Dynamic Information Visualization Using 3D Metaphoric Worlds

C. Russo Dos Santos, P. Gros, and P. Abel
Multimedia Dept.
Eurécom Institute
2229, Route des Crêtes
06904 Sophia-Antipolis, France
email: {cristina.russo,pascal.gros,pierre.abel}@eurecom.fr

ABSTRACT
In this paper we discuss the mechanisms that have been used to deal with the dynamic nature of data in the CyberNet project. The purpose of this project is to study how three-dimensional (3D) metaphoric visualization may help the user in the process of monitoring large amounts of dynamic information. We present the dynamic data model and the visualization model and then focus on the problem of updating a 3D world without disorienting the user.

KEY WORDS
Dynamic Information Visualization, 3D Visual Metaphors

1. Introduction

Nowadays there is more and more demand for information visualization systems in order to supervise large amounts of dynamic information such as stock exchange, for example [1]. Specific problems arise when dealing with dynamic information because the visualization evolves with time and this may disorient the user. This issue is especially pertinent when one needs to move objects in the visualization in order to make room for newly created objects. Another problem is to define a data model that can handle the dynamic nature of data.

The CyberNet project [2] goal is to deliver an interactive 3D dynamic information visualization tool designed to evaluate the added value that this technology brings to the visualization of large volumes of dynamic information. Although the tool has also been designed to deal with any kind of dynamic data, the system has been experimented in the context of network management and an example is reported further in the paper.

In this paper, we present some of the mechanisms that have been used to deal with the dynamic nature of data within the CyberNet project.

2. Dynamic data model

We focus this paper on the visualization of dynamic data. Dynamic data is data that varies in time. Data may vary in amount (new data may appear and older data may disappear) as well as in value (the value of existing data may vary in time). In addition to the time dependency, a fundamental point is that the system must process data on-line and, thus, is not aware of the modifications before they happen.

2.1 How to handle data dynamics?

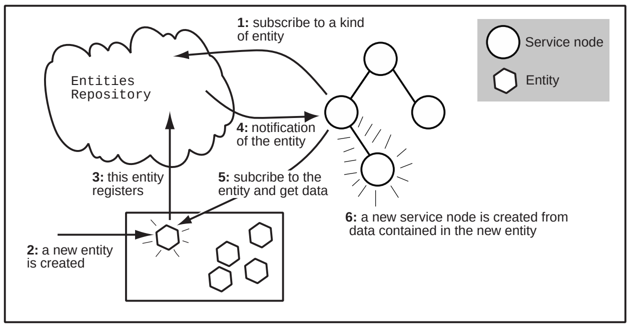
Online processing of dynamic data requires translating the dynamic data into visual elements. This translation requires several steps. The first step toward this process is to logically structure the data in order to obtain a data model – this step is called *data transformations* according to the terminology of [3]. Basically, structuring the data requires to group data according to common properties and to identify relationships. In our structuring process the final data model is a tree.

The data model will also be used to handle the dynamic nature of the data, each time new data is created or existing data is modified, the system will automatically update the data model to reflect this modification. Because the data model is automatically translated into a visual representation, the modification will be visualized.

The data model we have developed is based on the concept of *entities*. Entities are used to group all the values that are necessary to describe some logical element of the data model. An entity can represent a physical device (e.g., a router, a hub) or a conceptual item (e.g., a process, a file-handle). The entities are created by an entity collecting process described in [4].

Since we are dealing with a dynamic environment, entities have a life cycle: new entities may be created, existing entities may disappear, and the values of the entity may vary. In order to cope with this dynamics all the entities are stored in an *entity repository*. The role of the repository is to keep track of all the existing entities and to be able to answer to queries concerning entities (i.e., all the hubs that have an IP address in a given range). The difference between the repository and traditional database management systems is that the queries are persistent. When an object makes a query to the repository, all the entities that match the query are returned to the caller; if, later on, a new entity that matches the query is created, the caller will be notified. This *persistent query* capability is the basis of the dynamic data model.

The data model tree is composed of *service nodes*. A service node is an object that knows how to interpret part



1: subscribe to a kind of entity
Service node
Entity
Entities Repository
4: notification of the entity
3: this entity registers
5: subcribe to the entity and get data
6: a new service node is created from data contained in the new entity
2: a new entity is created

Figure 1. Dynamic creation of a new service node in the model tree when a new entity is created.

