Journal of Visual Languages and Computing 21 (2010) 209–229

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

DaisyViz: A model-based user interface toolkit for interactive information visualization systems

Lei Ren [a,*], Feng Tian [b], Xiaolong (Luke) Zhang [c], Lin Zhang [a]

[a] School of Automation Science and Electrical Engineering, Beijing University of Aeronautics and Astronautics, Beijing 100191, China
[b] Human–Computer Interaction Lab and the Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China
[c] College of Information Sciences and Technology, Pennsylvania State University, PA 16802, USA

ARTICLE INFO

Keywords:
User interface
Information visualization
Toolkit
Multiple coordinated views
Model-based interface development

ABSTRACT

While information visualization technologies have transformed our life and work, designing information visualization systems still faces challenges. Non-expert users or end-users need toolkits that allow for rapid design and prototyping, along with supporting unified data structures suitable for different data types (e.g., tree, network, temporal, and multi-dimensional data), various visualization, interaction tasks. To address these issues, we designed DaisyViz, a model-based user interface toolkit, which enables end-users to rapidly develop domain-specific information visualization applications without traditional programming. DaisyViz is based on a user interface model for information (UIMI), which includes three declarative models: data model, visualization model, and control model. In the development process, a user first constructs a UIMI with interactive visual tools. The results of the UIMI are then parsed to generate a prototype system automatically. In this paper, we discuss the concept of UIMI, describe the architecture of DaisyViz, and show how to use DaisyViz to build an information visualization system. We also present a usability study of DaisyViz we conducted. Our findings indicate DaisyViz is an effective toolkit to help end-users build interactive information visualization systems.

1. Introduction

In the past 20 years, an abundance of novel information visualization (*infovis*) techniques have emerged to support visual representation and interaction techniques. Various layout algorithms have been developed to assist the visualization and understanding of complex datasets organized in diverse structures, such as tree, network, multi-dimensional, and temporal data [1]. Interaction designs provide powerful support for core user tasks, such as those described in the visual information-seeking mantra [6], dynamic query [2], overview+detail [3], focus+context [4], panning+zooming [3], and multiple coordinated views [5].

Infovis is generally recognized as an effective way to help users acquire knowledge [1], although effectively applying interactive information visualization in domain-specific problems (e.g., scientific research, business intelligence) still faces some challenges. One of the challenges is to provide a balance between the flexibility required to integrate domain-specific models into visualization tools and the simplicity needed to build and access visualization tools. Despite the challenges various *infovis* toolkits, such as PAD++ [8], Jazz [9], Piccolo [10], Snap-Together [11], GeoVista [12], Improvise [41], PRISMA [13], Polaris [14], XML toolkit [15], Fekete's *infovis* toolkit [16], prefuse [7], Many Eyes [17], and ComVis [18], have been developed. Most of the toolkits support the rapid prototype of visualization systems, the addition of domain knowledge, and domain-specific task models.

* Corresponding author.
E-mail addresses: leo.renlei@gmail.com (L. Ren), post.wimp@gmail.com (F. Tian), lzhang@ist.psu.edu (X. (Luke). Zhang), zhanglin@buaa.edu.cn (L. Zhang).

1045-926X/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jvlc.2010.05.003

Although the use of these toolkits involves a significant amount of coding, which prevents most end-users for being able to use them. On the other hand, software tools like Spotfire (http://spotfire.tibco.com) and Tableau (http://www.tableausoftware.com/) offer users a collection of visualization tools that end-user can directly use. These software packages are based on pre-defined models accounting for basic user needs and task requirements. Therefore, plugging in new domain-specific knowledge to these customize visualization tools is often not supported. As visualization tools become more important to data analysis and knowledge exploration in different domains, at different levels of analysis, and by people with different technical skills, this challenge cannot be ignored. This problem needs to be addressed so that visualization tools and systems are easy built by end-users, and are customizable with user defined domain-specific models, data, and user task.

One approach to tackle this problem is to use model-based design for interactive systems [19]. This approach allows a designer to build a declarative interface model that specifies the high-level domain requirements and then use the model to guide the interface development process. This model-based design paradigm offers a number of benefits, including user-centered development cycle, centralized user interface specification, comprehensive design tools for interactive, and automated development, and reuse of user interface designs [19]. For end-users, the benefits of this model-based approach are valuable, because they can build domain-specific *infovis* applications by crafting a model on user interface (UI) and interaction tools, rather than writing code and algorithms to create UI and interaction tools from scratch.

While this model-based approach allows end-users to focus on the primary tasks of exploring the data through visualization, end-users may still face a challenge in developing a model to accurately describe the relationship between data, visualization tools, and user tasks. For example, when analyzing the cause of the bottleneck in a product line with visualization tools, users can use coordinated, multiple views [5] to analyze several facets of the product line: the elements in the product line, the overall relationship of all components in the product line (e.g., elements, machines, transportation vehicles, and personnel), the task hierarchy that is related to the bottleneck, and the process and progress of individual tasks. A possible visualization-based solution to this analytics task is to have a set of visualization tools for different types of data, such as a multi-dimensional visualization tool to illustrate the different attributes of individual elements, a network visualization tool to present the relationship among all components of the product line, a tree view to visualize the task hierarchy, and a timeline to map the process and progress of tasks. By linking information artifacts in these different visualization tools, users can examine what tasks have been delayed and what the consequences of the delay might be. However, to use a model-based approach to design visualization tools for this task, users need to develop a model that describes not only what visualization tools to use and what datasets to analyze, but also what information facets should be mapped to individual tools, what data sources these facets correspond to, what relationships all the facets of interest have, etc. Such a complicated model could be difficult for users.

To address this challenge users face when using the model-based approach to design visualization systems, we developed DaisyViz, a model-based user interfaces toolkit for the development of interactive *infovis* systems with multiple coordinated views. The key features of DaisyViz include the following:

- unified data models suitable for multiple data types and multiple coordinated views;
- diverse and extensible layout algorithms for tree, graph, multi-dimensional, and temporal data types;
- rich interactive tools (e.g., dynamic query, overview +detail, panning+zooming, focus+context, and multiple coordinated views) to support diverse *infovis* tasks; and
- model-based interface development for end-users.

The goal of DaisyViz is to simplify the design and implement of *infovis* applications. DaisyViz is based on an interface model called UIMI consisting of three declarative models—data model, visualization model, and control model. End-users construct interface models and then generate profiles that are parsed by DaisyViz to automatically generate an interactive *infovis* application.

The paper is structured as follows. Section 2 reviews related work. Section 3 defines the UIMI model, and Section 4 describes the architecture of DaisyViz. In Section 5, we discuss a novel visualization technique, DOI-Wave, in DaisyViz and its integration into DaisyViz. Then, the paper presents an example of using DaisyViz to build a visualization data system for data analysis in a manufacturer. After the introduction of a usability study on the use of the DaisyViz in Section 7, the paper concludes with future work.

2. Related work

Over the last few decades, a massive amount of research has been done on *infovis* to explore novel visual representations of data, to design new interaction techniques for the manipulation of information artifacts to reveal underlying knowledge, to build new toolkits for the creation of *infovis* applications, etc. Some examples of popular visual representation designs include tree visualizations (e.g., TreeMaps [20], Cone Trees [21], Hyperbolic trees [22], SpaceTrees [23], DOITrees [24], InterRing [25], and CirclePacking [26]), graph visualization (e.g., Clustered Graph [27], Force-Directed Graph [28], and Radial Graph [29]), multi-dimensional visualizations (e.g., Scatterplots [30] and Parallel coordinates [31]), and temporal visualizations (e.g., Theme River [32], Circle View [33], and TimeWhell [34]). Advances were also made on interaction techniques to facilitate user navigation and exploration of visual information, including dynamic query [2], overview+detail [3], focus+context [4], panning+zooming [3], and multiple coordinated views [5].

Some *infovis* tools have been developed especially for the visualization of a given data type, such as static graph

visualization (e.g., Graphviz [35] and GVF [36]) and multi-dimensional visualization (e.g., Polaris [14]). These tools cannot cover all seven types of data [6], especially the popular data types of tree, graph, multi-dimensional, and temporal data. Facing complex information datasets, users need to visualize and explore many facets of the information conceptual entity to gain insight into each facet and, most importantly, to understand the hidden relationships among the facets. Thus, it is important to integrate these type-specific visualization tools.

Multiple coordinated views are an effective design to support interactive visual analysis by combining two or more coordinated visualization tools. Examples of such design include Snap-Together [11], GeoVista [12], Improvise [41], PRISMA [13], and ComVis [18]. These tools, however, largely focus only on implementing the coordination mechanisms, and provide less support for the combination of a wide range of visualization and interaction techniques, such as the visualizations for tree, graph, multi-dimensional, and temporal data, and the interaction techniques for overview, zoom, filter, and details-on-demand. DaisyViz, in contrast, can provide generic data models suitable for different data types in multiple coordinated views, and can support multiple common tasks that extend Shneiderman's taxonomy [6] in each view.

Various toolkits, such as XML toolkit [15], Fekete's InfoVis toolkit [16], prefuse [7], Many Eyes [17], JViews of ILOG [46], and Tom Sawyer Perspective [47], have been developed to simplify the design and implementation of *infovis* applications. These tools often provide generic data models to cover a variety of data types and visualization components that encapsulate algorithms into widgets. XML toolkit, based on the standard Java data structures (e.g., Treemodel), is a package of *infovis* algorithms, rather than a toolkit for end-users. Although Fekete's InforVis toolkit provides a library of existing visualization widgets to support often-seen layout algorithms and interaction techniques (e.g., focus+context and dynamic query), it lacks the flexibility and reusable components for constructing novel *infovis* applications [7]. Prefuse [7], in comparison to Fekete's *infovis toolkit,* offers a simpler way to develop visualization applications by encapsulating the details of layout algorithms and interactive tools, and allows users to build systems by assembling pre-developed program blocks. However, it does not provide a design tool that includes GUI for end-users, so users still have to write codes according to predefined standards when customizing complex domain-specific applications. The goal of Many Eyes [17] is to support collaboration for visualization at a larger scale (e.g., the whole Internet). This system pays more attention to creating online visualizations tools for end-users. Its mechanisms for data models, visualization and interaction techniques are relatively simple, so it can hardly meet the demands of interactive *infovis* applications for complex domain-specific applications. The commercial systems, such as ILOG JViews Diagrammer [46] and Tom Sawyer Perspective [47], are used to help end-users to build web and desktop *infovis* applications. While ILOG JViews Diagrammer is a powerful tool that provides a broad range of modules for data integration, layout algorithms, rendering, and application deployment. It focuses more on the presentation of complex visual objects, such as graphs, rather than complicated user interaction activities. For example, the tool can support such activities as object selection, zooming, and panning, but more advanced user interaction techniques like dynamic query and focus+context are not included. Tom Sawyer Perspective is a tool to quickly and easily translate data into different types of synchronized views, but its support for diverse visualization layouts as well as complex interaction techniques is weak. In contrast to current solutions mentioned above, DaisyViz can provide a model-based design toolkit for end-users to simply create domain-specific interactive *infovis* applications that support unified data models and a wide range of commonly seen visualization tools and interaction techniques in an environment of multiple coordinated views.

3. UIMI: a user interface model for information visualization

DaisyViz is based on the idea of model-based interface development [19], which uses a declarative model of user interfaces to drive the development process. A user interface model abstracts the features of a user interface and represents all the relevant aspects of the user interface in a formal language. The user interface model, the core of development process, is then parsed according to knowledge bases to generate applications. From the perspective of end-users, their only design concern is to construct an interface model.

Fig. 1 shows our user interface model for *Infovis* (UIMI). In this model, users can construct a model by simply answering several questions:

- What facets of the target information should be visualized?
- What data source should each facet be linked to and how the facets are related to each other?
- What layout algorithm should be used to visualize each facet?
- What interactive techniques should be used for each facet (i.e., view) and for what *infovis* tasks?

The answers to these questions are then used to construct three declarative models of data, visualization, and control.

