INCORPORATED DYNAMIC SYMBOLS IN DRAWING (CAD) & SPATIAL (GIS) SYSTEMS

by
Nikos C. Zestas, Ioannis G. Paraschakis,
Athanasios D. Styliadis

Abstract: In this paper an application framework is presented, which extends the object model of an Object-Oriented CAD or GIS system by incorporating user-defined dynamic graphic symbols in its class hierarchy tree.

In the proposed framework, the dynamic graphic symbols are database persistent objects of point type, defined as collections of standard graphics entities including annotation text. The symbols are associated with other graphic entities, or a data source, and incorporate intelligence on synchronizing their own contents with the current status of the source data. Standard graphics entities are used as graphics containers, while the initial contents of text entities are used as keywords to retrieve actual information from data sources or other entities. Thus, each instance of a dynamic symbol class, transforms and renders at a given position each one of the entities, for every time the map or the graphics display needs to be updated.

Keywords: information visualization, dynamic symbols, CAD, GIS

1. INTRODUCTION

Annotation text and symbolic icons are essential parts of a digital drawing or a map's content. They are used to visualize spatial data and help the users to better understand spatial relationships. Depending on the point of interest there is a lot of literature in this field, such as the content, location, visual hierarchy and representation [1,2,3,4]. Automated placement of labels has been also a fundamental task in GIS and CAD systems. The "Find Best Placement Label Problem" has received considerable attention from researchers, due to both its practical applications in the areas of cartography and to its theoretical significance [6,7]. We have also mentioned here dynamic labeling that is used for information exploration of a spatial database. Dynamic labels are created in an interactive process when the user moves from one area of interest to another. Modern Spatial Database Management Systems (SDBMS) support dynamic query interfaces for visual database exploration.

The work presented so far is related to the comprehension of a map by the user, which is a critical point in the design process but not always the major one. Specific maps and drawings in engineering mapping, e.g. construction maps, topographical diagrams, architectural or mechanical drawings require standardization. Specification sets impose rules of how the information must be organized, where exactly it must be placed and how it must be displayed. Labels are often combined together with graphics in a single symbol according to a specifications set. Although in the age of information there is a lot of application-oriented software for some

specific uses (sometimes called vertical applications software), the specific needs of different user groups cannot be usually met satisfactorily.

Previous demands in drawing appearance and content cannot be met of course by predefined automated processes, unless the specifications set is permanent and known at development time. Unfortunately specifications differ from country to country and often change. So the common practice of CAD users today is to use graphic blocks with text attributes or text entities with the desired information as content. Excluding dimensioning which are natively supported in today's CAD systems, all other kind of information has to be rendered as text. This method has some disadvantages: the user has to acquire manually the data, convert them to text and then to append the appropriate text or block entity to the drawing. Furthermore, the implicit relation between the annotation entities that renders the information and the original entity, which actually maintains that information (source entity), cannot be established with the above procedure. In this case the functionality is clearly restricted, for instance, an annotation entity may become invalid if during the editing process the targeted source entity has to be modified.

Spatial systems incorporate procedures dedicated to map production and data visualization. They can create static or dynamic labels, but their symbols repository is restricted to its predefined shape library for point-feature representation. These symbols are mainly used for the representation of a qualitative attribute at specific location. In a map containing numerous annotations and symbolic information the symbols readability may be reduced [5], due to a limited shape repository.

2. GENERIC PROBLEM FORMALIZATION

From this paper point of view a graphic symbol is an object, which encapsulates two different kinds of data components: the information for which it stands for and its graphics display representation. The source information component acquires its data employing methods that may relate more than one object. More than one spatial relations or constraints may also be represented by a symbol.

The display representation of the symbol also transfers information to the user of the map, so display representation entities may be data acquired from the information component. Graphic objects in an Object-Oriented schema always incorporate logic of how to draw themselves. In a dynamic environment drawings and maps may be updated continuously under an automated process, or may be regenerated due to a specific user's action. A symbol evaluates and draws itself dynamically anytime its virtual 'draw' function is called.

