Electronic Journal of SADIO

http://www.dc.uba.ar/sadio/ejs

vol. 1, no. 1, pp. 88-101 (1998)

Telescope: An Object-Oriented Architecture for Visualization Systems

Ricardo Orosco

Universidad Autónoma de Madrid, Depto. Ingeniería Informática Campus Cantoblanco, (28049) Madrid, España, Univ. Nacional Centro Prov. Bs. As, Fac. Ciencias Exactas, ISISTAN, Grupo de Objetos y Visualización, Tandil, Argentina and Comision Investigaciones Científicas (CIC) Prov. Buenos Aires, Argentina. E-Mail: orosco@acm.org

Abstract

The construction of information visualization systems is a difficult task. The provision of an object-oriented software architecture for this kind of systems can help to reduce this difficulty. However, this approach is not currently used in most visualization systems. In this work, Telescope, an object-oriented architecture for visualization systems, is presented. Its main goal is to facilitate this construction, taking advantage of the benefits of the object-oriented paradigm (reusability, extensibility, and maintainability). The Telescope architecture is based on six main components, which are common to all information visualization systems: data representation, data abstraction, data objects- graphical objects mapping, presentation, interaction and visualization state. CityVis, a visualization system for city data, developed using Telescope architecture is described, showing the implementation of each Telescope component. Also, the implementation of several features in visualization systems, such as visualization techniques and management of abstraction levels and revealed information is described. Finally, current work and conclusions of the project are explained.

1. Introduction

The understanding of great quantities of data is a very difficult task [Duce 93]. Furthermore, in many cases, data have a multidimensional nature that increases this difficulty. One of the most common ways to understand data is by means of the use of graphical representations or visualizations of them, through the so-called *information visualization systems*. These systems have the goal of allowing the analyzer (i.e., the *system user*) to perform data exploration processes, so to obtain more information about the visualized data (e.g. to find items satisfying particular requirements or to detect patterns in a set of items). Nevertheless, the construction of information visualization systems is difficult and complex [Kazman 96]. There are several factors contributing to this difficulty:

- *great variety of domains.* (E.g. numerical data, geographical data, financial data, software, etc.) Each one of these domains has its own requirements and characteristics.

- *different kinds of visualizations*. The analyzer can perform several types of visualization processes [Bergeron 93]: exploratory (when the user does not know what is looking for), analytical (the user knows what is looking for in the data, trying to determine if it is there) or descriptive (when the phenomenon represented in the data is known, but the user needs to present a clear visual verification of it).

- *diversity of user tasks and goals*. Users can perform visualizations with different goals (e.g. find patterns in data, search for items satisfying some criteria, comprehension of the data space, etc.), and/or use different visualization techniques according to their purposes.

- *low reusability degree of previous visualizations*. This fact is a consequence of the hand-crafted nature of visualization systems (generally, they are built using a high-level programming language, such as C++ or FORTRAN) or because they are oriented to some particular domain.

In order to reduce this difficulty, the provision of a software architecture can help in the visualization systems construction. Particularly, object-oriented frameworks are useful in providing a generic solution for given application domains. A framework implements, in terms of object classes, the generic behavior of an application domain, while the particular features of each application are implemented through subclasses. However, neither software architectures nor object-oriented frameworks are currently used in information visualization systems. Most of current works on this area consist in the development of new visualization techniques (e.g. [Furnas 86], [Robertson 93], [Chuah 95], [Ahlberg 95], [Keim 94]), or in the provision of assistance to the user in the data exploration process (e.g. [Rennison 94], [Lucas 96], [Andrienko 97]). VANISH [Kazman 96] is one of the few works in visualization construction, although its usability is limited to hierarchically organized data.

In this work, *Telescope*, an object-oriented architecture for the structuring and building of visualization systems is presented. Its main goal is to facilitate the construction of visualization systems, taking advantage of the benefits of the object-oriented paradigm (such as reusability, extensibility, and maintainability). The design of *Telescope* was motivated by several goals:

- generality, allowing the construction of visualizations in different domains;

- allow the easy incorporation of new visualization techniques, in order to support a wide variety of user tasks and goals;

- reusability of previous visualization systems (or part of them), to reduce the time and difficulty of visualization systems construction; and

- extensibility, allowing the incorporation of new functionality in the visualization.

This paper is organized as follows: section §2 enumerates the main components of an information visualization system, and some of the software architectures reported in the bibliography. Section §3 describes the structure of *Telescope*. Section §4 presents *CityVis*, an information visualization system for city data, implemented using the *Telescope* architecture. Section §5 describes the implementation of several features of visualization techniques in *CityVis*, such as visualization techniques and abstraction levels management. Finally, section §6 explains conclusions and future work.

2. Information Visualization Systems components

According to [Duce 93], a visualization process consists of three phases, each one corresponding to one of the major transformations performed over the visualized data:

1. 'Data Access': accessing and structuring of the visualized data (generally stored in a database);

2. 'Mapping': conversion of data in a graphical representation, and the assignment of visual properties to the graphical objects (e.g. color, texture, etc.); and

3. *'Rendering'*: creation of the presentation.

Most of current visualization systems (e.g. *AVS* [Upson 89], *Iris Explorer* [Edwards 92]) present components for each of these phases. However, in order to provide an adequate support for the full visualization process, others components must be identified.

