3: Data Table Section and Automatic Structure Representation of the Selected Data Table in the DVS

We select the relational data table C0 from the source database. Our framework will use a two-level tree-structured DVS C1 to represent the structure of the table C0.

Step 4: Data Structure Conversion

Each visual object, for example, each polygon in this case, does not directly correspond to any data tuple in C0. We need to define a set of new data objects by defining new derived attributes and converting the data structure of these tuples.

In other visualization tools, this structure conversion and the new attribute definition are supported by one of the following two methods: (1) pre-defined by system developers, or (2) direct programming by users. Method (1) enables the system to automatically process data and to construct charts. It simplifies the visualization process, but restricts users to construct only those charts that are pre-defined in the visualization system. Method (2) enables users to define custom-made charts, but require users to have programming skills.

In order to remove these restrictions, our framework enables users to perform these two manipulations simply through directly manipulating the DVS.

In this example, we convert the DVS C1 to a new four-level DVS C2. The DVS C2 defines a hierarchical relation among four attributes, "Group", "Direction", "X" and "Y" of each data object. The structure of the data object defined by the DVS C2 matches with the structure of the polygon template defined in the four-level Template subtree of the CVS C5.

Step 5: Visual Mapping Definition

We directly link nodes in C2 to the corresponding nodes in C5, using dashed lines to define the mapping from the attribute of data objects to the visual properties of

visual objects.

Step 6: Automatic Visualization Generation based on Users' Specifications of the DVS and the CVS

After this linkage definition, the system will automatically generate a query and issue this query from the DVS to the source database (A.6) to retrieve a set of tuples (A.7). Then our system converts these tuples to a set of data objects having the structure defined by the DVS C2. Using the CVS C5, and the user-specified linkages between the DVS C2 and the CVS C5, the system will then automatically construct the visualization chart C7 (A.8) by instantiating each copy of the visualization template with the attribute values of each retrieved data object. Because the structure of data objects matches with the data structure required by the visual objects, our system can instantiate each visual object without any further structure conversion of each data object.

2.1.3 System features

The most important feature of our framework is that the intermediate model is visual and can be directly manipulated. It enables users to process data, and to customize charts as they want. Users can visually define some complex charts only through structure manipulation. While the other systems can define complex charts only through explicit programming.

Both the DVS and the CVS are tree-structured schemata. Such kinds of tree structures are used in many areas. They are much easier to understand and to manipulate than abstract programming languages. We will explain further details of these two schemata in the following two sections.

Users can save all these defined schemata, i.e., both DVSs and CVSs, for reuse. Users can also duplicate the subtree of an existing schema, and add this copy to another one. Such operations simplify the chart definition process by reusing exiting subtrees to avoid repetitive schema definitions of the same subtrees. The details of our framework will be given in the following sections. For simplicity, we only consider those cases in which the base data set is a relational database, and the visual output is a static 2D chart.

Our framework also has the limitations. Because of the direct connections between visual objects and data objects, it is difficult to use our framework to create the charts which do not satisfy the specified conditions, which would be the case in examples such as assembly drawing, flow charts.

Furthermore, some charts do not directly represent data but perform some data modifications are not supported as well, such as, a map of the subway that change the locations of some stations to make it easier for reading. Our framework does not focus on the visualization design and only represent data directly from the base data without human changes. If the chart creator is a programming expert, our framework also allows users to create advanced visualization templates by programming directly, and then share it with others.

2.1.4 Data View Schema (DVS)

Usually, the source data set for creating a chart is stored as a flat table. Each record in the table is the basic unit of the source data set. However, such a structure cannot satisfy all the requirements for creating various charts. As introduced in Section 2.1.1, the system need to convert the data structure before using the tabular data set for creating some kinds of charts, such as line charts or nested charts.

Our framework provides the DVS to support users to visually define how to retrieve and to process the source data. The retrieved data in our framework is organized as a set of data objects with a user-specified structure. The DVS has three functions:

(1) retrieving the necessary sub-set from the original data set;

(2) defining derived attributes;

(3) performing structure conversion of the retrieved data set;

(4) specifying IDs to data objects.

Each DVS has a tree-like structure. Our framework provides five different basic types of schema-nodes for composing a DVS. They are explained in Table 2.1. An initial DVS is always a two-level tree with several attributes. Such a two-level structure represents a flat table. As shown in Figure 2.3, the tree in (c) represents the table in (a). To realize the function (1) in the above list, users can directly remove one or more attribute nodes from the DVS to filter out the unnecessary attributes. Users may add a quantification node as a child of some attribute node to specify a condition to filter out the unnecessary records in the original data set.

For realizing the function (2), users can use a function node to define a derived attribute in the DVS. Figure 2.4(a) shows an example in which the fourth input of the function node Fun uses the output of another function node Fun'. Each function node has a single output connected to its parent node, and one or more

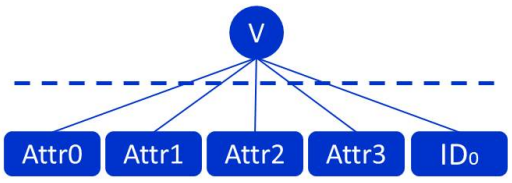
inputs coming from its children. The input of a function node may be an attribute node or the output of another function node. For creating a derived attribute, the order of the children of a function node determines the parameter order of the corresponding function. For example, the definitions in Figure 2.4(b) and (c) create a derived attribute 'a' respectively defined as "a:=b-c" and "a:=c-b".

Attr0	Attr1	Attr2	Attr3	ID0
$V_{0,0}$	$V_{1,0}$	$V_{2,0}$	$V_{3,0}$	ID0_0
$V_{0,1}$	$V_{1,0}$	$V_{2,1}$	$V_{3,1}$	ID0_1
$V_{0,1}$	$V_{1,0}$	$V_{2,2}$	$V_{3,2}$	ID0_2
$V_{0,0}$	$V_{1,0}$	$V_{2,3}$	$V_{3,3}$	ID0_3
$V_{0,1}$	$V_{1,1}$	$V_{2,4}$	$V_{3,4}$	ID0_4
$V_{0,1}$	$V_{1,1}$	$V_{2,5}$	$V_{3,5}$	ID0_5
$V_{0,1}$	$V_{1,1}$	$V_{2,6}$	$V_{3,6}$	ID0_6
$V_{0,0}$	$V_{1,1}$	$V_{2,7}$	$V_{3,7}$	ID0_7
$V_{0,0}$	$V_{1,1}$	$V_{2,8}$	$V_{3,8}$	ID0_8

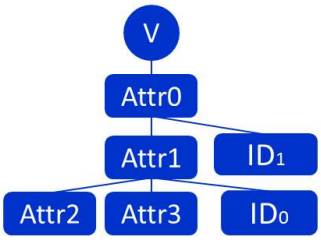
(a) A flat table representing a normalized relational view

Attr0	Attr1	ID1	Attr2	Attr3	ID0	
$V_{0,0}$	$V_{1,0}$	ID1_0	$V_{2,0}$	$V_{3,0}$	ID0_0	do0
		ID1_1	$V_{2,3}$	$V_{3,3}$	ID0_1	do1
	$V_{1,1}$	ID1_2	$V_{2,7}$	$V_{3,7}$	ID0_2	do2
		ID1_3	$V_{2,8}$	$V_{3,8}$	ID0_3	
$V_{0,1}$	$V_{1,0}$	ID1_4	$V_{2,1}$	$V_{3,1}$	ID0_4	
		ID1_5	$V_{2,2}$	$V_{3,2}$	ID0_5	
	$V_{1,1}$	ID1_6	$V_{2,4}$	$V_{3,4}$	ID0_6	
		ID1_7	$V_{2,5}$	$V_{3,5}$	ID0_7	
		ID1_8	$V_{2,6}$	$V_{3,6}$	ID0_8	

(b) An aggregate table representing an unnormalized relational-view



(c) The two-level DVS representing the normalized relational view shown in (a)



(d) The multi-level DVS representing the unnormalized relational-view shown in (b)

Figure 2.3: A normalized relational view, an unnormalized relational view, and their DVS represtations

Group-by and Ungroup-by operations

Users can define structures of data objects by manipulating the DVS. Our framework allows both the flat structures and hierarchical structures of data objects. For example, the multi-level tree shown in Figure 2.3 (d) defines a hierarchical structure of a data set, whose tabular representation is shown in Figure 2.3 (b). For defining a hierarchical structure, our framework provides a pair of special operations, i.e.,

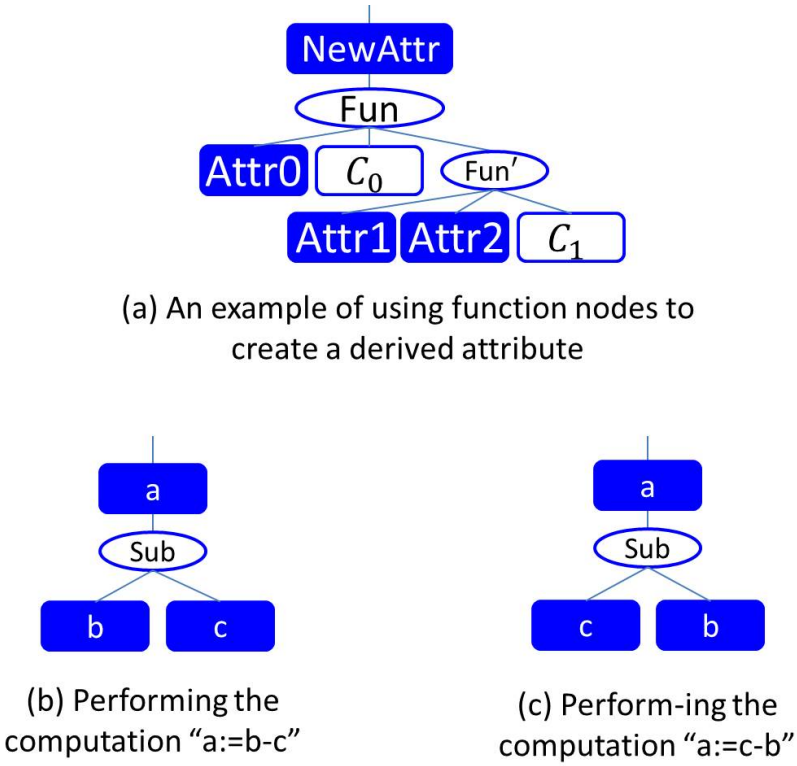


Figure 2.4: The examples for how to use the function node

"Group-by" and "Ungroup-by". As shown in Figure 2.5, users can use the "Group-by" operation to define a hierarchical structure of a DVS by dividing a set of records into several sets according to the value of one specified attribute.

It is different from the "Group-by" statement of SQL, which is always used in combination with an aggregate function, such as average, to obtain a single value. For example, to convert the DVS in Figure 2.3(c) to the DVS in Figure 2.3(d), we can apply "Group-by" operations to the "Attr0" and to the "Attr1". The "Ungroup-by" operation is the inverse operation of the "Group-by" operation.

Data object ID

Last, the DVS can also specify the IDs to data objects defined by it. A two-level DVS can only define one kind of data objects, while a multi-level tree can define more than one kind of data objects. As shown in Figure 2.3, the two-level DVS in (c) defines every record in (a) as a data object. The tree in (d) can define three kinds of data objects. The DO0, DO1 and DO2 are respective examples of these

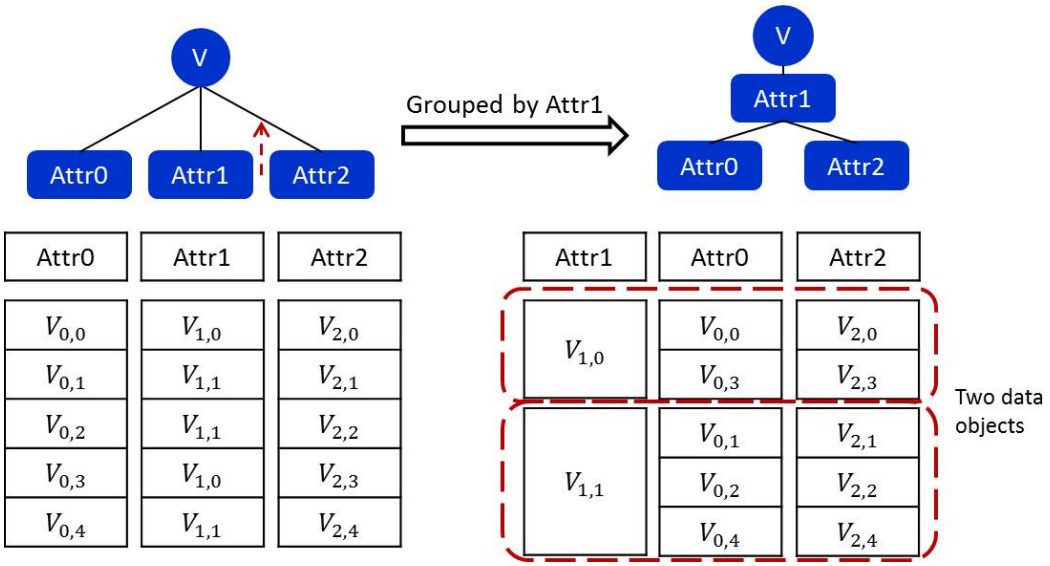


Figure 2.5: "Group-By" operation

three different kinds of data objects. According to the hierarchical structure among attributes, we define that DO0 is a sub-object of DO1, and DO1 is a sub-object of DO2.

Our framework requests users to specify the IDs to some or all data objects for manipulating them conveniently. Our framework treats the IDs as one or more attributes of data objects, and uses ID nodes in the DVS to represent such ID attributes. For example, in the tables shown in Figure2.3 (a) and (b), the attributes "ID_0" and "ID_1" are ID attributes, and in the DVSs shown in Figure2.3(c) and (d), the nodes "ID_0" and "ID_1" are ID attribute nodes.

When users initially define a two-level DVS to represent a flat data table, the system automatically adds an ID node as the brother of all the attribute nodes to generate the ID of every record (Figure 2.3 (a) and (c)). We name such IDs auto-specified IDs (asIDs). In Figure 2.3(d), the ID node "ID_1" is automatically added as the brother of Attr1 according to the users' request. Such an ID is called user-specified ID (usID). Then, the table of Figure 2.3(b) includes the ID attributes, ID_0 and ID_1.

Each ID node is protected in the following sense: users can only ask the system to add an ID node for specifying identifiers for data objects in the DVS, or remove an existing ID node for deleting the identifiers of data objects, but can not further manipulate it.

Another important function of the ID attribute node is to build the association

between the data objects and the visual objects representing these data objects. Our framework requires each data object represented by a visual object to have an ID. Our framework uses this ID to manipulate the corresponding visual object. In Section 2.1.6, we will explain the details of the relationship among a data object, its ID, and its corresponding visual object.

Data set join operation

Our framework allows users to perform the join operation between two data sets from different data sources through the manipulation of their DVSs. Users may join two two-level DVSs in one of the five ways: a cross join, an inner join, a left outer join, a right outer join, and a full outer join. Users can use a function node to realize the join operation. The title of each function node indicates which join operation is performed.

For defining a join of two relational data sets V0 and V1, users may use a function node as the parent of the two view nodes V0 and V1 (Figure 2.6). The join result is a DVS "V". Figure 2.6 (a) shows the DVS which defines the cross join of the V0 and V1. Figure 2.6 (b) shows another DVS which defines an inner join or an outer join operation of the V0 and V1. In Figure 2.6 (b), the dashed line connecting the attribute nodes Attr2 and Attr3 indicates the join condition. Although the inner join and the three outer joins may be defined in the same DVS structure, the join result includes different data tuples.

In this example shown in Figure 2.6 (b), to specify the join condition as "V0.Attr2=V1.Attr3", we use the dashed line to link the node "Attr2" to the node "Attr3".

Users can open a dialog, named the "Virtual Node Generator", from the right-click menu of this dashed line. It is used to request the system to generate a virtual attribute node to replace these two linked nodes. The dialog is shown in the Figure 2.6 (c). In this dialog, we specify a node "jAttr" to represent these two linked nodes. The virtual attribute node has a name starting with the '*' symbol and has two parents.

In order to keep the tree structure of the DVS, our system automatically hides the part, which defines the join operation of two DVSs. As shown in the Figure 2.6 (a) and (b), the system will hide the parts which are enclosed by boxes. The Figure 2.6(c) is a DVS with the join definition part being hidden. To indicate that the DVS has a hidden part, we add the '*' mark before the title of the view node as shown in Figure 2.6 (c). Through the right-clicked menu of the view node,

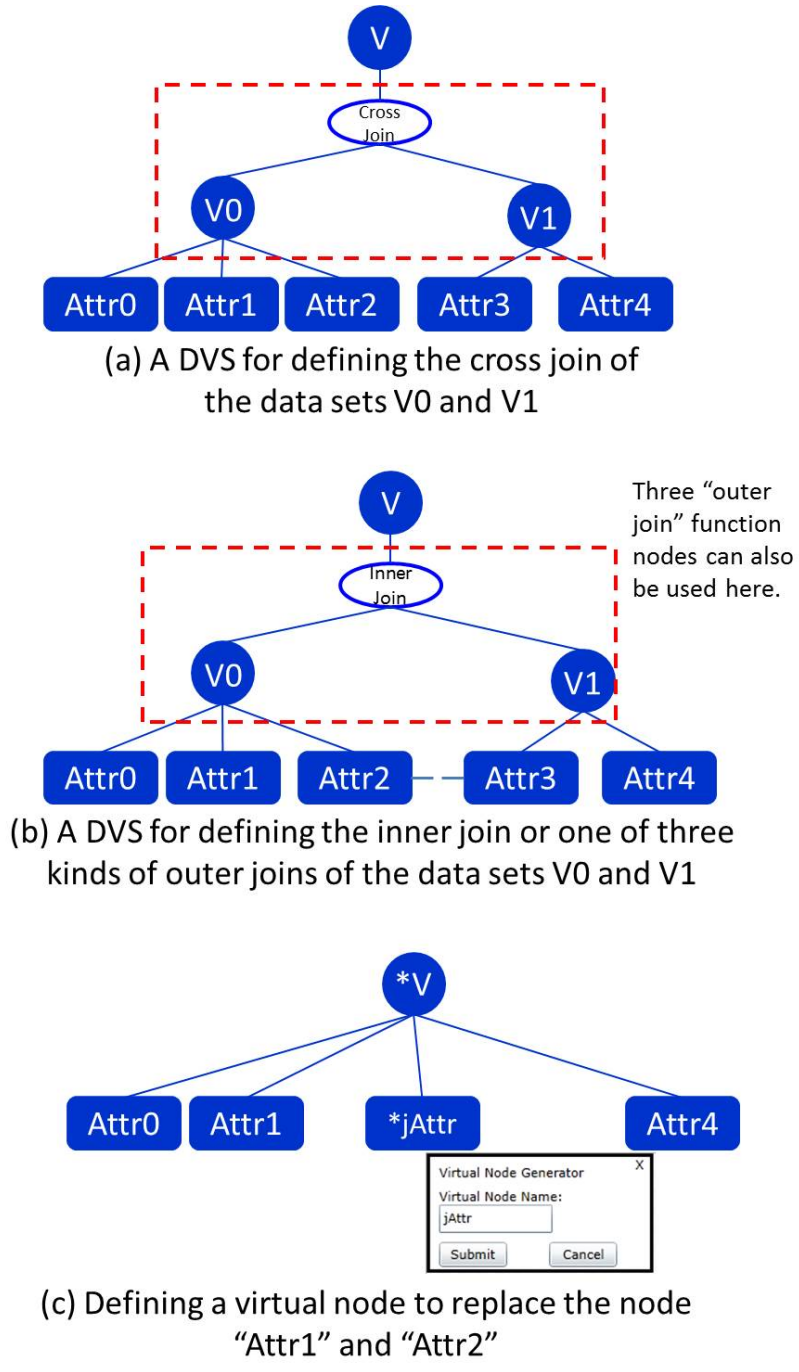


Figure 2.6: The DVSs for defining join operations of two data sets

users can request the system to show the hidden part. Users can perform further manipulations introduced in this section 3.2.1 after defining the join operation. Or users can further join this DVS with another DVS.

