Integrated Visualization Framework for Relational Databases and Web Resources

Tsuyoshi Sugibuchi and Yuzuru Tanaka

Meme Media Laboratory, Hokkaido University 060-8628 Sapporo, Japan
{buchi,tanaka}@meme.hokudai.ac.jp

Abstract. In this paper we propose an integrated visualization framework for relational databases and Web resources. Using our *VERD framework*, users can dynamically integrate local databases and Web resources and interactively construct various visualizations. Users can define data integration and visualization by constructing a flow diagram consisting of various *operators* through direct manipulation. The principal idea of this framework is to manipulate Web resources, visualizations, and database relations through visual components based on relational schemata and integrate them based on the relational database model. In VERD framework, two types of special view relations are used for representing visualizations and for treating Web resources as database relations. Hence, in VERD framework, a Web resource can be combined with database relations as if it is also a local relation. Similarly, VERD framework specifies visual mapping between source data relations and visualization representations by means of relational operations. VERD framework supports various interactive visualization operations including 'details-on-demand', 'drill-down', and 'brushing-and-linking' These operations are performed by modifying view relations associated with visualizations.

1 Introduction

Good information visualizations can reveal significant abstraction of data when the scale and the complexity exceeds the human capability to interpret data records from reading only them, and help users to find hidden trends or structures from a data set. In addition, adding interactivity to information visualizations is useful for exploring a large set of multivariate data. The interactive manipulation of visualization allows users to dynamically change visualization parameters, so that user can scan a large data set, focus on a meaningful portion of data, and find trends or patterns. Many interactive techniques including brushing-and-linking, *dynamic queries* [14], multiscale visualization, and multiple visualization coordination have been studied. Moreover, the interactive definition or construction of visualization is another useful technique for data exploration tasks and has been applied in many scientific visualization systems and database visualization systems. In these systems, users can construct a visualization system, suitable both for the source data and for the user's mental model, quickly to test user's hypotheses.

G. Grieser and Y. Tanaka (Eds.): Intuitive Human Interface 2004, LNAI 3359, pp. 159–174, 2004.

Dynamic information integration is also a useful technique to find rules from a complicated data set. By combining other related data with data of our current concern, we can introduce another viewpoint on current data. For instance, we may find an unforeseen trend from a market data by comparing it with a weather data. The World-Wide-Web is a universal data repository providing a huge amount of data for various application domains. Contents distributed over the WWW include more dynamic or graphical contents than records stored in DBMSs. Moreover, many Web application pages provide various services like Web searching, online dictionaries, shopping, and engineering calculations. If we can extract such contents and services from arbitrary Web pages and dynamically integrate them together with the target data of our current concern, we can extensively extend our data analysis capabilities.

In this paper we propose a new visualization framework we call VERD (Visualization Environment based on the Relational Database model). VERD framework allows users to integrate local database relations with related Web contents, and provides an interactive environment to visualize integrated data resources in various ways.

The following is an outline of VERD framework.

The relational database model has been adopted as a theoretical basis of VERD framework to integrate heterogeneous data resources and visualizations. In VERD framework, each data resource and each visualization display are manipulated through operations on relational schemata. In particular, we define special view relations to uniformly treat both visualizations and Web resources as relational schemata. A *visualization view relation* is a view relation to represent a visualization. A *Web view relation* is also a view relation to represent a set of related Web resources. Hence, data integration and visualization are both performed by uniformly applying relational operations to local database relations, Web view relations, and visualization view relations.

VERD framework introduces interactivity into data integration and visualization tasks by using the following mechanism. VERD framework provides a set of interactive components called *operators*. Each operator corresponds to a user's primitive operation for specifying data integration and visualization. Each visualization is defined as a flow diagram consisting of operators and connections among them. To integrate arbitrary Web resources with our visualization, we use Web wrappers to define a view relation among mutually related contents over the Web. We have developed a lightweight Web wrapper algorithm and a graphical interface for users to specify Web wrappers through direct manipulations. By using this interface, users can interactively and quickly specify Web wrappers on demand in their data exploration task.

This paper is organized as follows. In Section 2, we describe our basic framework for interactive database visualization. In particular in Section 2, we focus on how VERD framework works with visualization view relations. In Section 3, we show how Web resources can be integrated with database visualizations. We describe the details of Web view relations and an outline of our Web wrapper. Then in Section 4, we describe how VERD framework is implemented. In Sec-

tion 5, we discuss the related research works. Finally, we conclude our paper with some remarks.

2 VERD Framework

2.1 Visualization View

A visualization view is a relational representation of a visualization. In the relational database model, a virtual relation which provides a schema for some application is called a view relation. Our visualization view relation is an extension of this view relation to deel with graphical representation and location of each records. Visualization of source data is equivalent to defining a visualization view relation on a source relation. In VERD framework, a source data can be associated with a visualization scheme by using relational operations. Visualization view relations are dynamically constructed and modified in a visualization task.

A schema of a visualization view has *visualization attributes*. Each visualization attribute corresponds to a visualization property such as location, color, size, and angle. A type of each visualization attribute is identified by its name. In other words, the name of each attribute is used by our visualization system to associate its value with an appropriate visual property.