- In most cases, a visualization process often performs a semantic processing of the data to be visualized, such as a filtering phase. In consequence, may be necessary to add an *'Abstraction'* component before the creation of the graphical objects (*Mapping*).

- An information visualization system can be described as a graphical presentation of data, and direct manipulation tools for exploring relationships among these data [Fishkin 95]. Therefore, an *'Interaction'* component must be included in every visualization architecture.

- Finally, because visualization systems must support data exploration processes, it is necessary to keep information about user operations and current state of visualization process. Accordingly, it is necessary to include a *'Visualization State'* component.

In general, current visualization systems present some of these components, although not necessarily all of them. Most of these systems are centered on only one or two of these components. For example, *SDM* [Chuah 95] is focused in manipulation aspects, *Visage* [Lucas 96] in presentation construction, *Galaxy of News* [Rennison 94] in data structuring, and *Pad++* [Bederson 94] in provision of abstraction levels (through the use of zooming).

The design of software architectures for visualization systems including all of these components has

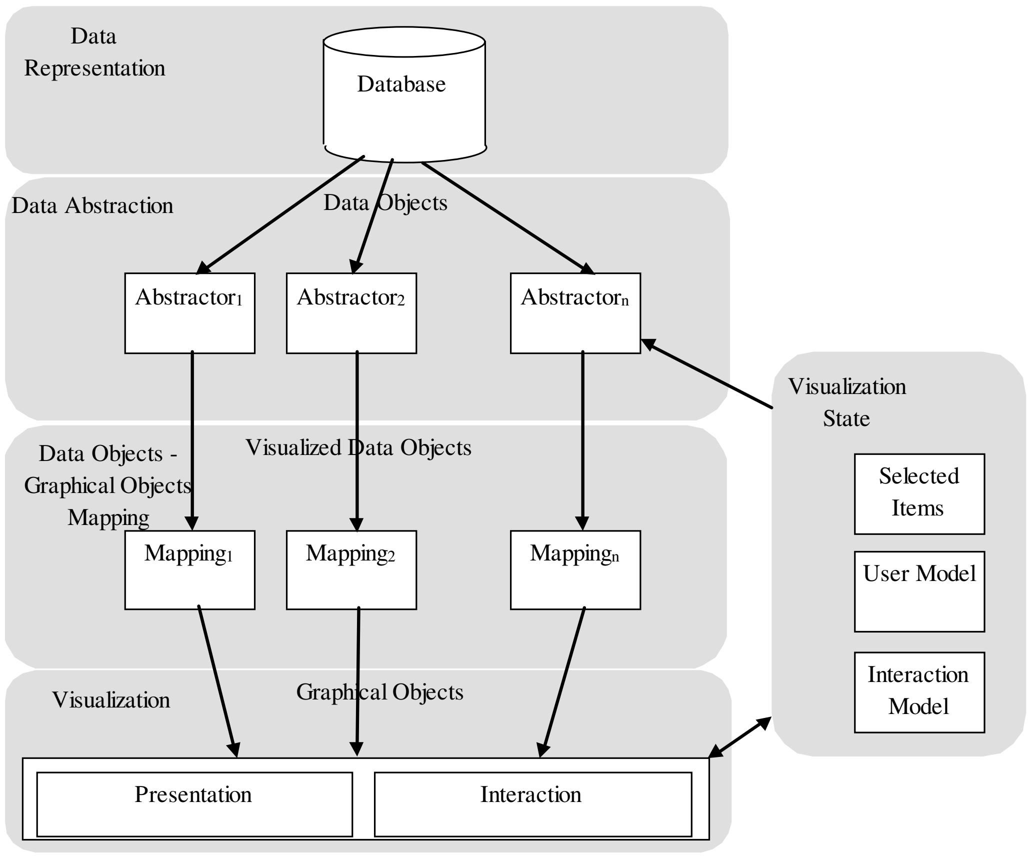

Fig 1. *Telescope* components

not been sufficiently explored. *VANISH* [Kazman 96] is one of the few works in this area; it consists of an adaptation of the *Arch/Slinky* metamodel [SIGCHI 91], designed for user interfaces architectures. Although it allows an easy and quick prototyping of visualizations, its range of applicability is limited to hierarchically organized data.

Cognitive Coprocessor [Robertson 93] is another visualization system architecture, whose main goal is to minimize the information access cost. It is used in the *Information Workspace* system, along with three-dimensional workspaces (*Rooms 3D*) and specialized browsers (*Cone Tree*, *Perspective Wall*, *Calendar Visualizer*). Its design is mainly focused on the satisfaction of temporal constraints, both for the provision of animation capabilities and to allow immersive navigations.

3. *Telescope* framework

The *Telescope* framework was designed taking into account the six components identified in the previous section. Fig. 1 summarizes the *Telescope* structure, describing the main relationships among its components.

Data Representation: storage and access to the data items to be visualized. Because data objects are generally stored in a database, this component must provide an interface to this storage (which can be implemented in a language such as SQL). Additionally, once the data was accessed, they must be organized according to the current visualization tasks and goals, to allow a quick access to the information during the exploration process.

Data Abstraction: this level determines which elements will be visualized, and classifies them into sets of related items. It is implemented by *abstractors* [Campo 97a], which are entities capable to access the data objects and determine which of them will be presented. The input of each abstractor is a set of data objects, organized in a convenient way, and its output is another set of items (e.g. in the filtering case, the output is a subset of the input), containing the data to be presented in the visualization. The determination of the visualized items may be performed based on several factors, such as visualization state, zoom level or current task.