Symbols are distinguished from the spatial objects that model the real world. They may be temporal objects and their size must be in paper or display units in order to be readable. Database processes may also use symbols in a graphics screen in order

to inform process control operators about significant information that may be otherwise hidden or hardly readable. Lastly symbols must be user defined.

Generally, a map is created from a spatial database using a finite set of available methods (functions). Let *D* be a finite set of database available functions.

$$D = \{m_1, m_2 \dots m_n\} \quad (1)$$

Symbols are independed objects and incorporate their own methods about retrieving and displaying information. The set *S* is defined here as the finite set of the methods that a symbol uses for information retrieval.

$$S = \{a_1, a_2 \dots a_n\} \quad (2)$$

Members of set *S* are generally knowledge retrieval functions that depend on one or more members of *D*. Consequently we consider the following mapping structures:

a. *M : K → F (where K is a literal keyword and F is a function)*: it relates attributes names to functions of set *S*

b. *N :* O → G (*where O is literal keyword and G is a set of graphics entities)*: it relates the Symbol names to geometry.

3. THE PROPOSED FRAMEWORK

For the above specifications to be met we introduce a new layer for information access. This layer may be incorporated in host CAD/GIS application or developed using a Software Developer's Kit (SDK) by a client application. SDKs are widely used today by developers, because they let the user to extend the object model of the host application. For example, ESRI provides to users ArcObjects(*), a software developer's kit which uses Microsoft's COM (component object model) in order to let the users to extend ArcGis's object model. AutoDESK(**) provides ObjectDbx and ObjectArx its own software developer's kit, which also encapsulates this technology. Smallworld(***) has its own object-oriented language named magic, which supports object oriented technology. Developers working on an open source product do not need a developer's kit.

The proposed framework manages the *M*,*N* mappings already introduced in previous section, via a an object named **InfAccessorManagerObject.** This object is a unique instance of the framework class, also serving as a repository for two class types, named here **CSymbol** and **CInfAccessor.** These classes define the interface and implement logic about symbols and methods for information access respectively. **InfAccessorManagerObject** registers mapping structures of type {attribute-name,

(*) http://www.esri.com
(**) http://www.autodesk.com
(***) http://
//www.gepower.com/dhtml/network_solutions/en_us/smallworldtechnology/magikstudio.jsp/

CInfAccessor-pointer}, where attribute-name is a literal key and **CInfAccessor**-pointer is a pointer to the object that implements the method. **CInfAccessor** is a base class that defines a virtual function for data access. Let **FetchInfo** be the name of this function. Lastly we define our CSymbol class, which is a derived class inheriting directly from a graphics object of point type. **CSymbol** as the other graphics objects has to implement it's own 'draw' method.

Each **CSymbol** class has the following members:

a. Its name.
b. A point that identifies the symbol's position.
c. Three rotation angles for the orientation of the **CSymbol** in 3D space.
d. A set of instances to other objects that that actually maintain the information represented by the symbol.
e. The **Draw** method which draws the symbol, as next defined.

We can now easy define the logic of our keyword-based framework here. The user draws text entities as a part of a symbol's graphical representation, and registers them to the framework as a symbol using a name. **InfAccessorManagerObject,** performs the registration using the previous defined *N* mapping. Framework can be used in order to instantiate symbols at a given position now. When a symbol instance needs to be drawn that instance first acquires its graphics representation from the framework using its name as a key. The normal behavior of symbol's 'draw' function is to transform and draw each one of the entities of the graphics representation set. This normal behavior is not applied to the text entities of the graphic set, since they are treated differently. Usually each text entity has text content and some graphical attributes (color, style, etc) that define its display representation shape. Text entity content is interpreted as the keyword. We use this keyword to retrieve from the framework a virtual pointer to a **CInfAccessor** object. Once this pointer has been successfully fetched, its virtual function **FetchInfo** is executed, the arguments being the key and the object instances that are currently associated with this symbol instance. A clone of the text entity is created with exactly the same graphical attributes and with the acquired information as content. The clone is translated and drawn at symbol position.