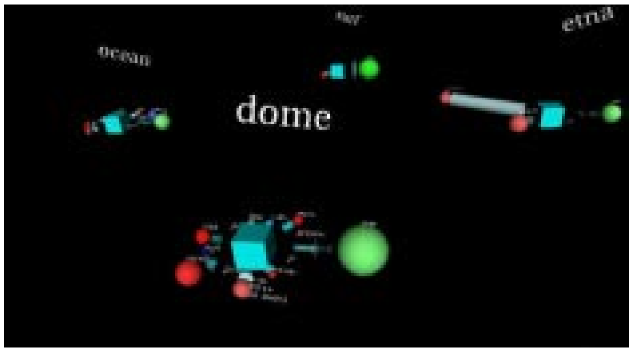


Figure 2. Solar system metaphor.

of the data model. The way a service node acts is the following: the service node issues a persistent query to the entity repository, the repository returns what we call a *relation* which groups references to all the entities that satisfy the query. The service node adds that relation as one of its children, hence constructing a tree-like structure. Figure 1 depicts this mechanism.

2.2 Slaving the visualization to the dynamic data model

Using the previous mechanisms, the data model is always reflecting a hierarchical structure based on the existing entities. This model must be translated into a 3D world. This translation is called the *mapping process* and will be briefly discussed in Subsection 3.3. The visualization is slaved to the dynamic data model and the system will continuously update the world in order to present an up-to-date visualization of the data model. We take advantage of the hierarchical structure of the data model that matches quite naturally the hierarchical representations of 3D scenes generally supported by most 3D-scene description libraries.

3. The visualization model

Our visual structure is based on the concept of 3D visual metaphors. We have designed real-world based metaphors such as a solar system (Figure 2) or a cityscape (Figure 6) metaphor, that allow the user to better and more rapidly comprehend the visualizations, since their underlying structure is already familiar. For example, it is straightforward that houses in the same district share some common properties. Examples of those and other metaphors can be found at CyberNet's project webpage [2]. It is important to note that a metaphor is not designed for a specific service. In fact, a service can be visualized using any metaphor provided it offers sufficient visual and structural capabilities for displaying the service information.

3.1 Graphical and metaphoric components

A 3D metaphoric world is a set of *metaphoric components* (MCs) organized hierarchically. Metaphoric components are *graphical components* with interaction and navigation add-on capabilities. Each graphical component has visual parameters that can be used to visualize information. CyberNet supports two classes of graphical components: *3D glyphs and layout managers*.

- *3D Glyphs* (3DGs) are 3D objects that may represent data through their visual parameters (e.g., color, size, and texture). The level of complexity of a 3DG can be related to the number of visual parameters it offers for modification and, hence, to the dimension of data that can be displayed.

- *Layout Managers* (LMs) are responsible for one of the most important visual choices: the use of space. They organize their children – either 3DGs or other LMs – in space according to a built-in policy and may also animate these positions along paths. For instance, a orbit LM organizes elements in orbit around a center, and a chess LM on a plane in rows/columns.

Using these two kinds of graphical components, we can build a 3D-scene hierarchy. The result is a tree where internal nodes are LMs and leaves are 3DGs.

To create a new visual metaphor, one needs to define the set of MCs that illustrates the metaphor. For example, a solar system metaphor (Figure 2) is composed of star, satellite, and planet glyphs, and orbital layout managers. The more GCs are used in a metaphor, the more numerous are the ways of visualizing the data.

3.2 Building and updating the worlds

There are two different operations, *creation* and *update*, which can be used to update the metaphoric world. Creation, as the name implies, involves creating one or more new graphical components – glyph or layout manager – and update involves bringing up to date the visual parameters of one or more GC (e.g., updating the size of a box to reflect the new value of the mapped data). LMs can receive orders to add new children (3DGs or LMs) to themselves and any GC can update itself upon receiving up-to-date data.

Before sending data that will update the world we need to know which kind of graphical components must be created from the data model. This is the role of the mapping process.