2.1.5 Chart View Schema (CVS)

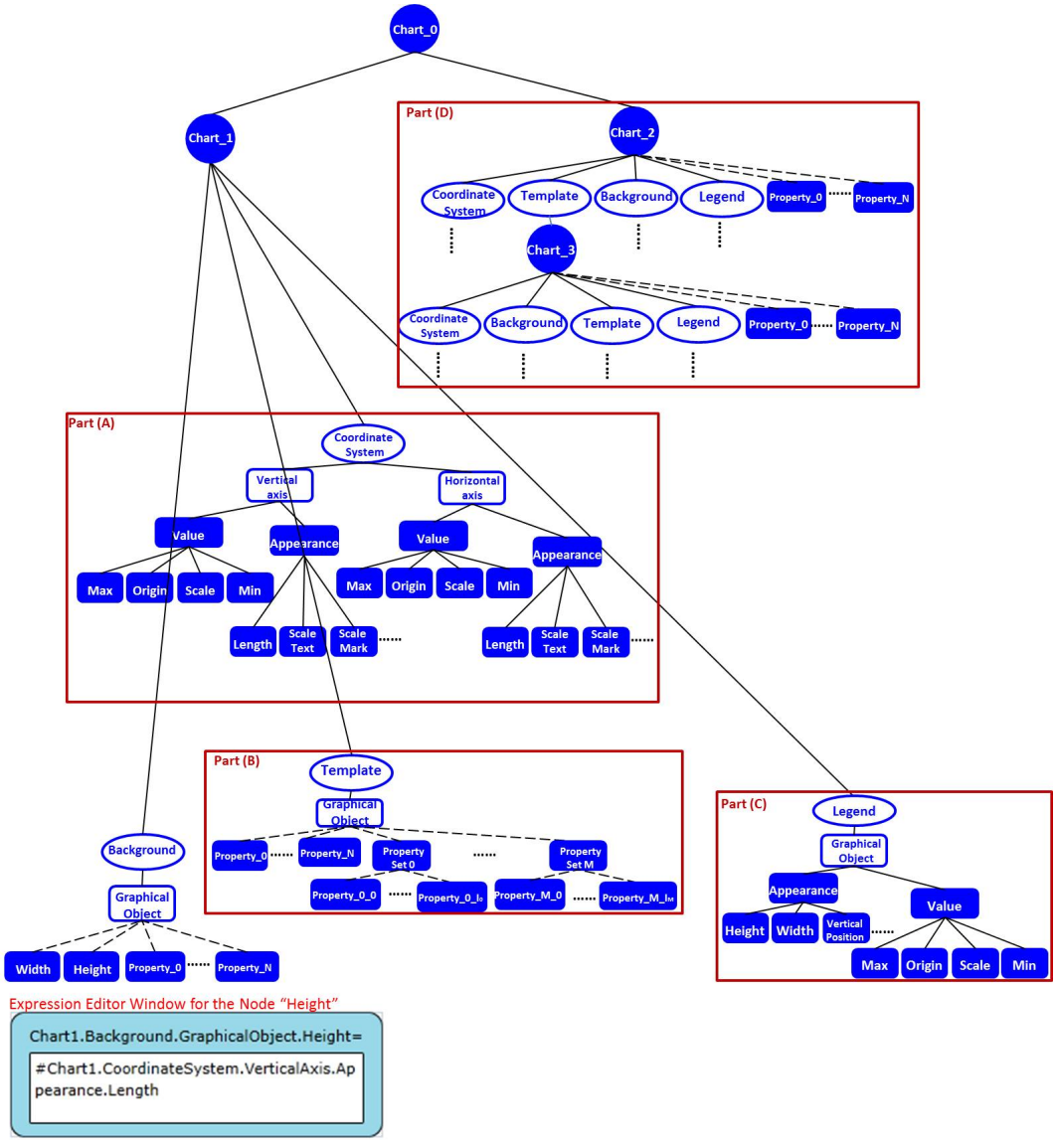


Figure 2.7: A CVS example

In our framework, every chart is composed of four components. Each component is one or a set of graphical objects. We define that each chart has four components:

1) the set(s) of visual objects created from the visualization template(s), 2) its coordinate system, 3) its legend(s), and 4) its background. To define a chart, users can manipulate the CVS of defining this chart. Table 2.1 lists the different nodes used to define a CVS, Figure 2.7 is an example CVS. We name each subtree of a CVS with its root node name. For example, a subtree with *Template* node as its root node is named as *Template* subtree.

As shown in Figure 2.7, every CVS should have four subtrees with the Chart-Component nodes as their roots, i.e., its Template subtree, its Coordinate-system subtree, its Legend subtree and its Background subtree. They are respectively used to define the four chart components. Each Chart-Component node may have one or more Graphical-Object nodes as its child, and they indicate what graphical object will be used as the corresponding chart component.

In the Template subtree and the Background subtree, each Graphical-Object node has a set of property nodes as its child nodes. These property nodes indicate the visual properties of each graphical object. The structure of the subtree of a Graphical-Object node represents the relation among its visual properties. As shown in Part (B) of Figure 2.7, the graphical object represented by the graphical object node in the template sub-schema has n properties and m property sets. In each property set, it has $i_k (0 \leq k \leq m)$ properties.

In the coordinate-system sub-tree (Part (A) of Figure 2.7), users can separately define each axis. Each axis has two property sets. One is the appearance property set, represented by the Appearance node, which defines the properties that determine the rendering within the coordinate axes. The other is the value property set, represented by the Value node, which has four properties to define what information is indicated by the axis. Three widely-used coordinate systems, the Cartesian coordinate system, the Polar coordinate system, and the Parallel coordinate system, are provided by the system library. In the legend sub-schema, users can choose a graphical object to create a chart legend and then customize it. The generic approach of our framework enables us to treat every chart legend as a small chart. As in the case of defining the coordinate-system, we need to define both an appearance-property set and a value-property set, to specify the chart legend, represented respectively by the Appearance node and the Value node in the Legend sub-schema (Part (C) of Figure 2.7).

Every default CVS will show all the subtrees and all the property nodes for defining a chart. Users can hide some nodes or subtrees in a CVS for simplicity. Our framework will use the default values of these hidden visual properties to create

the chart.

Our framework allows users to change the value of a visual property of a graphical object by manipulating the corresponding property node. Users can (1) directly change the value of a property node, (2) specify a mathematical expression to compute the value of a property from the values of other properties, or (3) wire a property node to some attribute node of the paired DVS. Method (1) is applicable to all the property nodes, Method (2) is applicable to the property nodes in all the subtrees except the Template subtree, and method (3) is applicable to the property nodes only in the Template subtree. Figure 2.8 shows an example of Method (1) and (3). We will show more details of Method (3) in Section 2.1.6.

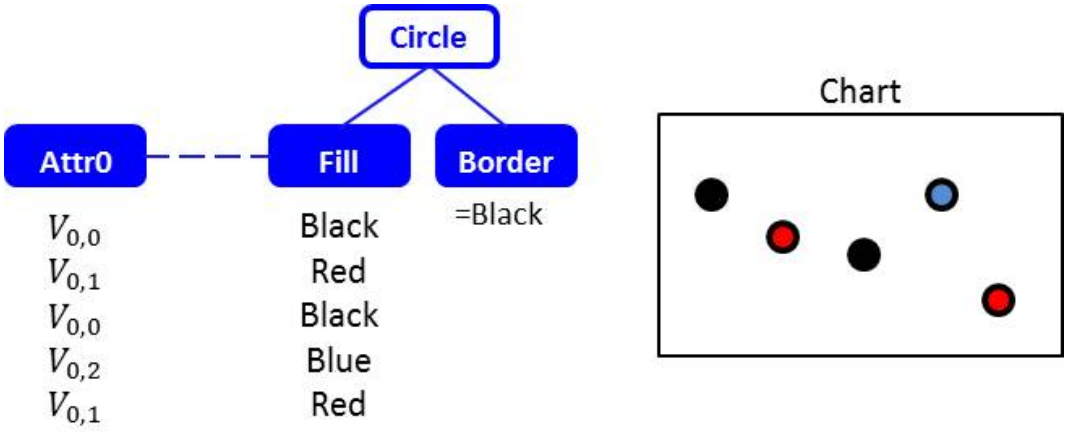


Figure 2.8: Two methods for determining attribute values of each visual object, i.e., by linking the attribute node to a property node, and by directly specifying the property value

Users can select a property node and open an expression editor form from its right-click menu to perform the method (2). In the editor, users can directly input a mathematical expression to define how to use the value of other properties to automatically compute the value of the selected property node. In Figure 2.7, there is an expression editor at the bottom of the figure. In this editor, we define that the height of the background is equal to the length of the vertical axis of the coordinate system. If there are more than one CVS in the workspace, users can also use the properties in other CVS to define the expression. Such a definition can define the relation of two charts, such as defining the relative positions of two charts. We will show an example of combining two charts together by this method in Section 2.1.8.

Users may define more than one CVS in the workspace, and each CVS separately

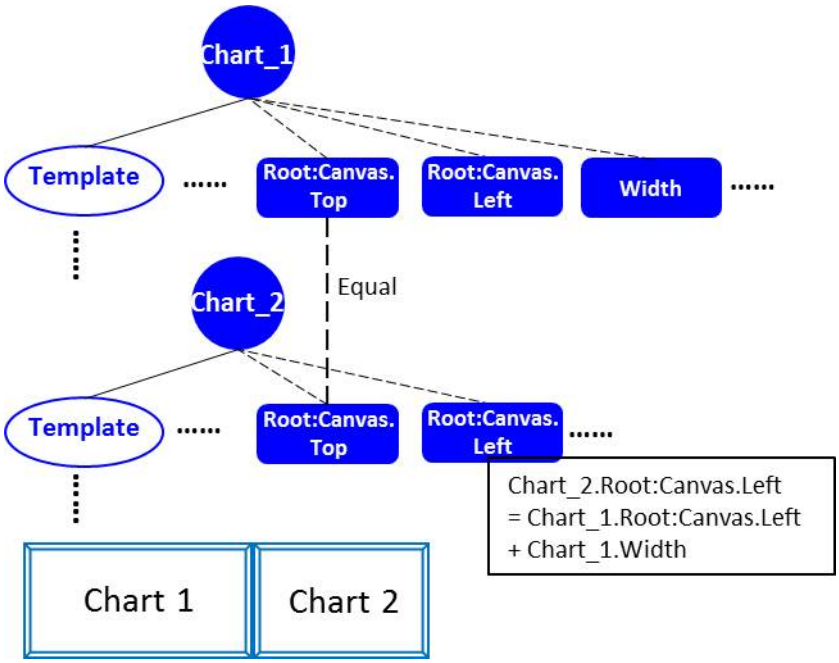


Figure 2.9: Defining the relative positions of two sub-charts

defines a sub-chart of the whole chart. Users can define an expression to specify the value of some property of a CVS using the properties of other CVSs. Such a definition can relate two charts. For example, in Figure 2.9, we specify that Chart2 should always be attached to the left boundary of Chart1 just by defining the expression in the expression editor window of the node "Root:Canvas.Left" in the Chart2's CVS. The combination of two sub-charts of Napoleon's Campaign chart can be similarly defined by this method, which will be shown in Section 2.1.8.

In our framework, users can create nested charts, by using another chart as a visualization template for each visual object. Figure 2.10 shows an example of the CVS for defining a nested chart. In this example We use Chart_3 as the visualization template of Chart_2 by using the CVS of Chart_3 as a child of the template node of the CVS of Chart_2.

2.1.6 Linkage between DVS and CVS

In our framework, we can define node bindings. Each node binding is defined between a property node of the Template subtree of a DVS and some attribute node of the paired CVS. This mechanism is sometimes called the visual mapping, which specifies how to use the visual properties of each visual object to represent the

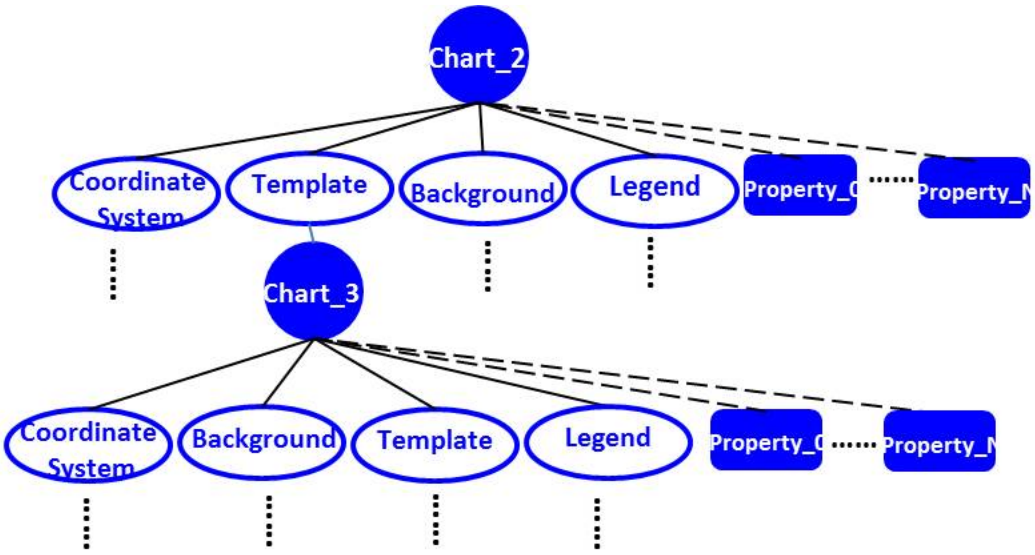


Figure 2.10: A partial CVS for defining a nested chart

retrieved data.

To build the visual mapping in our framework, users first need to wire the ID node of the nominated DVS to the graphical-object node in the template subtree of the paired CVS using a dashed line. This binding defines a direct connection between each data object and a visual object whose template is the bound graphical-object. This connection has three functions:

1. transferring data from a data object to its connected visual object for changing the values of its visual properties, and

2. synchronizing the selection states of a data object and its connected visual object.

In a resultant chart, we may use the axes of the coordinate system and legends to present the values of some attributes. To specify such associations, users need to link the Attribute node in the DVS to the corresponding Graphical-Object node in the CVS. For example, in Figure 2.2, the attribute "Path_X" is linked to the "Horizontal Axis" node of the Coordinate-System subtree. When such a linkage is set up, our system uses the value range of the linked attribute "Path_X" to specify the values of the four Property nodes, i.e., "Max", "Min", "Original", and "Scale", which are the children of the Graphical-Object node, "Horizontal Axis", in the CVS.

Users may link an Attribute node in the DVS to the Property node, namely either the node "Root:Canvas.Top" or the node "Root:Canvas.Left", in the Template subtree of a CVS, to specify either the vertical position or the horizontal position of each visual object. Then our system will also automatically set up a linkage between this Attribute node and either the "Vertical Axis" node or the "Horizontal Axis" node in the Coordinate-System subtree. For example, as shown in Figure 2.16, users need to manually set up the two linkages "Manual Linkage (1)" and "Manual Linkage (2)". Since they are respectively linked to the node "Root:Canvas.Left" and the node "Root:Canvas.Top", the system will then automatically set up the two other linkages "Auto Linkage (1)" and "Auto Linkage (2)". These bindings between some attribute and either the legend or the axes of the coordinate-system of the chart can be manually added, edited, or deleted after they are generated.

Figure 2.11 shows how our framework works when users select a data object or visual object. When a visual object in the chart is selected, the system will find out the graphical object (visualization template) used to define it (Arrow(1)), using the pointer stored in each visual object. According to the binding between the graphical object node to the ID node (Arrow(2)), the system will find out the ID of the data object represented by the selected visual object. Using the ID, the system will find out the corresponding data object (Arrow(4)) and rewrite the selection state of this ID in the SST (Arrow(3)). In the opposite direction, when a data object is selected, the system will first get its ID (Arrow(5)) and rewrite the SST (Arrow(3)). Then the system will find out to the corresponding visual object in the chart according to the binding between the ID node and the graphical object node (Arrow (7)), and the connection between the graphical object node and the visual object(Arrow (8)). Last, when the state of an ID is changed in the SST, the system will use the ID to find out the data object (Arrow (6) $\rightarrow$(4)) to change the selection state of the data object. Simultaneously, the corresponding visual object in the chart will be found, and its selection state will be updated (Arrow(6) $\rightarrow$ (7) $\rightarrow$ (8)).

Our framework provides a selection state table (SST) (Table fig:selectionTable) to show and change the selection state of data objects. It is stored in the root node of the DVS. In each table, there are one or more pairs of columns. Each pair contains an ID attribute and a "State" attribute. According to the ID nodes in the DVS, the SST may be a flat table or a hierarchical table. The hierarchical relation between IDs are directly extracted from the DVS. Table 2.3 shows two SSTs stored in the root nodes of the two DVSs in Figure 2.3 (c) and (d).

In an SST, if the state of an ID is 'F', it means the corresponding data object and

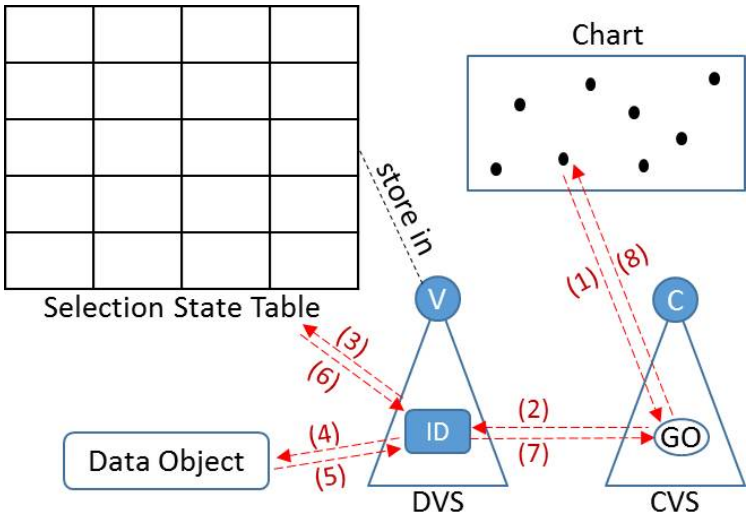


Figure 2.11: An illustration of how the system work when the users select a visual objects

visual object are not selected. If the state of an ID is 'T', it means the corresponding data object and visual object are selected. The default state of every ID is 'F'. When the selection state of a data object is changed, the SST will update automatically. Similarly, if a state in the SST is changed, our framework will change the selection state of the corresponding data object. Because of the direct connection between the data object and its connected visual object, our framework can keep the selection states of some data object, its ID in the SST, and its connected visual object same.

In a hierarchical selection-state table, the selection states among the ID of a data object and the IDs of its sub-objects will influence each others. We define a rule for the state propagation. Suppose that the ID of a data object is "oID" and that this object has n sub-objects with "soID_i" $(0 \leq i < n)$ as their IDs. We define the selection propagation rule as follow:

- Downward selection propagation:
  if oID=True, then, for all i, $soID\_i$ = True.

- Upward selection propagation:

  - Any-to-One:
    if there exists i such that $soID\_i$ =True,
    then oID=True;

Table 2.3: Two selection-state tables separately stored in the view nodes of the two DVSs in Figure 2.3

(a) TheSST in the DVS of Figure 2.3 (c)

oID_DT	State
DT_0	F
DT_1	F
DT_2	F
DT_3	F
DT_4	F
DT_5	F
DT_6	F
DT_7	F
DT_8	F

(b) TheSST in the DVS of Figure 2.3 (d)

ID_V_DT	State	oID_DT	State
V_DT_0	F	*DT_0*	F
		DT_1	F
V_DT_1	F	*DT_2*	F
		DT_3	F
V_DT_2	F	*DT_4*	F
		DT_5	F
V_DT_3	F	*DT_6*	F
		DT_7	F
		DT_8	F

– All-to-One:
  if, for any i, soID_i =True, then oID=True.

In the downward selection propagation, when the state of a data object is 'T', all the states of its sub-object will become 'T'. In the upward selection propagation, there are two possible modes. In the All-to-One mode, when the states of all the sub-objects of a data object are 'T,' the state of the data object will become 'T.' In the Any-to-One mode, if the state of one sub-object of a data object is 'True,' then the data object state will become 'T.' The update of a selection-state table will be responded by selecting or releasing corresponding visual objects in the chart.

2.1.7 Relating the visual objects of different charts

Our framework supports to relate the visual objects of different charts by defining the bindings among the view nodes among different DVSs. This function is called the correlation among charts. This function is very important and effective for improving users' comprehensions of multidimensional data sets [8][14].

Other visualization systems always provide such function as packaged techniques, while we introduce a common mechanism of defining such a function. Our framework

relates the visual objects in different charts by alternatively relating their connected data objects, which are defined in different DVSs. There are two situations:

1. if two or more DVSs defined by the same original table, their data objects will automatically relate to each other by the asIDs;

2. users can manually relate the data objects generated by different data tables by the values of one pair of specified attributes.

Figure 2.12 and Figure 2.13 are two examples to respectively explain the two situations. In each example, there are two charts, the DVSs for creating them, and their SSTs. In Figure 2.12, our framework will automatically wire the view nodes of the DVSs to each other though the same ID node "asID_T1", because the two DVSs are created by the same data table. If the selection state of any asID is changed in the SST of one DVS, the system will propagate this change to the other SST and update the state of the same ID.

In Figure 2.12 , if users select any black point in the left chart, the system will select the data object which is used to create the selected point by their connection. Next, the system will get the ID (e.g. the ID "asID_T1_2") of the data object connected and change the state of the ID in the left SST to 'T'. The change will be propagated to the right SST to change the state of the same asID to 'T'.

If we apply the Any-to-One mode (Section2.1.6) to the right SST, the state of the ID "urID_T1_V1_0" will become 'T'. The system will select the data object with the ID "urID_T1_V1_0" and the green line in right chart, which is connected with the selected data object.

Figure 2.13 (b) is the example of the situation (2). The two charts are created by different data tables. We can link two tables through two attribute nodes and using dashed lines. Suppose that a data object defined by the left DVS is DO_L, a data object defined by the right DVS is DO_R, and the value of the attribute AT of the data object DO is DO.AT. In this example, we define that if DO_L.Attr1 is equal to DO_R.Attr5, DO_L and DO_R will have the same selection state. We need to extend the two SSTs using the "Attr1" and "Attr5" to realize this function.

As shown in Figure 2.13, we respectively connect the nodes "Attr1" and "Attr5" to the two view nodes using the dashed line with the text "SSTE" to extend the SSTs stored in the two view nodes (the extended SSTs is on the top of Figure 2.13).