3.1. Data model

The data model in UIMI is to describe the unified data structures of multiple relevant facets of the target information. We use relational data schemas to manage data and to define the relationships between data attributes. The target data can be analyzed from different information facets. For a facet, data can be organized into tables in which each row corresponds to a basic data item and each column represents an attribute of data [37]. Foreign keys of the tables can describe the relationships

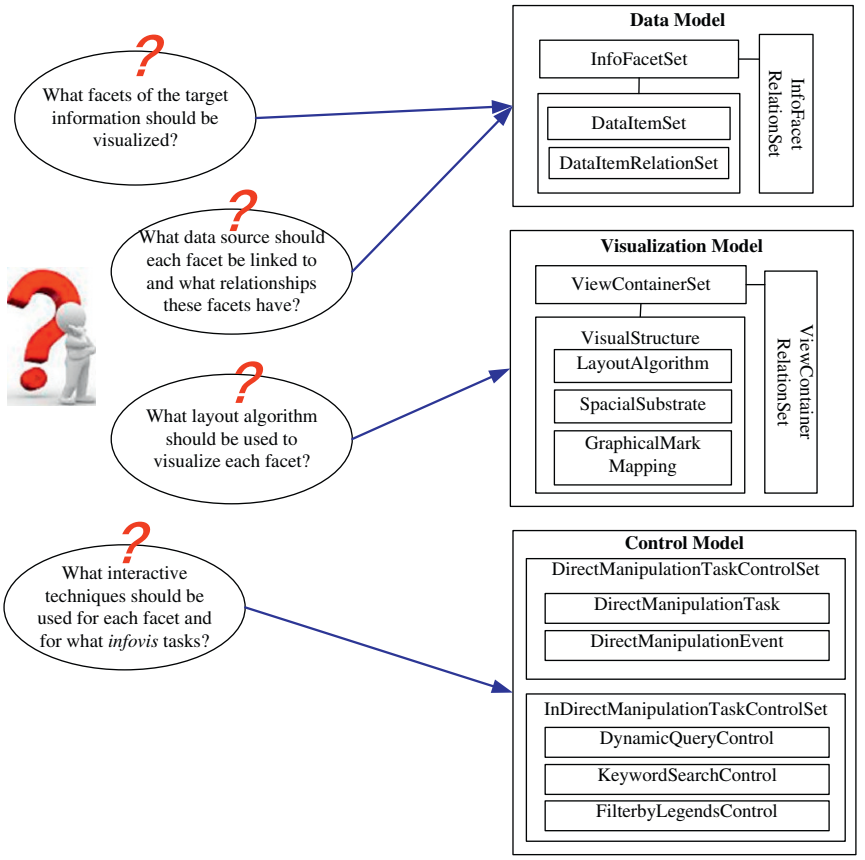

Fig. 1. The conceptual model of UIMI.

among the facets. Take the aforementioned example of bottleneck analysis in a product line that involves four information facets: product element, overall relationship among all components related to the product, task hierarchy, and the process and progress of tasks. The multi-dimensional data of product elements can be represented by a multi-column table, in which each data dimension of a product element is mapped to a column. The tree data of task hierarchy and graph data of the relationships among all components related to the product can be represented with two tables: one to specify individual nodes (tasks or components) and the other to define the relationships among nodes. The temporal data of the task process and the progress can also be managed by a table, in which each column is a time point and foreign keys can tie the data to the tables of product element and task hierarchy.

To formalize the representation of data, data facets, and relationship among facets with relational data schemas, we developed the following definitions.

Definition 1. A ***DataItem*** $DI = <DIID, Attribute_1, Attribute_2, \ldots, Attribute_n>$. *DIID* is the ID of a data item. $Attribute_i = <AttrName_i, MetaData_i>$ is a property of *DI*, where $AttrName_i$ is the property name of $Attribute_i$, and $MetaData_i$ is the metadata type of nominal, ordinal, and quantitative [1]. Here interval data are treated as quantitative data.

Definition 2. A ***DataItemRelation*** $DIR = <SourceDIID, TargetDIID, Direction>$ where *SourceDIID* and *TargetDIID* are the *DIID* of two data items of a link. $Direction \in \{direct, undirect\}$.

Definition 3. An ***InfoFacet*** $IF = <IFID, DISet, DIRSet, DataSource>$ is a facet of the target data. *IFID* is the ID of the facet of the target data. *DISet* is a set of *DI*. *DIRSet* is a set of *DIR*. *DataSource* is a pointer to a data source, which could be a formatted text file, an XML document, or a relational database. For a formatted text file or an XML documents, *DataSource* would be the path. For a relational database, *DataSource* would be the connection strings including server, database, username, password, etc.

Definition 4. An ***InfoFacetRelation*** $IFR = <SourceIFID, TargetIFID, KeyAttri, Direction>$ is a relationship between two facets. *SourceIFID* and *TargetIFID* are IDs of two relevant facets. Let *SourceAttriSet* be the set of attributes of data items in facet *SourceIFID* and *TargetAttriSet* be the set of attributes of data items in facet *TargetIFID*. Then, *KeyAttri* is defined as $KeyAttri \in SourceAttriSet \cap TargetAttriSet$. $Direction \in \{direct, non\text{-}direct\}$ defines whether the relationship is directional or non-directional between the source and target facets.

Definition 5. ***Data*** $= <$ *IFSet*, *IFRSet* $>$. *IFSet* is a set of facets of the target data, and *IFRSet* is a set of relationships among the facets.

3.2. Visualization model

The visualization model in UIMI defines the visual representations of data in user interfaces. This model concerns two primary issues: coordinated views and visual structure in each view. Coordinated views are a set of views that will be offered. Individual views are coordinated by sharing common data attributes. Each view defines a visual structure that describes layout algorithms, spatial substrate, and graphical mark mappings [1]. Here, spatial substrate specifies a set of axes that are involved in a view and data attributes that are mapped to individual axes. Graphical mark mappings express how the attributes of data should be encoded with the attributes of graphical marks (e.g., shape, color, size, orientation, texture, etc.). From the perspective of end-users, this model defines how many views should be included in a user interface, what visualization technique should be used in each view, and how the information is visualized. The visualization model contains the following formal definitions.

Definition 6. An ***Axis*** $= <$ *AxisID*, *AxisType*, *DIMappingAttri*, *ValueRang* $>$. *AxisID* is the ID of the axis. *AxisType* $\in$ {*NoAxis*, *NominalAxis*, *OrdinalAxis, QuantityAxis*} is the type of the axis. *DIMappingAttri* is the encoded attribute of data items. *ValueRange* is the predefined range of the axis.

Definition 7. A ***SpatialSubstrate*** is a set of axes.

Definition 8. A ***GraphicalMarkMapping*** *GMM* $= <$ *LabelMappingAttri*, *ShapeMappingAttri*, *ColorMappingAttri*, *SizeMappingAttri*, *OrientationMappingAttri*, *TextureMappingAttri* $>$ where *LabelMappingAttri*, *ShapeMappingAttri*, *ColorMappingAttri*, *SizeMappingAttri*, *OrientationMappingAttri* and *TextureMappingAttri* are the attributes of data items encoded with the attributes of graphical marks, including label, shape, color, size, orientation, and texture.

Definition 9. A ***VisualStructure*** *VS* $= <$ *VSID*, *Layout Algorithm*, *SpatialSubstrate*, *GMM* $>$. *VSID* is the ID of *VS*, and *LayoutAlgorithm* is the name of layout algorithm. *LayoutAlgorithm* $\in$ {TreeMaps, RadialGraph, ParallelCoordinates, ScatterPlot, DOITrees, 3DBarChart, …}. The set of *LayoutAlgorithm* can be extended when a new layout algorithm is added.

Definition 10. A ***ViewContainer*** *VC* $= <$ *VCID*, *IFID*, *VSID* $>$. *VCID* is the ID of a view container, *IFID* is the ID of a facet that will be visualized in this view container, and *VSID* is the ID of the visual structure in this view container.

Definition 11. A ***ViewContainerRelation*** *VCR* $= <$ *VCRID*, *SourceVCID*, *TargetVCID*, *KeyAttri* $>$ where *SourceVCID* and *TargetVCID* are the IDs of two coordinated views with the common attribute *KeyAttri* of data items.

Definition 12. ***Visualization*** $= <$ *VCSet*, *VCRSet* $>$ where *VCSet* is a set of view containers, and *VCRSet* is a set of *VCRs.*

3.3. Control model

The control model in UIMI describes the *infovis* tasks and interaction techniques used in each view. Based on the task taxonomy by Shneiderman [6] and new task requirements seen in recent toolkits [7–18], we developed a more detailed taxonomy for *infovis* tasks, which includes overview of multiple views, overview of visualization in a view, pan, zoom, filter by data attributes (i.e., dynamic query), filter by legends, keyword search, detail tooltip (i.e., detail-on-demand), and coordination (e.g., brushing-and-linking and drill-down). For the tasks of coordination, brushing-and-linking refers to a technique in which when an item is selected in one view highlights, the corresponding item(s) in another view will be highlighted; drill-down is a tool that loads related items in a view when they are selected in another [11].

The control model allows users to define what tasks should be involved and how tasks should be executed. They can specify the tasks needed by the requirements of applications and the concrete control model for every task that has been chosen. The interaction controls can be divided into two categories: direct manipulation and indirect manipulation. Direct manipulation is mainly used for overview of multiple views, overview of visualization in a view, pan, zoom, detail tooltip, and coordination. For the direct manipulation, users can describe which interaction events should be applied to achieve a task based on a set of predefined events (e.g., double-click on the blank area in a view to overview the visualization, or drag and drop a graphical node from one view to another to establish the coordination between two views). Indirect manipulation involves the use of control tools, such as sliders, text boxes, and legends, and is targeted by dynamic query, keyword search, or filter by legends. When defining indirect manipulation tools, users need to specify what data attributes should be mapped to these tools.

The control model includes the following components:

Definition 13. A ***DirectManipulationEventSet*** *DMES* $=$ $<$ OnItemLeftClick, OnItemLeftDoubleClick, OnItemRightClick, OnItemRightDoubleClick, OnItemHover, OnItemLeftDragDrop, OnItemRightDragDrop, OnViewLeftClick, OnViewLeftDoubleClick, OnViewRightClick, OnViewRightDoubleClick, OnViewHover, OnViewLeftDragDrop, OnViewRightDragDrop,… $>$. Here, some usual interaction events are predefined for users to choose to realize direct manipulation.

Definition 14. A ***DirectManipulationTaskControl*** *DMTC* $=$ $<$ *DMTask*, *DME* $>$. *DMTask* $\in$ {OverviewMultipleViews, OverviewVisualization, Pan, Zoom, DetailTooltip, Brushing-and-LinkingCoordination, DrillDownCoordination,…} is one of the tasks using direct manipulation. *DME* $\in$ *DMES* is the interaction event for *DMTask.*

Definition 15. A ***DynamicQueryControl*** *DQC* = < *DQCID*, *VCID*, *DIAttri*, *ValueRange* > . *DQCID* is the ID of a dynamic query slider. *VCID* is the view container that contains the slider. *DIAttri* is the attribute of data items that will be used in dynamic query. *ValueRange* is a predefined range of the slider.

Definition 16. A ***KeywordSearchControl*** *KSC* = < *KSCID*, *VCID*, *DIAttri* > . *KSCID* is the ID of a text box for keyword search. *VCID* is the view container for the text box. *DIAttri* is the attribute of data items that will be used to search for items in a view.

Definition 17. A ***FilterbyLegendsControl*** *FLC* = < *FLCID*, *VCID*, *GraphicalMarkAttribute* > . *FLCID* is the ID of a legend control. *VCID* is the view container where the legend control resides. *GraphicalMarkAttribute* is the attribute of graphical marks that will be used to filter items in a view.

Definition 18. ***Control*** = < *DirectManipulationTaskControlSet*, *IndirectManipulationTaskControlSet* > . *DirectManipulationTaskControlSet* is a set of controls using direct manipulation to support related tasks. *IndirectManipulationTaskControlSet* = < *DQCSet*, *KSCSet*, *FLCSet* > is a set of controls using indirect manipulation to support related tasks.