3.3 Mapping the data model visually

Mapping is the process that automatically constructs a 3D metaphoric world from the information contained in a service tree. The result is a graphical hierarchy where internal nodes are layout managers and leaves are 3D glyphs. In the CyberNet system, special objects called *adaptors* handle the mapping. These adaptors are dependent on the type of metaphor used for displaying the information. Basically, there are two types of visual mapping: hierarchical and visual parameters mapping.

Each metaphor is defined as hierarchy of MCs – i.e., a city metaphor (Figure 6) is based on a metaphoric component called City, which contains Districts, Districts MCs contain Streets and Buildings, Buildings contains Floors, and so on. We call *hierarchical mapping* the process of defining which service element (entity or relation) will be visualized using which MC. For instance, a workstation supervision service using a solar system metaphor like in Figure 2, the model mapping rules states that computers=stars, users=planets, and processes=satellites.

The main idea behind the mapping process is to define a set of association rules for each service, based on the type of each service element and its position in the service tree. In particular, since an entity may be part of several relations (thus being located at more than one position in the service model tree), it may have several visual counterparts in the presentation domain. So far, the mapping process is mostly hard-coded but we have been developing an automatic mapping process based on data and visual parameters characterization. Some prior work in those fields has already been done, namely by [5], [6], and [7].

Besides mapping the structural elements onto the graphical components, the attributes of each entity must also be translated into visual information. This is the purpose of the *visual parameters mapping*. Each MC has a number of visual parameters that may be dynamically modified in order to display information (e.g., position, orientation, size, color, etc). How we map the data values on these visual parameters (e.g., CPU percentage on color, memory on size, and so on) is the responsibility of the visual parameters mapping. When defining the mapping rules, care must be taken to preserve metaphor coherency. For example, if one rule uses the color for identification purposes, other rules cannot use this visual parameter to represent a data value.

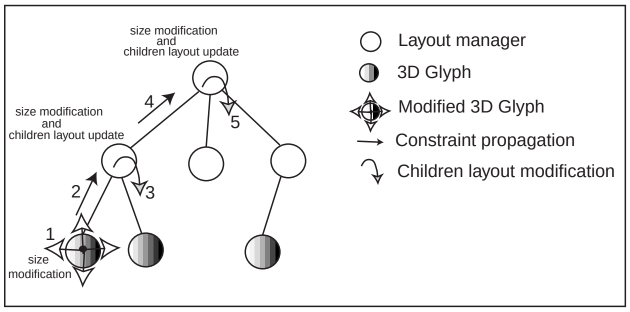

size modification and children layout update
size modification and children layout update
size modification
1
2
3
4
5
Layout manager
3D Glyph
Modified 3D Glyph
Constraint propagation
Children layout modification

Figure 3. Bottom-up constraint propagation.

4. Impact of data dynamics on the visualization model

While the system automatically modifies the 3D world, it must take care of global coherency in the world. For example, when an object is growing in size, the system must verify that its geometry does not intersect other objects in the world. If it detects a potential intersection, the system should be able to modify the neighboring objects (for instance it can slightly push them apart in order to make some room for the growing object). Thus, the modification of one object may have impact on its neighborhood, and the modified neighborhood may require the modification of other objects. This phenomenon is called modification propagation. Several types of causes lead to this propagation, namely: the size of a child is modified, a new child is created or an existing child is removed.

Managing the modification propagation is an important topic directly related to dynamical layout management. The traditional way to handle static layout management is to use global layout constrained optimization algorithms. However, because of performance problems, dynamic management requires avoiding centralized solutions since global optimization algorithms cannot be re-evaluated each time a small modification arises in the world. We chose to develop a distributed propagation mechanism: in our system each LM has the responsibility of modifying itself and its direct children layout. There are two basic strategies for propagating the modifications: *bottom-up* and *top-down*.