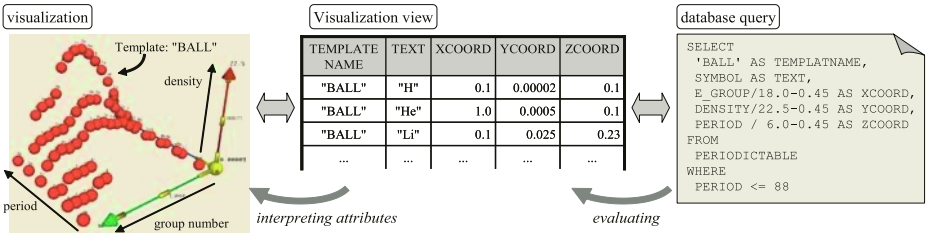

Fig. 1. Visualization view example

Figure 1 shows an example visualization and a visualization view relation corresponding to this visualization. This 3D scatter plot visualization represents a periodic table of elements. In this visualization, each element is represented as a ball. The three coordinate values of each ball are associated with the group, the period, and the density of each element. A textual label attached to each ball shows the name of each element. The schema of a visualization view relation illustrated in Figure 1 has visualization attributes corresponding to the four visual properties, X, Y, Z coordinates and the text label. A `TEMPLATENAME` attribute is included in this schema to specify a template. A template is a visual component to represent each database record. Some template has both a visual shape and a data representation function for displaying the text, changing the color, and morphing the shape. Users can compose a new template by combining visual

primitive components. The template used in this example visualization consists of a simple ball primitive and a component displaying a text string. The virtual dimension of visualization can be expanded by combining additional visual primitives with the template. This operation is equivalent to adding new visualization attributes to a visualization view.

A visual mapping for associating source data with a visualization is equivalent to a relational operation which projects a source relation onto a visualization view schema. Figure 2 illustrates an example of a visual mapping. In this example, the `SYMBOL` attribute is directly projected onto the `TEXT` attribute. On the other hand, the computed attributes derived from the attributes `E_GROUP`, `PERIOD`, and `DENSITY` are associated with the three coordinate attributes. These computations normalize the locations of visible data elements to arrange all of them in the visualization area. This projection operation can be described as a database query given in Figure 1. In this query description, the name of the source data relation appears in the `FROM` clause, and the visual mapping is specified in the `SELECT` clause. This query also specifies conditions to visualize only those elements with their atomic numbers less than 88. When our visualization system performs this visualization, this query is evaluated by the DBMS. Then, our system creates this visualization from the query evaluation result.

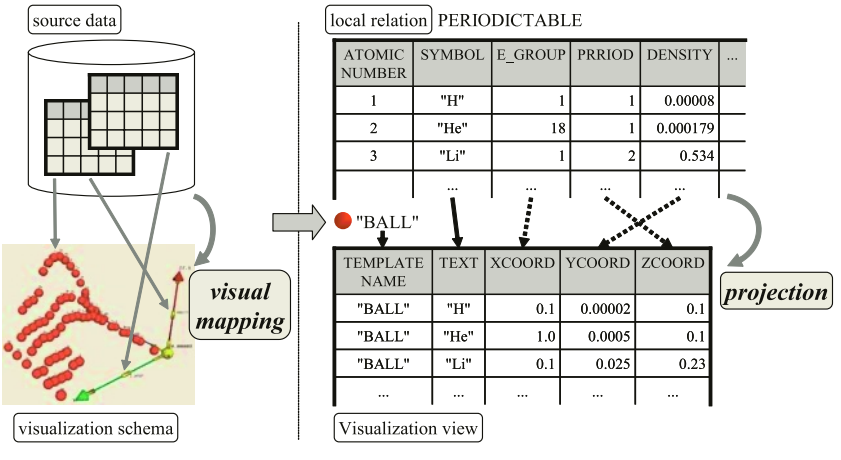

Fig. 2. Visual mapping

Each visualization operation is performed by deriving a new visualization view relation from another visualization view relation. VERD framework allows users to magnify a portion of visualization, to combine other data with the current visualization, and to coordinate more than one visualization. Figure 3 shows that each visualization operation is performed as a view modification operation. This figure shows that not only a visualization but also its portion can be represented as a visualization view relation. This feature is useful for performing interactive data exploration tasks by directly combining visual components.

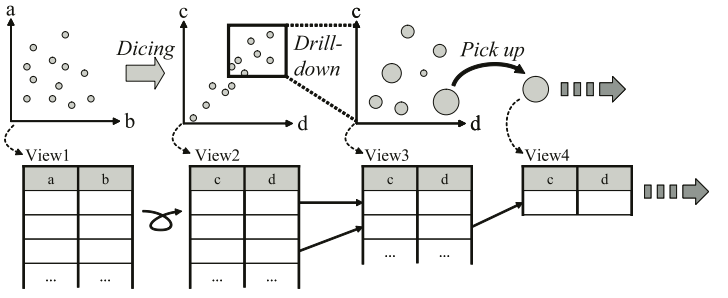

Fig. 3. View modification

2.2 Operators

VERD framework allows users to interactively specify visualizations. For this purpose, VERD framework provides a set of *operators*. Users can define a visualization by constructing a flow diagram consisting of operators.