4. Architecture of DaisyViz

4.1. Overview

Fig. 2 shows the architecture of DaisyViz. DaisyViz is composed of four main parts: DaisyViz tools, DaisyViz

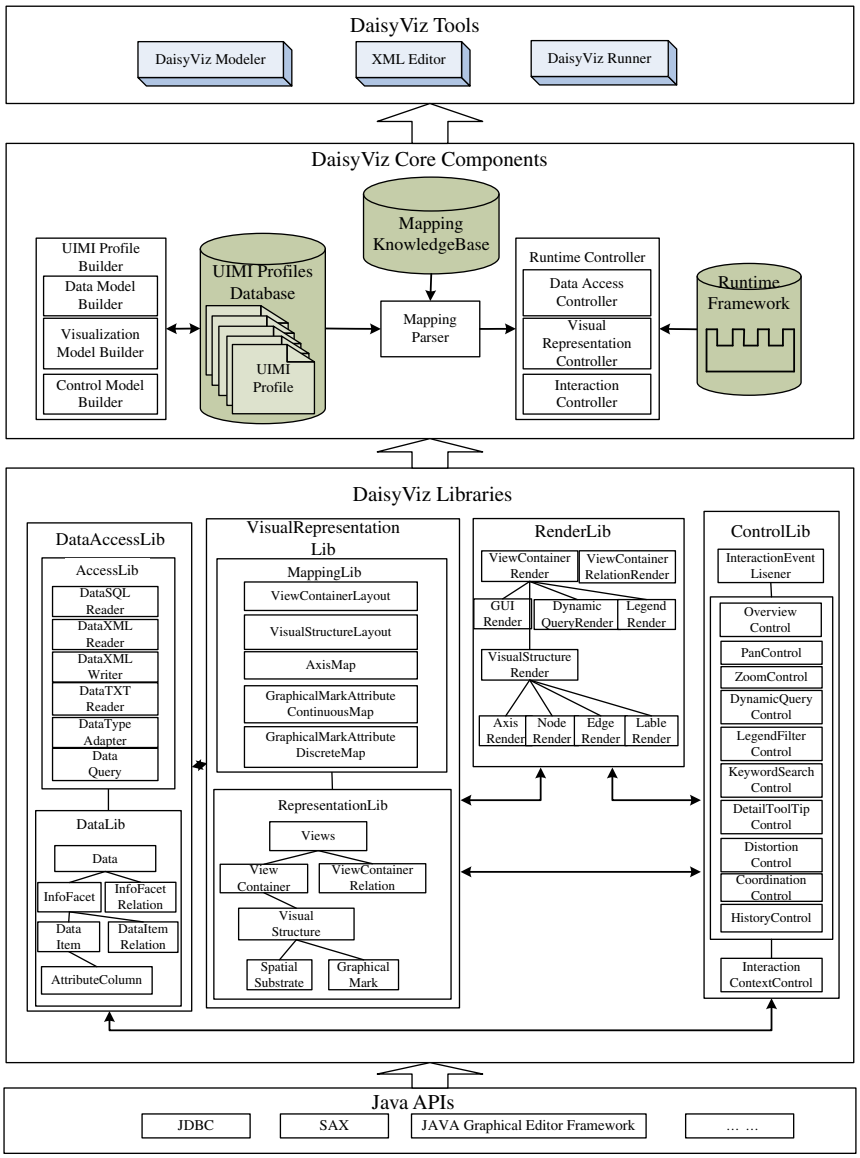

Fig. 2. Architecture of DaisyViz.

core components, DaisyViz libraries, and Java APIs. Java APIs include such components as JDBC, SAX, and Graphical Editor Framework, and provide the underlying support for all other parts. DaisyViz libraries, as the foundation of DaisyViz core components, cover the libraries used for data structures of information, data access, visual representation structures, visual mapping, graphical element rendering, and interaction events controlling.

DaisyViz core components support the main development process. The UIMI profile builder is responsible for UIMI modeling based on the requirements of the domain applications and then constructing UIMI profiles. The mapping parser, built on mapping knowledge bases, translates the declarative language in a UIMI profile into runtime parameters needed by the system runtime framework. The runtime controller, the master of runtime systems, controls the runtime mechanisms such as data access, visual representation mapping, graphics rendering, and interaction event processing.

DaisyViz provides end-users with tools to design and run domain-specific applications. The DaisyViz modeler can help users build UIMI models and construct profiles. The DaisyViz runner, a runtime tool, can generate applications through parsing UIMI profiles and thus support user's visual analysis. End-users can modify a UIMI profile with an XML editor.

4.2. DaisyViz core components

DaisyViz core components can be broken down into six parts: a UIMI profile builder, a UIMI profiles database, a mapping parser, a mapping knowledge base, a runtime controller, and a runtime framework.

A DaisyViz development process starts with UIMI modeling. The requirements of domain-specific applications are formalized to a UIMI profile. The UIMI profile builder includes three model builders to construct the data model, the visualization model, and the control model, as well as their profiles. Because of the wide acceptance of XML as the format for information exchanging, the UIMI profiles are built in XML schemas and can be edited by any XML editing tool. XML-based profiles are stored in a UIMI profile database. The profiles in the database can serve as templates for reuse and incremental development of domain-specific applications.

Because UIMI is a declarative model, it needs to be translated into a program language for the runtime system. The mapping parser, which is built on the mapping knowledge base, is responsible for the translation. The reasoning rules of knowledge are represented by Predicate Logic [38], which performs the reasoning based on the declarative profiles. The results of the mapping parser are the parameters that are essential for the runtime system.

Once the UIMI profiles are parsed, the runtime controller can embed the parameters generated by the mapping parser into the runtime system framework. As illustrated in Fig. 3, to run an application system, the runtime framework requires three types of parameters that are generated from the data model, the visualization model, and the control model, respectively. The parameters from the data model are used to access data sources and generate data structures. The parameters from the visualization model support the mapping process from data to visual representations. The parameters from the control model mainly handle the responses to interaction events and execute *infovis* tasks.

The runtime framework has five function modules and four modules related to data processing (Fig. 3). Five function modules include a module for data access, a module to map visual representations, a rendering module, a module of interaction event control, and a module for meta-task control. Four modules that concern data are a data module to manage data acquired from data sources, a visual representation module to manage view graphs, a graphical element module to handle rendered graphical objects, and an interaction context module to respond user actions.

All runtime processes are built on these nine modules. A runtime process starts with an interaction event triggered by a user's interaction with the graphical elements in the user interface. Based on the interaction context module, where current interaction state (e.g., selected items and search results) are recorded, the interaction event is recognized by the interaction event control module.

Interaction events here are classified into two types of tasks. The first type of tasks only concerns the transformation of graphical elements, such as overview, panning, zooming, and filtering by legends, without the need for data access. The second type of tasks (e.g., dynamic queries, keyword search, and view coordination) is tightly coupled with datasets. These two types of tasks are processed by the meta-task control module in different ways. For tasks only concerning graphic transformation, the meta-task module performs graphical operations on the graphical elements stored in the interaction context module and then renders them to the interface. For tasks requiring access to data sources, the meta-task control module communicates with the data access module for data query and update.

To produce visual results, raw data must be converted into structured data and then mapped to visual representations that can be rendered. The data access module connects to the data sources and formats the raw data into structures that can be processed by the data module and the meta-task control module. Then, the visual representation mapping module establishes the correspondences between data structures (e.g., facets, data items, attributes) in the data module and view structures (e.g., view types, spatial substrate, and graphical marks) in the visual representation module. These view structures are drawn on the interface by the render module.

4.3. DaisyViz libraries

The above DaisyViz core components rely on DaisyViz libraries. These libraries facilitate the development of *infovis* applications by providing the components

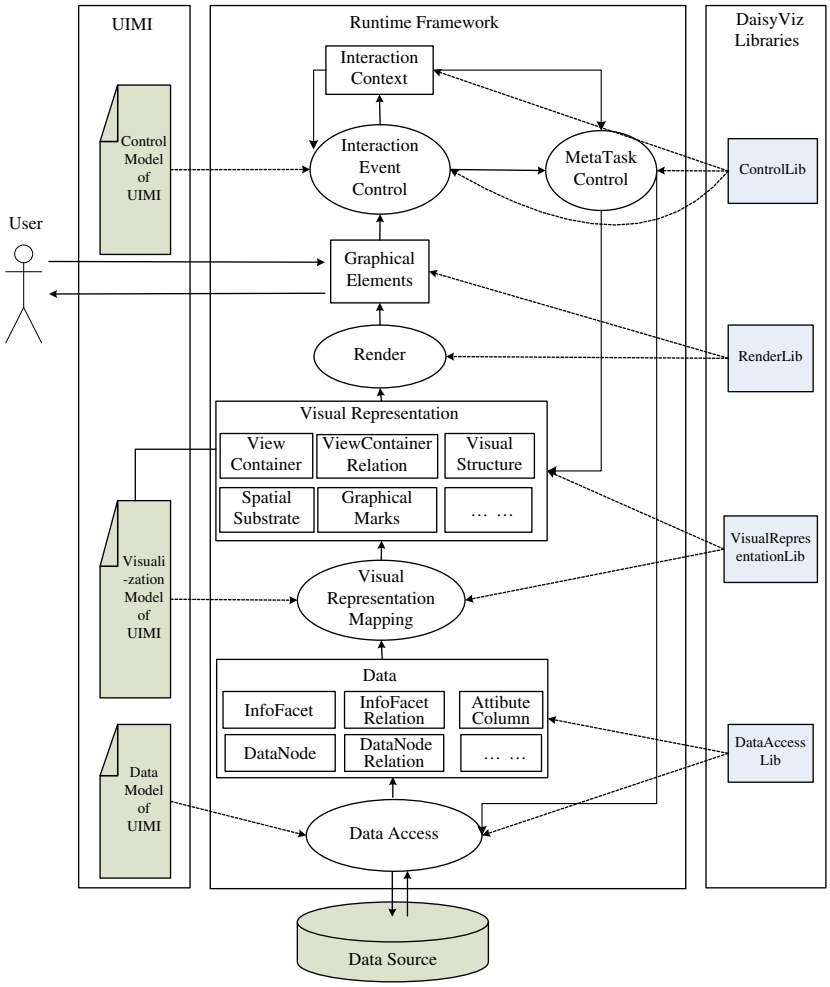

Fig. 3. Runtime framework of DaisyViz. (Squares: data Modules; ellipses: function modules.)

frequently used. Four libraries are included in DaisyViz: *Data Access Lib*, *Visual Representation Lib*, *Render Lib*, and *Control Lib*, as illustrated in Fig. 2.

4.3.1. Data access lib

To support data access and to define the unified underlying data structures, DaisyViz has two sub-libraries in *Data Access Lib*: *Access Lib* and *Data Lib*. *Access Lib* provides components (*DataTXTReader*, *DataXMLReader*, and *DataSQLReader*) to access and format data from text files, XML documents, and relational database; a component (*DataXMLWriter)* to construct XML profiles; and a component (*DataTypeAdapter)* to transform UIMI data types, i.e., nominal, ordinal and quantity, to the variable types in program language (i.e., string, float, etc.). *Data Lib* defines unified data structures for tree, graph, multi-dimensional, and temporal data by providing such basic components as *InfoFacet*, *InfoFacetRelation*, *DataItem*, *DataNodeRelation*, and *AttributeColumn*.

4.3.2. Visual representation lib

To aid visual mappings and to provide generic data structures for visual representations, *Visual Representation Lib* includes two sub-libraries: *MappingLib* and *RepresentationLib*. The *MappingLib* has five components. *VisualStructureLayout* hosts commonly seen layout algorithms (e.g., TreeMaps [20], DOITrees [24], CirclePacking [26], Force-Directed Graph [28], Radial Graph [29], scatterplots [30], Parallel coordinates [31], 3D bar charts) as well as our new designs, such as DOI-Wave (see Section 5). *AxisMap* generates spatial substrate and encapsulates the mapping functions for different axis types. *GraphicalMarkAttributeContinuousMap* and *GraphicalMarkAttributeDiscreteMap* are used for assigning color, shape, size, and other attributes to graphical marks based on a collection of mapping functions. *RepresentationLib* organizes graphical elements to the generic data structures by providing components for defining views, visual structure, spatial substrate, graphical marks, etc.