In the Figure 2.13 , if we select the leftest point in the left chart, the system will get its ID "asID_T1_0" and change the state of the ID in the left SST to 'T'. Next,

asID_T1	State
asID_T1_0	F
asID_T1_1	F
asID_T1_2	F
asID_T1_3	F
asID_T1_4	F
asID_T1_5	F
asID_T1_6	F
asID_T1_7	F
asID_T1_8	F

State Propagation

urID_T1_V1	State	asID_T1	State
urID_T1_V1_0	F	asID_T1_0	F
		asID_T1_2	F
		asID_T1_3	F
		asID_T1_5	F
		asID_T1_6	F
		asID_T1_7	F
urID_T1_V1_1	F	asID_T1_1	F
		asID_T1_4	F
		asID_T1_8	F

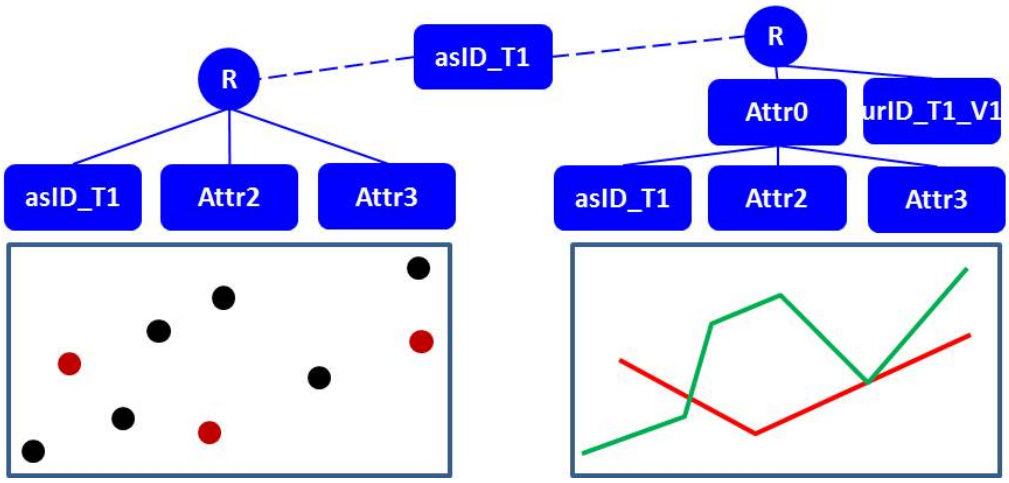


Figure 2.12: An example of relating the objects by asID

State Propagation

asID_T1	Attr1	State
asID_T1_0	B0	F
asID_T1_1	B0	F
asID_T1_2	B0	F
asID_T1_3	B1	F
asID_T1_4	B1	F
asID_T1_5	B1	F
asID_T1_6	B0	F
asID_T1_7	B0	F
asID_T1_8	B0	F

urID_T2_V1	Attr5	State	asID_T2	State
urID_T2_V1_0	B0	F	asID_T2_1	F
			asID_T2_3	F
			asID_T2_5	F
			asID_T2_6	F
urID_T2_V1_1	B1	F	asID_T2_0	F
			asID_T2_2	F
			asID_T2_4	F
			asID_T2_7	F

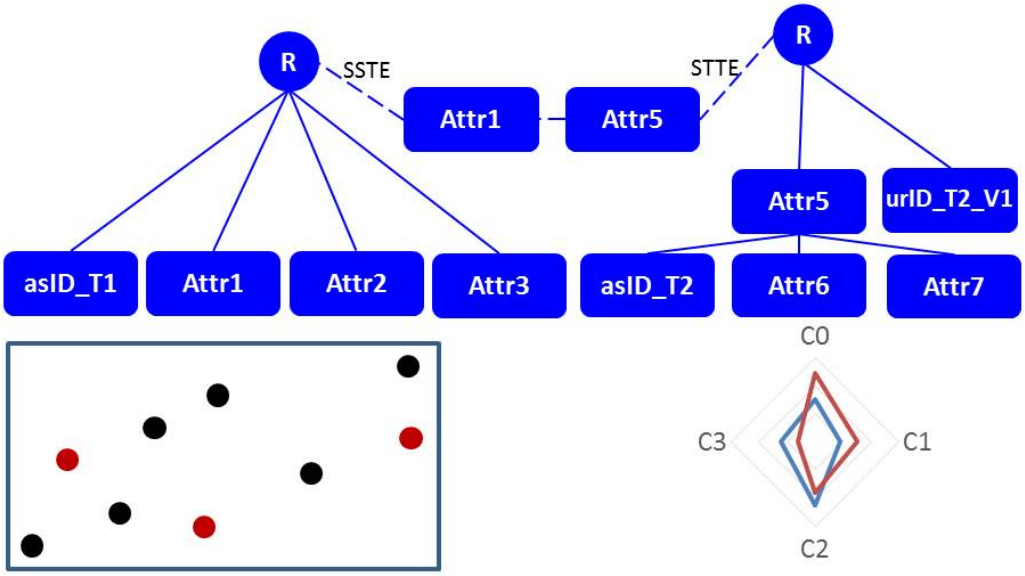


Figure 2.13: An example of relating the objects by the values of specified attributes

the system will read in the value "B0" of "Attr1" which is in the same row of the ID "asID_T1_0". Next, The system will check the value of the "Attr5" of each urID in the right SST. If the value is equal to 'B0', the state of the corresponding ID will be changed to 'T'. In this example, the state of "urID_T2_V1_0" will become 'T'. Then, the data object identified by ID "urID_T2_V1_0" and the connected visual object (blue line) in the right chart will be selected. In Section 3.3, we will use another example to show this function.

2.1.8 Recreating Napoleon's chart using our framework

Here we will show how to use our framework to recreate the Napoleon's Campaign chart. The source data are stored in two flat tables. The first has five attributes, the "longitude", the "latitude", the group", the "direction", and the "survivors". It records the number of survivors in each army group at each different key location during the campaign. We will define a flow map in which six polygons are used to represent the data in this table. The second table has two attributes, the "longitude" and the "temperature". It shows the temperature change during the retreat, which we represent as a line sub-chart. Finally, we will combine these two sub-charts together to recreate the chart.

To create the flow map, we use a polygon webble as a visualization template, and the Microsoft Bing map as the background. Each visual object represents the campaign path and the change of the total number of survivors in each army group in either the advance or the retreat direction. The vertical width of each path represents the number of survivors. First, we need to define the DVS from the two-level tree representing the flat source data table. Five Attribute nodes, which respectively denote the five attributes in the first data table, are in the second level of the DVS. We apply the "Group-by" operations to the Attribute nodes "group" and "direction" in this order. The tuples in this set are now divided into six sets, because the attribute "group" has three different values and the attribute "direction" has two different values.

We can use the three attributes, "longitude", "latitude" and "survivors" to define four new derived attributes "X_Upper", "Y_Upper", "X_Lower" and "Y_Lower". The expressions defining these derived attributes can be given as follows:

$$X\_Upper := Para1 \times longitude;$$

$$Y\_Upper := Para2 \times latitude + 0.5 \times Para3 \times survivors;$$

$$X\_Lower := Para1 \times longitude;$$

$$Y\_Lower := Para2 \times latitude0.5 \times Para3 \times survivors.$$

In these expressions, we use three parameters, "Para1", "Para2", and "Para3". They determine the mapping relation between the attribute values of each data object and the corresponding property value of each visual object. For example, the "Para3" and the value of the attribute "survivors" determine the width of the campaign path at each key location. In this example, the value of the attribute "survivors" may range from 6000 people to 340000 people. By setting the "Para3" to 1/2000, the converted value, namely the path width, may range from 3 pixels to 170 pixels.

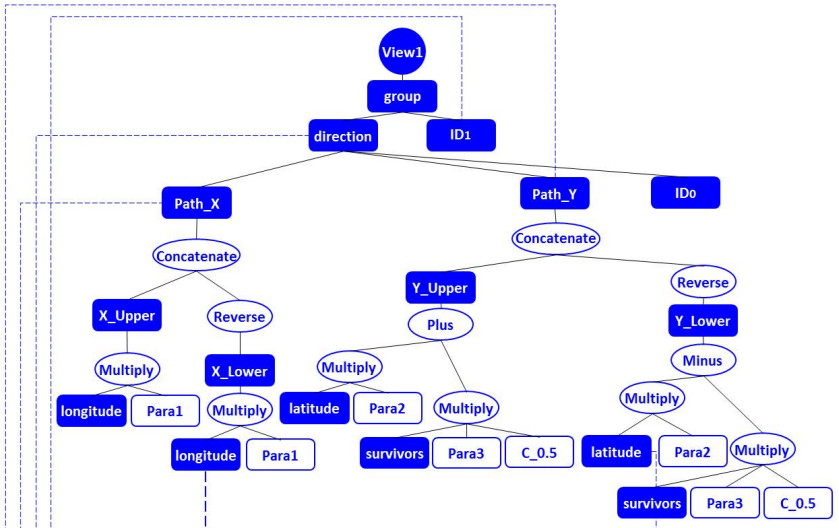
The values of these four derived attributes, respectively, represent the X and Y coordinates of the vertices along the upper and the lower borders of each polygon from left to right. We need to arrange the coordinate values of all the vertices in a clock-wise order to define a polygon by a sequence of vertices. Therefore, we reverse the order of the vertices in the lower border by using the Function node "reverse" as the parent node of the "X_Lower" node and the "Y_Lower" node. We then separately concatenate the "X_Upper" and the reversed "X_Lower", and also the "Y_Upper" and the reversed "Y_Lower", to define two new attributes the "Path_X" and the "Path_Y". Next, we can add an ID node as the sibling of the node "Direction" as shown in Figure 2.14 (a). Then the system will generate data objects to store the retrieved data, and automatically generate an ID for each data object.

Next, we define the CVS as shown in Figure 2.14 (b). This CVS has only three of the four subtrees, i.e. its Template subtree, Background subtree and Geo-Coordinate System subtree.

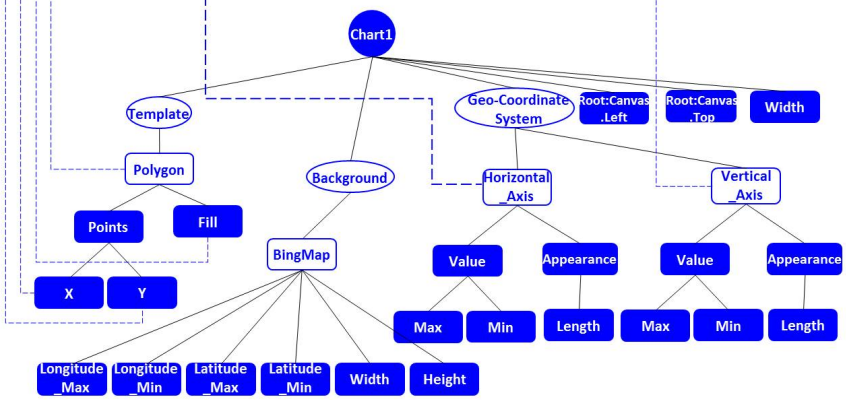
Next, we define linkages from the nodes of the DVS to the nodes of the Template subtree in the CVS using dashed lines (Figure 2.14 (a) and (b)). We then specify in the expression editor window that the width and the height of the background map object are equal to the lengths of the coordinate axes. This specification makes the coordinate system have the same size as the background.

We then define another DVS and another CVS for creating the temperature chart. They are shown in Figure 2.14 (c) and (d) respectively. Finally, we combine these two charts, by defining the relative position between them. The definition of the relative position of these two charts uses the expression editor windows of the three nodes in the CVS of the Chart2. These nodes are "Width", "Root:Canvas.Top" and "Root:Canvas.Left". The expressions we specify are as follows:

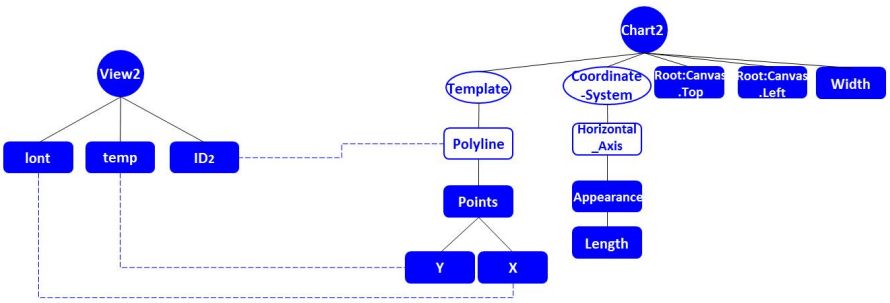
$$Chart2.Root : Canvas.Top = Chart1.Root : Canvas.Top + Chart1.Height;$$



(a) The DVS for creating the nonstandard path chart



(b) The CVS for creating the nonstandard path chart



(c) The DVS for creating the temperature chart (d) The CVS for creating the temperature chart

Figure 2.14: Schemata for defining the nonstandard path chart (the upper part) and the temperature chart (the lower part) of the Napoleon's campaign chart

$$Chart2.Root : Canvas.Left = Chart1.Root : Canvas.Left;$$

$$Chart2.Root : Canvas.Width = Chart1.Root : Canvas.Width$$

Based on these relations, the system will attach the temperature chart to the bottom of the path chart, and make these two charts have the same width. Figure fig:NapoleonChart shows the resulting composite visualization chart.

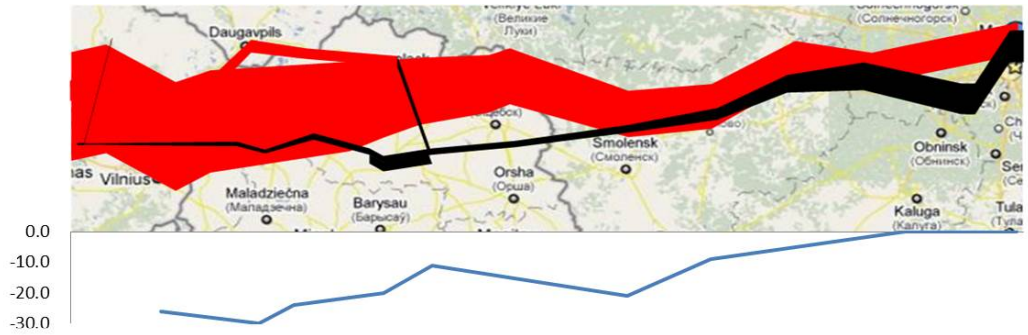


Figure 2.15: The chart of Napoleon's Russia champaign that is automatically recreated from DVSs and CVSs in Figure 2.14 using our framework

2.1.9 Extending Napoleon's chart using our framework

Here we will show how easily we can make extensions to the original Napoleon's Campaign chart. We first introduce color gradation only to the path taken by the main retreating troop to show the temperature change. Next, we add six pie charts at six key locations of the path taken by the main troop advancing to Russia. Each pie chart shows the composition change of the troop.

To create these pie charts, we use a fake data table "CorpsData" with four attributes, "longitude", "latitude", "corps", and "number". The attribute "corps" may take three different values, "cavalry", "infantry", and "artillery". The total number of these three different types of soldiers determines the diameter of each pie chart. We distinguish different sectors with three different colors "Cyan", "Orange" and "Yellow" respectively for the "cavalry", "infantry", and "artillery".

The DVSs and the CVSs for this chart are shown in Figure 2.16 and the visualization chart shown in Figure 2.17.

In this example, we reused the same DVSs "View1" and "View2" in Figure 2.14 for defining the data objects, and reused the same CVS "Chart1" in Figure 2.14

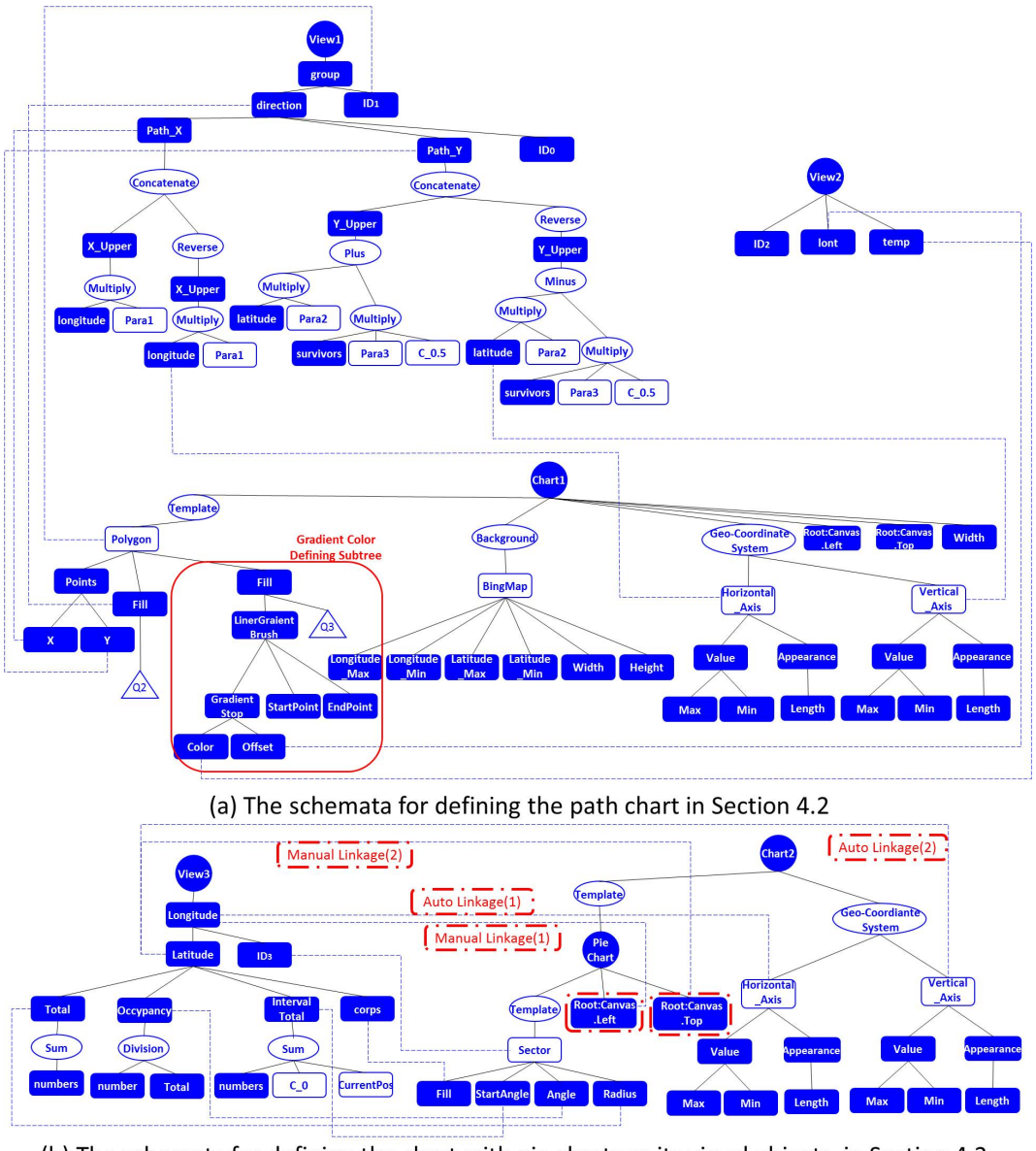


Figure 2.16: The schemata for creating the extended version of the Napoleon's campaign chart in Section 2.1.8

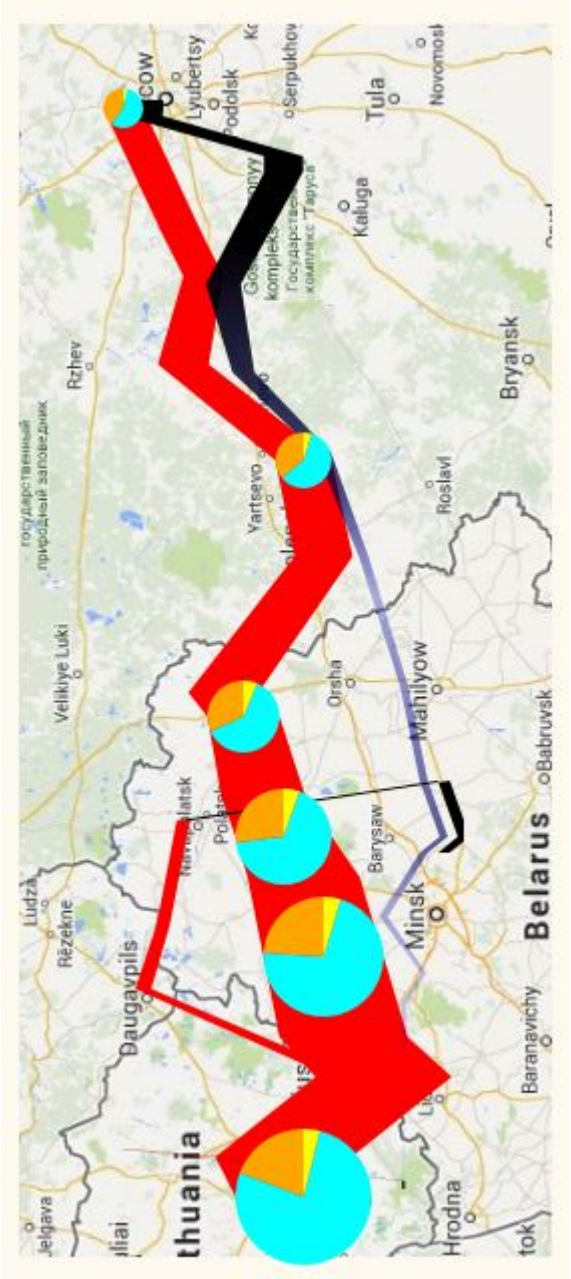


Figure 2.17: An extended version of the Napoleon's chart that is automatically created form DVSs and CVSs in Figure 2.16 by our framework

with an additional subtree as shown in the enclosing box in Figure 2.16 (a) to define the gradient color. In Figure 2.14, we connected the two Attribute nodes in the DVS "View2" to the two Property nodes of the CVS "Char2". In Figure 2.16, we connected these two Attribute nodes to the two Property nodes, "Color" and "Offset" of the newly added subtree, which defines the gradient color of the main retreat campaign path.

To create the six pie charts, we assume that we have a pre-defined CVS "Pie Chart" in the library, and then added it to the CVS "Chart2" as the child of its Template node. We also introduced a new DVS "View3" to represent the newly introduced table "CorpsData". This DVS defines data objects for creating these six pie charts. By linking the six Attribute nodes in "View3" to the six Property nodes in "Chart2", we can make the system automatically generate the six pie charts in Figure 2.17.

2.2 Template Composition Method

2.2.1 Composing a complex template by Transformer and Shadow Functional Components

This method is used to help users to expand the template library by combining primitive shape object together.[20] Using this template composition method, we can even create some complex templates without any programming, which is convenient for people who are less skilled at programming. A well-designed complex template can effectively and clearly represent multivariate data. In some situations, a well-designed visualization result can also increase the data-analysis speed or help people find out the hidden information of the data. It is important to provide users with an easy way to search an existing template or to compose visualization templates using primitive components. However, because of the diversity of users' needs, it is almost impossible to provide all the templates to satisfy their needs. Thus, we provide the template composition method to help users to create templates to satisfy their needs.