Table 1. Operators for specifying representations

Name	in / out	parameter	function
TemplateManagerBox	$V \rightarrow V'$	*template name*	applying a template
AxisBox, OriginBox	$V \rightarrow V'$	*attribute name*	specifying a coordinate system
ContainerBox	$V \rightarrow V$		displaying visualization results
OverlayBox	$V^1 + V^2 \rightarrow V'$		overlaying several visualizations

Table 2. Operators for specifying queries

Name	in / out	parameter	function
TableBox	$\phi \rightarrow V$	*table name*	selecting a database relation
SelectBox	$\sigma(V) \rightarrow V'$	(spacial position)	selecting records it encloses
RecordFilterBox	$\sigma(V) \rightarrow V'$	*condition*	adding a new condition
JoinBox	$V^1 \bowtie V^2 \rightarrow V'$	*join condition*	joining two relations

Table 1 and 2 show operators provided by VERD framework. These operators can be classified into two categories, visual specifiers and query specifiers. Note that each operator does not correspond to either a relational operation or a clause of a database query. Each operator is designed to correspond to a user's primitive operation for specifying visualizations and database queries.

Operators perform their functions by executing composite operations consisting of several relational operations. Each operator takes single or multiple visualization view relations as its input. Some types of operators take parameters specified by users. Most parameters (e.g. the name of the table to visualize) are directly specified. Parameters of some types of operators (e.g. SelectBox) are specified by spatially arranging these operators. Each operator modifies input

visualization view relations according to its function and given parameters, then it outputs a modified view relation.

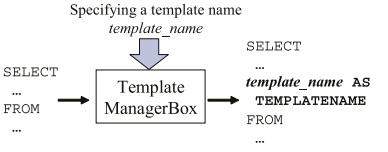

Fig. 4. Example of query rewriting by an operator

Each operator modifies visualization views relation by rewriting database queries that specify that views. Figure 4 shows that a TemplateManagerBox performs its function by rewriting an input query to specify the value of `TEMPLATENAME` attribute. *A query flow diagram* is a flow diagram consisting of such operators and connections among them, and it works as an SQL query generator for specifying visualization view relations. Figure 5 shows a query flow flow diagram that specifies the visualization in Figure 1. Database queries flow through this diagram, and are modified by each operator.

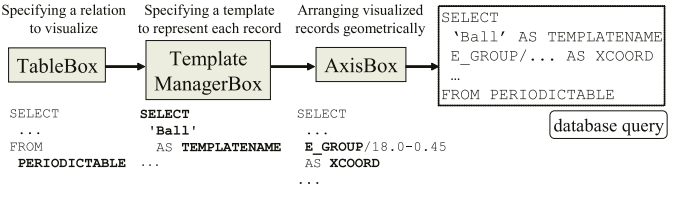

Fig. 5. Example of a query flow diagram

Operators are also interactive components. Users can combine operators by connecting their jacks and plugs as if they connect AV components. Figure 6 shows a screenshot of a query flow diagram specifying the visualization illustrated in Figure 1.

2.3 Visualization Examples

Magic Lens. Figure 7 illustrates an example query flow diagram that defines a magic lens [1]. It displays detailed information about records included in the selected region. In this example, a base of a magic lens, which is represented as a rectangular frame, consists of a SelectBox and an OverlayBox. The SelectBox specifies a region to apply additional representations, and the OverlayBox overlays new representations over the original visualization. By just dropping operator components into this frame, users can specify representations that the magic lens applies to each target record. Users can apply additional representations by dropping such a magic lens into some visualization result.

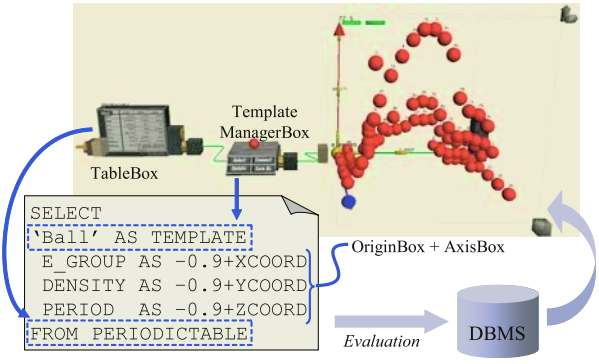

Fig. 6. Screenshot of a query flow diagram

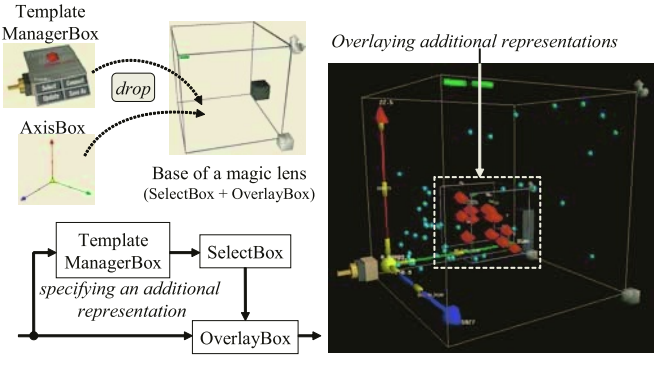

Fig. 7. Magic lens

Visualizing Details on Demand. In VERD framework, any portion of visualization corresponds to some view relation. Therefore, each component contained in a visualization result can also work as a view relation. Figure 8 illustrates a visualization process starting with the visualization result obtained in Figure 6. In this example, visualization (A) corresponds to a visualization view relation (i), and each ball representing an atom corresponds to a small visualization view relation with a single tuple represented by this ball.