4.3.3. Render lib

This library is to simplify the drawing of all kinds of graphical elements displayed in interfaces. *View ContainerRender* and *ViewContainerRelationRender* are used to draw the views and the links between related views. *VisualStructureRender*, *AxisRender*, *NodeRender*,

EdgeRender, and *LabelRender* mainly serve for the visualization in a view where axes, labels and graphical marks are drawn. *DynamicQuderyRender* is used for dynamic query sliders drawing. *LegendRender* assists the visualization of legends. *GUIRender* includes some traditional GUI components, such as text box, and dropdown box. The design of *Reder Lib* is based on the Factories approach [39], which can be extended and modified by users.

4.3.4. Control lib

To simplify the customization of the interaction tasks of *infovis*, *Control Lib* provides components of those interaction events that are frequently used in *infovis*, and encapsulates commonly seen *infovis* tasks into components. *InteractionEventListener* gives a list of events for users to choose, such as *OnItemLeftClick(),* an event qfor left-clicking a graphical item. Other components like *OverviewControl*, *PanControl*, *ZoomControl*, *DynamicQueryControl*, *LegendFilterControl*, *KeywordSearchControl*, *DetailTooltipControl*, *DistortionControl*, *Brushing-and-LinkingCoordinationControl,* and *DrillDownCoordinationControl* can each support a specific *infovis* task. *InteractionContextControl* is used to manage interaction states, such as currently selected items and search results, which offers input parameters for the encapsulated components for task control.

5. DOI-wave: a novel visualization technique in DaisyViz

Visual representations in DaisyVis can be extended. In addition to commonly seen visualization layout algorithms, such as TreeMaps [20], DOITrees [24], CirclePacking [26], Force-Directed Graph [28], and Radial Graph [29], our current version of DaisyViz also includes a new design for network data visualization, DOI-Wave. In this section, we introduce this new design and use it as an example about how new layout algorithms can be integrated into DaisyViz.

5.1. DOI-wave overview

DOI-Wave is a novel focus+context visualization technique for interactive exploration of networks and data. This technique dynamically adjusts the layout of the network based on a user's estimated degree-of-interest (DOI). In graph visualization, it is important to allow network analysts to explore a node's degree of connectivity and network distance from other nodes [29]. Several approaches have been studied. One group of visualization techniques use the paradigm of clustered graph navigation to support network exploration [27,28,42]. However, this paradigm has its limitations in practice because it assumes that the graph of interest has a hierarchical structure, which is suitable for clustering. Another group of designs transform a graph into a tree (e.g., spanning tree) by using a fast algorithm and support user navigation through this resulting tree rather than the original graph [43,44]. Because of the transformation of data from a graph to a tree, some links among nodes are hidden and the change of the graph structure cannot be faithfully represented.

A third approach applies the focus+context paradigm [4] and visualizes a graph. Such focus+context techniques consider a user's attention in network analysis and navigation, and dynamically allocate display resources by calculating the user's degree-of-interest (DOI) [40]. In general, designs falling into this category can be further divided into two classes: designs with geometric distortion (e.g., graph fisheye [45]) and designs without distortion (e.g., Radial Graph [29]). Designs with geometric distortion are difficult to use. In particular, comprehending the hierarchical relationships of a complex graph in a distorted view is a challenge because of the cluttered nodes and edges, as a result of the distortion. Techniques without distortion, on the other hand, cannot deal with large graphs well, because algorithms in this class often follow a design model that put nodes in centric rings. For large graphs, this layout is not efficient and is difficult to explore nodes in every directions, all 360°.

Our DOI-wave design, based on attention-reactive user interface [40], follows a hierarchically clustered graph paradigm, in which the view of a graph is dynamically determined by the selection of a focused node to which the user pays the most attention. The layout algorithm changes the view of a graph dynamically and animatedly when the focused node is shifted and the DOI values of each node in the graph are changed. By hierarchically clustering nodes, our DOI-Wave design can dynamically adjust the number of nodes in the visualization and allow the display of the graph to be easily adapted to available display space.

Fig. 4 is an example of using our DOI-wave approach to visualize a social network. A user can easily explore the social network by shifting the focus from one person to another in the social network. Fig. 4a shows a view in which the user selected a person called "Otomi" as the focus to examine his social relationships. When the user discovered that one of his friends, "Ben", had many friends and deserved more attention. The user then selected "Ben" as the new focus and got a new shown in Fig. 6b.

In comparison, Fig. 5 shows the views of the same social network by an ASK-Graphview [42] (Fig. 5a), which uses force-directed clustered techniques, and Radial Graph (Fig. 5b). As shown in Fig. 5a, the relationships between two actors within a cluster can be understood easily, especially when the focused actor is the center node of a cluster, such as "Ben". However, for those actors who do not apparently belong to any clusters, such as "Otomi", it is difficult to comprehend their relationships with and network distances to others. In the view of Fig. 5b, exploring the network has to be conducted in all direction around the rings, especially when actors of interest are far away from each other in network distance, such as "Otomi" who is at the center and "Sam" who is in the outer ring.

The DOI-wave design offer some advantages over other methods. The layout of the DOI-wave differs from Radial Graph in that users can understand the degree of

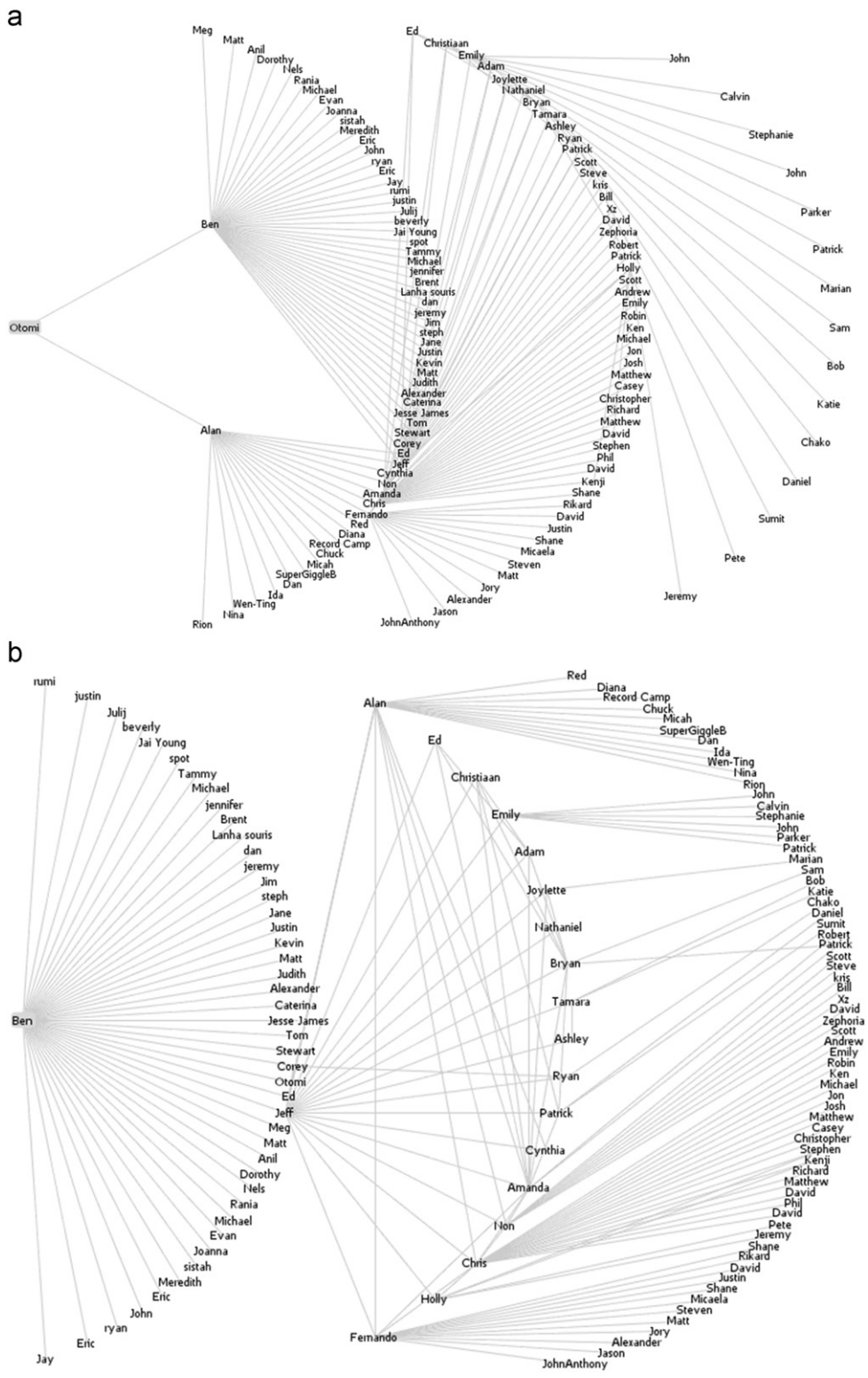

Fig. 4. DOI-wave tool for social network exploration: (a) DOI-wave Layout with actor "Otomi" as focus and (b) DOI-wave layout as focus shifted to actor "Ben".

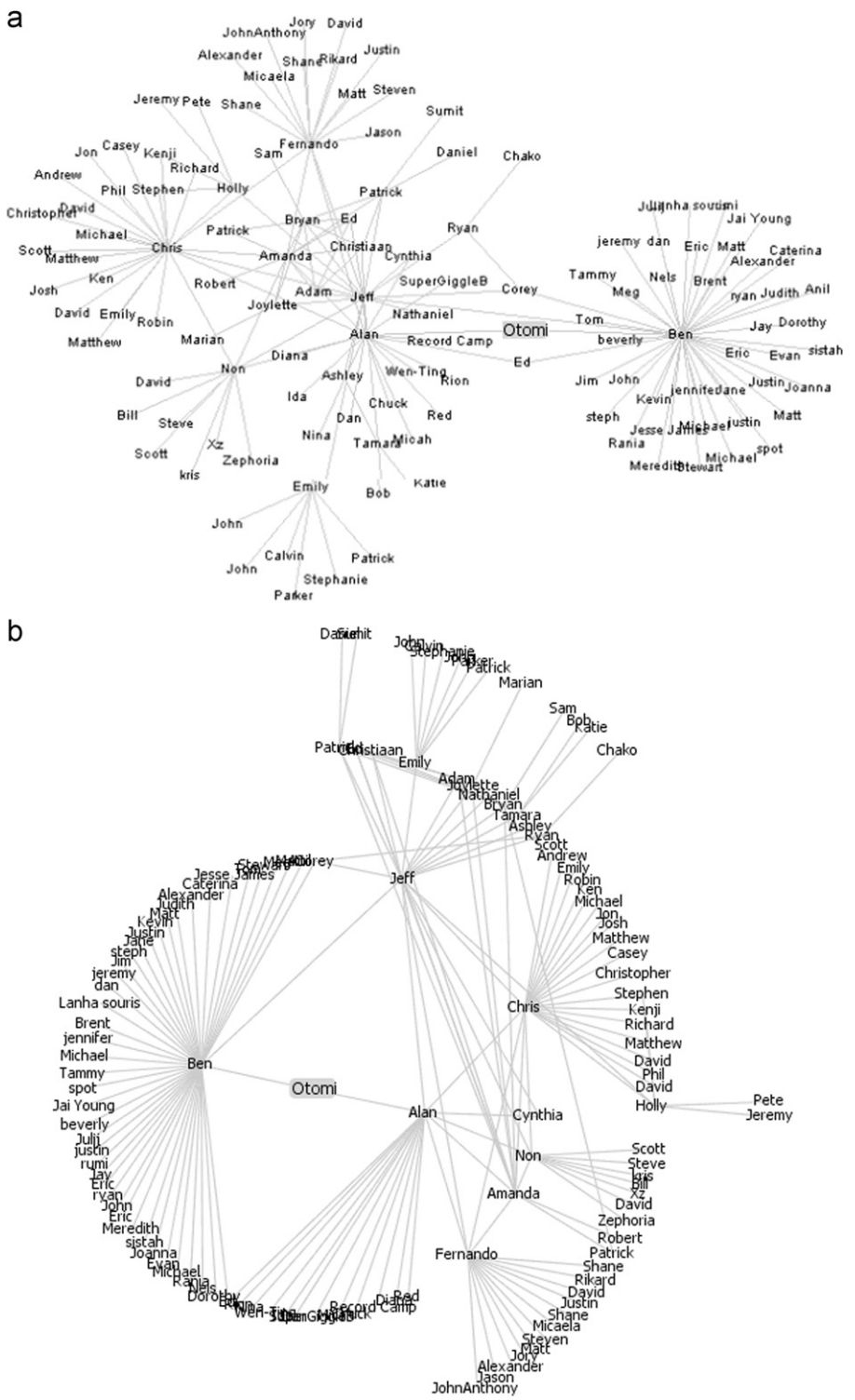

Fig. 5. ASK-GraphView and Radial Graph for social network exploration: (a) ASK-Graphview and (b) Radial Graph.

connectivity of the focused node and its network distance to other nodes in a single direction in the DOI-Wave, rather than exploring nodes in every directions, all 360°. In addition, the adaptive visualization mechanism in the DOI-wave can achieve more effective display space utilization because the layout takes advantage of the rectangular space rather than the circular space used by Radial Graph.