Data Objects - Graphical Objects Mapping: this component performs the association between the data objects (and their properties) and the correspondent graphical objects. These associations are described by the system-designer in a mapping specification, which also can be interactively modified by the user at run-time. Besides, the mapping specification can dynamically change at any time, as a consequence of changes performed by the system (for example, the presentation of an item may differ according to current zoom level - *semantic zooming* -).

Visualization: it comprises graphical data presentation and user interaction. It is sub-divided into two components:

Presentation**:** it consists of presentation building, according to the graphical objects specified in the mapping.

Interaction**:** it consists of all aspects related to the user interaction: item selections and manipulations, user navigation, etc.

Visualization State: it contains the current visualization state, which can include the current selected items, user model, interaction model, etc. This state may be accessed by other system components in order to perform their tasks (e.g. the presentation component may consult which are the current selected items, to display the presentation focus).

3.1. *Telescope* software architecture

Each *Telescope*'s component is implemented by means of classes (generally abstract classes) providing its generic behavior. Each visualization application developed using *Telescope* must implement the particular behavior for each component, as shown in section §4.1. Fig. 2 shows relationships among the main classes of each component.

The visualization component consists of a set of *Views*, hierarchically organized. Each view contains a graphical presentation (an instance of a *GraphicalObject* subclass), and an interactor for each allowed operation on it (for example, a *SelectInteractor* is used to perform item selections).

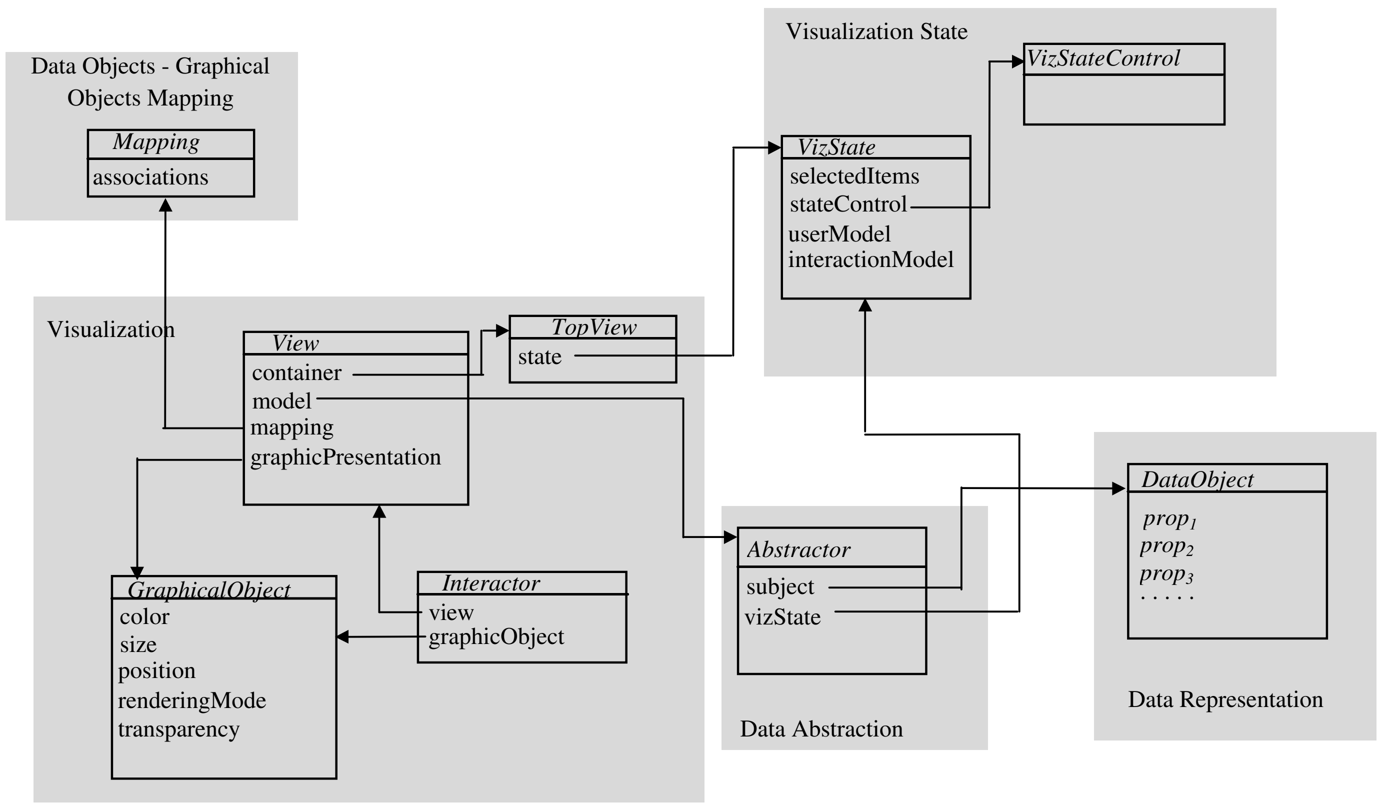

Fig. 2. Class relationships in *Telescope*

Views have associated a *Mapping* that performs the conversion between data objects and graphical objects. These mappings can be customized by the user at run-time or modified by the system according to changes in the visualization state. The *Mapping* class can be specialized through subclasses, for example to implement distortion-based visualization techniques (as described in section §5.1.1)

Additionally, each view gets the information to be displayed through an associated model. This model consists of an abstractor (instances of an *Abstractor* subclass), which determines objects and properties to be effectively visualized. This abstractor has a reference to the data object (which may be stored in a database), which contains the actual values to be visualized.