Such a portion working as a visualization view relation can be joined with another relation. In this example, the isotopes of ruthenium are retrieved by joining a relation storing major isotope data with a relation (ii) corresponding to ruthenium, and visualized as the visualization (B). A user can visualize isotopes of various elements by only dropping balls into a SelectBox.

Nested Visualization. Figure 9 shows an example of nested queries and nested visualizations. This example visualizes the prefectural cabbage production in Japan. In this example, a ContainerBox is used as a template to visualize each prefecture. This template has two tiny AxisBoxes. This template is designed to visualize an annual cabbage production change of each prefecture. A Container-

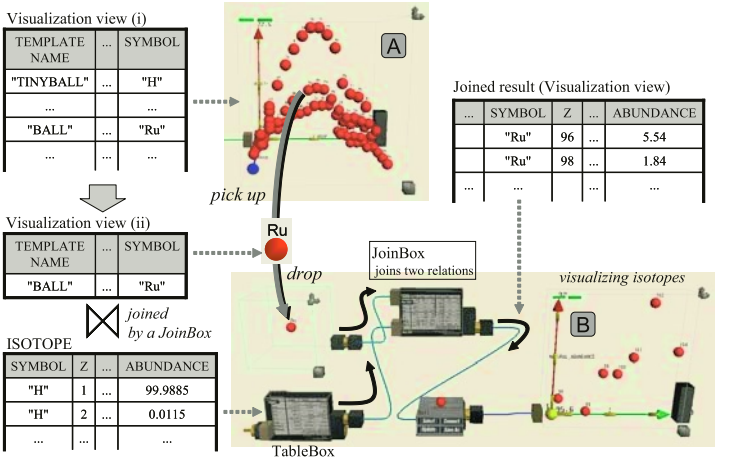

Fig. 8. Visualizing details on demand

Box used as a template gives users a nested visualization. In this example, a large ContainerBox visualizes prefectural cabbage production amounts in 1991 as shown in (A). Each small ContainerBox used as a template visualizes the annual cabbage production change of the corresponded prefecture as shown in (B). Users can select each of these small ContainerBoxes as a source data set to visualize its related information. In this example, three small ContainerBoxes are selected. The visualization (C) shows annual production changes of three similar vegetables, cabbage, Chinese cabbage, and lettuce, in the selected different prefectures. In this example, users can see the general trend of lettuce production which have increased instead of Chinese cabbage.

3 Integration with Related Web Resources

In this section, we extend our VDRD framework to integrate database contents with related Web contents.

3.1 Web View Relations

In VERD framework, Web resources are integrated with local databases based on the relational database model. In our approach, we extract data from a target Web document into a tabular form, and access it through a relational view called *a Web view relation*. A Web view relation associates a relational schema to relevant Web resources. Therefore, data integration between local relations and related Web resources is performed by specifying relational operations between local relations and Web view relations.

To perform data integration among heterogeneous Web resources, we have to associate the set of these Web resources to a general data structure like a relational table. This is a difficult task because each Web document has a complicate

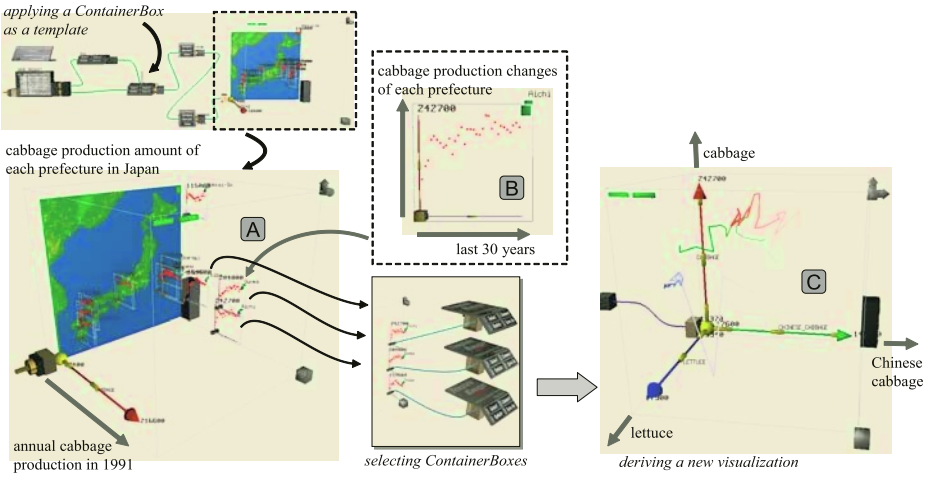

Fig. 9. Example of a nested visualization

and different structure. However, we can make this task easier by focusing only on required portions of documents and by associating them to a relational table. To perform this operation, we need *a Web wrapper* for extracting mutually related resources from Web documents. Moreover, we need an interactive way of quickly developing a Web wrapper.