The implementation logic of **CSymbol's** 'draw' method is defined step-wise below:

1. *Get the global unique instance of **InfAccessorManagerObject** and save a pointer to it on **pIAObj***

*2. Using the symbol name and the referenced objects get the current graphic representation set from **pIAObj**[(a)]*
*3. Create a transformation matrix from world to symbol's position with respect to rotation angles, and save it to variable **MATRIX***
*4. For each graphic object **GOBJ** in the current graphic representation set*
5. If this is a text or annotation object then
*6. Get the text content of the object and save it to variable **KEY***
*7. Use **KEY** as a keyword in order to get a virtual pointer to a **CInfAccessor** object from the **InfAccessorManagerObject**.*
*8. If the pointer to the **CInfAccessor** object is valid then*
*9. Invoke **FetchInfo** of **CInfAccessor** with*
*10. arguments the **KEY** and the referenced object instances*
11. that associated with this symbol
*12. If **FetchInfo** returns successful then*
13. Create a clone of the text object
*14. Transform the clone using **MATRIX***
15. Set clone's content according to
*16. **FetchInfo** return value*
*17. Call clone's **'Draw'** method*
*18. Else if **FetchInfo** fails then throw an exception*
*19. Else if the pointer to **CInfAccessor** object is not valid then*
*20. Transform text object using **MATRIX***
*21. Call text's **'Draw'** method*
22. Else if it is not a text object
*23. Transform object using **MATRIX***
*24. Call object's **'Draw'** method*
25. End of for loop.

4. AN IMPLEMENTATION EXAMPLE

An implementation of the above method has been made using AutoCAD as main platform. The framework was developed using AutoDESK's ObjectArx

(a) In dynamic environment the framework may return different graphics representations, depending on the state of objects referenced by the symbol.

technology, and its task was to draw symbols acquiring data from an also ObjectArx network application, namely a managing a sewage network.

4.1 The hypothesis

In Fig. 2 we see a small piece of a sewage network consisting from two pipelines and two manhole-joints N1, N2. N1 represents a main manhole-joint, while N2 stands for a connection joint for buildings, connected to the main network via a service connection pipe. From above the ground only manholes represented by N1 & N2 are visible. Given the distances A and L, the exact location of the service connection pipes can be easily determined in field (see Fig. 2).

Lets say that we want to create two symbols Symbol1 & Symbol2. Symbol1 will be used for visualization purposes of main manholes displaying a framework of lines and some attributes, e.g. name, ground elevation and depth. Symbol2 will be used to display properties of manholes that are used as connection joints for buildings as N2 in fig2.

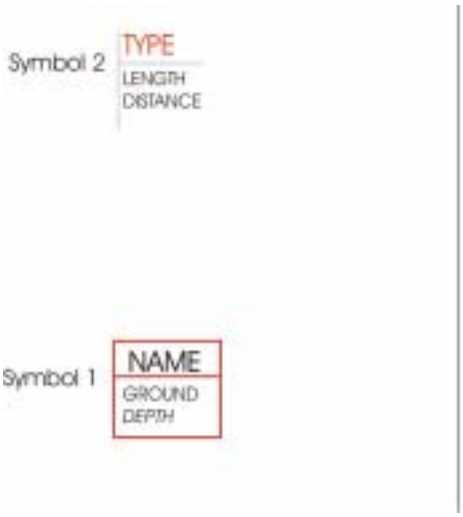

Fig 1. Design and definition of Symbols

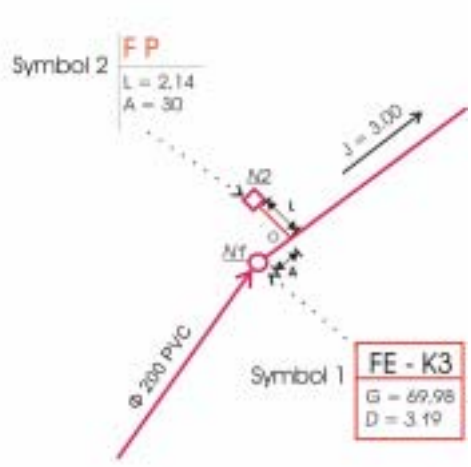

Fig 2. Placement of symbols

4.2 The implementation of the framework

The framework behavior has been implemented by registering in the following list of mapping structures *{keyword, method}*:

1. {LENGTH, GetLength}. GetLength evaluates the total length of the objects in the argument list. When GetLength is applied to a manhole

used as connection joint the length of the service connection pipe is returned, i.e. L=2.14 in Fig. 2.