The visualization state component is mainly implemented by means of a *VizState* class, which manages all the information related to the current state of the visualization process. It also contains an instance of *VizStateControl* class, allowing to the user to control and manipulate interactively this state.

4. *CityVis*

CityVis [Orosco 97a] is an information visualization system for city data, implemented in *Visual Works - Smalltalk*, using *OpenGL* [Neider 93] as 3D graphics platform. Its main goal is to support the exploration of city information, providing techniques to assist the user in this process. Among its main features, the following ones can be included:

- 2D and 3D presentations, allowing the user to switch between the two types of presentations or to combine them. This feature enables the user to choose the best presentation paradigm for each task.
- Immersive navigation in the visualized space.
- Provision of different abstraction and zooming levels, according to the user viewpoint.
- Presentation organized in layers, allowing the differentiation of several topics in the visualization. The system visualizes different kinds of information associated to the city (hotels, apartments, restaurants, transportation, statistics, etc.), that may be combined by the user to build the desired visualization.

- Possibility of saving and restoring information deduced during the exploration process. The user can perform an exploration process, save its results, and latter resume the operation. Also, it allows the reuse of revealed information in future explorations.
- Possibility of user customization of presentations. The system-designer provides a default mapping for the presentation of each data item class, but the user can modify it at any time during the visualization.
- Provision of assistance techniques for the exploration process. These techniques attempt to alleviate the user's work in the exploration, using both user and interaction models.

Fig. 3 is a snapshot of *CityVis*, showing a 3D user navigation within the city information space.

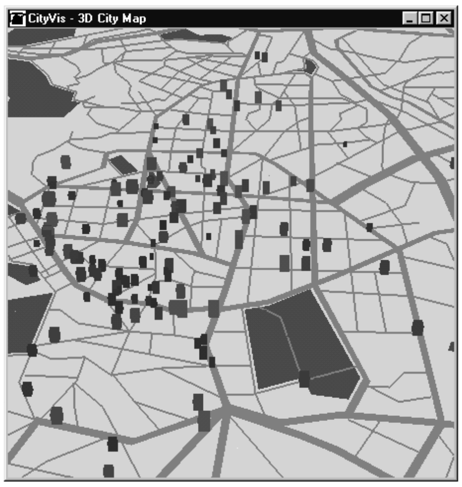

Fig. 3. *CityVis* - 3D Visualization of a city map

4.1. *CityVis* implementation

CityVis was developed using the *Telescope* framework, incorporating capabilities for city data visualization, through specialized classes for the management of this kind of data.

The *CityVis* visualization component is implemented by specialized views for each class of visualized data object. For example, views for streets, places, hotels, restaurants, transports, apartments, etc. are provided. In a similar way, for user interaction and item manipulation, different classes of interactors are provided (e.g. select and move-interactors).

Also, several mapping specifications are provided, indicating the associations between data objects and the graphical objects that must be used as their representations. For example, in fig. 3 apartments on sale are shown on the map, where each apartment (the data object) is represented by a 3D box (the graphical object), and the apartment price is represented by the box color (items with red color has a high price than blue items).

The information presented to the user is determined by means of abstractor objects. In *CityVis*, there is a specialized abstractor (a *CityAbstractor* class) for the global management of the city information to be shown, and different abstractors (instances of *CityTopicAbstractor* class) for the management of related information (e.g. topics such as hotels, apartments, map, etc). Additionally, there are other abstractors that manage information revealed during the exploration process (e.g. statistics, as described in section §5.2.)

Finally, a CityVisState class, containing all the information about the current visualization and exploration process manages the current state of the visualization.

5. Implementation of Visualization Systems Features

In this section, the implementation of several features included in most information visualization systems is described. These features are:

- incorporation of different kinds of visualization techniques;

- management of revealed information during the exploration process; and

- automatic management of different abstraction levels.

All descriptions are based on their current implementation in *CityVis*.

5.1. Visualization Techniques

Currently, three kinds of visualization techniques are implemented in *CityVis*: distortion-based techniques, layering techniques and assistance techniques.

5.1.1. Distortion Techniques

These techniques consist of a distorted graphical presentation, with the goal of emphasize those items of current interest to the user (the focus). *Fisheye* [Furnas 86], *Stretching* [Sarkar 93], and *Bifocal Display* [Spence 82] are some of the most common distortion-based techniques.

CityVis implements this kind of techniques by means of specialized mappings for each technique. Before the mapping returns the graphical object to be displayed, the object is distorted according to the technique implemented by this mapping. For example, a fisheye distortion is implemented by a *FisheyeMapping* class (subclass of *Mapping*), in which the method *getObject:* returns the distorted graphical object to be used in the presentation:

```
getObject: aDataObject
    "returns graphical object associated to aDataObject"
          "1: gets the associated graphical object"
   graph_obj := super getObject: aDataObject.
          "2: distorts the graphical object"
   self distort: graph_obj.
          "3: returns the distorted graphical object"
   ^graph_obj
```

The implementation of the *distort:* method depends on each technique, because each one of them performs distortions over different attributes. For example, in fisheye views, the distorted attributes are position and size.