For the implementation of our framework, users can use primitive graphical objects that directly work as graphical-templates. They can also create composite templates by combining primitive graphical objects together. To create a composite template, we need to connect graphical objects together, and then to define the geometrical constraints among them to make them interoperable with each others. Two kinds of function object, Shadow and Transformer (Figure 2.18), are provided

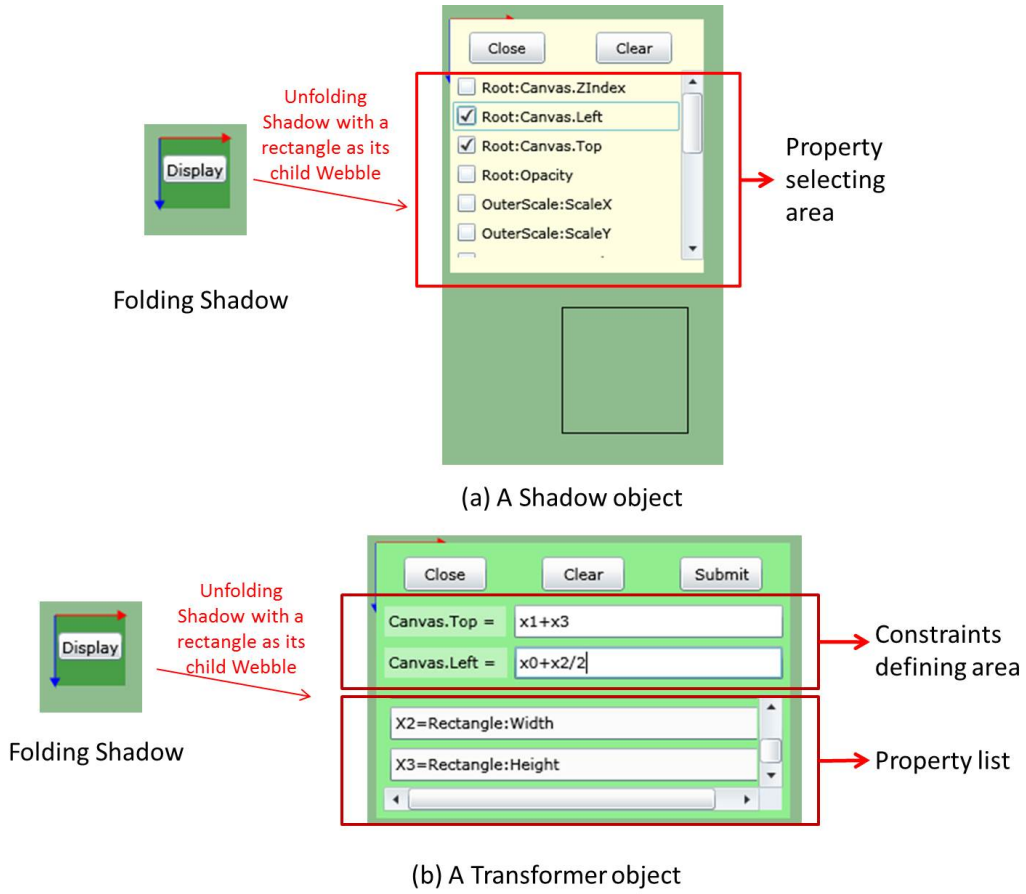


Figure 2.18: Shadow and Transformer for defining geometrical constrains

by our framework to define such kind of constraints.

Figure 2.18(a) shows a Shadow and the Figure 2.18(b) shows a Transformer. A Shadow should be the parent of a component of the composite template, and it is used to manage the visual properties of its child in two aspects. One is to extract the property information of its child. When an object is connected to a Shadow, the Shadow will read all of its child's visual property names and list them. Users can select the properties they need by checking the corresponding checkboxes. The Shadow will then create a message that includes the pairs of each selected properties' name and its value, and send out this message. The other function of Shadow is to renew the values of its child's visual properties according to the message sent to its #Input slot. A Transformer is an object used to define and to make a transformation between its #Input slot and #Output slot. The output messages of a

Transformer should also be the properties' name-and-value pairs which may be used by a Shadow to renew its child's visual properties. The property values included in the output message are generated by computing the transformation expressions that are specified by users.

Figure 2.19 shows how to define such constraints between two components in the meme media architecture. After connecting these objects as shown in Figure 2.19, the Shadow_1 extracts visual property information of Component_1 (Arrow①) according to the user's property selection. The list of the property names and values will form a message (Arrow②) and the message will be sent to the Transformer (Arrow③). The message is stored and transmitted in XML using the following format:

```
<Properties>
<Property Name="Property-Name1"
         Value="Property-Value1"/>
......
</Properties>
```

The information will be transformed according to the expressions specified in the Transformer (Arrow④). The transformed result will then be sent to Shadow_2 (Arrow⑤). Shadow_2 will generate the property modification commands, such as "XPosition, value", according to the received message (Arrow⑥) to change the corresponding property values of Component_2 (Arrow⑦).

As we explained above, the geometrical constraints among template components are defined in an external way. This external geometrical constraints definition method has two advantages. One is that any existing graphical object can be used as a template component without any modification. The other one is that the internal function of the component will be preserved after combining them together. Our framework also allows us to embed some additional functions in the visualization results, such as invoking some web search, just by connecting such components as child components. In section 2.2.2, we will use an example to explain such functions.

2.2.2 Case study

In this example, we use a composite record template to visualize stock price data as shown in Figure 2.20. To create this template, we need a rectangular object and two liner objects. They are connected with each other using Shadows and Transformers as shown in Figure 2.20. We use the Shadow1 to extract the information of

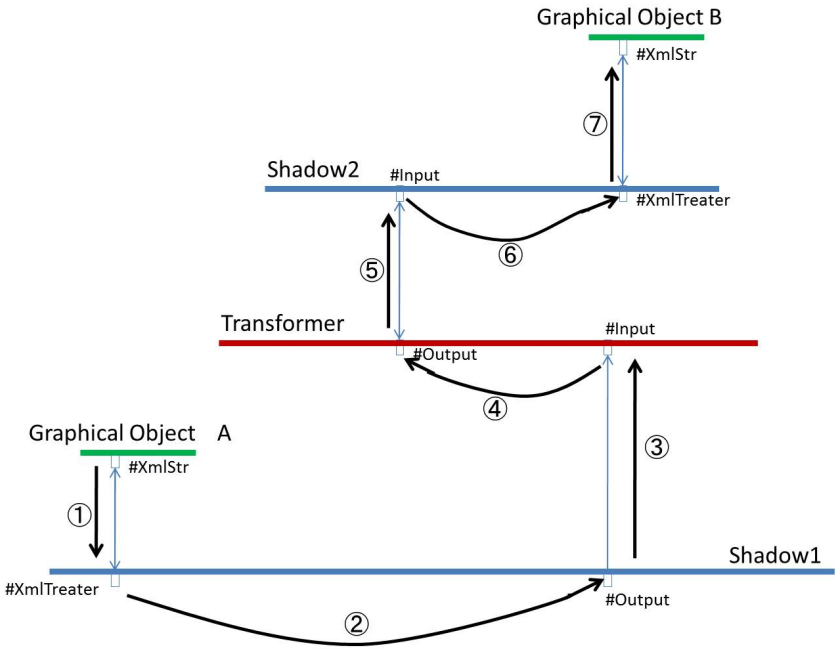


Figure 2.19: Specifying geometrical constrains among graphical objects using Shadows and Transformers

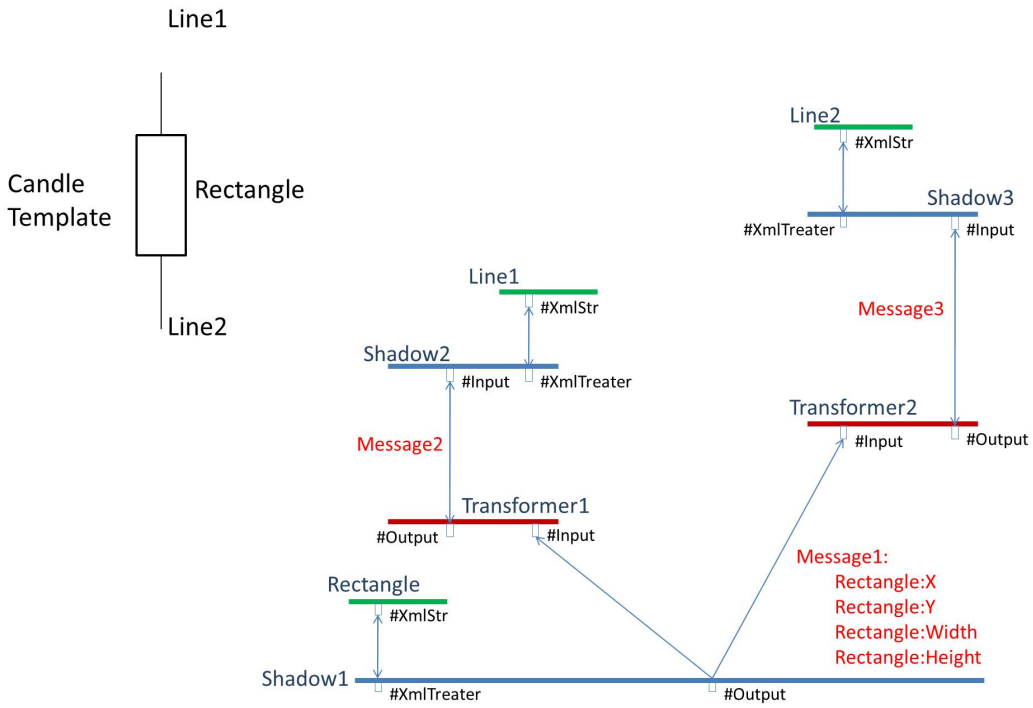


Figure 2.20: Composite record-template used for visualizing stock price information in candlestick chart

the rectangular object's four visual properties, Rectangle:X, Rectangle:Y, Rectangle:Width and Rectangle:Height, to generate Message1. Message1 will then be sent to two Transformers separately. The information of extracted properties is stored and transmitted using XML document.

In the two Transformers, Message1 will be transformed according to the specified expressions of two Transformers to generate Message2 and Message3. In this example, the expressions in two Transformers are specified as shown in Table 2.4. Message2 and Message3 will then be sent to Shadow2 and Shadow3 respectively. After receiving these messages, the Shadows will modify the visual property values of Line1 and Line2 according to the two messages.

We can also utilize the graphical objects' internal functions to append some additional functions to the template. For example, we can use a Google search object as the rectangle component of the template. If users click this object, it will pop up a window to show the Google search result of the key word(s) specified in its #Keyword slot. When we perform the visualization, we associate the #Keyword

Table 2.4: Expressions specified inTransformer1 and Transformer2 in Figure 2.20

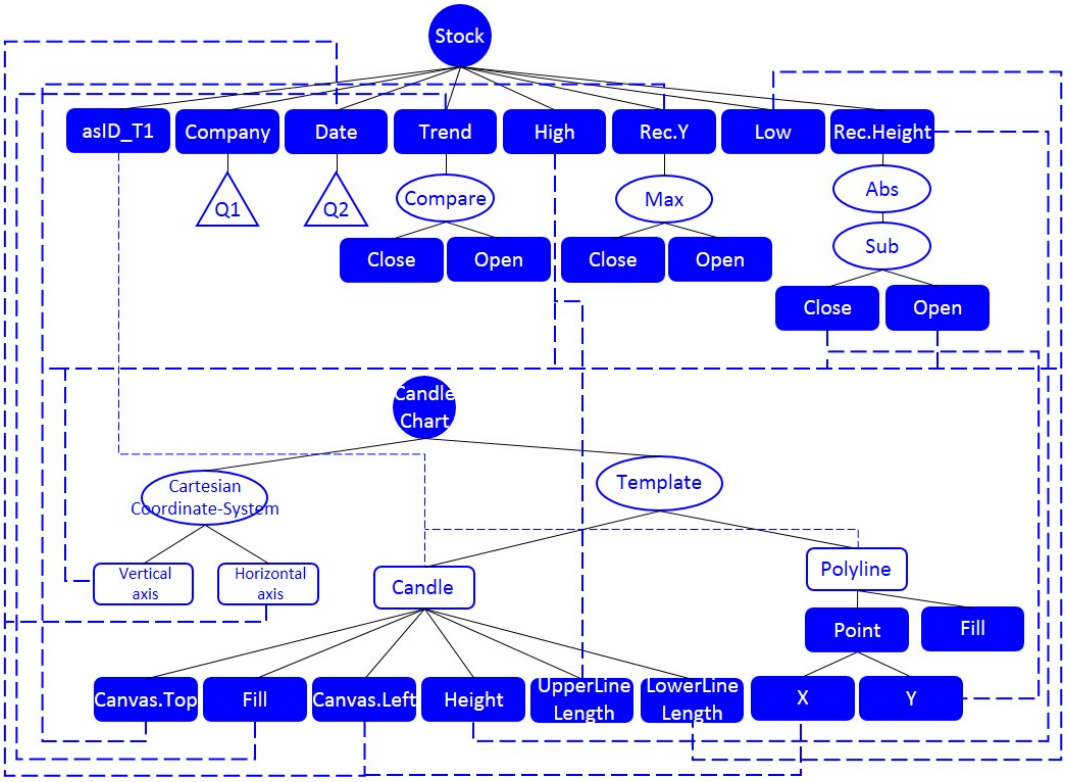
Transformer	Expressions
Transformer1	Line1:X = Rectangle:X + Rectangle:Width/2
	Line1:Y = Rectangle:Y
Transformer2	Line2:X = Rectangle:X + Rectangle:Width/2
	Line2:Y = Rectangle:Y + Rectangle:Height

slot with the Stock Name of the original data. If the user clicks the rectangular Google search object of a visual item in the visualization result, the Google search result of the corresponding stock will be shown in a pop-up window.

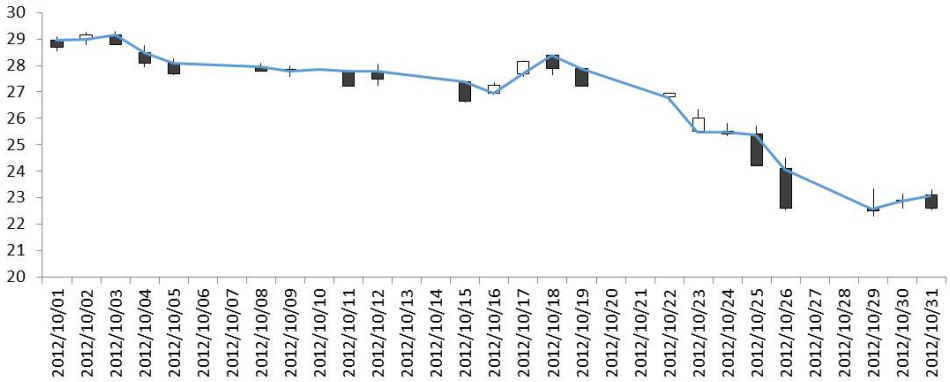
In this example, we visualize a data table with six attributes. They are "Company", "Date", "Open", "High", "Low" and "Close". In this table, there are the daily stock-price information of all companies in 2011. We try to create two different charts. The first chart represents the stock price change of the Company "ComA" in October and the second one extends the first chart which includes the information of three different companies. We use a candle-shape graphical object as the visualization template to represent the different stock prices of one company on one day. If the start price is higher than the end price in the same day, the corresponding candle-shape visual-object will be filled with black. In each chart, we further add one or three poly-lines to the chart to indicate the change trend of the close prices.

First, we define the DVS of the first chart. We need to add two quantification nodes "Q1" and "Q2" as the children of the attribute nodes "Company" and "Date" to obtain the data we need. The condition in "Q1" is "Value=ComA", and the condition in "Q2" is "Value≥2012/10/1 & Value≤2012/10/31". Next, we manipulate the other nodes to define the DVS as the upper tree in Figure 2.21(a). Next, we define the CVS for this chart and binding the nodes between the DVS and the CVS (Figure2.21(a)). Finally, the system will automatically generate the result chart as shown in Figure 2.21(b).

We extend this first chart by adding the information of other two companies. We need to change the condition in the node "Q1" and then perform the group operation to the attribute "Company". We set the condition in "Q1" as "Value=ComA&ComB&ComC". The DVS of the new chart is shown in Figure 2.22(a). The bindings between the new DVS and the CVS will not be changed, except wiring the node "urID_T1_Stock" of the new DVS to the node "Polygon" of the CVS to specify the data objects will be used to three different lines. The



(a) The DVS (upper) and CVS (lower) for visualizing the stock price change of the company "ComA" in October



(b) The chart created by defining the schemata in (a)

Figure 2.21: The stock price chart of the company 'ComA' in October and the schemata for creating the chart

resultant chart is shown in Figure 2.22(b).

2.3 Applying Our Framework for Creating GeoVisualization

2.3.1 Special functions for supporting to create GeoVisualization

In this framework, we provide three chart interactions, i.e., zooming in/out, panning, and brushing and linking to help users to flexibly explore the generated charts, and the following three functions for supporting the creation of GeoVisualization Charts.

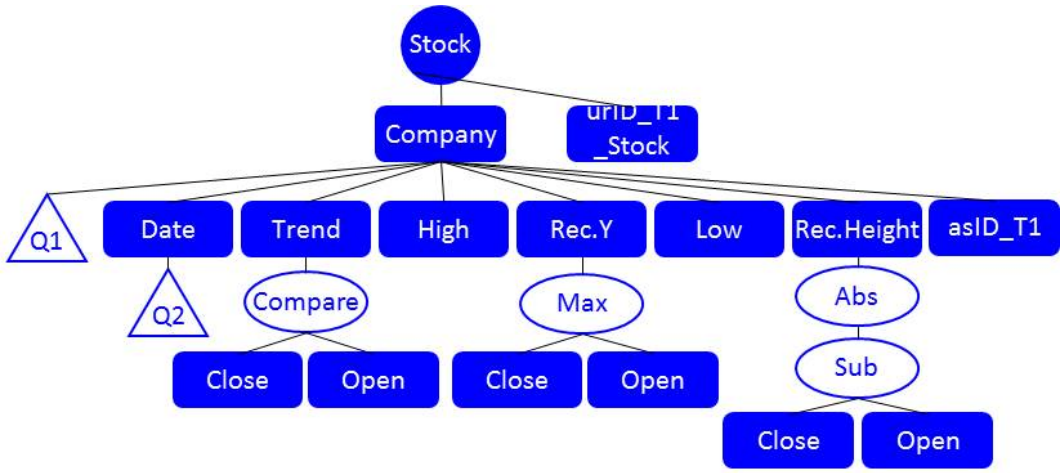
First, our framework can automatically define the respective mapping from the longitude and latitude attributes of a data object to the X and Y coordinate properties of a visual object. Our framework will check all the attributes of the original data set. Any attributes with names contain the string "lon" (longitude) or "lat" (latitude) are respectively mapped to the X and Y positions of the visual objects. If users are not satisfied with the auto-mapping result, they can reject or modify these mappings through manual definition, which was introduced in Section 2.1.6.

Second, the source database may use the geographic name to specify the location of each visual object. Users can use two special function nodes "GetLat" and "GetLon" to separately define the latitude and longitude attributes from a geographic name. As shown in Figure 2.23, the "Long" and "Lat" nodes respectively use "Getlat" and "GetLon" node as their child nodes. These two function nodes use attribute nodes whose values are geographic names as their child nodes. Our framework will automatically query a web service (for example, latlong.net) to obtain the latitude and longitude values of each geographic name. Next, our framework will automatically define the mapping from the converted subtree to the CVS.

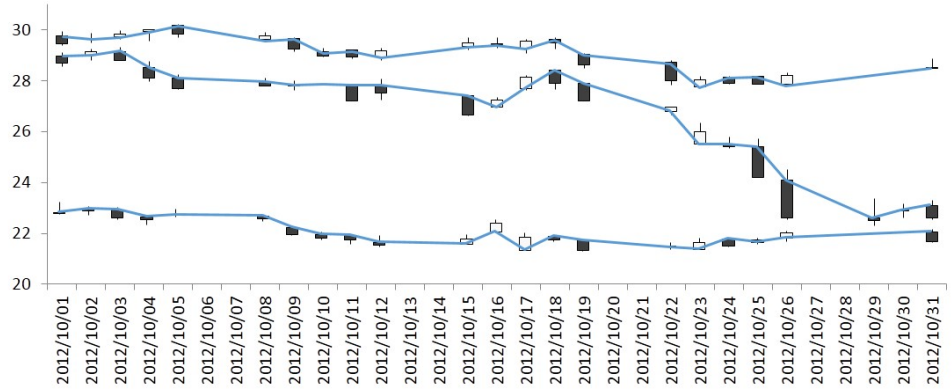
Third, our framework can automatically adjust the size and the display area of the map object used as a chart background. Our framework will first check the Max and Min values of the longitude and latitude attributes. Then according to the required size and the map area of the result chart, our framework will automatically adjust the used map object to make the background display correctly.

2.3.2 Case study

In this section, we will show how to use our framework to create a 2.5D GeoVisualization chart, which represents the trajectories of two typhoons that came close to Japan at different times.



(a) The modified DVS for visualizing the stock price change of three companies in October



(b) The chart created by defining the schemata in (a)

Figure 2.22: The stock price chart of three companies in October and the DVS for creating the chart

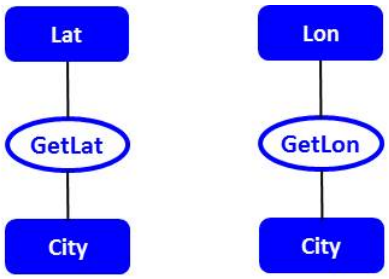


Figure 2.23: The Function nodes "GeoLat" and "GeoLon" and the subtrees for defining the latitude and longitude attributes

In this chart, the Z coordinate axis denotes the duration time. The bottom panel is an ArcGIS map object. The typhoon data is obtained from National Institute of Informatics of Japan. We used the center positions of the bottom faces of a set of cylinders to represent the trajectory of a typhoon. The radius of a cylinder's bottom face represents the radius of Major Storm Axis. We use the red and blue color to respectively show the trajectories of the Typhoon No. 201311 from Aug. 8th to Aug. 18th, 2013, and No. 201323 from Sep. 19th to Oct. 7th, 2013. We also added a text with a link in each cylinder. Users can click the text to open a web page to check related meteorological satellite observation images.

In Figure 2.24, we show the CVS and DVS for defining this chart, and Figure 2.24 shows the resulting chart.