5.2. DOI model

Here we will give the definitions of networks that can be used in the DOI-wave algorithm.

Definition 19. A ***DWNode*** is a 7-column table $<$*NodeID*, *DOI*, *CenterPoint*, *NodeWidth*, *NodeHeight*, *Level*, *NodeList*, *Text*, *Flag*$>$. *NodeID* is the ID of a node, *DOI* is the value of degree-of-interest. *CenterPoint* is the centerpoint of the bounding box of the node. *NodeWidth* and *NodeHeight* are the width and the height of the bounding box. *Level* is the layer level in the clustered hierarchy the node belongs to. *NodeList* is the set of neighbor nodes' IDs which are linked to the node. *Text* is the label. *Flag* is a Boolean flag with the initial value false.

Definition 20. An ***DWEdge*** is a 4-column table $<$*EdgeID*, *StartNode*, *EndNode*, *Text*$>$, where *EdgeID* is the ID of an edge, *StartNode* and *EndNode* are the *NodeID* of two endpoint nodes of the edge, and *Text* is the label.

Definition 21. A ***DWNetwork*** is a 2-column table $<$*DWNodeSet*, *DWEdgeSet*$>$, where *DWNodeSet* is a set of nodes, and *DWEdgeSet* is a set of edges.

Let N be a *DWNetwork* consisting of n nodes. Let Nd_f be the focus node users have selected. Let Nd_i be any node in N, $Nd_f \neq Nd_i$. Let Tr be a spanning tree of N with Nd_f as the root node. Let Nd_p be the parent node of Nd_i in Tr. A simple DOI function is defined as follows:

$$\mathrm{DOI}(\mathrm{Nd}_i) = \mathrm{DOI}(\mathrm{Nd}) - C \tag{1}$$

In the DOI function, C is a constant. Then, the DOI computing algorithm can be described as follows:

Step 1: Initiate DOI value of the focus node, $\mathrm{DOI}(\mathrm{Nd}_f) = n^*C$. Let the Flag of Nd_p be true, Nd_p.Flag=true.

Step 2: Traverse N with breadth first order and get the node set NodeList of Nd_f.

Step 3: For any $\mathrm{Nd}_j \in$ NodeList, if the Flag of Nd_j is false, $\mathrm{DOI}(\mathrm{Nd}_j) = \mathrm{DOI}(\mathrm{Nd}_f) - C$. Then let the Flag of Nd_j be true, Nd_j.Flag=true.

Step 4: For any $\mathrm{Nd}_j \in$ NodeList, let Nd_j be the new Nd_f. Then go to step 2.

5.3. DOI-wave layout

Fig. 6 shows the layout method when a node is selected as the focus of attention. Let the left-top point of the rectangle display area be the origin of a two-dimensional coordinate system, the horizontal axis be the X-axis, and vertical axis be the Y-axis. Let w be the width of the rectangle in the X-axis, and h, the height in the Y-axis. For Nd_f and N, the DOI-wave layout algorithm is described as follows:

Step 1: For each node $\mathrm{Nd}_i \in$ NodeSet, compute $\mathrm{DOI}(\mathrm{Nd}_i)$ by the DOI computing algorithm.

Step 2: For each node $\mathrm{Nd}_i \neq \mathrm{Nd}_f$, sort all the nodes in descending order according to $\mathrm{DOI}(\mathrm{Nd}_i)$. Then put them into sets SL_0, SL_{01}, SL_2,…,SL_m, where for $\mathrm{Nd}_j \in \mathrm{SL}_i$, $\mathrm{DOI}(\mathrm{Nd}_i) = (n-i-1)^*a$.

Step 3: Put Nd_f at the point (a+NodeWidth (Nd_f), $h/2$).

Step 4: For the nodes in each set SL_k (k=0 to m), place them evenly in the arc L_k. The equation of L_k is $(x-s-k^*d)^2+(y-h/2)^2=R^2$, where $t \leq y \leq h-2t$. If k=0, draw each edge between Nd_f and Nd_i where $\mathrm{Nd}_i \in L_k$. If $k \neq 0$, draw each edge between Nd_i and Nd_j, where $\mathrm{Nd}_i \in L_{k-1}$ and $\mathrm{Nd}_j \in L_k$.

Since the display region is often like a rectangle, the focus node is moved to the middle on one side, and the other nodes are dynamically re-arranged on a serial of arcs. Each node lies on an arc according to its DOI value. Immediate neighbors of the focus, on the nearest arc, have the maximum DOI values, and their neighbors with the minor DOI values lie on the second nearest arc, etc. The

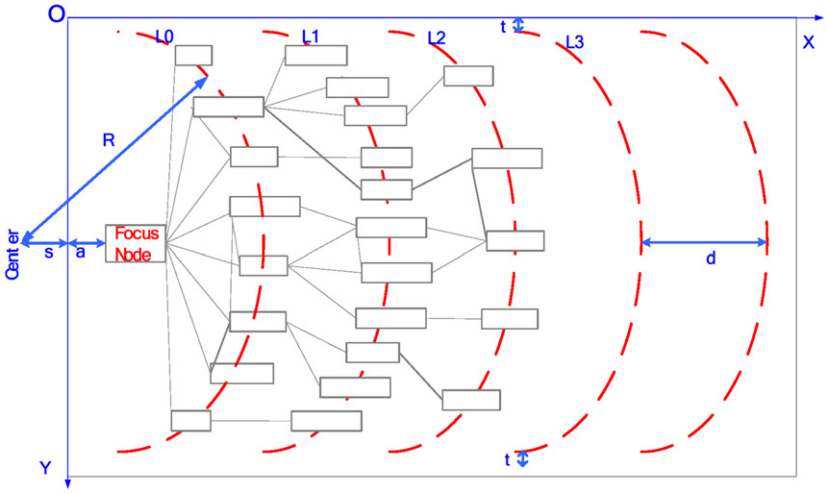

Fig. 6. DOI-wave layout.

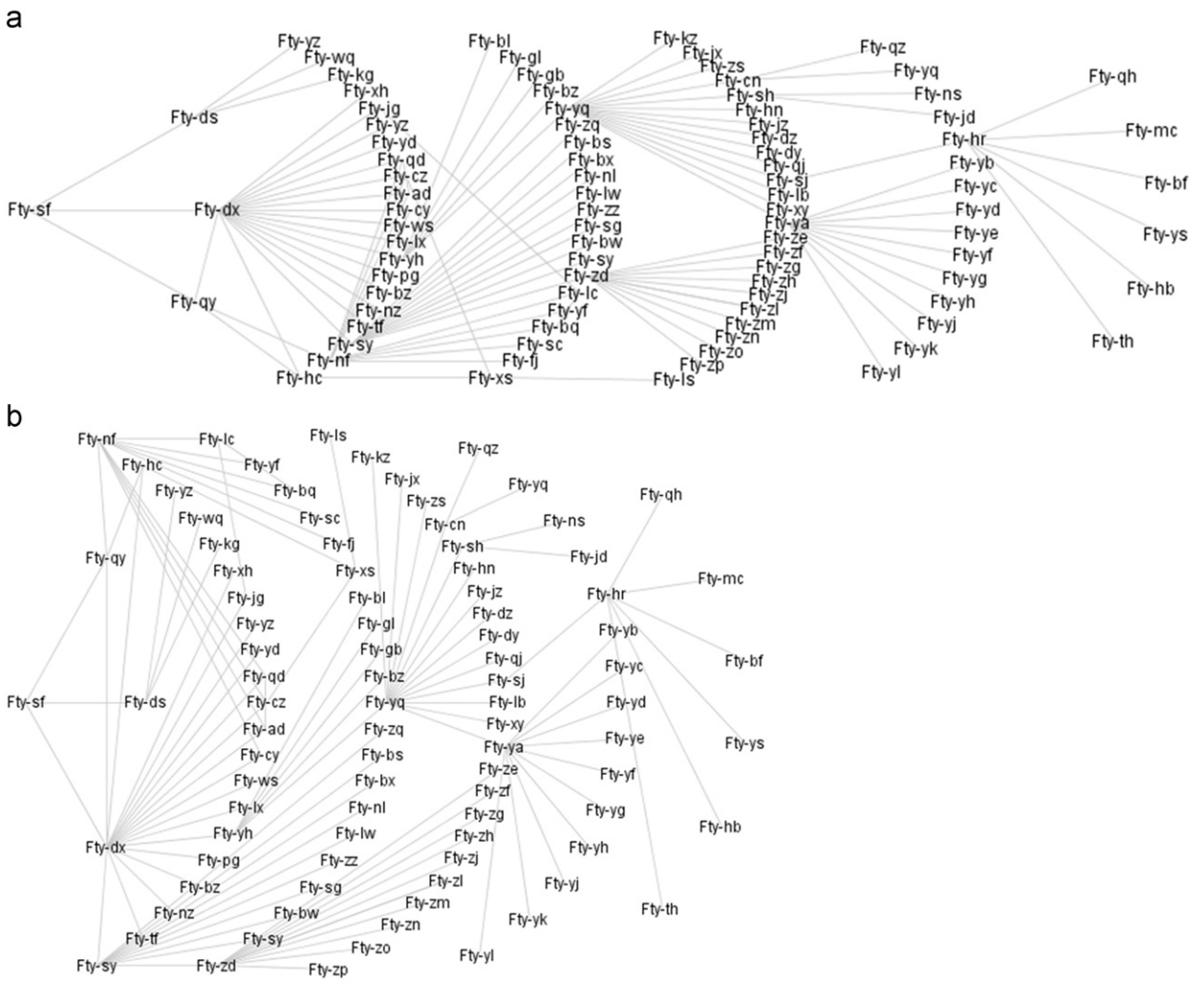

Fig. 7. Dynamically adaptive layouts to DOI-wave.

layout can be adaptive dynamically to the display space, i.e., the radius R (see Fig. 6) of the arc varies with the height of the rectangle and the distance d between two adjacent arcs varies with the width, as shown in Fig. 7.

5.4. Integration of DOI-wave into DaisyViz and its use

New layout algorithms can be added into the DaiszVis toolkit. For example, to include the DOI-wave design in DaisyViz, we can integrate the above definitions and layout algorithms into the *VisualStructureLayout* sub-library of *Mapping Lib.*

Table 1 shows part of the UIMI profile of designing a DOI-wave view for a network. The profile includes the definitions of a data facet—*Factory*, and a view container that has a DOI-Wave tool to visualize the data facet and two indirect manipulation tools—*Dynamic query* and *Keyword Search*. Fig. 8 is the system generated based on this UIMI profile.

6. Application example

In this section, we present how DiasyViz works with an example of building an interactive *infovis* application in a manufacturer.