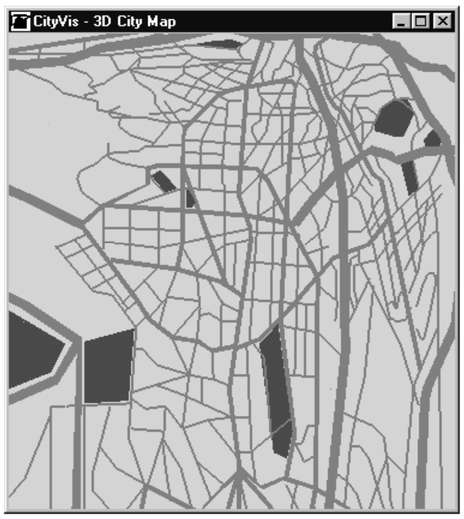

Fig. 4. Distorted view in *CityVis*

```
distort: aGraphicalObject
 "gets original object position"
 pos := aGraphicalObject position.
        "d = distortion factor"
 aGraphicalObject position: ((1 + d) * pos) / ((d * pos) + 1).
        "gets original object size"
 size := aGraphicalObject size.
 aGraphicalObject size:  ((1 + d) * pos) / ((d * pos) + 1).
```

In this way, adding a new distortion technique consists only in the implementation of a specialized mapping, and the correspondent distortion functions. Fig. 4 shows a snapshot of a distorted map in *CityVis*.

5.1.2. Layering techniques

These visualization techniques are generally used to compose different topics of information, based on the *'transparent paper'* metaphor. For example, in *CityVis* there are different layers for the city map, transports, hotels, restaurants, apartments, etc. The user can manipulate these layers, adding and/or removing some of them at any time during the visualization, to display only the information relevant for the current task. Some of the most common layering techniques are *Livemap* [Silvers 95] and *Back to the Future* [Belge 94].

CityVis implements layering techniques through different views for each layer, each one with an associated abstractor (a *CityTopicAbstractor*, as it is shown in fig. 5) that determines the visible information in the layer. The management of visible topics at run-time is done automatically by the *CityAbstractor* (generally in response to user requests), which determines if each *CityTopicAbstractor* is enabled or not. Fig. 6 displays a snapshot of the *CityVis* layering, showing the city map, subway lines, restaurants and hospitals layers.

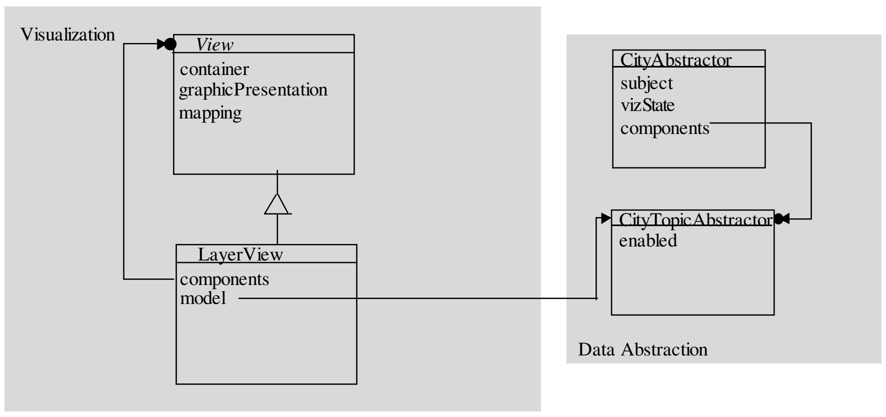

Fig 5. Class relationships in layering technique implementation

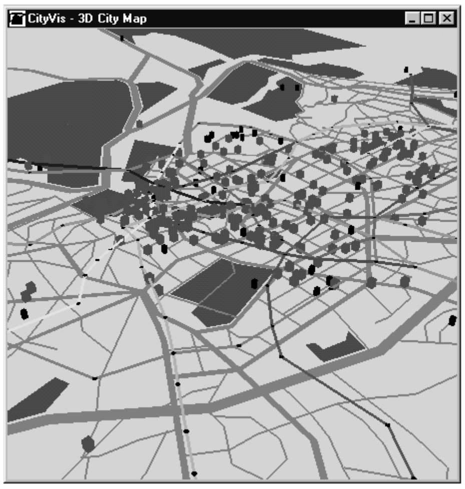

Fig 6. Layering techniques in *CityVis*.

5.1.3. Assistance Techniques

In the last years, several visualization techniques providing some degree of user assistance in the information visualization process have been developed (e.g. *Galaxy of News* [Rennison 94], *Information Workspace* [Robertson 93], *IRIS* [Andrienko 97]). *AutoFocus* [Orosco 97b] is an example of assistance technique implemented as part of *CityVis*. Its goal is to assist the user in the exploration process, providing help in the focus specification and using 3D presentations conveying more information. The assistance in the focus specification is implemented by a statistical inference mechanism, that attempts to detect the criteria used by the user in item selections. Once these criteria has been estimated by the system, item presentations are updated accordingly to them, reflecting the relevance of each item with respect to the currently defined focus (using an item opacity proportional to the importance of each item). In this way, a quick approximation to the desired final focus is obtained, facilitating the task and reducing the user workload. Particularly, *CityVis* uses *AutoFocus* to assist the user in the searching of apartments on sale.