4.1 Bottom-Up constraints propagation

The first solution to handle constraint propagation is to use a bottom-up approach. The main idea is to let the children influence the upper part of the hierarchy. For instance, if the size of a 3DG increases, the space allocated by its parent LM to the 3DG should be modified accordingly. Thus, the parent LM should also modify its size and its own parent must take into account this modification. The modifications are propagated to the upper part of the hierarchy until the root (Figure 3). An important consequence of the use of

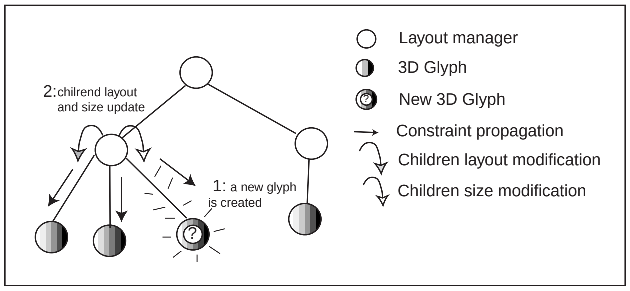

Layout manager
3D Glyph
New 3D Glyph
Constraint propagation
Children layout modification
Children size modification
2: chilrend layout and size update
1: a new glyph is created

Figure 4. Top-Down constraint propagation.

this propagation mode is that the overall world structure may be modified as soon as a leaf node is modified. This is due to the fact that when a child modifies its size, the information is transmitted to its father that in most cases has to modify the position of all its children according to its layout policy. In many cases, this global propagation mechanism can produce globally unstable 3D worlds.

4.2 Top-Down constraints propagation

The second solution to handle constraint propagation is to use a top-down approach. The advantage of this approach is to limit the influence of modifications. In this case, it is the role of the father to allocate space for its children and to control that they actually fit in the region defined. Since it is the father that determines the size of its children, the propagation of the space constraints is going from the root to the leaves (Figure 4). Each child should be able to fit itself into the region of space that its parent has reserved for it. When a child is not able to do so, it is the role of its parent to modify its size in order to fit the child to the region. The main problem of this propagation mechanism is that the world loses size coherency: since it is the parent that defines the space allocated to its children, two conceptually identical leaf nodes may differ in size and thus cannot be compared by the user. Another related problem is that when a part of the world is growing in number of elements, each element is generally decreasing in size in order to keep the aggregate size constant. These local modifications (as opposed to the global modifications generated by the previous strategy) can produce local instability in the 3D worlds.

4.3 Directional constraints

In the presentation of the two previous constraint strategies, we did not make any references to the constraints themselves or to the scheme used to define the regions of space. Our basic statement is that the volume managed by a father should include all the volumes managed by (or allocated to) its children. How these volumes are defined is a matter of implementation. However, why should the propagation be identical according to every direction in space? A good

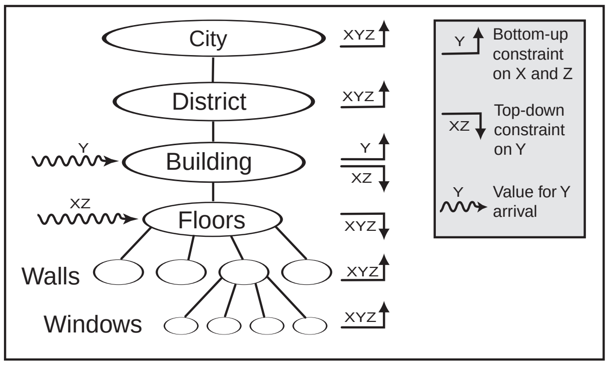


Figure 5. Constraints in a city metaphor.

example of non-uniform propagation can be found in the city metaphor implemented in the CyberNet project. In this metaphor, a district is defined as a LM that places buildings in space. This could be implemented either by using a top-down or a bottom-up propagation strategy. The top-down strategy will fix the size of districts (once created a district will not be able to grow in size, and adding new buildings will be done by resizing down existing ones). The bottom-up strategy will conserve building size and allow district to grow (when a district grows, the neighboring districts will be pushed apart in order to make room). In the first case, even if the districts are limited in the ground plane, there is no reason to limit the buildings in height. Hence, the district LM is designed to apply a top-down constraint to the buildings in the ground plane and leave freedom to the height direction by using a bottom-up strategy for that direction (Figure 5).

In other words, buildings have a defined ground size (X and Z), but they are free to grow in height (Y) according to the number and size of the floors. Thus, each floor is constrained to have the same (X and Z) size as the building. The walls are constrained in each direction because they should exactly match the floor size. They are composed of windows and the number of windows is used to represent some information. Because the walls size is constrained and since the number of windows may be variable, the size of the windows is not relevant (i.e., cannot be used to map information) although their relative size at a given floor may be.