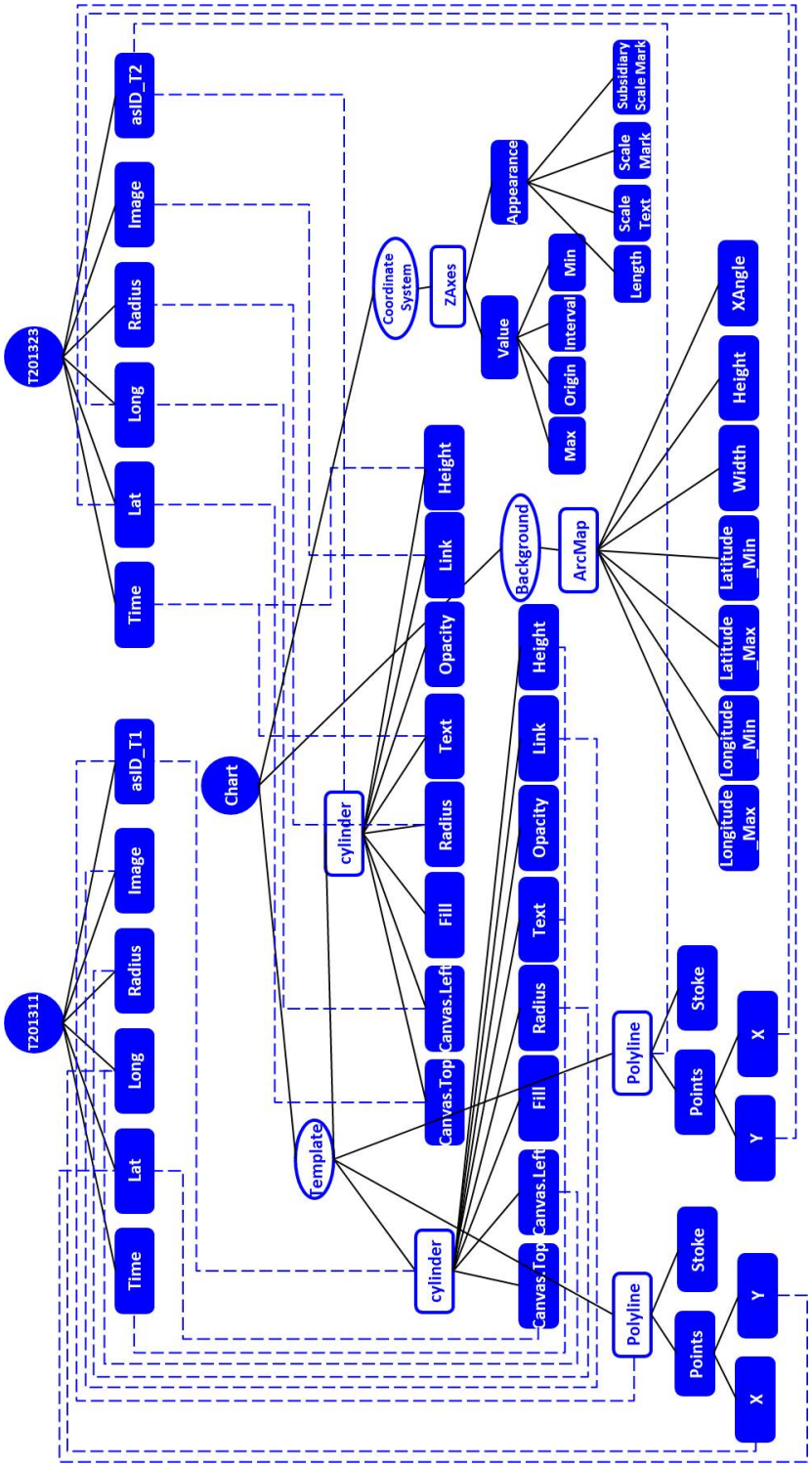


Figure 2.24: The DVSs and CVS for defining the typhoon chart in Figure 2.25

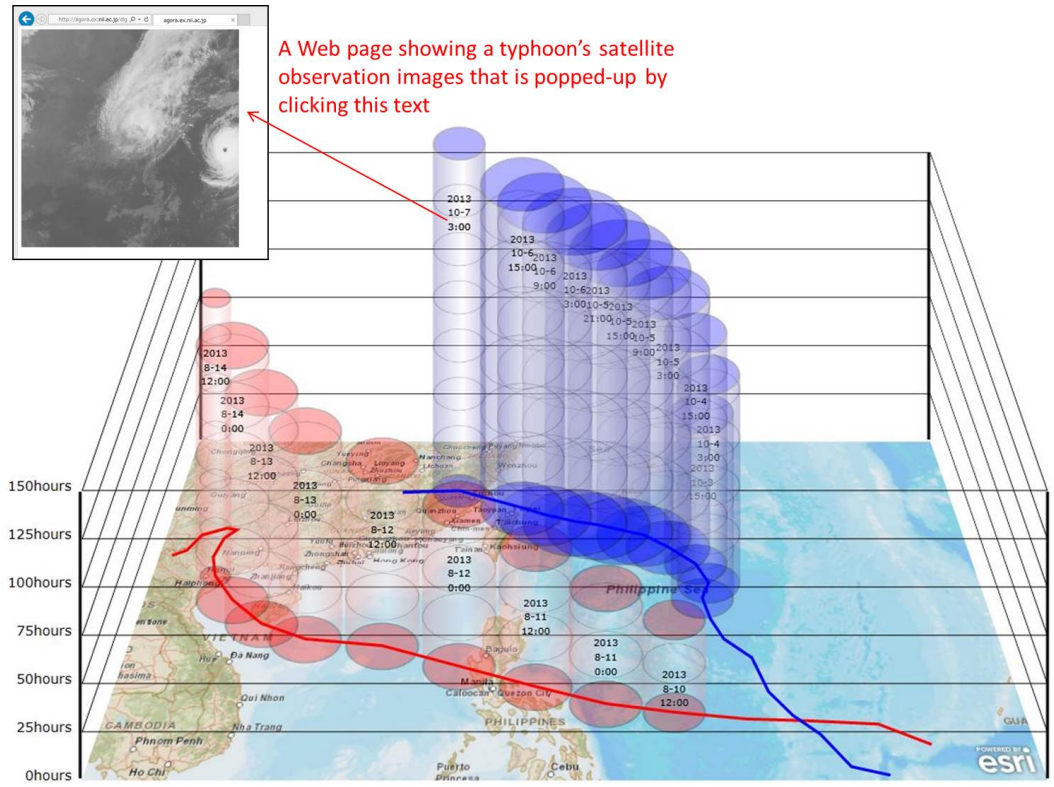


Figure 2.25: The resulting 2.5D typhoon chart defined by the DVSs and the CVSs shown in Figure 2.24

Chapter 3

Manipulating Visualization Charts

3.1 Common Interactions

Interaction is an important technology in information visualization. Interacting with a chart can help users to obtain more information than only observing this chart. Robert Spence introduces the functions of these interactions in his book[24]. In the existing visualization systems, many common interactions are applied. According to the different reactions of the visualization system to these interactions, we can classify most interactions in the following three types.

1. We users perform the "Zoom in/out" interaction and the "Pan" interaction, the system will change the mapping relationship between data objects and visual objects. For example, we users pan a chart, the mapping relationship between some attributes of the data objects with the position properties of visual objects will be changed. In these situations, the systems do not need to define new queries or re-query the data sources.

2. Users can define quantifications on some visual property. This interaction is always realized though a filter. After users define a filter, the system normally defines a new query with these defined quantifications. After re-querying the database, the system will obtain the data satisfying users' requests, and then visualize the new data. "Semantic zoom" and "Details on demand" also are the interactions in this type.

3. The last type of interactions is realized based on the linkage between data

and charts. "Linking and brushing" and "Multiple views" are two typical interactions in this type. By these interactions, different visual objects in the same charts or in the different charts are related with others though the data objects, which are used to define them. Such interactions normally do not change any part of the visualization definitions.

The interactions in different visualization may be different. The same interaction also may be realized by different methods. The above classification may not suitable for all the interactions in all the visualization systems. How to design the interactions and realize them in a generic method is still discussed by many researchers. In this thesis, we only discuss the interactions provided in our framework and their realizing methods.

3.2 Interactions of Manipulating the Intermediate Model

Beside the common interactions introduced above, our framework also provides a set of special interactions. Instead of directly manipulating the DVS and the CVS to define the query and the visualization, our framework also allows users to indirectly manipulate these two trees by directly manipulating the visualization results. [21] Users can

(1) define the binding between some attribute of the data set and some visual property of the visualization template by directly filling in an input form on a window called the binding editor which can be opened by right-clicking one of the visual objects in the visualization result,

(2) define a binding between two properties of two chart components to define the value-equality between them by filling in an input form in the same binding editor in (1)

(3) merge or divide visual objects

(4) use the legend or a visualization as a filter, or

(5) embed a graphical object or a visualization to another existing visualization as its new visualization template.

For performing the operations (1) and (2), we use the object called the binding editor. For the operation (1), the user can select a property of the visual object, a data table, and an attribute of the selected data table in the editor (Figure 3.1 (a)).

Similarly, for the operation (2) the user can select a property of the selected chart component, another chart component and a property of the second chart component in the binding editor (Figure 3.1 (b)).

According to the user operations on the binding editor, the system will automatically bind a node of the DVS and a node of the CVS, or bind two nodes of the CVS.

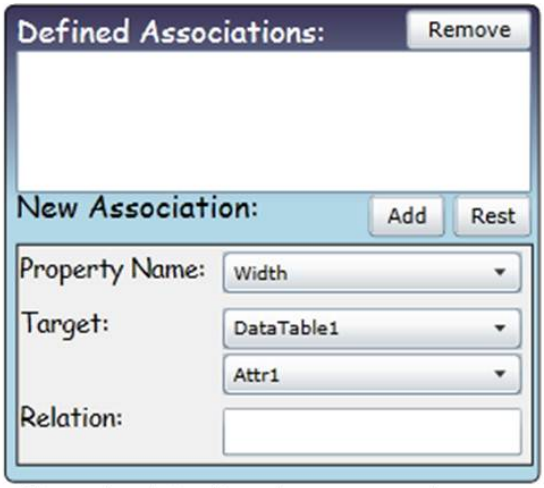
For the operation (3), users can select a visual object or a set of visual objects and use the menu to perform the merging or dividing operations. The system will ask which attribute(s) should be used to perform this operation. Figure 3.2 is an example for explaining these two operations. In this example, the visualization result, the DVS and the tabular representation of the retrieved data are shown.

Our framework allows users to quantify the values of the attributes by specifying a region on the legend or on the visualization (Figure 3.3). According to the specified region on the legend, the system will calculate the corresponding value range of the attribute associated with this legend. Then a quantification node will be added to the DVS as the child of the node representing the bounded attribute, and set the obtained value range as the quantification condition.

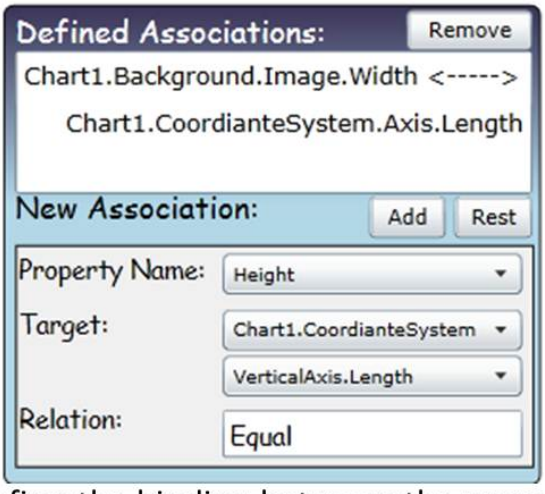
Users can also specify a polygonal region on the visualization to select all the visual objects in this region. The system can filter out the visual objects that are not selected in the visualization. The system will add the quantification nodes to the DVS as the child of its view node, and set the object IDs of the selected visual objects as the quantification condition.

If users set the visualization VS1 as the filter of the visualization VS2, and select the visual objects in VS1, the system will find out the corresponding visual objects in VS2 using the mechanism introduced in Section 2.1.7, and filter out the other visual objects.

For realizing the operation (5), users first need to load a graphical object "gObject" from the library or load an existing chart "newChart". Then users can drag it on an exiting visualization "existChart" and drop it there. Then the embedded object will become a new visualization template of the "existChart". Our framwork will insert a copy of the corresponding sub-tree whose root node represents "gObject" or "newChart" into the CVS of the "existChart" as the child of the template node. Section 3.3 shows an example of creating a nested chart by adding a pie chart

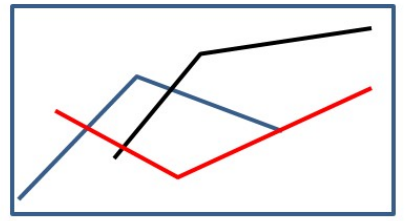


(a) Define the binding between the property Width of the visualization template and the attribute Attr1 of the DataTable1



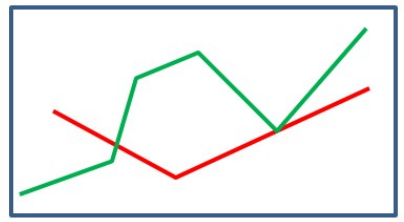
(b) Define the binding between the property Height of the background and the property Length of the vertical coordinate axis

Figure 3.1: The binding editors for defining two kinds of bindings



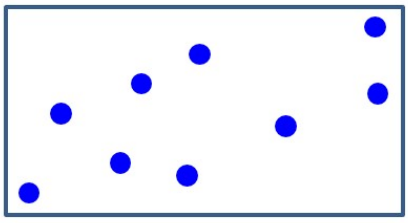
Ungroup-by Attr1

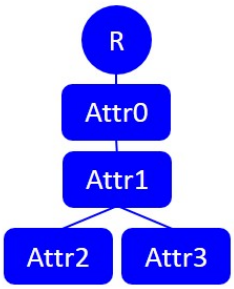
Group-by Attr1

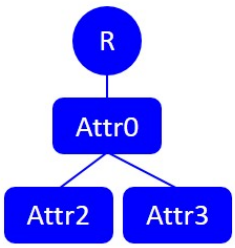


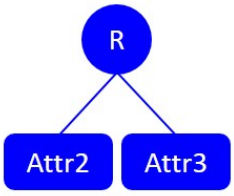
Ungroup-by Attr0

Group-by Attr0









Attr0	Attr1	Attr2	Attr3
A0	B0	0	0
		5	8
		10	5
	B1	3	4
		7	9
		13	10
A1	B0	2	6
		6	2
		13	7

Attr0	Attr2	Attr3
A0	0	0
	5	8
	10	5
	3	4
	7	9
	13	10
A1	2	6
	6	2
	13	7

Attr2	Attr3
0	0
5	8
10	5
3	4
7	9
13	10
2	6
6	2
13	7

Figure 3.2: Two situations for performing group-by or ungroup-by operation on the chart

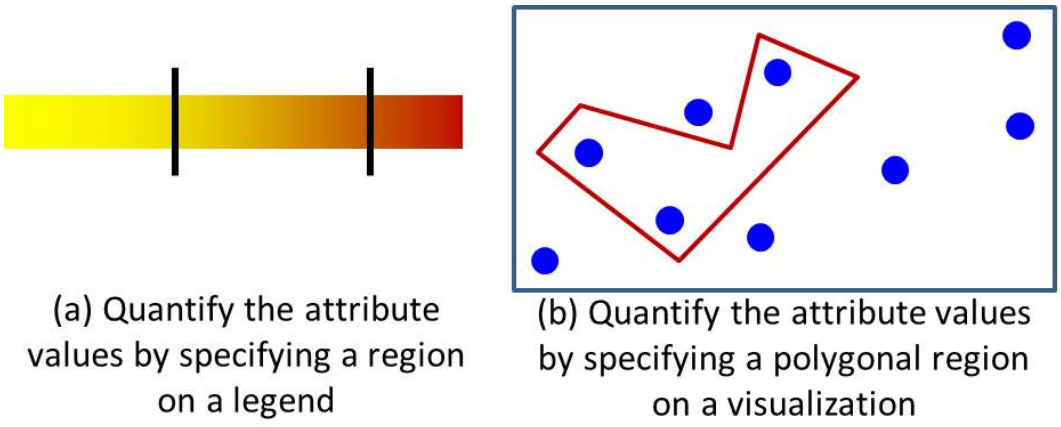


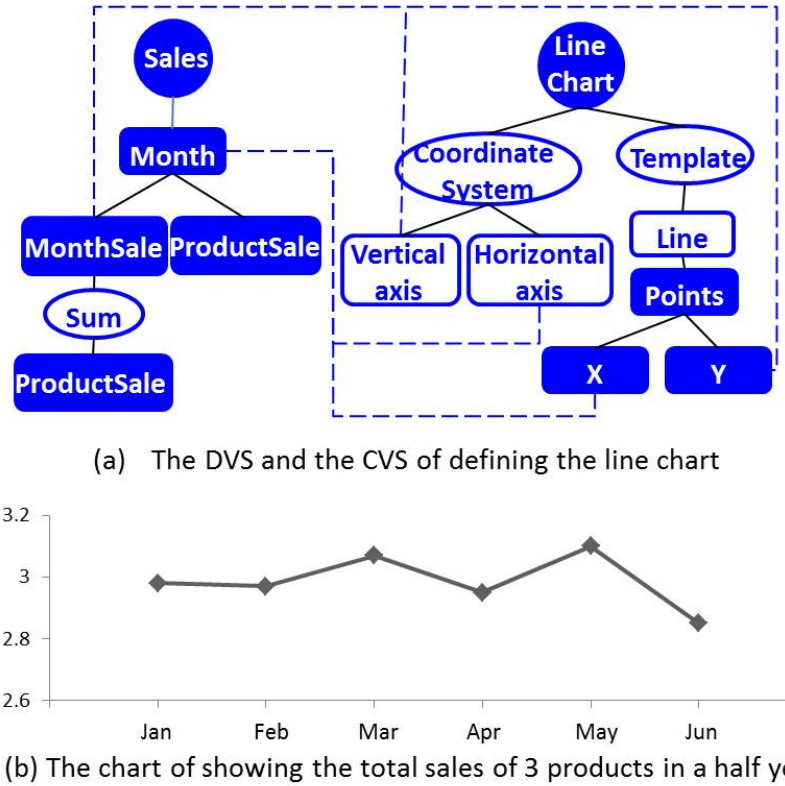
Figure 3.3: Quantifying the attribute value or visual objects by specifying a region on a legend or on a chart

as the new template to a line chart.

3.3 Case Study

In this example, we first create a line chart to visualize the change in total sales of three products during half a year. The DVS and the CVS are shown in Figure 3.4 (a), and the line chart are shown in Figure 3.4 (b). Now, in the resultant line chart, we want to add a small pie chart at each vertex of the polyline (Figure 3.5 (b)). Each pie chart visualizes the sale proportions of the three products in each month. To create it, we can first load a pre-defined pie chart from a library of charts, and drag and drop it onto the line chart. This will make the system automatically add the CVS used to define the pie chart as the child of the template node of the CVS used to define the line chart (Part I of Figure 3.5 (a)). Next, we can open the binding editor of the pie chart and define two bindings, i.e., one between the horizontal position of the pie chart and the attribute Month of database view, and the other between the vertical position of the pie chart and MonthSale (a derived attribute) of the database view. These operations will make the system automatically copy the pie chart five times and put them at the five vertices of the polyline in the line chart. Next, users can use the Attribute window and Attribute Details window to define the two attributes, Occupancy and IntervalSum, using the subtrees in Part II of Figure 3.5 (a)).

We click some sector of an arbitrary pie chart to open the binding editor to

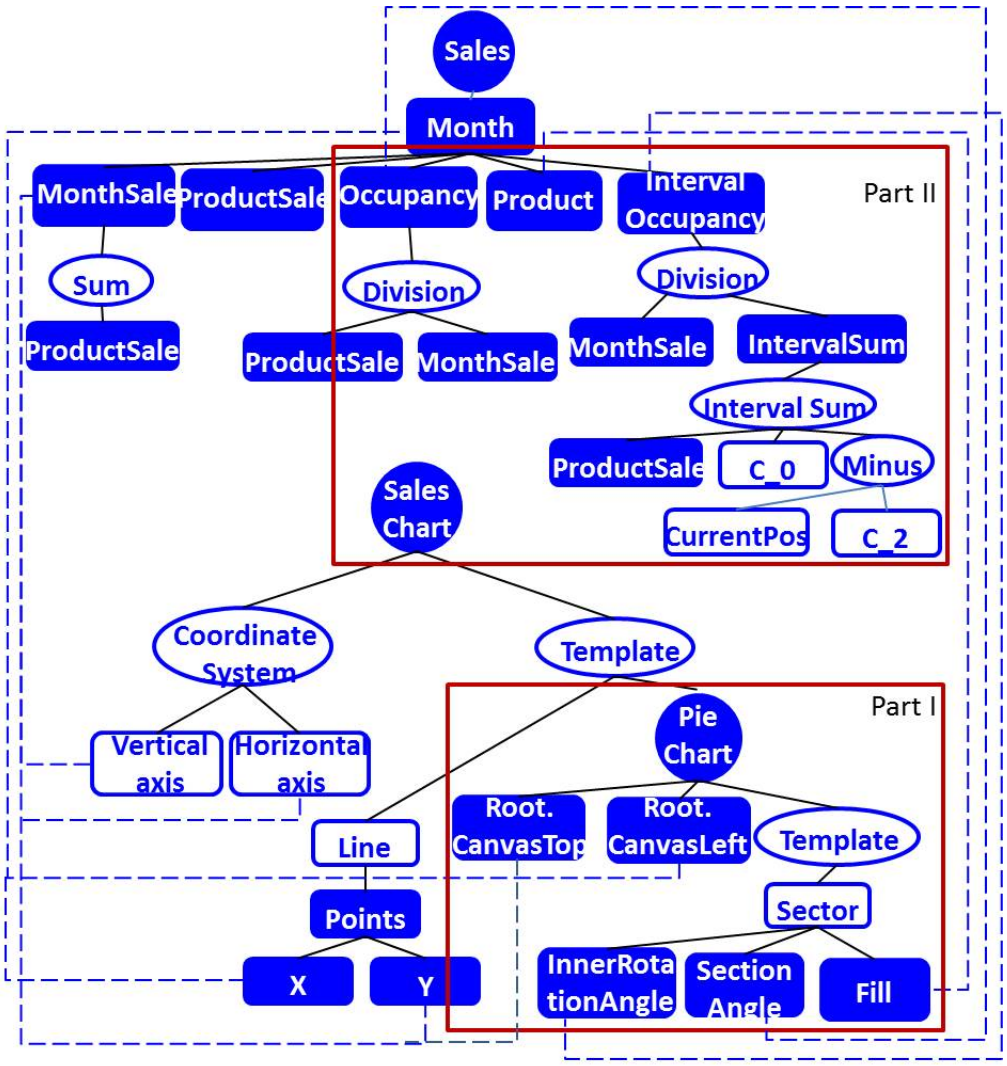


(a) The DVS and the CVS of defining the line chart

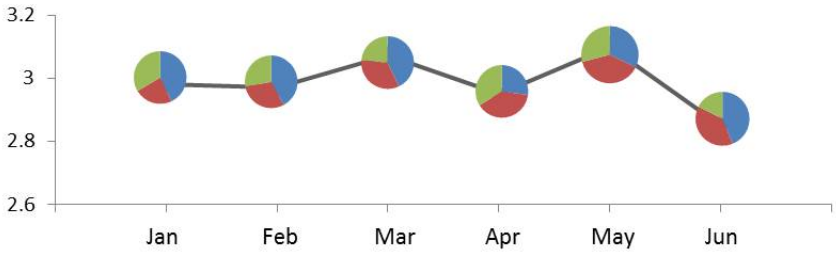
(b) The chart of showing the total sales of 3 products in a half year

Figure 3.4: The schemata for defining a line chart to represent product total sales and the resultant chart

define two new bindings, one between the property SectorAngle and the attribute Occupancy, and the other between the property InnerRotationAngle and the attribute IntervalSum. This will make the system automatically generate the final visualization result as shown in Figure 3.5 (b).