This kind of techniques can be implemented through the incorporation of a specialized abstractor (a *CriteriaInferenceAbstractor*) that performs the inference process. Additionally, the visualization state (*CityVisState*) is extended, adding a criteria list (instances of *Criteria* class), which represents criteria inferred by the system. Each of these criteria contains all the information about it, such as its values, its certainty and/or its associated attribute. Fig. 7 shows the main relationships among classes in the *AutoFocus* implementation, while fig. 8 shows a snapshot of *AutoFocus* operation.

The *AutoFocus* operation consists of two steps: first, the user criteria are estimated, and then the relevance of each item with respect to the current focus is computed. The first step is performed when a new item is selected (or deselected); in this case, the visualization state receives a message indicating the item selection:

```
addSelectedItem: anItem
 selectedItems add: anItem.
 criteria := criteriaInference inferCriteriaWith: selectedItems.
 stateControl setControlsWith: criteria.
```

Once this message is received, the state updates the selected item list, and then informs to the *CriteriaInferenceAbstractor* that it is necessary to recalculate the criteria (through the method *inferCriteriaWith:*). When the new criteria has been calculated, the state updates the widget controls for each criteria (e.g. range-sliders or range-buttons).

The method *inferCriteriaWith:* implements the process of criteria deduction. A simplified version of its code is:

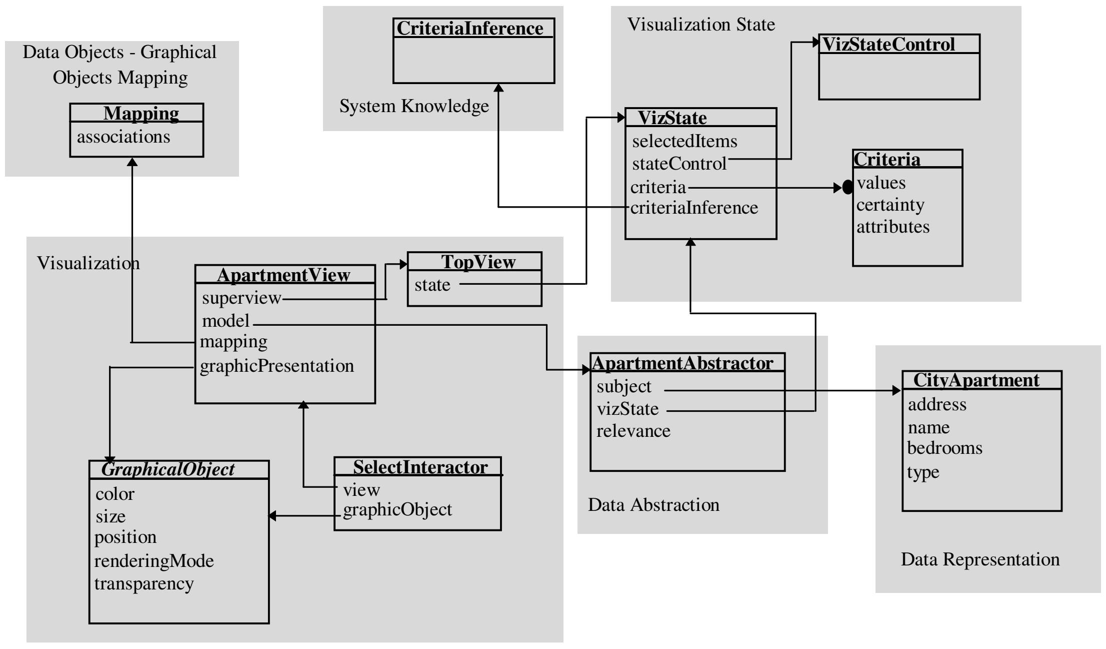

Fig. 7. Class relationships in *AutoFocus* implementation.

```
inferCriteriaWith: items
 crit := OrderedCollection new.
 visual_attrs := items getVisualAttrs.
 visual_attrs do: [:va | crit add: (self infer: va with:  items)].
 self removeDependencies.
 ^crit.
```

In this code, firstly the data attributes that may be used by the user in the selection are obtained, through the associated visual attributes shown in the presentation (using the method *getVisualAttrs*). Once these attributes are obtained, the system attempts to infer one criterion for each one of them. This task is done by the method *infer:with:*, which performs a statistical analysis of the values for each attribute in the selected items, determining the composition of the criteria (mainly, its values and its certainty). The implementation of *infer:with:* depends on the type of the attribute (e.g. the analysis used in attributes with continuous values is different from the one used in a boolean case). Finally, after one criterion for each attribute is deduced, the system attempts to detect redundant criteria, through a dependency analysis of the domain attributes (this step is performed by the *removeDependencies* method)

In the second step, each *ApartmentAbstractor* determines the current relevance of its associated item with respect to the current focus, using the criteria deduced in the first step, through the method *relevance*:. The relevance of each item is measured by a closeness function computing the distance from this item to each criterion.

```
relevance
 crit := vizState criteria.
 closeness := 0.
 crit do: [:c |
        closeness := closeness +((c critCloseness: self) normalized)].
^closeness
```

The item closeness is the sum of the individual closeness of each deduced criterion (which are computed by the *critCloseness:* method). The relevance is then mapped to the transparency of the graphical presentation, as is indicated in the mapping associated to the correspondent *ApartmentView*.