Table 2 shows the requirements of the application. The target information for analysis is "Manufacture". "Manufacture" is an abstract conception, so it is divided into four related concrete facets for analysis: *Manufactory*, *Product Order*, *Manufacturing Plan*, and *Plan Execution*, as shown in the column "InfoFacet" in Table 2. Each facet has a set of data attributes, which are listed in the column "Data Attributes". The relationships among the data instances within each facet are specified by the column "Data Type". The contents in the "Layout" column are the view layout for each facet, and possible interaction tasks associated with the view are listed in the column

Table 1
UIMI profile of a design to use DOI-wave for a network.

```
< Data >                                                                    // data definition
  < IFSet >                                                                 // information facets of interest
    < IF                                                                    // first information facet
      IFID="Factory"                                                        // name of the facet
      DISet="FactoryData"                                                   // source for data
      DIRSet="FactoryRelationData" >                                        // source for data relationship
       < DataSource >                                                       // parameters about data source
         < ConnetionString
           Driver="SQLServerDriver"
           Data Source="DBServer"
           Initial Catalog="ManufactoryDB" / >
         < FactoryData SQL="select * from factory" / >
           < FactoryRelationData SQL="select select * from factoryRelations" / >
       < /DataSource >
    < /IF >
  < /IFRSet >
< /Data >

< Visualization >                                                           // visualization definition
  < VS
    VSID="VS_Factory"                                                       // id of the layout
    LayoutAlgorithm="DOI-Wave" >                                            // DOI-Wave layout
  < /VS >
  < VCSet >
    < VC                                                                    // a view container
      VCID="VC_Factory"
      IFID="Factory"                                                        // data for the view
      VSID="VS_Factory"/ >                                                  // view style
    < /VCSet >
< /Visualization >

< Control >                                                                 // control definition
  < InDirectManipulationTaskControlSet >
    < DQC                                                                   // dynamic query control
      DQCID="DQC_OverallProduct"
      VCID="VC_Factory"                                                     // in which view container
      DIAttri="OverallProduct"
      ValueRange="default"/ >
    < KSC                                                                   // keyword search control
      KSCID="KSC_Manager"
      VCID="VC_Factory"                                                     // in which view container
      DIAttri="Manager"/ >
    < /InDirectManipulationTaskControlSet >
< /Control >
```

"Interaction Task". Between-facet relationships are defined by attributes shared by related facets. For example, the *Manufactory* and *Product Order* facets share a data attribute, ManufactoryID, which defines the relationship between factories and product orders—each manufactory has certain orders and each order belongs to a manufactory.

This table indicates the key demands that the *infoVis* application should meet, including the following:

- providing support for different data types, such as tree, graph, and multi-dimensional data, as shown in the column "Data Type";
- providing support for specific layout algorithms, such as Radial Graph, Force-Directed Graph, DOITrees, TreeMaps, and Scatter Plot, as shown in the column "Layout"; and
- providing support for various interaction techniques, such as dynamic query, overview+detail, panning+zooming, focus+context, and multiple coordinated views, etc., as shown in the column "Interaction Task".

Based on the requirements in Table 2, we can use the DaisyViz Modeler to create the UIMI profile of the application. As shown in Fig. 9, the DaisyViz Modeler provides three panels—the data modeler (Fig. 9a), the visualization modeler (Fig. 9b), and the control modeler (Fig. 9c). Here, the datasets are retrieved from a relational database, so parameters required to establish a connection to the database are specified and database tables that are needed are also chosen in the data modeler (Fig. 9a). Also, those key attributes that connect individual facets can be explicitly identified in this panel; or the common attributes among the facets, i.e., the foreign keys in chosen tables, are selected by default to coordinate data instances between different facets.

The visualization modeler allows a user to configure layout algorithm, spatial substrate and the attributes of

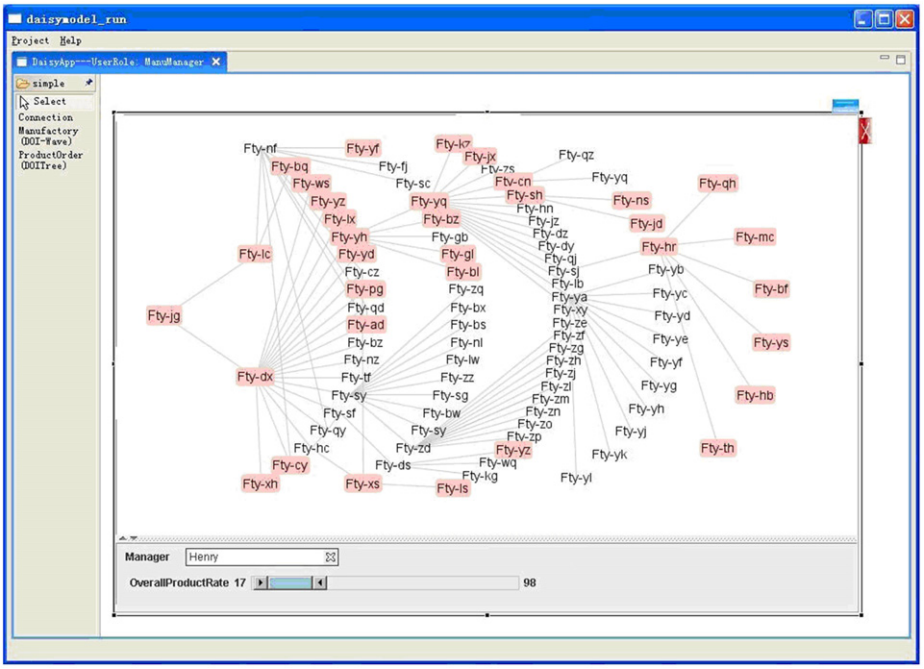

Fig. 8. An Application of DOI-wave created with DaisyViz.

Table 2
Design requirements for manufacture in an enterprise *Infovis* application.

InfoFacet	Data attributes	Data type	Layout	Interaction task
Manufactory	ManufactoryID, ManufactoryName, OverallProductRate, Address, Manager, AlertStatus	Graph	Radial Graph, Force-Directed Graph	Overview, Pan, Zoom, DynamicQuery, KeywordSearch, Detail, Brushing-and-linking, Coordination
Product order	ProductOrderID, Customer, ProductID, OrderNumber, Priority, ManufactureID	Tree	DOITrees	Overview, Pan, Zoom, DynamicQuery, FilterByLegends, KeywordSearch, Detail, FocusPlusContext, Brushing-and-linking, Coordination
Manu plan	PlanID, ProductOrderID, PlanNumber, FinishDate, ProductGroup	Tree	TreeMaps	Overview, Zoom, KeywordSearch, Detail, Brushing-and-linking Coordination
Plan execution	ExecID, PlanID, ProductID, Date, FinishNumber, IsExpire, Quality	Multi-dimensional	Scatter Plot	Overview, Pan, Zoom, DynamicQuery, FilterBy Legends, KeywordSearch, Detail, Drill-down, Coordination

graphical marks for a given facet. Fig. 9b shows an example of using the modeler to construct the view of the facet of "Plan Execution". In the figure, a scatterPlot is used for the facet, with its two axes mapped to two data attributes "Date" and "ProductID". The label for each axis is also specified in the filed of "Axis Mapping Attribute". The data attributes "Quality" and "IsDelay" are chosen to be mapped to the shape and color attributes of graphical marks. This modeler also allows the user to define which views would be coordinated. If such view coordination is not specified, the views for those related facets are chosen by default.

Fig. 9c shows how the control modeler works in customizing interaction tasks. Here, the target is the graph view for the facet "Manufactory". "OnItemRight-Click" is selected as the *DirectManipulationEvent* for the task *Brushing-and-LinkingCoordination*, which indicates the brushing-and-linking coordination can be initiated by a right-click event on nodes in the view graph. Data attributes "Manager" and "OverallProductRate" are chosen for the *KeywordSearchControl* and the *Dynamic-QueryControl*.

The results of data modeling, visualization modeling, and controlling are ported into an XML-based UIMI profile. Fig. 10 shows part of the UIMI profile. This profile can be edited with any XML editor.

Based on the UIMI profile, the DasiyViz Runner generates a prototype system. As shown in Fig. 11a, a list of facets are provided on the left panel. When a facet is dragged and dropped into the blank area of the right panel, the view corresponding to the facet can be created. Each view contains two parts: a visualization area and a control panel. The control panel contains necessary tools, such as keyword search boxes, legends dropdown boxes,

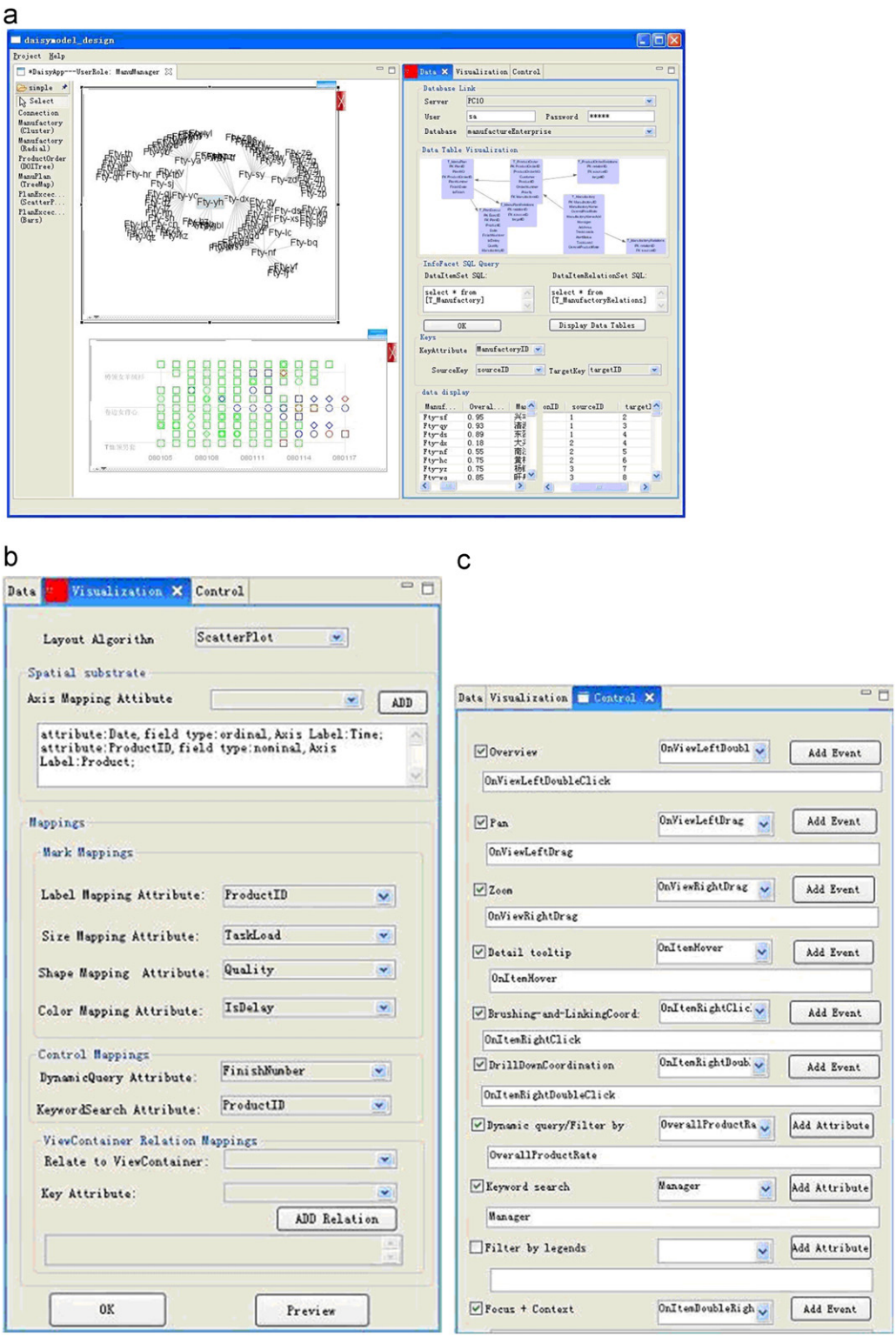

Fig. 9. Modeling UIMI with DaisyViz modeler: (a) Data model to define the facet of "Manufactory", (b) Visualization model for the facet of "PlanExecution", and (c) Control model for the facet of "Manufactory".