Generally speaking, a wrapper means a software tool for extracting structured information from unstructured data or semi-structured data. A Web wrappers extracts a relation from a set of HTML documents distributed over the WWW. There are many studies on information extraction from Web documents. Various ways of implementing Web wrappers have been already proposed. In this work, we adopted a DOM (document object model) -pattern-based wrapper. An HTML document is a semi-structured document whose structure can be represented as a DOM tree. Our wrapper scans a DOM tree of a source HTML document, and finds a sub tree that matches with a given template pattern. Then the wrapper extracts data values from the found sub tree, and imports them as attribute values of the Web view relation.

A specification of a Web wrapper is described using the extraction path notation. The extraction path notation is an extension of the XPATH [2] notation. Extended features include the notation for specifying branched patterns of DOM fragments. Data mapping between HTML nodes and the attributes of a Web view relation can be specified using the extraction path notation.

Figure 10 shows an example of a Web wrapper and its specification described using the extraction path notation. Our notation uses a comma and braces to specify branched patterns for matching. In this example, the first and the second $\langle TD \rangle$ nodes appearing in each row of a table are selected. Attribute values or texts of HTML nodes of the matching sub tree are associated with the attributes of the Web view relation schema. For this purpose, the description of each ex-

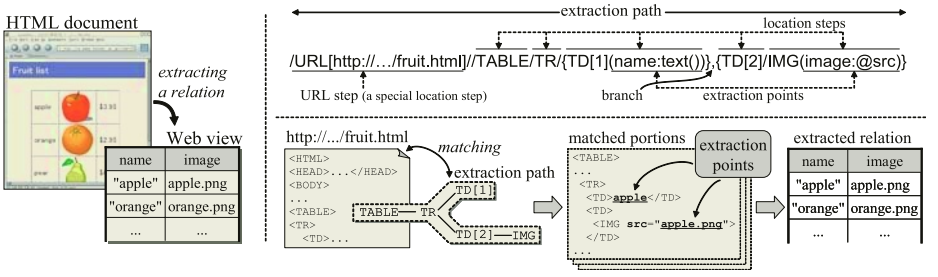

Fig. 10. Example of a Web view relation and an extraction path

traction path includes extraction points. In this example the text enclosed by the first $\langle TD \rangle$ node tags is extracted as the attribute 'name'. The attribute src of $\langle IMG \rangle$ node is extracted as the attribute 'image'.

Our notation uses backward references to define more complicated extractions. Moreover, source information may be often distributed in a set of HTML documents related to each other by hyperlinks. A Web view must be defined by visiting all the related HTML documents. Our notation for describing extraction paths is designed to satisfy these requirements.

3.2 User Interface for Specifying Web Wrappers

Our visualization system provides a graphical user interface to specify Web wrappers. Our interface has the dynamic previewing mechanism for supporting users to quickly and interactively specify an arbitrary Web wrapper. Usually, the wrapper specifying process has many repetitions of try-and-error tasks. To specify a wrapper that correctly works, a user tries to extract data using a provisional wrapper specification, and repeat the refinement of the current wrapper specification. The purpose of our dynamic previewing interface is to improve the wrapper specification task by shortening the time required for each try-and-error step.

Figure 11 shows on outline of this interface. To quickly specify a wrapper specification, our system uses a wrapper induction algorithm that generates a wrapper specification from given examples of extracted data. We developed a lightweight algorithm for the wrapper induction. With this algorithm, our system generates a wrapper from example HTML nodes selected from the source Web page within 100 miliseconds.

Our system provides a Web browser interface for users to directly specify extraction examples on a Web page. Users can select an HTML node as an example through mouse operations. Moreover, when the mouse cursor is moved, our browser immediately selects an HTML node under the mouse cursor as a candidate example, and generates a Web wrapper from a set of example nodes. The generated wrapper is executed immediately, then extracted HTML nodes by this wrapper are highlighted in the browser interface as an extraction preview. The latency time required for completing a preview update after a mouse cursor movement is less than 100 milliseconds.