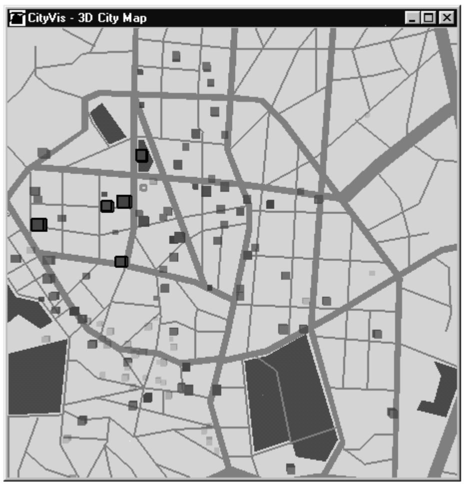

Fig. 8. *AutoFocus* snapshot. In this case, items with similar properties to selected items are displayed in a more emphasized way, while dissimilar items are displayed with a higher transparency degree.

5.2. Management of Revealed Information

CityVis allows the management of new information revealed during the exploration process. The user can define new relations over objects, and store them as a new topic. For example, it is possible to define a new topic *'transport'* as the union of the bus, subway and train items. Once this new topic has been defined, it can be used in future exploration processes.

Another possibility in *CityVis* is to build statistics about the visualized information. Particularly, *CityVis* allows to determine zones with a high concentration of given items. For example, the user may desire to know which is the city zone with a high concentration of hotels and restaurants; in this case, the system generate a statistic based on these items, which is displayed through a semi-transparent layer indicating the density of these items in the desired zone (fig. 9 shows an example of statistic layer). This statistic is saved as a new layer, which may be used in future explorations, where the user do not need to access to the items used to build it.

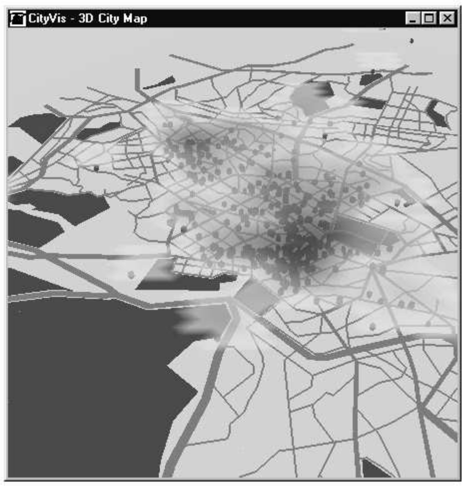

Fig. 9. Statistics layer in *CityVis*.

Specialized abstractors manage both kinds of revealed information. In the first case, a *RelationshipAbstractor* performs modifications to the data organization, while in the second one, an *StatisticsAbstractor* generates new data objects based on a previous set of objects.

5.3. Abstraction Levels Management

The management of abstraction levels is one of the main features in all data visualizations. However, the construction of techniques performing an automatic control of the abstraction or detail level is not easy. In addition, most of current implementations have little reusability, because the knowledge about this management is hard-coded into the visualization system.

One of the most common techniques managing detail levels based on the user viewpoint is *Semantic Zoom* [Muthukumarasamy 95], which is currently available in *CityVis*. In this case, the management of abstraction levels is performed automatically by the abstractors, which determine those items that must be visible at the current zoom level, filtering those elements that must not be visible at this level. For this, each abstractor contains a reference to an *AbstractionScale* object (as it is shown in fig. 11), which contains the information about the scale used. A more detailed description of the implementation of the automatic abstraction levels management can be found in [Campo 97b].

Fig. 10 shows a map view in a higher abstraction level than in previous figures, where only main streets and places are shown.

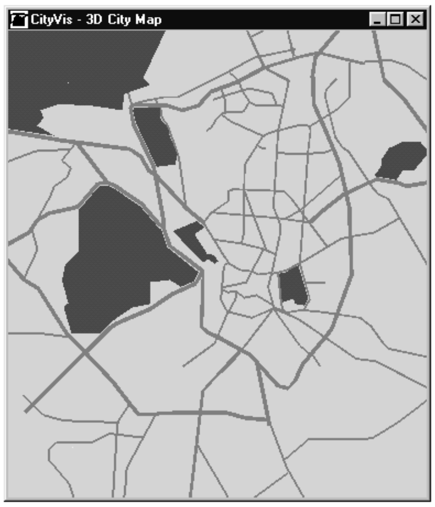

Fig. 10. Higher abstraction level in *CityVis*

6. Conclusions and Future Work.

In this work, *Telescope*, an object-oriented architecture for visualization systems is described. The architecture emphasizes the division among the main components in visualization systems, trying to provide an adequate support for the reusability and extensibility of these systems, therefore facilitating the construction task.

A visualization system, *CityVis*, was developed to demonstrate the suitability of *Telescope* to the domain of geographical data. Also, several types of visualization techniques were added to *CityVis*, showing the facilities provided by *Telescope* for the incorporation of new functionality and/or visualization techniques to existing visualization systems. Other features of information visualization systems were described, such as the management of abstraction levels and revealed information. In all of these cases, the implementation of these features is simple and allows a quick prototyping of visualization systems. Additionally, the extensibility of Telescope is high, due to the clear localization of possible changes and extensions.