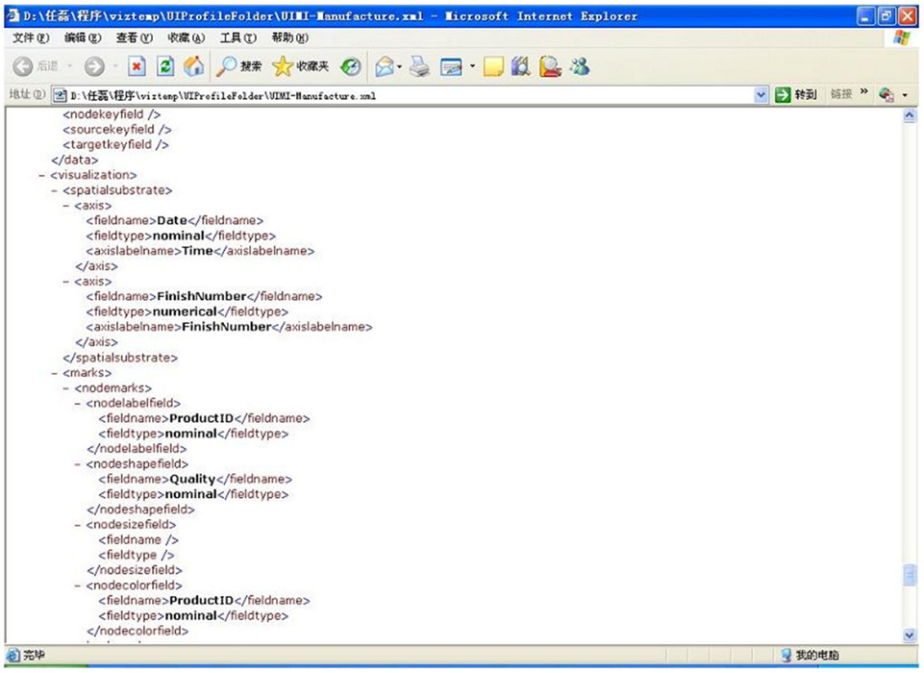

Fig. 10. Part of an XML UIMI Profile Generated by DaisyViz Modeler.

and dynamic query sliders. Coordinated views are connected by red lines. Fig. 11b shows the results of coordination in these coordinated views when a node in the Force-Directed Graph view is right-clicked: a mouse-clicking event triggers the brushing-and-linking coordination between DOITrees and TreeMaps, and also initiates the drill-down coordination in a Scatter Plot. Fig. 11 also illustrates some other interaction techniques, such as the detail tooltip and keyword search in the graph view, dynamic query and filter by legends in the Scatter Plot.

The DasisyViz is flexible; users can customize visualization tools by choosing different layout algorithms for the same dataset. Fig. 12 shows a prototype system built on the same dataset as that used to create Fig. 11, but modeled with different views. Here, the Force-Directed Graph in Fig. 10 is replaced with a DOI-wave view, the DOITree is switched to a nested fisheye view, and the Scatter Plot becomes a 3D bar chart.

As this application example shows, the DaisyViz toolkit offers the following features:

- Providing a unified data structure for tree, graph, and multi-dimensional data. (Since temporal, 1D, 2D, and 3D data are special cases of multi-dimensional data, DaisyViz can also cover them.)
- Integrating a variety of layout algorithms, including Radial Graph, Force-Directed Graph, DOI-Wave, DOITrees, TreeMaps, Scatter Plot, CirclePacking Fish-eyes, and Bar chart. New layout algorithms can also be added.
- Supporting a wide range of *infovis* tasks, including overview, pan, zoom, filter by data attributes/dynamic query, filter by legends, keyword search, detail tooltip/detail-on-demand, and coordination (e.g., brushing-and-linking, drill-down, etc.).

7. Qualitative usability study

To further understand the expressiveness, effectiveness, flexibility, and extensibility of DaisyViz, we conducted a usability study on the use of DaisyViz to create the manufacturing analysis application mentioned above by 10 non-expert users.

Ten subjects had different jobs: four were computer programmers, three were user interface designers, two were data analysts, and one was a researcher on *infovis*. All subjects, except the *infovis* researcher, lacked the experience in building *infovis* systems. They all used *infovis* applications in their daily work before the test.

The subjects were first given a tutorial about the DaisyViz for about 45 min, including the concepts of UIMI, the XML schema of UIMI profiles, and some samples developed with the DaisyViz toolkit. Then, the subjects were asked to perform three tasks with the DaisyViz toolkit by using an SQL Server database that stored the manufacturing datasets. The first task was to create an application with three views, covering graph, tree, and multi-dimensional data. The subjects were asked to refine their views by using color, size, or shape to represent information about one or more data attributes. The second task was to add interaction controls to implement the functions including overview, pan, zoom, dynamic query, filter by legends, keyword search, and detail tooltip. Finally, the subjects were asked to add two coordination techniques among the three

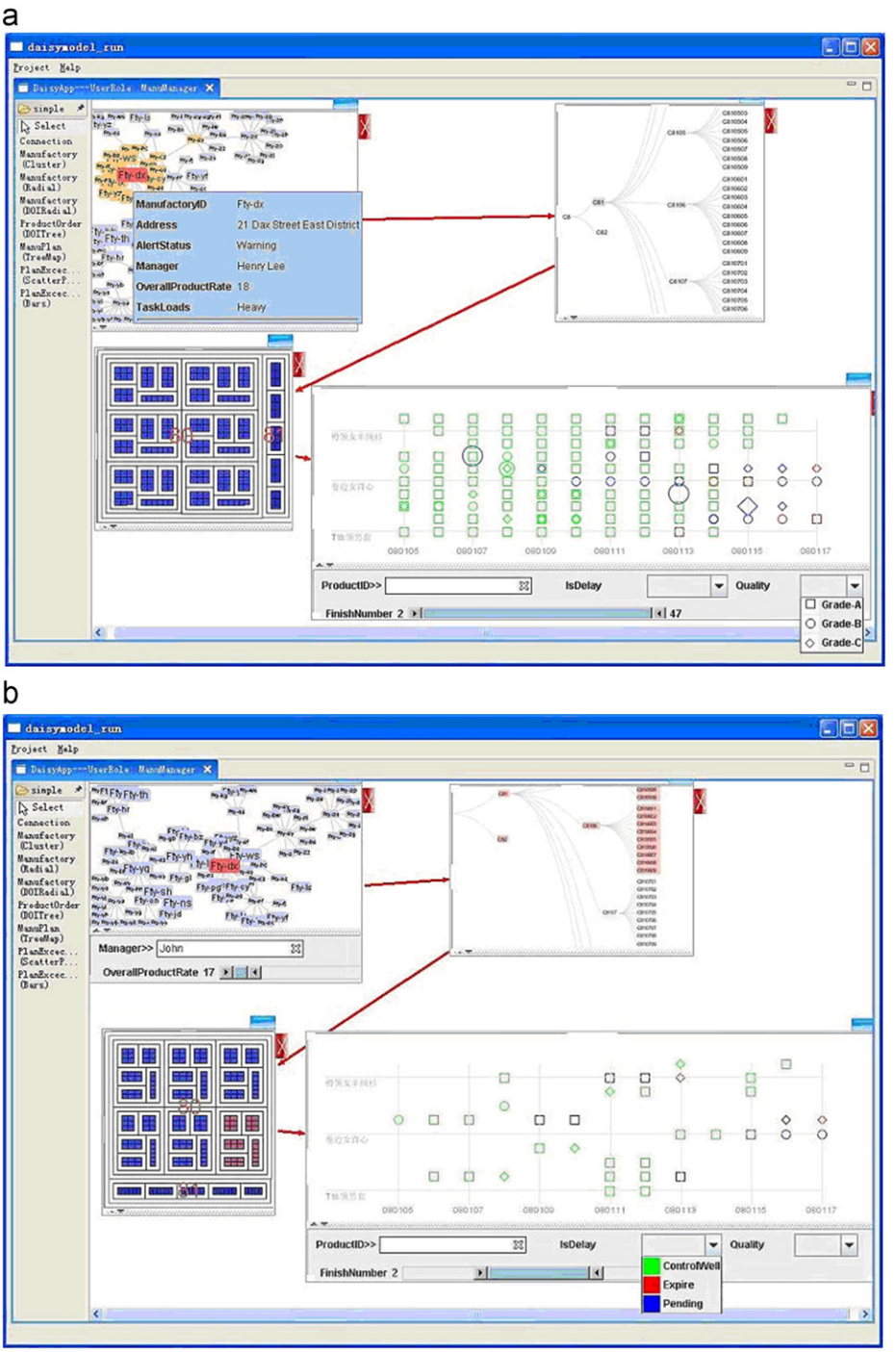

Fig. 11. (a) A prototype system generated by DaisyViz runner with four related views connected with red links: Force-Directed Graph, DOITrees, TreeMaps, and Scatter Plot. (b) Multiple views coordination triggered by a right-clicking the Force-Directed Graph: brushing-and-linking coordination between DOITrees and TreeMaps, and Drill-Down Coordination in Scatter Plot. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

views, including brushing-and-linking coordination and drill-down coordination respectively.

The tasks were carried out in a client-server environment and each subject was on a Windows PC client (2.6 GHz Pentium 4 with 2 GB RAM). The time limit was 150 min and the subject performance was recorded for further analysis. Each subject was interviewed about their experience during the three tasks.

Our results showed that all subjects developed executable applications with the DaisyViz. Eight of the ten subjects completed all tasks, and four subjects finished in 100 min.

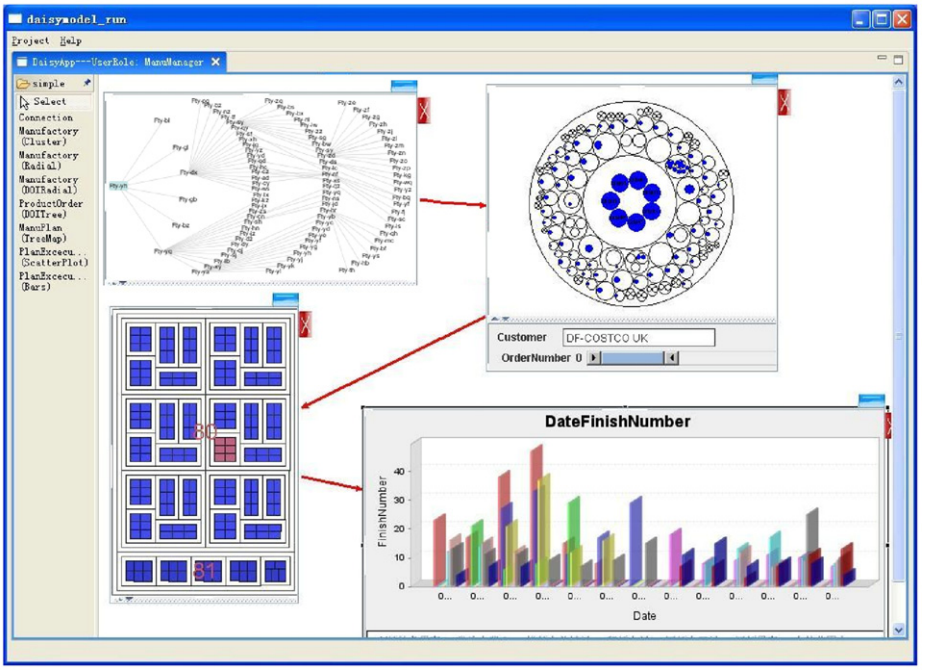

Fig. 12. An application with different views based on same dataset in Fig. 11.

The most common difficulty the subjects encountered was that they had to switch frequently between the data model configuration panel and the SQL Server database to write SQL queries, because they thought the overview of databases was not presented to them. Another interesting observation was that some subjects preferred to directly edit the XML-based UIMI profiles. They stated that editing the XML profile seemed faster once becoming familiar with the UIMI. However, the XML tags used in UIMI sometimes slowed them down. For example, *Brushing-and-LinkingCoordination* was initially called *BLC*, which was hard to interpret by subjects. We also found that eight subjects made mistakes while configuring interaction events because of the conflicting events they defined. For example, three subjects specified "OnItemLeftClick" as the *DirectManipulationEvent* for two tasks, *DetailTooltip* and *DrillDownCoordination*, which led to a runtime error. They suggested that such configuration conflict be detected automatically by the toolkit. As for the interface of the DaisyViz Modeler, seven subjects suggested that the controls for the mapping attributes configuration in the panel of control model, such as the dropdown box of the mapping attributes for dynamic query and keyword search, be placed in the panel of visualization, because they thought these configurations all belonged to the mapping problems.