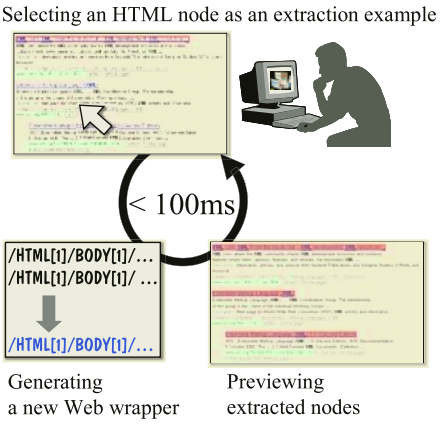

Fig. 11. The architecture of dynamic previewing

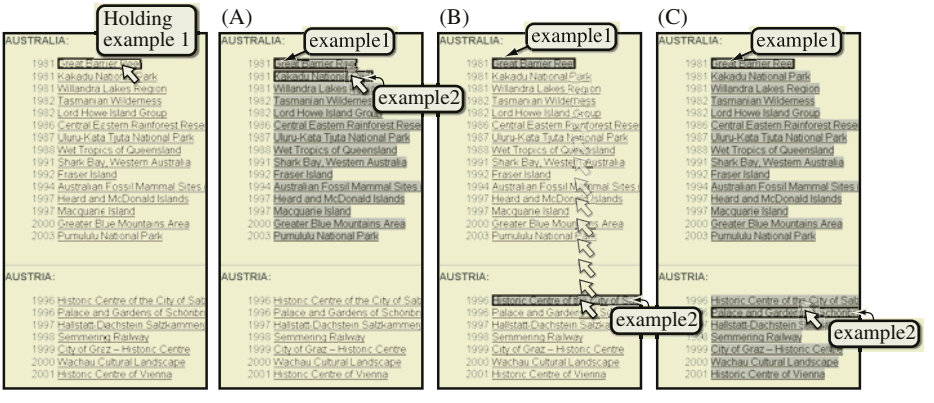

Fig. 12. Example of dynamic previewing for specifying wrapper

Figure 12 shows that our interface smoothly highlights items to be extracted in response to the mouse movement. In this example, a user is specifying a Web wrapper on 'The World Heritage List' page presented by UNESCO [3]. The dynamically updated preview suggests various patterns of extraction results such as (A) all heritages of one nation, (B) the heritage that appears first in each nation's heritage list, and (C) all the heritages of all the nations. User can quickly find good extractions by only moving the mouse cursor over the target Web page.

3.3 Visualization Example Integrated with Web Resources

VERD framework provides a special operator called a *WebViewBox* to execute a Web wrapper defined using an extraction path expression. Figure 13 shows an example of visualizing information of the world heritages by using this operator. In this example, the nation's name, the registration year, the name and photograph of each heritage are extracted from 'The World Heritage List' Web site

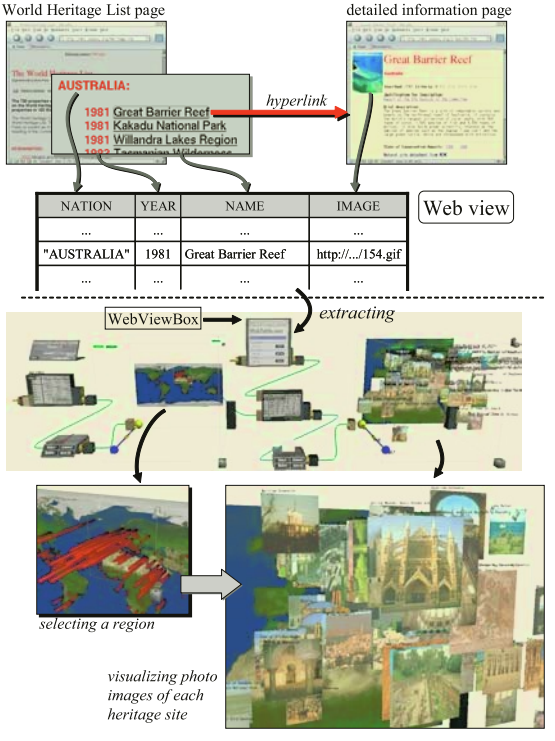

Fig. 13. Example visualization of The World Heritage List

by the WebViewBox. We can access these extracted data through the Web view relation.

In this example, we created a visualization from data stored in a local database. The GDP, longitude and latitude of each nation are obtained from a local database, and are visualized as the left visualization in Figure 13. Then, we select nations in Europe by using a SelectBox, and join the selected portion with the Web view relation. As a result, each heritage's photograph obtained from UNESCO's page is visualized in the right visualization, and is spatially arranged according to the longitude and the latitude obtained from a local database.

3.4 Wrapping Web Applications

In VERD framework, a function of each Web application is also abstracted as a relational view. A function of a Web application is treated as a relation between user's requests and system's answers. This relation is considered as a Web view relation. Each Web application's transaction that consists of a user's request and an application's answer is treated as an instance tuple of the Web view relation. Because each transaction corresponds to each tuple, this abstract method works well for stateless services that process each transaction independently of other transactions. Typical Web applications like search services and online dictionaries are of this type.