The design of complex 3D virtual worlds requires being able to use both constraint propagation strategies within the same metaphor hierarchy, as we have seen above. By mixing both constraint strategies, several propagation configurations may arise – Table 1 depicts the different potential configurations.

4.4 The city example

An example of the implementation of the city metaphor is given in Figure 6. This metaphor was used to represent a

Parent Children	Top-Down propagation	Bottom-Up propagation
Top-Down propagation	The father allocates space for its children – the children respect the allocated size and constrain their own children.	The children decide their own size (they constrain their own children) – the father allocates space accordingly.
Bottom-Up propagation	The father allocates space for its children – the children decide their own size. The potential conflict is solved by the parent using a scale operation to constrain the size of its children.	The children decide their own size (they compute their size from their own children) – the father allocates space accordingly.

Table 1. Mixing propagation

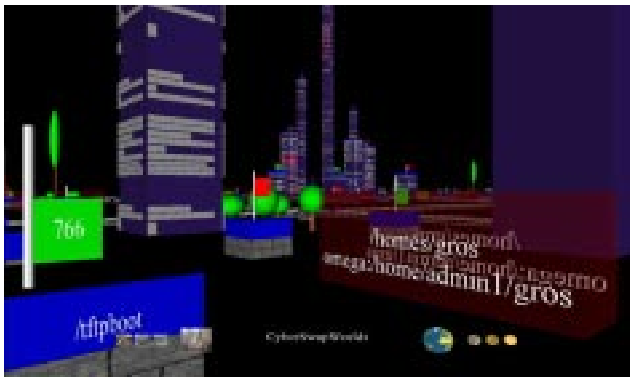


Figure 6. CyberNet Project: City metaphor. NFS data visualization tool.

NFS client/server configuration of about 60 computers. In this metaphoric visualization example, cities are subnets, districts are workstations, buildings are disks, and floors are remotely access disks (each time a client workstation remotely mounts the disk, a new floor is added to the building). Additionally, windows are file-handles – each time a file on that server disk is opened on the client workstation, a new window appears on the floor that represents the client workstation.

4.5 Movement animation

Movements of objects in the world must be presented in such a way that they do not confuse the user. A well-known technique to try overcoming this problem is to animate the movement of objects in the visual representation in order to make the visual representation evolve in time in a way that is not disruptive to the user [8].

The evolution in time is usually slowed down to make the changes in the world appear as a smooth transition. In other words, objects do not appear/disappear or are moved around simultaneously. There is a four step algorithm for a transition from a current state to a following state. First, all deleted objects are removed. Second, remaining objects are moved to their new position. Third, all resize operations are done. Finally, new objects are added to the world. This animation is also used to move the user as the world evolves, as we will see in next section.

5. Navigation and interaction

Navigation and interaction allow the user to explore and better comprehend the information being displayed. Navigation gives the user the possibility to travel among the data and thus explore it more directly. Interaction allows the user to interact with the world being displayed in order to change the visualization to better meet his needs.

In this section we will tackle some of the consequences of data dynamics regarding the navigation in and the interaction with the metaphoric world. However, as it is out of scope of this paper, we will not delve into a detailed description of CyberNet's navigation and interaction mechanisms. For further information on these subjects, particularly on our navigation system, please refer to [9].

5.1 Navigating in a dynamic world

In the CyberNet visualization framework, all metaphoric worlds are constructed using metaphoric components (MCs) (Section 3.). MCs have mechanisms that help the user to navigate logically inside the metaphor. The goal is that the user navigates in the world with the mechanism most suited to the metaphor itself. We call this principle *metaphor-aware navigation* [9].