(a) The DVS and the CVS of defining the nested chart



(b) The nested chart generated by directly manipulating (a)

Figure 3.5: The schemata for defining a nested chart by manipulating the line chart in Figure 3.4 and the resultant chart

Chapter 4

An Extended Framework for Visualizing Data Retrieved from Semantic Web

In this paper, we will extend our framework for supporting users to exploratorily search the Semantic Web resources, and use the search result to define charts.[19] In this extended framework, we provide an extended view node. We name such extended view node the Semantic Web node (SW node in Table 2.1). It uses the diamond shape as its appearance. After specifying a SPARQL endpoint, users can access the corresponding Semantic Web dataset through this node.

Users can select the "Show/Hide Semantic Web Search Workspace" in its right-click menu of the SW node to open a workspace as shown in Figure 4.1. In this workspace, users can exploratorily extract and expand a schema of RDF data. On this schema, users can visually define the SPARQL query. We re-implement the method of the Semantic Web search proposed by Piao. According to users' manipulations in this workspace, our system will automatically generate an extended DVS, whose root is a SW node, as the view of Semantic Web resource. The generated SPARQL query is also stored in this SW node. Users can manipulate an extended DVS as manipulating other normal DVSs.

4.1 Components Provided by the ESVSW Framework

In ESVSW framework, these are two basic components. One is the class component, which can automatically search and display properties of a class. The other is a

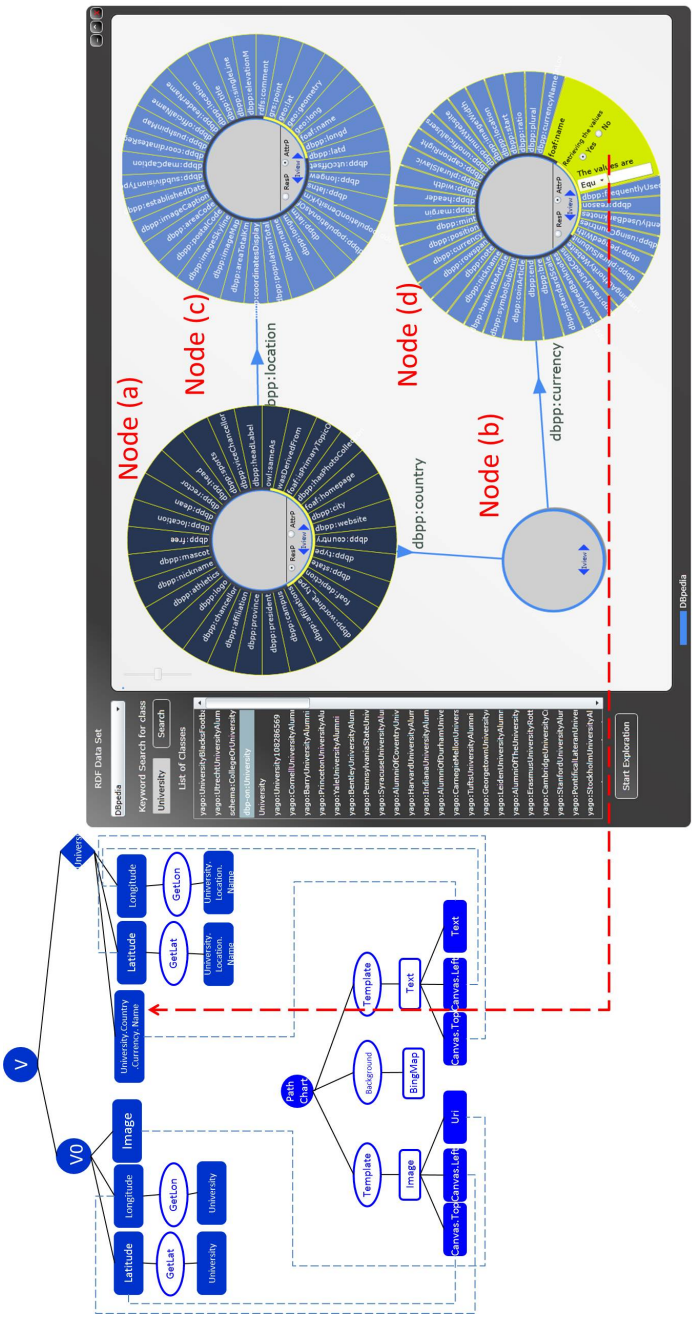


Figure 4.1: An example of using the ESVSW framework to search the Semantic Web resources

visualization component, which provides a table for displaying the retrieved data, and allows users to define several types of charts. In our extended framework, we only reuse the function of the Semantic Web search in this framework.

In an RDF data set, each class has multiple properties. Piao classifies these properties into two categories. One is named the relationship property, and the other is named the attribute property. The value of each relationship property is a set of resources identified by their URIs. The value of each attribute property is a set of literals.

Each class is represented by a class component, which is represented as a class node. It stores all the URIs of the instances belonging to this class. Each property is shown by a sector surrounding the center of the class node. The relationship property is dark blue, and the attribute property is light blue. Users can use the radio button to select which kind of properties is shown. As shown in Figure 4.1, Node (a) is a class node with its relationship properties being shown. Node (c) and Node (d) are two class nodes with their attribute properties being shown. Node (b) is a node without any property being shown.

4.2 Exploratorily Searching Semantic Web Resources

In order to search users' interested information from Semantic Web, two steps are necessary. In the following sections, we use an example shown in Figure 4.1 to explain how to exploratorily search Semantic Web. In this example, we want to search: (1) the locations of the universities stored in the DBpeida; (2) the currencies used in the countries where these universities belong to.

STEP1: Extracting and expanding the schema of RDF data

In order to search Semantic Web, users need to first specify a SPARQL endpoint and a keyword. The end point determined which RDF database will be accessed. The system uses the specified keyword to rename the SW node, and saves the selected end point to the SW node as its property. Then through the SW node, the system will search the classes whose names contain the specified keyword in the specified RDF database. The search result will be shown in the "List of Classes". Users can select a class in this list as the start class to extract and to expand the schema of RDF data.

In the example shown in Figure 4.1, we choose the DBpedia as the SPARQL endpoint and use the "University" as the search keyword. Then, we choose the class "dbp-on:University" in the "List of Classes". Next, the system generates the class

Node (a). We can click the Node (a) to request the system to extract the properties of this class and to show them.

According to our search target, we separately click the relationship properties "dbpp:county" and "dbpp:location" of Node (a). Then the system will generate two new class nodes, Node (b) and Node (c). There is an edge directed from Node (a) to each of the new generated node. The title of the edge is the same with the clicked property. Next, we further expand this local schema by clicking the property "dbpp:currency" of the Node (b) to generate the Node (d). Then we obtain the schema of RDF data which includes the classes of our desired resources.

STEP2: Visually defining a SPARQL query

After users obtain a schema of RDF data, they need to manipulate the schema to specify what resources they want to retrieve. Our system will automatically translate these specifying operations to a SPARQL query. The schema extraction and expansion is realized by using the relationship properties of each class, while the SPARQL query definition is realized by using the attribute properties of each class.

From all the attribute properties of each class node, users need to choose their interested one by clicking it. Then thesystem will show a query definition panel in the sector of this clicked attribute property. In the Figure 4.1, the Node (d) shows the query definition panel of the property "foaf:name".

Each attribute property has its own query definition panel. Users can use the radio button to specify whether the value of this property will be retrieved. As shown in Node (d) of the Figure 4.1, we select the "Yes" to retrieve the value of the property "foaf:name". Users also can specify a quantification condition to filter out the retrieved data in this panel.

In this example, besides of the property "foaf:name" of the Node (d) for retrieving the name of the currencies, we also select the property "foaf:name" of the Node (c) for retrieving the addresses of these universities, and the property "foaf:name" of the Node (d) for retrieving the names of these universities.

According to users' operations on the schema of the RDF data, the system will automatically generate a SPARQL query. Each manipulation on the schema will simultaneously update the generated query. The SPARQL query of this example is shown in Figure 4.2. The generated SPARQL query will be stored in the SW node.

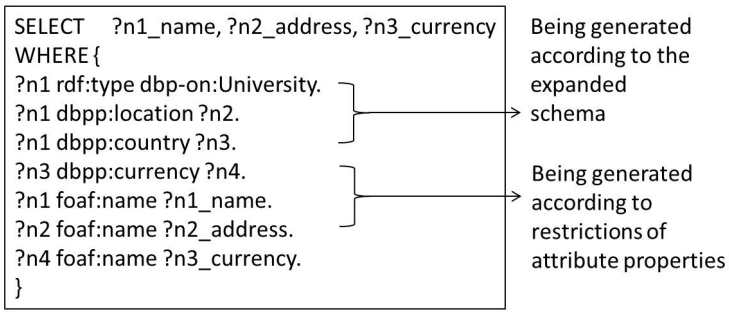


Figure 4.2: The SPARQL query automatically generated in the example of Section 4.1

4.3 Defining a View of the User-Interested Web Resources Using a DVS

A SW node can be used as the root of an extended DVS, which defines a view of the Semantic Web resources. The extended DVS is generated in a dynamic way during the process of the visual definition of a SPARQL query (Step 2 in the last section). Our framework will generate corresponding attribute nodes simultaneously according to users' operations in the workspace introduced above. When users click the "Yes" button on the query definition panel of some attribute property of a class, the system will generate the corresponding attribute node and set it as the child node of the SW node. As indicating by the arrow in Figure 4.1, when we click the "Yes" button in the property "foaf:name" of the Node (d), our system will generate an attribute node, name this node the "country.currency.name", and add this node as the child of the SW node. In the example shown in Figure 4.1, the system will finally generate a DVS including three attribute nodes.

Each automatically generated attribute is named, using edge titles and the selected property names, in the following way: *edgeTitle1.edgeTitle2. .propertyName.* The titles of all the edges from the start class node to the selected class node should be included in the name. Users can also rename this attribute node in the DVS. Users can hide the workspace of searching Semantic Web, and revoke it to continue the search as they want through the right-click menu of the SW node.

Users can further manipulate the extended DVS as they manipulate other normal DVSs. Users can perform the "Group-by" in the extended DVS to define the hierarchical relationship between different attributes. Users can also join an extended DVS and a normal DVS to integrate the data from a local database and the data from

Semantic Web. Such integration is ignored by most researchers. By manipulating the CVS in our framework, users can easily define both standard charts and non-standard charts to visualize the data from Semantic Web in our unified visualization framework.

As shown in the left hand side of Figure 4.1, there are two DVSs which respectively define a view of the local database and a view of the Semantic Web resources. We also define a CVS, and define the visual mapping from these two DVS to the CVS. According to these specifications, our system will automatically generate a resultant chart.

In order to realize the above functions, and to deal with data sets in a unified way, we add a data table in the local database for temporarily storing the data retrieved from Semantic Web. Each SW node has such a temporary table in the local database. When the system starts to visualizing process, the system will first evaluate the SPARQL query stored in the SW node. The evaluation result will be stored in the corresponding temporary table. Then the system will manipulate the data table as well as the original data tables in the local database using the NHibernate technology. When users close our visualization system or delete the extended DVS, the temporary table will be deleted.

4.4 Case study

4.4.1 An modified chart of Napoleon's Russian campaign

We explained the details on how to create this chart in Section 2.1.8. Here, we want to create a modified version of this chart. We remove the temperature chart and add some images to the flow chart by using the related resources that are obtained from Semantic Web. In the local database, there is a table including the information of the cities passed by the Napoleon's army during this campaign. This table has three attributes, i.e., "longitude", "latitude", "city". We define a DVS to represent it as shown in Part (a) of Figure 4.3.

We can search these cities in Semantic Web and retrieve their images. Then, we added these images to the original charts using the CVS. As shown in Figure 4.3, we first search the keyword "City" in DBpedia. We select the class "dbp-on:City" in the result list to generate a class node. In this node, we specified to retrieve the values of the attribute properties "foaf:name" and "foaf:depiction". The attribute "foaf:depiction" stores the URIs of the images of corresponding cities. The system automatically generated the DVS as shown in Part (b) of Figure 4.3. Then we

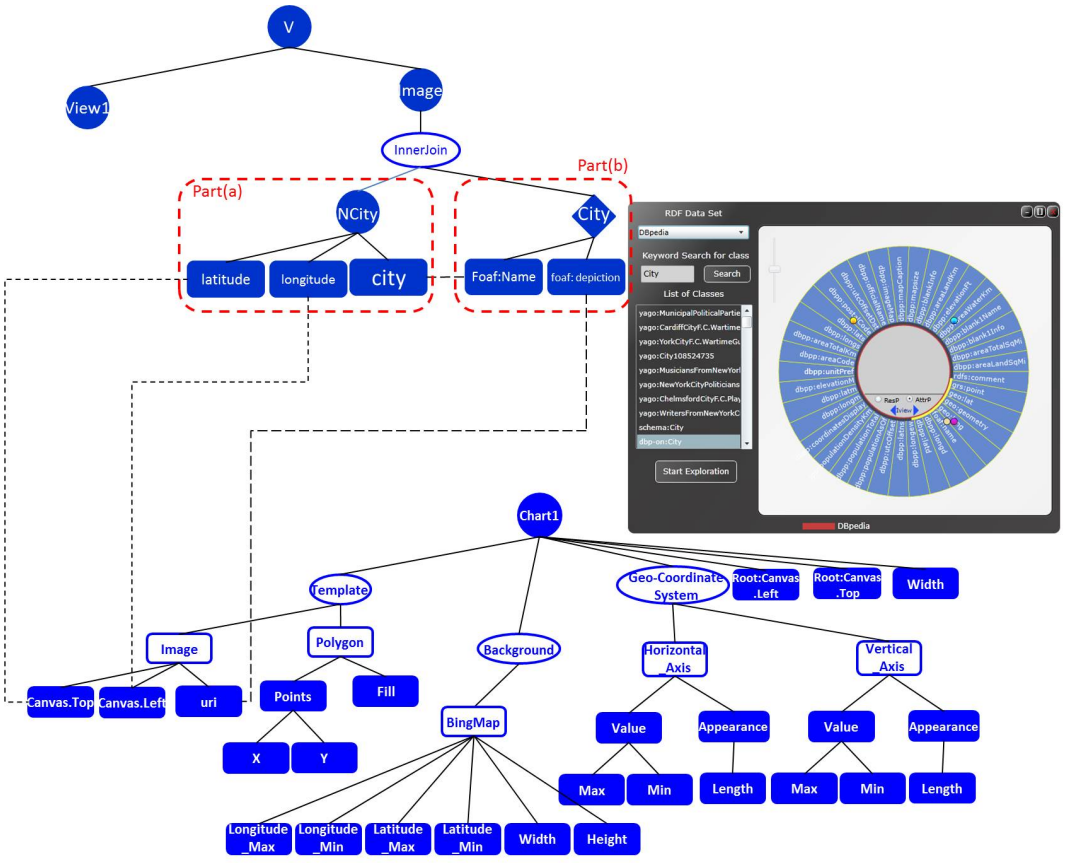


Figure 4.3: The DVS and CVS used to define the modified Napoleon's chart shown in Figure 4.4

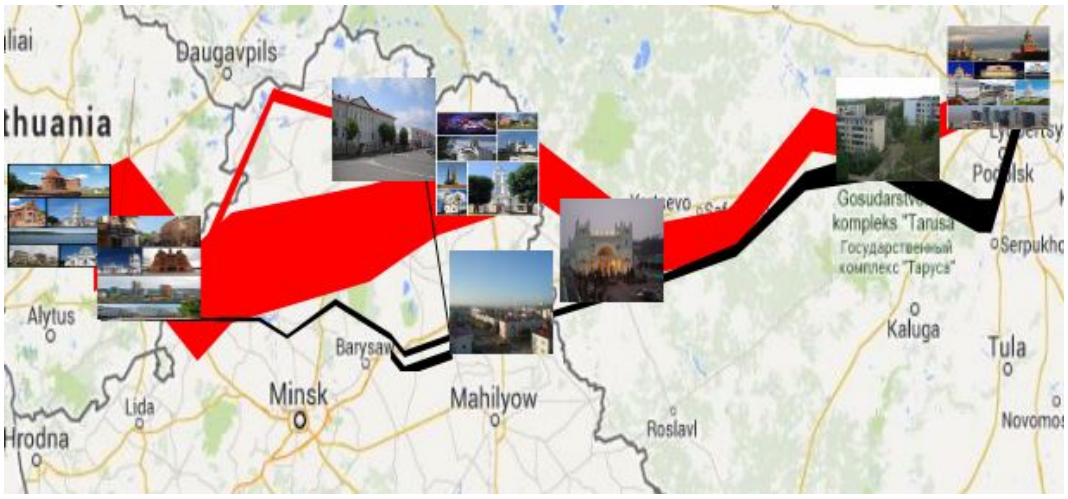


Figure 4.4: The modified Napoleon's chart with showing the images of some cities in this area

joined the DVSs in Part (a) and Part (b) to create a new DVS. The evaluation of joined DVS is a data table with four attributes, "latitude", "longitude", "City", and "foaf:depiction". This table describes the location and images of the cities stored in the local database.

As shown in the Figure 4.3, we defined a mapping from the joined DVS to the CVS to define the images in the resultant chart shown in Figure 4.4. In Figure 4.3, we omitted a DVS "View1" and the visual mapping from it to the CVS, which are used to define the paths in this chart.

4.4.2 An embedded chart visualizing the grosses of the movies

In this second example, we wanted to create an embedded chart to show the gross information about the films which are released in 2008.

We have a data table in the local database, which has two attributes, i.e., "MovieTitle", and "JapanGross". This data table describes the grosses of different movies in Japan.

In order to obtain more information, we searched Semantic Web for retrieving the data that describe the global gross of each movie, and the studio which created this movie.

By using the data from the local database and from Semantic Web, we defined an embedded chart as shown in Figure 4.5. We first grouped the movies in 2008 by their studios. In this chart, we used the black lines to represent the sum of the global

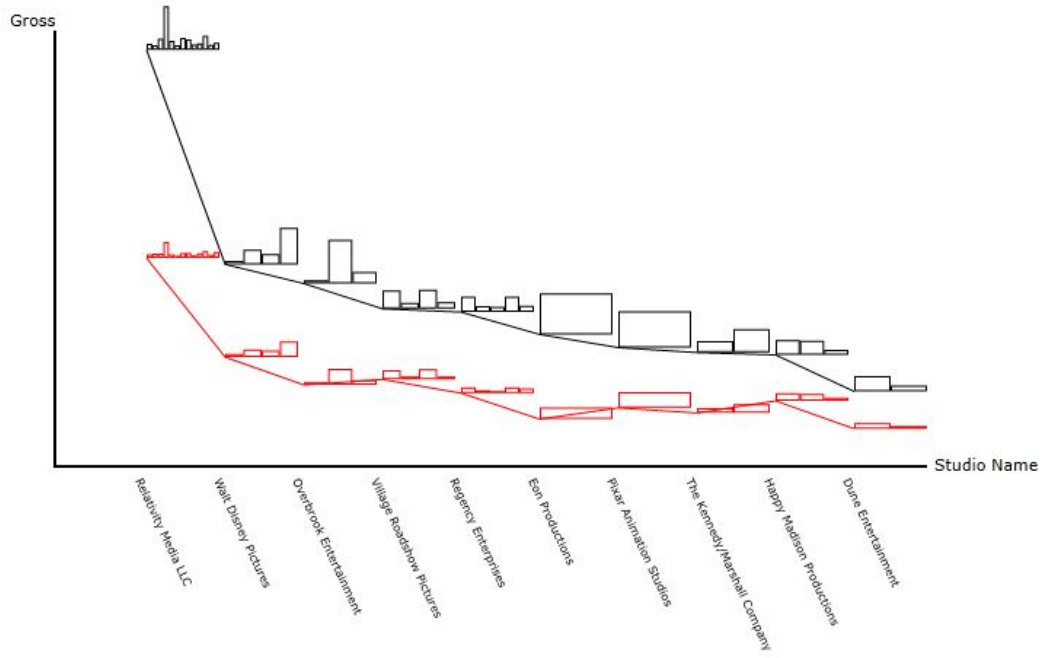


Figure 4.5: An embedded chart used to visualize the gross information both in Japan and in the world of the films created by 10 famous studios in 2008

grosses of all the movies created by the same studio. The red line describes the sum of their grosses in Japan. We only choose the top 10 studios in this example.

At each vertex of a line, we used a small bar chart to represent the gross of each movie in 2008 created by the corresponding studio. The height of each bar in these charts represents the gross of the corresponding movie. Black bars in these charts represent the global grosses of these movies, and the red bars represent the grosses in Japan.

To create this chart, we first searched Semantic Web. As shown in Figure 4.6, the schema of RDF data has two class node, "yago:2008Films" and "dbpp:studio". We specified to retrieve the values of the attribute properties "foaf:name" and "dbpp:gross" in the Node (a), and the attribute property "foaf:name" in the Node (b). The system automatically generated the corresponding attribute nodes in the modified DVS. We respectively renamed these attribute Nodes corresponding to these three properties as "MovieTitle", "WorldGross", and "Studio". The modified DVS generated by the above operations is shown in Part (a) of Figure 4.6. Part (b) in the DVS (b) represents the data table in the local database. We performed an inner join between them by specifying that "V1.MovieTitle=2008Film.MovieTitle". The join result is the whole DVS in Figure 4.6.