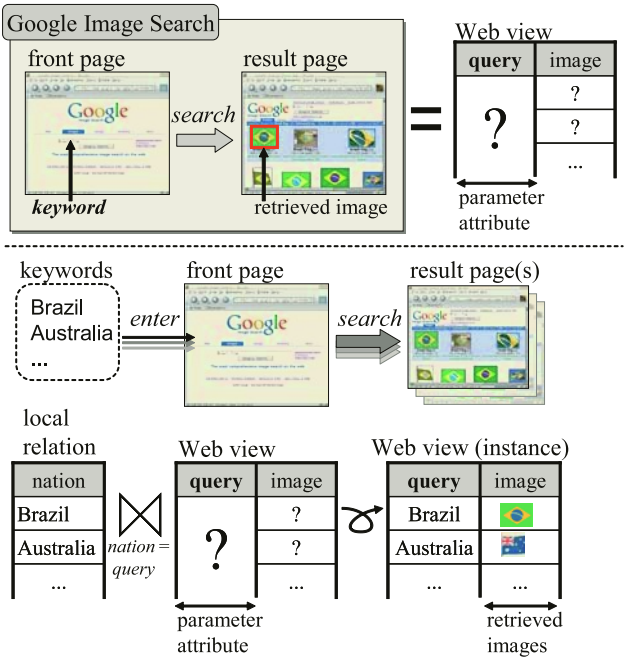

Fig. 14. Wrapping a Web application

To use a Web application, users have to fill in a form of an HTML page with parameter values, and to send a request as well as these parameter values to the corresponding Web server by using HTTP. We associate parameters of a Web application with the special attributes of the Web view relation. These Web view attributes are called *parameter attributes*. Usually a Web application returns an HTML document as a reply to the user's request. Therefore, we can extract return values from a Web application by specifying a Web view on the returned page.

A Web view relation with parameter attributes is also a view relation. However, when we try to obtain instance values of such a Web view relation, we need to specify a set of candidate values of its parameter attributes. These candidate values of parameter attributes are sent to the Web application page one by one, and, for each of them, a relation is extracted from the reply pages to fill in other attribute values of the Web view relation. In most cases, such candidate values of parameter attributes are specified by a relational join with another relation.

In Figure 14, we define a Web view relation by wrapping the Google Image Search service. This Web view relation extracts a set of URLs that point to images with the highest rank in the Google image search for each given keyword. This Web view relation has a parameter attribute to specify a keyword. This parameter attribute can be associated with nation names through a relational join with a local relation about nations in the world. As a result, for each nation, the image with the highest rank is extracted. By joining a Web view relation

with another Web view, we can define a functional composition combining two Web applications.

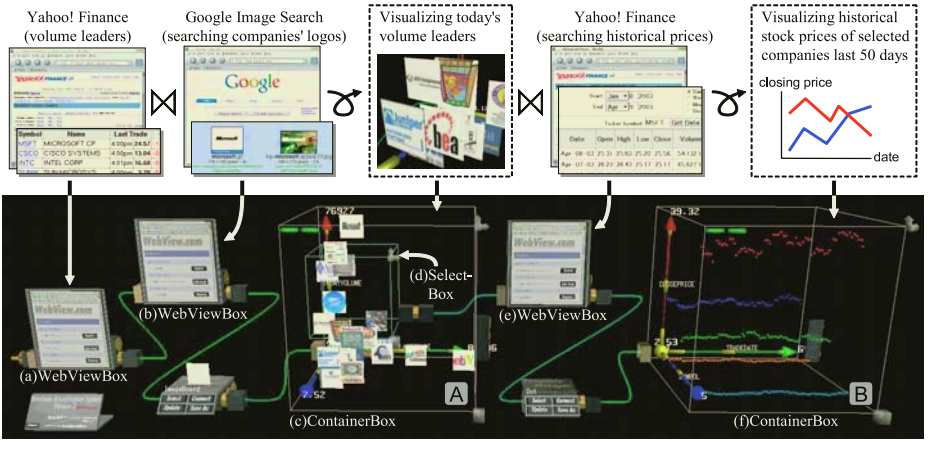

Fig. 15. Integrating multiple Web applications

Figure 15 shows an example functional composition of Web applications. In this example, the Web view relation specified by the WebViewBox (a) extracts today's volume leaders of NASDAQ market, and follows hyperlinks to the profile pages to extract the full names of the companies. The WebViewBox (b) defines a Web view relation by wrapping the Google Image Search service. This Web view is joined with the Web view specified by (a) to search for the logo images of the retrieved companies. The ContainerBox (c) visualizes extracted data by representing each company with its logo image and by arranging such logos geometrically in a 3D space according to the last trade price, the price change, and the trading volume of company stocks. The SelectBox (d) is used to select companies it encloses. A view relation corresponding to the selected companies is further joined with another Web view relation that is specified by the WebViewBox (e) to retrieve stock prices in the last 50 days. As a result, the stock prices of the selected companies during the last 50 days are visualized by the ContainerBox (f).

4 Implementation

We have implemented our system using a 3D visual component-ware system IntelligentBox [4]. The IntelligentBox system architecture provides 3D visual components called boxes. Each box is an interactive and functional component, and has slots as a logical interface for accessing its internal state. Users can specify slot connections between two boxes to define operation inter between them. We have implemented operator components as boxes. This allows us to connect operators using slot connections.