2. {TYPE, GetType}. GetType retrieves the type of the first object in the argument list, i.e. FP in Fig. 2.
3. {DISTANCE, GetDistance}. GetDistance expects only one object in the argument list which must be a manhole used as connection joint. It evaluates the distance A, which is useful in order to identify the location of the service connection pipe i.e. A=30 in Fig. 2.
4. {GROUND, GetElevation}. GetElevation retrieves the elevation of the first object, which is supposed to be a manhole joint.
5. {DEPTH, GetDepth}. GetDepth retrieves the depth of the first object, which is supposed to be a manhole joint i.e. G=69.98 in Fig. 2.

4.3 The Usage

First we create interactively in AutoCAD's drawing editor the symbols Symbol1 & Symbol2, using AutoCAD native entities. Secondly we register them as symbols in the framework. Next we created an instance of Symbol1 associated with joint N1 and an instance of Symbol2 associated with joint N2. The properties related to the symbol of each object appeared to those symbols. Both symbols reflect database changes every time the drawing is regenerated.

5. CONCLUSION

A generic framework for automated information retrieval has been developed, by employing user symbols. Information visualization with dynamic symbols fits better to the user needs, since one can use any shape for display representation. As a result, user aesthetics concerning drawing representation are much better met. The framework instead of queries uses user-friendly keywords associated with information retrieval functions in order to display data. Keywords here act as information retrieval patterns.

The presented procedure greatly facilitates the interactive placement of compound labels for information retrieval. In continuously updated dynamic displays, symbols are also continuously updated. Since symbols are independent objects they can be used to record specific properties of database objects over time.

REFERENCES

[1] **Kraak M.J. and F.J. Ormeling** (1996) *Cartography. Visualization of spatial data. Addison Wessley Longman Limited. ISBN 0-582-25953-3*

[2] **Cuff , David J. and Mattson Mark T.,** (1982). *Thematic maps. Their design and production* Methuen & Co, ISBN 0-416-33500-4 Pbk.

[3] **Dickinson, G.C.** (1963) *Statistical mapping and the presentation of statistics. 2nd edition* Edward Arnold ISBN 0 7131 5683 X

[4] **Wang, Peter C.C.** (1978) *Graphical representation of Multivariate Data* Academic Press Inc, ISBN 0-12-734750-X

[5] **Robinson, Arthur H., Joel L. Morrison, Philip C. Muehrcke, A.Jon Kimerling and Stephen C.Guptill** (1995) *Elements of Cartography* John Wiley & Sons, Inc 6th Ed. 1995 ISBN 0-471-55579-7

[6] **Christensen, J., Joe Marks and Stuart Shieber** (1995) *An Empirical Study of Algorithms for Point-Feature Label Placement.* ACM Transactions on Graphics, Vol. 14, No.3, July,1995, Pages 203-232.

[7] **Doddi Srinivas, Madhav V. Marathe and Bernard M.E.Moret** (2000) *Point set labeling with specified positions* Proceedings of the sixteenth annual symposium on computational geometry, Clear Water Bay, Kowloon, Hong Kong Pages 182-190.

Authors:

Nikos C. Zestas, The Aristotle University of Thessaloniki, Dept. of Cadastre, Photogrammetry & Cartography, P.O.Box 469, 541 24, Thessaloniki, GR, polyedro@otenet.gr

Ioannis G. Paraschakis, The Aristotle University of Thessaloniki, Dept. of Cadastre, Photogrammetry & Cartography, P.O.Box 469, 541 24, Thessaloniki, GR, jpar@topo.auth.gr

Athanasios D. Styliadis, The Alexander Institute of Technology, Dept. of Information Technology, P.O.Box 14561, 541 01 Thessaloniki, GR, styl@it.teithe.gr