For defining the chart in Figure 4.5, we need to further manipulate the joined DVS as shown in Figure 4.6. We performed the Group-by operation to the attribute node "Studio", and defined the DVS as shown in Figure 4.7.

We also need to define a CVS and to specify the connection between the DVS and this CVS as shown in Figure 4.7 to define the chart in Figure 4.5.

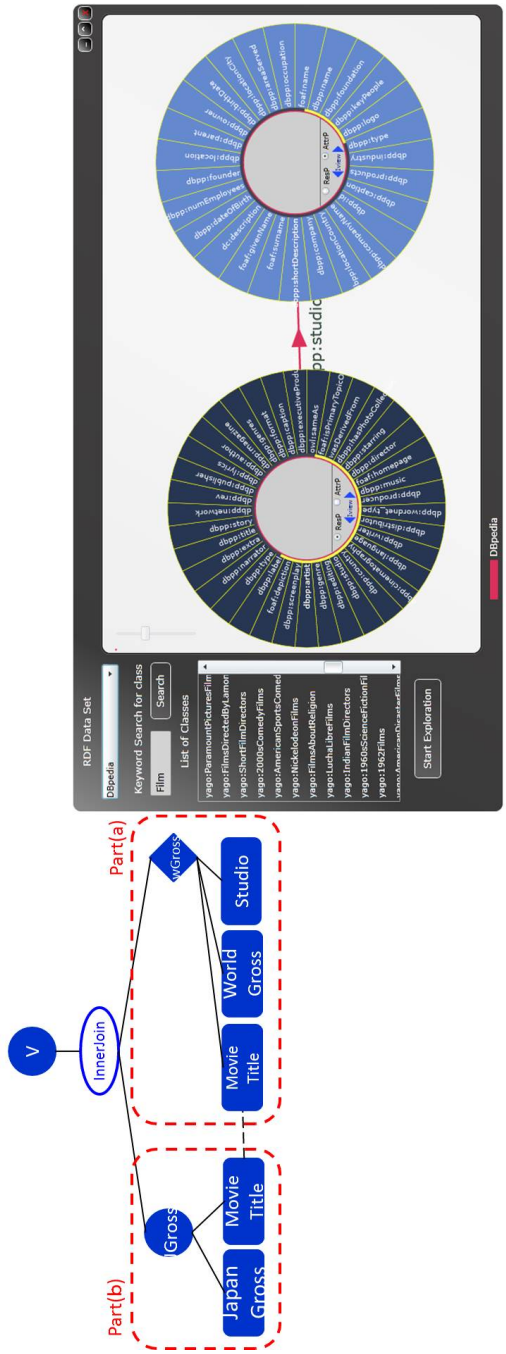


Figure 4.6: The DVS used for defining the chart shown in Figure 10, and the workspace of the Semantic Web search for retrieving the data to define Figure 4.5

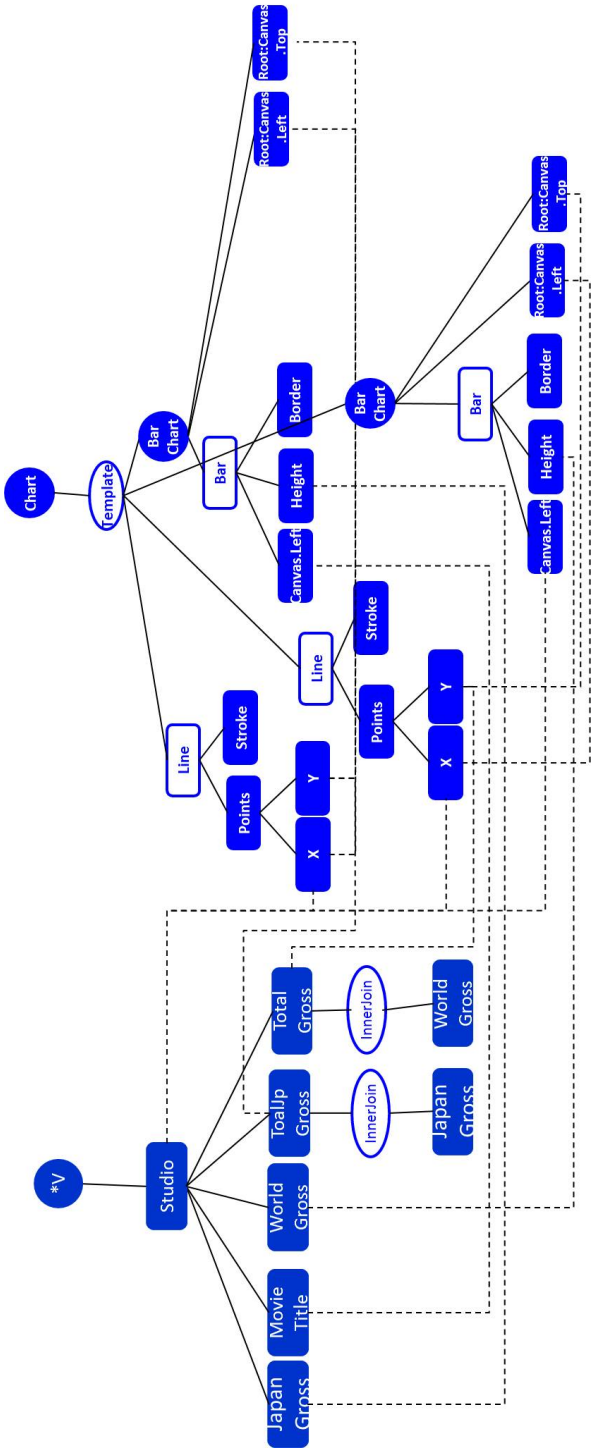


Figure 4.7: The DVS and CVS for defining the chart shown in Figure 4.5

Chapter 5

Implementing Our Framework Based on Webble World

To properly implement our framework, we need to answer the following two questions:

1. What technology can be used to retrieve and to process data?

2. What technology can be used to draw graphical objects and to enable users to interact with them?

The first technology should be able to deal with not only the flat tabular data structures, but also hierarchical data structures. As to the second question, most mark-up visualization languages such as SVG and XAML can provide a user interface for drawing graphical objects. However, a further requirement is that the selected language should support the parameterization of graphical objects. Such a language should be able to extract the required visual properties of each graphical object, to represent them in a tree structure, and to modify the appearance of each graphical object just by changing their values.

We have implemented our framework using NHibernate [6] and Webble World [11]. NHibernate supports retrieval of data from a database, and the manipulation of those data in a flexible way. Webble World is a component-based object-oriented web technology based on the idea of Meme Media [25]. Using any *de facto* Web browser with the Silverlight plug-in, users can freely access the Webble World system, which is currently being reimplemented in HTML 5.0. Section 5.1.2 will introduce the detail of Webble World system.

Every visual object in this system is called a webble, and has slots to store its property values and to exchange messages with other webbles. Users can compose advanced applications as composite webbles just by combining different webbles together and by defining slot connections among them[13]. In our implemented framework, any primitive or composite webble can be a graphical object. Each primitive webble has not only a geometrical shape but also an internal function. Users can publish both primitive and composite webbles over the Web, and share them with other users.

Our implemented framework has the following three major advantages:

(1) Webble World provides a library of webbles for users to choose appropriate webbles to define their charts.

(2) Advanced users can develop new primitive webbles through programming and use them to compose charts.

(3) The implemented framework supports the sharing and the reediting of newly composed charts. Users can easily create an online map.

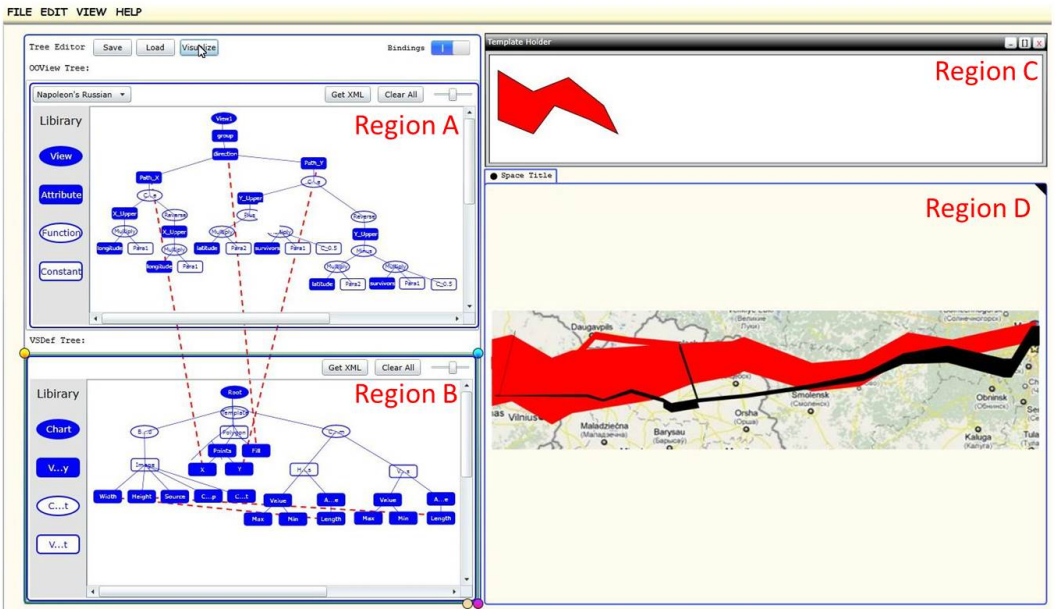


Figure 5.1: The user environment of the implemented framework

Figure 5.1 is a screen shot of our implemented framework, running in the MS Internet Explorer. As shown in this figure, the Region A and B are separately used

to manipulate the DVS and CVS. On the left side of these two regions, basic nodes are provided. Users can use them to define the DVS and CVS by drag-and-drop operations. The webbles which are used as visualization templates are shown in Region C. Region D is used to display the resulting chart.

In our implemented framework, we also provided a Webble which wraps the ArcGIS 10.2.2 object and enables users to use the ArcGIS 10.2.2 functions through the ArcGIS API v3.2 for Silverlight. Using this Webble, users can define maps, query routes, and access ArcGIS Server services. This expands the users' ability for defining GeoVisualization charts by reusing the objects and functions provided by ArcGIS 10.2.2 [3].

5.1 IntelligentPad and Webble World

Our framework is developed based on Webble World system. It is a web version of IntelligentPad. Both Webble and IntelligentPad are 2D instances of the Meme Media architecture. [25] Meme Media is proposed by Yuzuru Tanaka, and using it, it becomes possible to reuse the various existed resources and to integrate with other resources. In this chapter, we will introduce these two systems.

5.1.1 InelligentPad

Abstract

The IntelligentPad system uses a uniform method to deal with various kinds of resources. The basic unit in the IntelligentPad system is called a pad. It also a resource component with a sheet-like appearance on the computer display. Each pad has a set of communicating interface. These interfaces are called slots. Users can combine more than one pad together to create a composite pad to integrate these resource components. Such kind of composition is realized by defining the communicate channel between two pads. Each channel is defined by connecting two slots each of which are respectively belonged to two different pads.

These resource components which can be represented by pad include the media components, such as texts, figures, audios, videos and so on, and the software components, such as web services, applications, database systems and so on. For each pad, it will have several slots, which is used to hold the parameters of itself. And two pads can be combined together to create a composite pad by users' mouse operations freely. Such a composition defines not only the physical layout of components in a composed pad, but also the functional linkage between components. Users can

publish, organize or publish them as a whole. Furthermore, each component pad, called primitive pad, can be peeled off from the composite pad.

The created pads will be put onto the web as web resources. Then, users can download the existed pads and reedit them to satisfy their own requirements. Then the reedited pads can be updated to the web again. This process also goes with the evaluation and natural section of these pads and lead to the thriving development.

Architecture

Inner Architecture

A pad with the unique function is called primitive pad, which is the most basic unit of IntelligentPad systems. It is created based on the MVC construct as shown in Figure 5.2.In this construct, M is a Model, V is a View, and C is a Controller. The Model is used to define pad's internal state and behaviour. The View and the Controller worked together as the display part. The View is used to decide the appearance of the pad on the screen, and the Controller defines the reactions to user events through a mouse or a keyboard. Between the Controller and the View, there is an unidirectional message sending. When the Controller detected the user events, it can inform View through a message. Between the View and the Model, there is bidirectional message sending. The view can send a message to the Model to inform the user events from the Controller or change itself. The Model also can send a message to the View by update propagation to inform its state or change to the View.

For a non-developer user, the inner architecture is unnecessary, because every pad has a uniform interface. The user can operate them through mouse operations directly without considering about inner message communication.

A pad developer can define some slots in the Model or in the View of a pad, but not in the Controller. The slot is used to store the parameter values of each part, called, respectively, model slots and view slots of this pad. And the message exchange between 2 pads is also realized through the slot.

Pad Composition

The composition is defined by pasting a pad onto another pad. After the paste operation, a parent-child relationship is specified synchronously between these two pads. The pad on the bottom is called parent pad, and the pad on the top is called child pad. Any two pads can specify this relationship. One pad can only have one

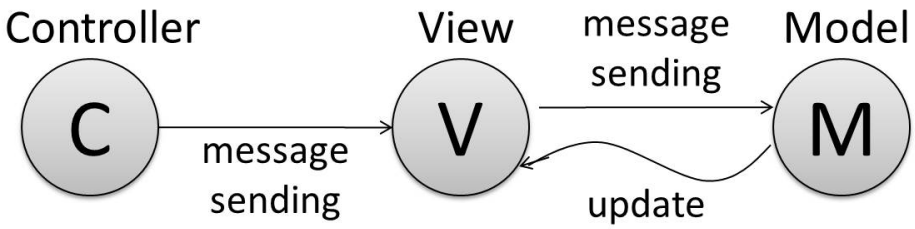


Figure 5.2: MVC construct of each pad

parent at most, but it can have more than one child. Then, in order to build the function linkage between parent pad and child pad, the slot connection is needed. Between 2 pads, there is only one connection is available. Through, this connection, 2 pads can exchange information. Figure 5.3 is an illustration the composition and slot connections of three pads.

When the slot value of a pad is changed, the new value will be sent to the connecting pad through the slot connection. The connecting pad will invoke its inner function when it receive the new value. Through the message exchanging mechanism, the system can realize the integrate of the functions of two connecting pads.

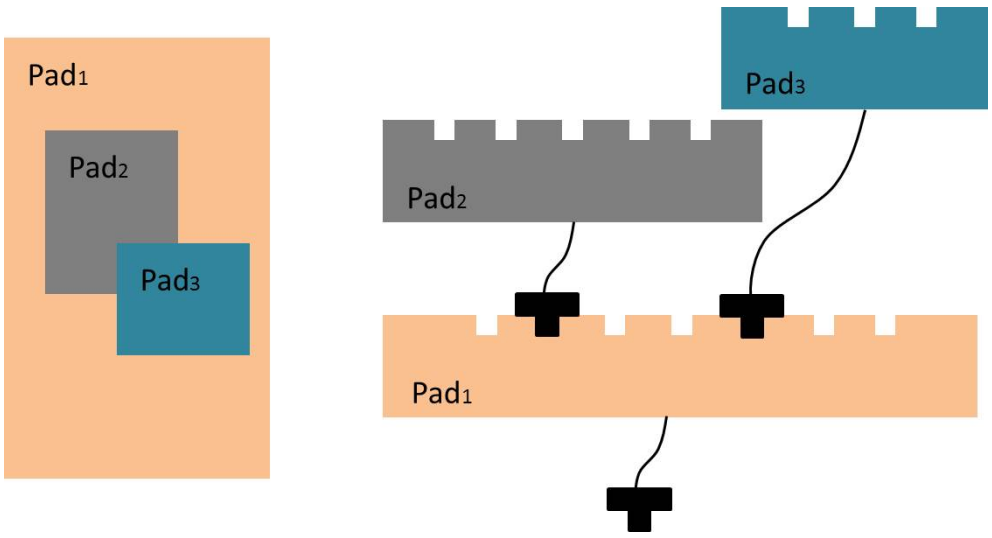


Figure 5.3: The composition and the slot connections of pads

The message exchange between two pads are shown in Figure 5.4. When the value of a slot of parent pad is changed, it will broadcast the update message. A

child pad receiving it maybe invokes its function and sends the gimme message to its parent to obtain the changed value of the slot. And in a pad, it is possible to forbid any message as user's specifications.

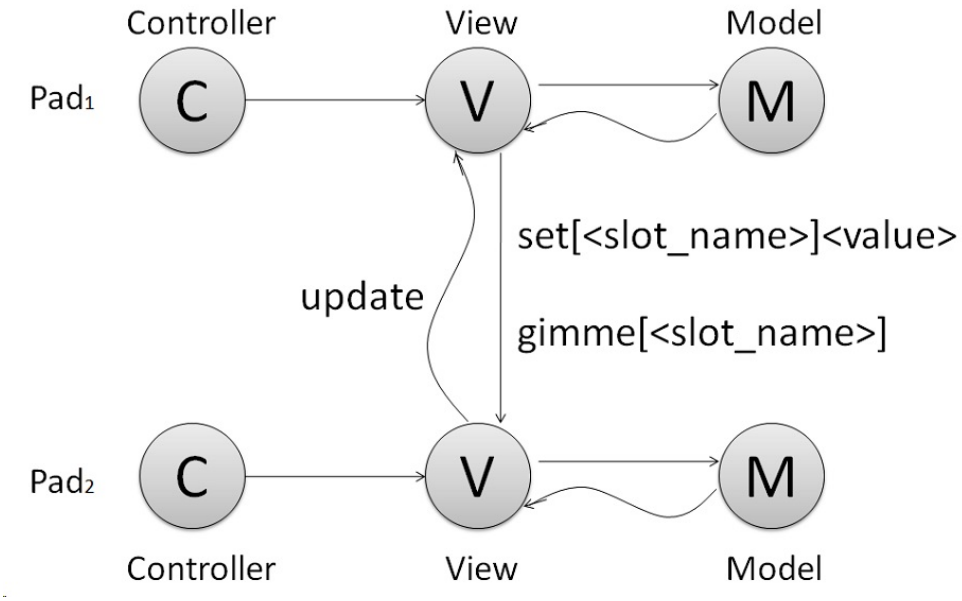


Figure 5.4: The standard Messages between pads

Meme Pool and Evolution of IntelligentPad

A primitive pad or a composite pad can be published into the Meme pool. Meme pool is a meme media object repository. We can search and load the existing pads from the meme pool to reuse or reedit them. We also can publish the self-created pads or self-composed pad into the meme pool for using by other users. The meme pool will promote the development of pads and the natural selection and evolution will happen on the pads in the meme pool.

Among the pads in the meme pool, the powerful and useful pads will be loaded and reused frequently. Certainly, this kind of popular pads will be reedited frequently as well. In opposite, the pads which are not good enough or have pleonastic functions will be loaded few. After a certain term, fewer and fewer users will load and reuse it. This kind of human activities makes the natural selection and evolution of pads happen.

5.1.2 Webble World

Webble World system [11] is a web version of 2D Meme Media Architecture (Figure 5.5). The basic architecture is very similar to IntelligentPad system. And it is more powerful than IntelligentPad because of some modifications and technology update. In Webble World system, the pad is renamed as Webble. The Webble World system:

- Works on the web and uses Silverlight technology;
- Can define the appearance of a Webble using any geometrical shape, while in IntelligentPad system each pad only have a rectangular appearance;
- Needn't be put onto another Webble to define parent-child relationship.

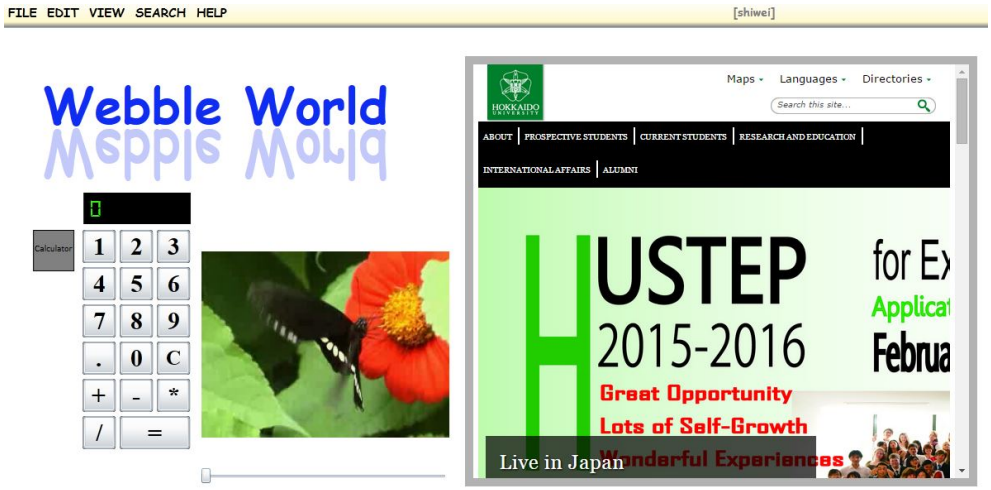


Figure 5.5: The Webble World system

Starting to use Webble World system, any client kernel is unnecessary to install in your PC. All we need are internet connection and a web browser with Silverligt [4] plugin and to access the website: `http://cow.meme.hokudai.ac.jp/WebbleWorldPortal/`. Silverligh is a new technology developed by Microsoft and provided to web-user for free. It is lightweight, cross platform system. Using Silverlight technology makes the Webble World system be a crossplatform system, too. The appearance of each Webble is defined by XAML(Extensible Application Markup Language) which is integrated in Silverlight technology to describe the appearance of Silverlight object. This means the appearance of a Webble can be any geometrical shape, which can only be rectangle in IntelligentPad system.

The third item introduces an important improvement of Webble World system comparing to the IntelligentPad system. In IntelligentPad system, child pad must be on the top of parent pad, while in the Webble World system, the position of the child pad can be arbitrary. This improvement enriches the compose method of pads greatly, and make it more powerful.