5 Related Work

There are many studies on interactive information visualization. DEVise [5] and Tioga-2 [6] are similar systems to the system proposed in this paper. These systems are based on the relational database model. In contrast to these systems that can represent an individual visualization display as a relation, VERD framework can further represent each constituent element of visualizations as a relation. This feature of VERD framework allows us to flexibly construct exploratory data visualization environment. For instance, the details-on-demand interface we showed in 2.3 can be easily constructed by naturally applying VERD framework because each visual component is also associated with a visualization view relation.

The flow diagram is a popular model for defining visual programming environments, and has been adopted by many interactive visualization systems such as AVS [8], DataExplorer [9], and Tioga-2 [6]. However, these systems distinguish components constituting flow diagrams from components constituting visualizations. On the other hand, in VERD framework, both types of components are equivalent in outputting some visualization view relations. Therefore, our system allows mixed use of operators and visualizations, and allows us to construct complicated visualizations such as the nested visualization example in Section 2.3.

Supporting OLAP (online analytic processing) operations is a future topic of our study. Polaris [10] is a visualization system for OLAP and it allows us to explore in a large dataset by using several OLAP operations such as drilling-down and pivoting. This system works well with one multidimensional dataset. Extended VERD framework will support dynamic data combination between multidimensional datasets and other datasets.

Data extraction from Web documents is one of the active research areas in the last decade. Many studies for the Web wrapping have been reported. In particular, several studies [11] [12] proposed graphical interfaces for specifying wrappers, but these studies have not focused their attention on repetitive try-and-error refinement of wrappers. LEXiKON [13] allows us to incrementally refine a wrapper through try-and-error repetitions. However, each trial step takes long because it has only a conventional Web form interface. Our wrapper definition interface for repetitive refinement is closely related to *dynamic query interfaces* [14]. Both a dynamic querying interface and our interface tightly associate a visualization with user's manipulations and allow users to quickly scan in a large exploration space of parameter values or extraction examples.

6 Concluding Remarks

In this paper we have proposed a new framework which provides an integrated environment for visualizing both database records and Web content objects. In our VERD framework, each visualization corresponds to a view relation, and is specified by a database query. Users can create such queries by interactively

combining operator components. In addition, we have extended this framework to support the relational data extraction from HTML documents over the Web. Users can extract relations from the Web by specifying Web view relations, and combine them with local database relations to visualize database records together with their related Web contents. VERD framework can also define functional compositions of Web applications by using relational operations.

References

1. Benjamin B. Bederson and James D. Hollan and Ken Perlin and Jonathan Meyer and David Bacon and George W. Furnas. Pad++: A Zoomable Graphical Sketchpad For Exploring Alternate Interface Physics. *Journal of Visual Languages and Computing*, 7(1), pp. 3-32, 1996.
2. XML Path Language (XPath). http://www.w3c.org/TR/xpath/
3. The World Heritage List. http://whc.unesco.org/heritage.htm
4. Y. Okada and Y. Tanaka. IntelligentBox:a constructive visual software development system for interactive 3D graphic applications. *Proc. of the Computer Animation 1995 Conference*, pp. 114-125, 1995.
5. Miron Livny, Raghu Ramakrishnan, Kevin Beyer, Guangshun Chen, Donko Donjerkovic, Shilpa Lawande, Jussi Myllymaki, and Kent Wenger. DEVise: Integrated Querying and Visual Exploration of Large Datasets. *Proc. of ACM SIGMOD '97*, pp. 301-312, May, 1997.
6. Alexander Aiken, Jolly Chen, Michael Stonebraker, and Allison Woodruff. Tioga-2: A Direct Manipulation Database Visualization Environment. *Proc. of the 12th International Conference on Data Engineering*, pp. 208-217, February, 1996.
7. Chris North, Ben Shneiderman. Snap-Together Visualization: A User Interface for Coordinating Visualizations via Relational Schemata. *Proc. of ACM AVI 2000*, May, 2000.
8. Craig Upson, Jr. Thomas Faulhaber, David Kamins, David Laidlaw, David Schlegel, Jeffery Vroom, Robert Gurwitz, and Andries van Dam. The Application Visualization System: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4), pp. 30-42, 2000.
9. Bruce Lucas, Gregory D. Abram, Nancy S. Collins, David A. Epstein, Donna L. Greesh, and Kevin P. McAul. An architecture for a scientific visualization system. *Proc. of IEEE Visualization '92*, pp. 107-114, October, 1992.
10. Chris Stolte, Diane Tang and Pat Hanrahan. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Visualization and Computer Graphics*, 8(1), pp. 52065, Jan/Mar 2002
11. Ling Liu, Calton Pu, Wei Han. XWRAP: An XML-enabled wrapper construction system for Web information sources. *Proc. of the 16th International Conference on Data Engineering*, pp. 611-621, March, 2000.
12. lberto H. F. Laender, Berthier Ribeiro-Neto, and Altigran S. da Silva. DEByE - Data Extraction By Example. *Data and Knowledge Engineering* 40(2), pp. 121-154, 2002.
13. Gunter Grieser, Klaus P. Jantke, Steffen Lange. Gunter Grieser, Klaus P. Jantke, Steffen Lange *Proc. of 13th Int. Conference on Algorithmic Learning Theory*, pp. 173-187, 2002.
14. Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6), pp. 70-77, 1994.