In order to assist the user in its navigation task, the system maintains three parameters regarding the user: the *user's current object* in the 3D world (the closest object to the user – the user is always associated with a MC), his *current object of interest* (the node that currently has his attention) and a *target object of interest* (where the navigation mechanism is requested to take the user to). When the user

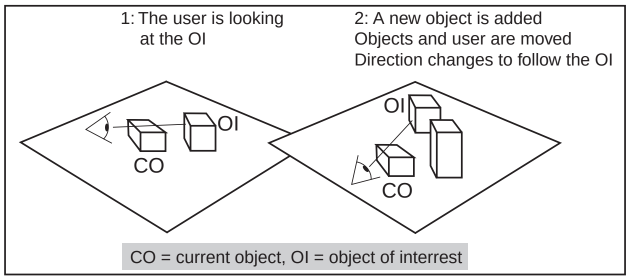

1: The user is looking at the OI
2: A new object is added
Objects and user are moved
Direction changes to follow the OI
OI
CO
OI
CO
CO = current object, OI = object of interrest

Figure 7. Automatic movement of the user when objects are moving.

is simply moving around in the world, the user's current object (current location) and his current object of interest are identical. But some navigation tasks require the user to be located in a place and to have his center of interest on another object (an example is the *look at* navigation mode).

The difficult part is to know what is the movement that the system must apply to the user when the world evolves. If the current object and the object of interest are identical, the simplest solution is to move the user move in the same way that his current object is moving. This strategy produces good results but may not be sufficient when the current object and the object of interest differ, since the movement of the object of interest and the movement of the current object may also differ. In this case, for instance for the *look at* navigation mode, the direction the user is looking at must also be changed towards the new position of the object of interest (Figure 7). There are still occlusion problems that may occur, for instance, when the new object that is created occludes the current object of interest. The solution is to move the user to a new location from where the occlusion does not apply.

5.2 Interaction and selection

In addition to the built-in navigation capabilities, the metaphoric components also provide some interaction features. We found it important to be able to interact with the world in order to modify its appearance. However, the dynamic nature of the world implies that the current appearance parameters must be saved in order to be later applied to the new objects that may be created. If newly created objects would not inherit those parameters, the world would lose its coherency.

One of most common ways of interacting with a visualization is the selection mechanism. The user may select objects to cluster, to render transparent or to isolate from other objects in the world, are just a few examples. As the world evolves the selection also evolves. For instance, if the user selects all the elements of a district in a city metaphor, each new building added to this district must be automatically selected.

6. Conclusion

In this paper, we have seen that building 3D visualizations of highly dynamic information leads to some requirements that do not exist in the case of static information. The data model must be able to evolve in time and these evolutions must have repercussions on the visualization model. Strategies are applied to the visualization model to minimize the effect of objects movement inside the 3D world so that the user is not disoriented by the modifications. To our knowledge, there is no ideal solution to overcome the problem of visualizing a world that evolves, but a well designed metaphor with special focus on its layout management constraints helps the visualization to be more stable, thus more comprehensible.

References

[1] Ben Delaney. The NYSE's 3D trading floor. *IEEE Computer Graphics and Applications*, 19(6):12–15, November/December 1999.

[2] CyberNet Project's Webpage. http://www.eurecom.fr/˜abel/cybernet.

[3] S. Card, J. Mackinlay, and B. Shneiderman. Readings in information visualization: Using vision to think, 1999.

[4] P. Abel, P. Gros, C. Russo Dos Santos, D. Loisel, and J.-P. Paris. Automatic construction of dynamic 3d metaphoric worlds: An application to network management. *Visual Data Exploration and Analysis VII*, volume 3960, pages 312–323. SPIE, 2000.

[5] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, April 1986.

[6] Michelle X. Zhou and Steven K. Feiner. Data characterization for automatically visualizing heterogeneous information. In *Proceedings IEEE Symposium on Information Visualization*, pages 13–20. IEEE, 1996.

[7] Lisa Tweedie. Characterizing interactive externalizations. In *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 of *PAPERS: Information Structures*, pages 375–382, 1997.

[8] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: Animated 3D visualizations of hierarchical information. *Proc. ACM Conf. Human Factors in Computing Systems, CHI*, pages 189–194. ACM Press, 28 April–2 May 1991.

[9] C. Russo Dos Santos, P. Gros, P. Abel, D. Loisel, N. Trichaud, and J.-P. Paris. Metaphor-aware 3d navigation. *Proceedings of the IEEE Symposium on Information Visualization,2000 – InfoVis2000*, pages 155 – 165. IEEE, October 2000.