Future work in *Telescope* includes the development of other applications in new domains, mainly those dealing with abstract data (e.g. WWW networks or bibliographical databases). Also, the implementation of new visualization techniques and the provision of a better support for the user interaction is currently being explored.

Acknowledgments

The author would like to acknowledge Dr. Marcelo Campo by his insight and ideas in the design of *Telescope*. The author would also like to thank to Dr. Roberto Moriyon for his continuous support and advice provided during the development of this project. Finally, thanks to Alvaro Ortigosa for his comments on drafts of this paper.

References

[Ahlberg 95] Ahlberg, Christopher. Wistrand, Erik. ***"IVEE: An Environment for Automatic Creation of Dynamic Queries Applications"*** Proc. CHI '95 Conf. Companion, pp 15-16 05/1995.

[Andrienko 97] Andrienko, G. Andrienko, N. ***"IRIS: a Knowledge-Based System For Visual Data Exploration"*** Proc. CHI '97, 1997.

[Bederson 94] Bederson, B. Hollan, J. ***"Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics"*** Proc. UIST '94, pp 17-26.

[Belge 94] Belge, M. Lokuge, L. Rivers, D. ***"Back to the Future: A Graphical Layerong System Inspired by Transparent Paper"*** SunSoft, Chelmsford, MA, 1994

[Bergeron 93] Bergeron, D. ***"Visualization reference models".*** Proc. of Visualization '93. IEEE Computer Society Press, 1993.

[Campo 97a] Campo, M. ***"Compreensao Visual de Frameworks através da Introspecao de Exemplos"*** Phd. thesis, UFRGS, Brasil., 1997 (in portuguese)

[Campo 97b] Campo, M. Orosco, R. Teyseyre, A. ***"Automatic Abstraction Management in Information Visualization Systems"***, Proc. V Intl. Conference on Information Visualization, London, August, 1997.

[Chuah 95] Chuah, M. Roth, S. mattis, J. Kolojejchick, J. ***"SDM: Selective Dynamic Manipulation of Visualizations".*** Proc. UIST '95, pp 61-70, 1995.

[Duce 93] Duce, D. A. ***"Visualization"*** Visualization '93 Conf.

[Edwards 92] Edwards, G ***"Visualization - the second generation"*** Image Processing, 1992

[Fishkin 95] Fishkin, K. Stone, M. ***"Enhanced Dynamic Queries via Movable Filters".*** Proc. CHI '95, pp 415-420. 05/1995.

[Furnas 86] Furnas, George. ***"Generalized fisheye views"*** Proc. ACM SIGCHI Conf. on Human Factors in Computing Systems, pp 16-23, 1986.

[Kazman 96] Kazman, Rick. Carriere, Jeromy. ***"An Adaptable Software Architecture for Rapidly Creating Information Visualizations"*** Proc. Graphics Interface '96, Toronto, 05/96.

[Keim 94] Keim, D. Kriegel, H.P. "VisDB: Database Exploration Using Multidimensional Visualization" IEEE CG&A, 09/94, 1994.

[Lucas 96] Lucas, Peter. Roth, Steven. Gomberg, Cristina ***"Visage: Dynamic Information Exploration"*** Proc. CHI 96 Conf. Companion, pp 18-20, 1996.

[Muthukumarasamy 95] Muthukumarasamy, J. Stasko, J. ***"Visualizing Program Executions on Large Data Sets Using Semantic Zooming"*** Georgia Institute of Technology, 1995, Tech. Report GIT-GVU-95-02.

[Neider 93] Neider, J. Davis, T. Woo, M. ***"OpenGL Programming Guide"*** Addison-Wesley, 1993, 516 pp.

[Orosco 97a] Orosco, R. Moriyon, R. ***"CityVis: an Intelligent Information Visualiza-tion System for City Data"*** Tech. Report, Dept. Ing. Informatica, UAM, in preparation.

[Orosco 97b] Orosco, R. ***"AutoFocus: User Assistance in Information Visualization"*** Hypertext, Information Retrieval and Multimedia '97 (HIM '97) Conf., Dortmund, Germany, October, 1997, 18 pp.

[Rennison 94] Rennison, Earl. ***"Galaxy of News: An Approach to Visualizing and Understanding Expansive News Landscapes"*** Proc. UIST '94, pp. 3-12, 1994.

[Robertson 93] Robertson,G. Card, S. Mackinlay, J. ***"Information Visualization using 3D Interactive Animation"*** Comm. of the ACM, 36(4), 04/93, pp 57-71, 1993.

[Sarkar 93] Sarkar, M. Snibbe, S. Tversky, O. Reiss, S. ***"Stretching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens"*** Proc. UIST 93, pp 81- 91, 11/1993.

[SIGCHI 91] UIMS Tool Developers Workshop. ***"A Metamodel for the Runtime Architecture of an Interactive System"*** SIGCHI Bulletin, 24(1), 1991, pp 32-37.

[Silvers 95] Silvers, Robert. ***"Livemap - A System for Viewing Multiple Transparent and Time-Varying Planes in Three Dimensional Space"*** Proc. CHI 95 Conference Companion, pp 200-201, 05/1995.

[Spence 82] Spence, R. Apperley, M. ***"Database navigation: An office environment for the professional"*** Behav. Inf. tech., 1,1. pp 43-54, 1982

[Upson 89] Upson, C. ***"The Application Visualization System: a computational environment for scientific visualization"*** IEEE CG&A, July 1989