Furthermore, the Webble World system is a web system. And itself is a meme pool for Webbles as well. We needn't to build a particular meme pool to accelerate the evolution of the Webble World system.

Figure 5.6 shows an application created in Webble World system. There are 4 gears and a controlling square. In these four gears, each one will be driven by its left gear except the first one. The First Gear is controlled by the square. With the horizontal moving of the controlling square, the Gear1 will rotate. Then, all the gears will move together. In this example, the square and 4 gears are all Webbles. Figure 5.7 is another Webble application. It is a clock.

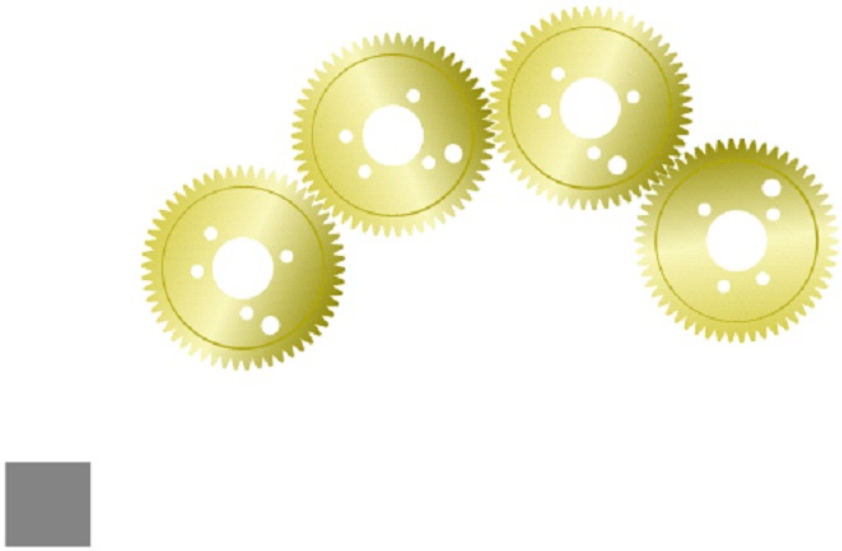


Figure 5.6: A geer rotation application in Webble World system

In recent years, the staff in our lab start to apply the Webble World system in many practical fields, such as medical science and child education[10][22]. Furthermore, in order to fit for the next generation Web environment, the Webble World system has been updated. The new version of Webble World system, Webble World 3.0[12], is developed based on HTML5. The new Webble World system can run on

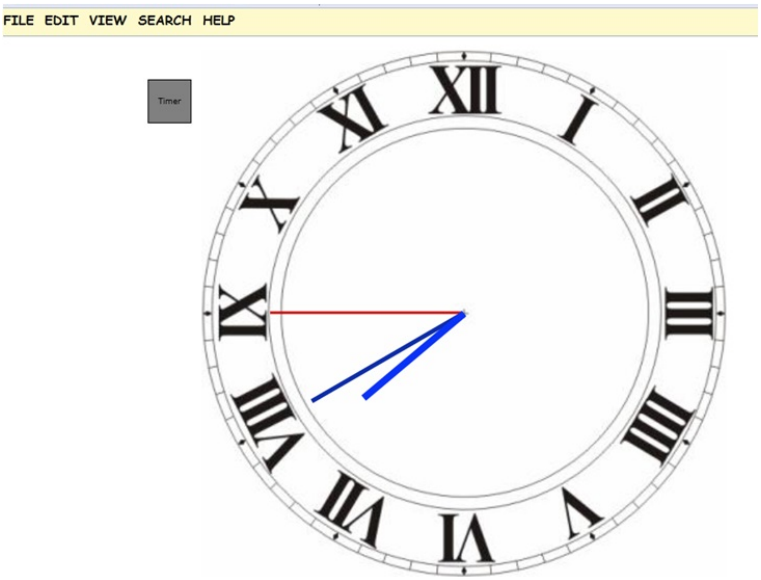


Figure 5.7: A clock created using Webble

most popular Web browsers. Even a smart phone and a tablet can run this system. We hope users can access this system anywhere and anytime. We expect a more wider application of this system in future.

Chapter 6

Evaluation

6.1 Related Research

6.1.1 Information visualization

From the survey of Michael Friendly [26], we know that past research efforts of Napoleon's chart mainly focused on the three issues:

1. how to extend this chart to effectively represent more information;
2. how to enable users to more easily obtain the information; and
3. how to use modern visualization tools to automatically construct this chart as well as its extended versions.

Solving the first two issues depends on the design of the chart. For solving Issue (3), there are modern visualization tools that can create the Napoleon's Campaign chart and other non-standard charts by procedural programming. For example, Shaw and Tigg wrote a program using Mathematica to recreate the Napoleon's Campaign chart [17]. This is a not an easy task for casual users because it requires significant programming skills.

Leland Wilkinson [30] proposed a powerful language, which was used to create a chart from declarative logical specifications. Protovis [2] and ggplot2 [29] provide methods similar to the method proposed by Leland to create nonstandard charts. However, these systems still have three major problems. First, the power of such systems depends on the sizes of their libraries of registered graphical objects. The developers of such systems need to frequently update the library of graphical objects

to satisfy users' requirements. This is a tedious, expensive and never ending endeavour, and users' requirements cannot be satisfied immediately. Second, the system developers determine how to parameterize these graphical objects. Users have to read the instructions in detail before using these systems. Third, selecting an appropriate graphical object from the library is not easy. Two or more pre-defined graphical objects provided by the system may look similar, because they may use the same geometrical shape as their appearance. For example, ggplot2, provides two graphical objects, geom_histogram" and geom_bar", which are both defined as rectangles, but used in different cases.

Many geographic information systems, such as ArcGIS v10.2.2 [3], also enable users to define nonstandard GeoVisualization charts. They provide wizardry systems that guide users to define various GeoVisualization charts on top of various map objects loaded from their map libraries. They enable users only to define charts with predefined types. While their wizard systems simplify the visualization definition process, they make all created charts look similar to each other. New designs are difficult to define with these systems. Because they provide no generic method for defining different types of charts, users have to learn how to specify parameters in different wizard systems for defining different types of charts.

Many expert systems are also proposed in cartography. Such research studies focus on the use of designers' knowledge for automatically choosing the best values of some specified properties of charts, or providing design advices on them. For example, AUTONAP supports geometrical arrangement of labels on a map, and ColorBrewer gives advices on the best color assignment to graphical objects on a map. However, in this paper, we do not discuss the use of designers' knowledge for the best choice of such property values of charts. Our research aims at supporting users to easily realize their custom-made design of a chart. Furthermore, these studies do not focus on the automatic creation of standard and nonstandard charts from their logical specifications.

6.1.2 Semantic Web search

If users want to visualize the data from Semantic Web, they may have two methods. First, users may retrieve their interested data and then convert the retrieved data into the format requested by some visualization system. Such a data retrieving and converting process always needs the help of other tools.

Second, users may use the visualization parts provided by the Semantic Web search systems to define charts. For example, Tabulator [1] allows users to define

some charts, such as bar charts on calendars and point charts on the maps. But these visualization parts only support a few chart types. They also do not support users to integrate the data retrieved from Semantic Web with the data from local database.

Users may also have two problems when using the Semantic Web search systems. First, the background knowledge about Semantic Web is necessary for using these systems. For example, iSPARQL[9] requests users to understand the schema for defining a SPARQL query to retrieve their interesting information.

Third, many systems, such as PGV [5], only allow users to search instances in Semantic Web. They do not provide a schema of RDF data. Users cannot simultaneously search a set of instances and their properties that belong to the same class. They have to repeat the search operations to obtain their desired result. It may be a difficult task when the users' interested instances are too large.

To solve the above problems, this paper will extend our visualization framework by integrating our visualization framework with Piao's ESVSW framework.

6.2 Function Comparison

As we mentioned in Section 6.1.1 , many powerful visualization systems have been developed to create both standard and nonstandard charts. In Table 6.1, we show a comparison between our framework and five major systems with respect to the eight different aspects, we think fundamental in interactive database visualization with standard and nonstandard charts. In these eight aspects, the first five aspects are used to evaluate whether a system enable users to flexibly customize a chart in different ways. The sixth aspect is used to evaluate whether a system enable users to dynamically explore a visualization chart. The last two aspects are used to evaluate whether a system can simplify the creation of similar charts. We will explain each these eight aspects in the following paragraphs. These five systems are Many Eyes[27], Protovis [2], Mathematica[17], ggplot2[29] and ArcGIS 10.2.2 [3].

In this table, "Definition Method" refers to how a system defines a chart, namely either declaratively without programming or procedurally with programming. Among the five major systems and our system, only Mathematica requires users to define a chart through procedural programming.

"Full Parameter Manipulation" denotes whether a system allow users to change the value of any parameter of each provided graphical object. Only our system supports this function. "Chart Embedding" denotes whether a system supports the

Table 6.1: A comparison of our system with other five systems in terms of eight major functional aspects

	Definition Method	Full Parameter Manipulation	Chart Embedding	Template Customization	Appearance Customization	Interactive Chart	Resource Sharing	Reusability of Definitions
Our System	Declaratively	○	○	○	○	○	○	○
Many Eyes	Declaratively	×	○	×	×	○	○	○
Protovis	Declaratively	×	○	○	○	○	×	○
Mathemetica	Procedurally	×	×	○	×	×	×	○
ggplot2	Declaratively	×	×	○	×	×	×	○
ArcGIS	Declaratively	×	○	×	×	○	○	○

embedding of existing charts into another chart. Neither Mathemaetica nor ggplot2 supports this function.

"Template Customization" denotes whether it is possible both to define new templates and to modify existing templates. Many Eyes and ArcGIS 10.2.2 do not support this function. "Appearance Customization" denotes whether it is possible to modify the appearance of the coordinate system, the background and the legend of a chart. Only our system and Protovis support this function.

"Interactive Chart" denotes whether a system supports user interactions with visualization charts. Neither Mathemetica nor ggplot2 support this function.

"Resource Sharing" denotes whether a system supports users to easily share user-defined objects though the Web. Our framework, Many Eyes and ArcGIS 10.2.2 support this function.

Finally, "Reusability of Definitions" denotes whether a system allow users to reuse a part of definitions of existing charts to define new charts. All the systems somehow support this function.

6.3 Visualization Process Comparison

Next, we compare our framework with Mathematica when it comes to the process of creating the Napoleon's chart. In Mathematica, someone has already provided a function called "NapoleonicMarchOnMoscowAndBackAgainPlot[]" (that was defined as a program consisting of 122 lines of Wolfram codes) to produce this chart. This pre-programmed functions procedurally defines the details of how to convert data, and how to draw the chart using the converted data. It is easy to use such kind of pre-programmed function to reconstruct similar charts only by adjusting parameter values and the source data.

For creating the Napoleon's chart based on our framework, we used two pairs of the DVSs and the CVSs. In total we used 41 nodes and 9 linkages. For defining the derived attributes, we used 11 Function nodes. For defining the relation between the two sub-charts, we specified 3 expressions in three Property nodes. These numbers are remarkably smaller than 122 lines of codes used to define the macro function in Mathematica, and furthermore, our definitions of the DVSs and CVSs are purely declarative. Users can directly reuse the saved DVSs and CVSs for defining similar charts with fewer steps. However, when users want to define a nonstandard chart from scratch, it will be difficult with other visualization tools, especially for users who are not good at programming. Our framework supports even such users when

defining such nonstandard charts just by directly manipulating the intermediate model using mouse operations and expression specifications.

Chapter 7

Conclusion

In this thesis, we have proposed a new information visualization framework. By using this framework, users can define their desired charts by the declaratively specifying the charts.

We have proposed an intermediate model consisting of one or more pairs of Data View Schemata (DVSs) and Chart View Schemata (CVSs) that respectively realize the data querying and manipulating process, and the chart defining process. By visually manipulating the DVSs and CVSs, and defining node linkage among them, users can request our framework to generate their desired charts. Our framework includes a method for supporting users to compose a new visualization template by combining two primitive graphical objects together. This method aims to solve the insufficiency of the visualization templates in our framework.

Our framework also supports users to interact with the generated charts. Besides the common chart exploring operations, users can use the chart modification operations to indirectly modify the DVS and the CVS. We have also extended our framework to visualize the data retrieved from Semantic Web.

In contrast to systems that define charts by procedural programing, our system allows users to declaratively define a chart without any explicit programming, and hence they are much easier to understand and use. The tree structured schemata in our framework enables users to visually manipulate the source data and graphical objects without reading any detailed instructions. Other systems require users to follow developer-defined rules to specify the values of one or more parameters of different a *priori* registered graphical objects. In such systems, users cannot compose a chart that requires an unregistered basic graphical object. Our framework also supports other seven important functions listed in Table 2. However, other major systems do not support some of these functions.

First, this thesis introduces a new information visualization framework. This framework includes an intermediate model, composed by the Data View Schemata and the Chart View schemata. By visually manipulating the DVS, users can define an object-oriented view of the database. Then our framework can generate an object-oriented query to retrieve data from the database to generate data objects. By manipulating the CVS, users can define how to use the copies of visualization template to represent data object, and customize the appearances of the background, the legend and the coordinate-system of a chart.

Our framework also supports users to relate two DVSs to define the integration of the data from different data sources, and to relate two CVSs to combine two charts together.

For supporting users define a complex chart using a visualization template, which is not registered in our template library, our framework provides a template composition method by using two functional objects, the Shadow and the Transformer.

We have also provided three functions in our framework for supporting users to easily and quickly define custom-made GeoVisualization charts. By using these functions, our framework will automatically define the mappings between longitude and latitude attributes of the data objects to the X and Y coordinate properties of the visual objects, and automatically adjust the display area, the scale and the center of the map used as the chart background. Users can also use geographic names (e.g., country, building), that can be automatically converted by some Web service to geo-locations to place the visual objects in the chart. Our framework also allows users to interact with the generated charts by three operations: zooming in/out, panning, and brushing and linking. Our framework also allows users to wrap the map objects of ArcGIS 10.2.2, and utilize the functions provided by the ArcGIS API for Silverlight v3.2.

We have used our framework to recreate the famous Napoleon's Campaign chart and also shown how it can be easily extended based on this framework. We have also shown a 2.5D GeoVisualization chart example, which visualizes the trajectories of two typhoons passing over Philippines and Taiwan. This example as well as the example of Napoleon's Russia Campaign chart demonstrates that our generic framework enables users to easily create a variety of nonstandard GeoVisualization charts.

Our framework support users to interact with the generated charts. The interactions provided by our framework are two types. One is the common chart exploring interactions. The other is the chart modification interactions.

The common chart exploring interactions include zoom in/out, pan, brushing-and-linking. These interactions are provided by most visualization systems. They temporarily or permanently change the visual mapping from data objects to visual objects.

Chart modification interactions enable users to indirectly modify the intermediate model of a chart by directly manipulating the visual objects of a chart. Such interactions can provide more powerful change in a visualization chart.

Our paper using a chart of product sales demonstrates the power of these chart modification interactions. Users can extend a line chart to a nested chart only by manipulating the line chart. By these operations, even users who have no sufficient knowledge about the intermediate model of our framework can define their desired charts only through simple mouse and keyboard operations.

We have also discussed how to extend our information visualization framework. We have reimplemented a framework proposed by Piao, which supports users to exploratively search Semantic Web.

Our framework can automatically generate a DVS to represent the structure of the data retrieved from Semantic Web, when users perform the searching operation in the workspace provided by Piao's framework. Users can manipulate such DVS as manipulating the DVS which represents the structure of the data retrieved from a local database. Users can join such DVS with other DVSs defining derived attributes, and hierarchical structures. Then, by associating them with some CVS, users can define their desired charts.

The proposal of this thesis consists of three major contributions to support users to define both standard and nonstandard charts in a unified framework. First, we have introduced the visualization framework and the intermediate model of this framework. Users can manipulate the intermediate model to declarative define their desired charts. Second, we have introduced how to interact with the generated charts. Beside the common chart exploring operations, our framework also allows users to manipulate the chart to indirectly modify the intermediate model for defining this chart. Last, we have extended our framework. Users can retrieve data from Semantic Web and integrate the data from Semantic Web and those from a local database. Furthermore, the integrated data are mapped to define a standard or nonstandard chart just by defining the DVS and the CVS.

Related Publications

1. Wei Shi, Randy Goebel, and Yuzuru Tanaka. A New Database Visualization Framework for the Automatic Construction of Non-standard Charts: Recreating the Chart of Napoleons Russian Campaign of 1812. Catographica, Vol. 49, No.4, pp.241-261, University of Toronto Press, Dec. 2014

2. W. Shi and Y. Tanaka. Object-oriented Graphical-template Composition Framework for Information Visualization. In *Proceedings of The IET International Conference on Frontier Computing: Theory, Technologies and Applications*, pp.378-383,IET, 4-6 Aug. 2010.

3. Wei Shi, Bin Piao, and Yuzuru Tanaka. An Extended Framework for Visualizing the Data from Both Local Databases and Semantic Web Databases. In *Proceedings of the International Conference on Computer Graphics, Multimedia and Image Processing (CGMIP 2014)*, pp.76-90, SDIWC, 17-19 Nov. 2014.

Bibliography

[1] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *Proceedings of the 3rd International Semantic Web User Interaction*, 2006.

[2] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15:1121–1128, November 2009.

[3] E. Corp. Arcgis v10.2.2. http://www.esri.com/software/arcgis, April 2014.

[4] M. Corporation. Silverlight. http://silverlight.net/, 2014.

[5] L. Deligiannidis, K. J. Kochut, and A. P. Sheth. Rdf data exploration and visualization. In *Proceedings of the ACM First Workshop on CyberInfrastructure: Information Management in eScience*, CIMS '07, pages 39–46. ACM, 2007.

[6] N. Forge. Nhibernate reference documentation. http://nhforge.org/doc/nh/en/index.html, 2012.

[7] M. Friendly. Re-visions of minard. http://www.datavis.ca/gallery/re-minard.php, 2012.

[8] P. I. A. Godinho, B. S. Meiguins, A. S. G. Meiguins, R. M. Casseb do Carmo, M. de Brito Garcia, L. H. Almeida, and R. Lourenco. Prisma - a multidimensional information visualization tool using multiple coordinated views. In *Proceedings of the 11th International Conference Information Visualization*, IV '07, pages 23–32. IEEE Computer Society, 2007.

[9] C. Kiefer, A. Bernstein, and M. Stocker. The fundamentals of isparql - a virtual triple approach for similarity-based semantic web tasks. In *Proceedings*

of the 6th International Semantic Web Conference (ISWC), Lecture Notes in Computer Science, pages 295–309. Springer, MAR 2007.

[10] M. Kuwahara and Y. Tanaka. Advanced "webble" application development directly in the browser by utilizing the full power of meme media customization and event management capabilities. In *2012 IEEE International Conference on Multimedia and Expo Workshops, Melbourne, Australia, July 9-13, 2012*, pages 211–216, 2012.

[11] M. N. Kuwahara. Webble world. http://cow.meme.hokudai.ac.jp/ WebbleWorldPortal/, 2014.

[12] M. N. Kuwahara. Webble world 3.0. https://wws.meme.hokudai.ac.jp/, 2014.

[13] M. N. Kuwahara and Y. Tanaka. Webble world - a web-based knowledge federation framework for programmable and customizable meme media objects. In *Frontier Computing. Theory, Technologies and Applications, 2010 IET International Conference on*, pages 372 –377, aug. 2010.

[14] A. R. Martin and M. O. Ward. High dimensional brushing for interactive exploration of multivariate data. In *Proceedings of the 6th IEEE Visualization Conference*, VISUALIZATION '95, pages 271–278. IEEE Computer Society, Oct. 1995.

[15] U. of MANNHEIM. The linking open data cloud diagram, 2014. http://lod-cloud.net/.

[16] B. Piao and Y. Tanaka. Interactive framework for visual exploratory search and integration of semantic web contents and services. In *7th International Conference on Information Management, Innovation Management and Industrial Engineering,*, volume 1, pages 534–539, Oct. 2014.

[17] W. T. Shaw and J. Tigg. *Applied Mathematica: Getting Started, Getting It Done*. Addison-Wesley Longman Publishing Co., Inc., 1993.

[18] W. Shi, R. Goebel, and Y. Tanaka. A new database visualization framework for the automatic construction of non-standard charts: Re-creating the chart of napoleonfs russian campaign of 1812. *Cartographica*, 49(4):241–261, Dec. 2014.

[19] W. Shi, B. Piao, and Y. Tanaka. An extended framework for visualizing the data from both local databases and semantic web databases. In *Proceedings*

of the International Conference on Computer Graphics, Multimedia and Image Processing, CGMIP 2014, pages 76–90, Nov. 2014.

[20] W. Shi and Y. Tanaka. Object-oriented graphical-template composition framework for information visualization. In *Proceedings of The IET International Conference on Frontier Computing: Theory, Technologies and Applications*, pages 378 –383, Aug. 2010.

[21] W. Shi and Y. Tanaka. A new interactive information visualization framework based on the object-oriented views of querying and visualizing databases. In *Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications*, GRAPP & IVAPP 2013:, pages 495–504, Feb. 2013.

[22] J. Sjöbergh and Y. Tanaka. Geospatial digital dashboard for exploratory visual analytics. In *Information Search, Integration, and Personalization - International Workshop, ISIP 2013, Bangkok, Thailand, September 16-18, 2013. Revised Selected Papers*, pages 3–17, 2013.

[23] R. Spence. *Information visualization.* Addison-Wesley, 2001.

[24] R. Spence. *Information Visualization: Design for Interaction (2nd Edition).* Prentice Hall, 2007.

[25] Y. Tanaka. *Meme Media and Meme Market Architectures: Knowledge Media for Editing, Distributing, and Managing Intellectual Resources.* John Wiley & Sons, Inc., 2003.

[26] E. R. Tufte. *The Visual Display of Quantitative Information, 2nd edition.* Graphics Press, 2 edition, May 1986.

[27] F. B. Viegas, M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon. Manyeyes: A site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, Nov. 2007.

[28] C. Ware. *Information Visualization: Perception for Design.* Morgan Kaufmann Publishers Inc., 2004.

[29] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis (Use R).* Springer, 2nd printing. edition, August 2009.

[30] L. Wilkinson and G. Wills. *The grammar of graphics.* Springer, 2005.