Responding to these issues, we improved the DaisyViz in the following ways: (1) adding a graph visualization of data schema of the database to the DaisyViz Modeler (see Fig. 8a); (2) making the XML tags in the UIMI profile clearer and more understandable; and (3) adding a conflict detection module for interaction events.

In addition, our interview data showed that most subjects appreciated the toolkit. They were surprised to see such complex applications could be created so easily and without much coding. The *infovis* expert, who had used many *infovis* toolkits, stated "It's unbelievable that the only thing I need to do is editing a simple XML." Eight of the ten subjects decided to use the DaisyViz tool in their own work. Some visualization applications created in the study have been used for data analysis by the manufacturer.

8. Conclusions and future work

This paper presented DaisyViz, a model-based user interface toolkit for the development of interactive *infovis* systems with multiple coordinated views. The toolkit relies on UIMI, a declarative interface model, as the core of the development process to model data access, visualization design, and view control. The task of end-users is to build a UIMI profile according to domain-specific requirements, and then a prototype system can be generated automatically by DaisyViz. DaisyViz can support unified data structures and include a set of diverse layout algorithms, which can be easily extended. In addition, DaisyViz can support an extended taxonomy of *infovis* tasks by providing commonly seen interaction techniques.

We showed an example of using DaisyViz by building an *infovis* system for manufacturing data analysis. We also conducted a qualitative usability study to understand how non-expert users built *infovis* systems with DaisyViz. Both the application development example and the user study have demonstrated the effectiveness and flexibility of DaisyViz in assisting the development of domain-specific *infovis* systems.

In the future, we will improve DaisyViz in the following directions. First, we will include more layout algorithms into the toolkit to produce richer information visualization tools. At the same time, we will develop APIs that allow users to

add their own layout algorithms into the toolkit more easily. Second, we will add new prebuilt expressions and functions to support more data mapping methods so that users have more choices in constructing their visualization tools based their needs. Third, we will improve the toolkit by supporting visual analytics tasks. Compared with *infovis* tasks, visual analytics are more complex and demand support for the understanding of more sophisticated relationships among different data facets, among different views, and among different user interaction activities. Furthermore, to better support visual analytics, we will provide rich tools that allow end-users to not only view the visualization results, but also edit the results. Last but not least, we will extend DaisyViz libraries to support building visualization systems for thin clients, such as web-based applications.

Acknowledgments

This research was supported by the National High-Tech Research and Development Plan of China under Grant no. 2007AA04Z153, the National Grand Fundamental Research 973 Program of China under Grant no. 2007CB310900, China postdoctoral science foundation under Grant no. 20100470185, and the National Natural Science Foundation of China under Grant no. U0735004.

References

[1] S.K. Card, J.D. Mackinlay, B. Shneiderman, Readings in Information Visualization: Using Vision to Think, Morgan Kaufmann, San Francisco, 1999.
[2] B. Shneiderman, Dynamic queries for visual information seeking, IEEE Software 11 (1994) 70–77.
[3] K. Hornbæk, B.B. Bederson, C. Plaisant, Navigation patterns and usability of zoomable user interfaces with and without an overview, ACM Transactions on Computer–Human Interaction (TOCHI) 9 (2006) 362–389.
[4] G.W. Furnas, A fisheye follow-up: further reflections on focus+-context, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, New York, 2006, pp. 999–1008.
[5] M.Q. Baldonado, A. Woodruff, A. Kuchinsky, Guidelines for using multiple views in information visualization, in: Proceedings of the Working Conference on Advanced Visual Interfaces (AVI'00), ACM Press, New York, 2000, pp. 111–119.
[6] B. Shneiderman, The eyes have it: a task by data type taxonomy for information visualizations, in: Proceedings of the IEEE Workshop Visual Languages, IEEE Computer Science Press, Los Alamitos, 1996, pp. 336–343.
[7] J. Heer, S.K. Card, J.A. Landy, Prefuse: a toolkit for interactive information visualization, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'05), ACM Press, New York, 2005, pp. 421–430.
[8] B.B. Bederson, J.D. Hollan, K. Perlin, J. Meyer, D. Bacon, G.W. Furnas, Pad++: a zoomable graphical sketchpad for exploring alternate interface physics, Journal of Visual Language and Computing 7 (1996) 7–31.
[9] B.B. Bederson, J. Meyer, L. Good, Jazz: an extensible zoomable user interface graphics toolkit in Java, in: Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'00), ACM Press, New York, 2000, pp. 171–180.
[10] B.B. Bederson, J. Grosjean, J. Meyer, Toolkit design for interactive structured graphics, IEEE Transactions on Software Engineering 30 (2004) 535–546.
[11] C. North, B. Shneiderman, Snap-together visualization: a user interface for coordinating visualizations via relational schemata, in: Proceedings of the Working Conference on Advanced Visual Interfaces (AVI'00), ACM Press, New York, 2000, pp. 128–135.
[12] M. Takatsuka, M. Gahegan, GeoVISTA studio: a codeless visual programming environment for geoscientific data analysis and visualization, Computers and Geosciences 28 (2002) 1131–1144.
[13] P.I. Godinho, B.S. Meiguins, A.S. Meiguins, R.M. Carmo, M.B. Carcia, L.H. Almeida, R. Lourenco, PRISMA—a multidimensional information visualization tool using multiple coordinated views, in: Proceedings of the 11th International Conference on Information Visualization (IV'07), IEEE Computer Science Press, Washington, 2007 pp. 23–32.
[14] C. Stolte, D. Tang, P. Hanrahan, Polaris: a system for query, analysis and visualization of multi-dimensional relational databases, IEEE Transactions on Visualization and Computer Graphics 8 (2002) 1–14.
[15] K. Borner, Y. Zhou, A software repository for education and research in information visualization, in: Proceedings of the Fifth International Conference on Information Visualisation (IV'01), IEEE Computer Society Press, Los Alamitos, 2001, pp. 257–262.
[16] J.D. Fekete, The InfoVis Toolkit, in: Proceedings of the IEEE Symposium on Information Visualization (InfoVis'04), IEEE Computer Society Press, Washington, 2004, pp. 167–174.
[17] F.B. Viegas, M. Wattenberg, F. van Ham, J. Kriss, M. McKeon, Many eyes: a site for visualization at internet scale, IEEE Transactions and Computer Graphics 13 (2007) 1121–1128.
[18] K. Matkovic, W. Freiler, D. Gracanin, H. Hauser, ComVis: a coordinated multiple views system for prototyping new visualization technology, in: Proceedings of the 12th International Information Visualization, IEEE Computer Society Press, Washington, 2008, pp. 215–220.
[19] F. Paterno, Model-based Design and Evaluation of Interactive Applications., Springer, Heidelberg, 2000.
[20] B.B. Bederson, B. Shneiderman, Ordered and quantum treemaps: making effective use of 2D space to display hierarchies, ACM Transactions on Graphics 21 (2002) 833–854.
[21] G.G. Robertson, J.D. Mackinlay, S.K. Card, Cone trees: animated 3D visualizations of hierarchical information, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, New York, 1991, pp. 189–194.
[22] J. Lamping, R. Rao, The hyperbolic browser: a focus + context technique for visualizing large hierarchies, Journal of Visual Languages and Computing 7 (1996) 33–55.
[23] C. Plaisant, J. Grosjean, B. Bederson, Spacetree: supporting exploration in large node link tree, design evolution and empirical evaluation, in: Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02), IEEE Press, Boston, 2002, pp. 57–64.
[24] J. Heer, S.K. Card, DOITrees revisited: scalable, space-constrained visualization of hierarchical data, in: Proceedings of the Working Conference on Advanced Visual Interfaces (AVI'04), ACM Press, New York, 2004, pp. 421–424.
[25] J. Yang, M.O. Ward, E.A. Rundensteiner, InterRing: an interactive tool for visually navigating and manipulating hierarchical structures, in: Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02), Boston, IEEE Press, Boston, 2002, pp. 77 – 84.
[26] W.X. Wang, H. Wang, G.Z. Dai, H. Wang, Visualization of large hierarchical data by circle packing, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06), ACM Press, New York, 2006, pp. 517–520.
[27] P. Eades, Q.W. Feng, Multilevel visualization of clustered graphs, in: Proceedings of the Fourth International Symposium on Graph Drawing, Springer, Heidelberg, 1996, pp. 101–112.
[28] P. Eades, M.L. Huang, Navigating clustered graphs using force-directed methods, Journal of Graph Algorithms and Applications 4 (2000) 157–181.
[29] K.P. Yee, D. Fisher, R. Dhamija, M.S. Hearst, Animated exploration of dynamic graphs with radial layout, in: Proceedings of the IEEE Symposium on Information (InfoVis'01), IEEE Computer Science Press, Los Alamitos, 2001, pp. 43–50.
[30] R.A. Becker, W.S. Cleveland, Brushing scatterplots, Technometrics 29 (1987) 127–142.
[31] A. Inselberg, B. Dimsdale, Parallel coordinates: a tool for visualizing multi-dimensional geometry, in: First Conference on Visualization, IEEE Press, Washington, 1990, pp. 23–26.
[32] S. Havre, B. Hetzler, L. Nowell, ThemeRiver: visualizing theme changes over time, in: IEEE Symposium on Information Visualization (InfoVis'00), IEEE Press, Washington, 2000, pp. 115–123.
[33] D.A. Keim, J. Schneidewind, M. Sips, CircleView—a new approach for visualizing time-related multidimensional data sets, in: Proceedings of the Working Conference on Advanced Visual Interfaces (AVI'04), ACM Press, New York, 2004, pp. 179–182.

[34] C. Tominski, J. Abello, H. Schumann, Axes-based visualizations with radial layouts, in: ACM Symposium on Applied Computing, ACM Press, New York, 2004, pp. 1242–1247.
[35] Graphviz, 〈http://www.research.att.com/sw/tools/graphviz〉.
[36] M.S. Marshall, I. Herman, G. Melancon, An object-oriented design for graph visualization. Software, Practice and Experience 31 (2001) 739–756.
[37] E. Thomsen, OLAP Solutions: Building Multidimensional Information Systems, Wiley Computer Publishing, New York, 1997.
[38] W. Krentzer, B. Mckenzie, Programming for Artificial Intelligence, Method, Tools and Applications, Addison-Wesley, New Jersey, 1991.
[39] H. Jeffrey, A. Maneesh, Software design patterns for information visualization, IEEE Transactions on Visualization and Computer Graphics 12 (2006) 853–860.
[40] S.K. Card, D. Nation, Degree-of-interest trees: a component of an attention-reactive user interface, in: Proceedings of the Advanced Visual Interfaces, 2002.
[41] W. Chris, Metavisual exploration and analysis of DEVise coordination in improvise, in: Fourth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV'06), IEEE Computer Science Press, Washington, 2006, pp. 79–90.
[42] J. Abello, H.F. Van, N. Krishnan, ASK-GraphView: a large scale graph visualization system, IEEE Transactions on Visualization and Computer Graphics 12 (2006) 669–676.
[43] M.C. Hao, M. Hsu, U. Dayal, A. Krug, Web-based visualization of large hierarchical graphs using invisible links in a hyperbolic space, in: Proceedings of the Fifth Working Conference Visual Database Systems, 2000, pp. 83–94.
[44] B. Lee, C.S. Parr, C. Plaisant, B.B. Berderson, TreePlus: interactive exploration of networks with enhanced tree layouts, IEEE Transactions on Visualization and Computer Graphics 12 (2006) 1414–1426.
[45] M. Sarkar, M. Brown, Graphical fisheye views, Communications of the ACM 37 (1994) 73–84.
[46] IBM ILOG JViews Diagrammer. Available from: 〈http://www-01.ibm.com/software/integration/visualization/jviews/diagrammer/about/features.htm〉.
[47] Tom Sawyer Perspectives. Available from: 〈http://www.tomsawyer.com/products/perspectives/java/index.php〉.