In *Proc. IUI '97 (1997 Int. Conf. on Intelligent User Interfaces)* pages 129-136, Orlando, FL, January 6–9, 1997

Top-Down Hierarchical Planning of Coherent Visual Discourse

Michelle X. Zhou **Steven K. Feiner**
Department of Computer Science
Columbia University
500 West 120th St., 450 CS Building
New York, NY 10027
+1 212 939 7000
{zhou, feiner}@cs.columbia.edu

ABSTRACT
A visual discourse is a series of connected visual displays. A coherent visual discourse requires smooth transitions between displays, consistent design within and across displays, and successful integration of new information into existing displays. We present an approach for automatically designing a coherent visual discourse. A top-down, hierarchical-decomposition partial-order planner is used to efficiently plan the visual discourse. Visual representations are modelled as visual objects, graphical techniques are employed as planning operators, and design policies are encoded as constraints. This approach not only improves the computational efficiency compared to search-based approaches, but also facilitates knowledge encoding, and ensures global coherency.

Keywords: Top-down hierarchical planning, automated graphics generation, knowledge-based user interfaces.

INTRODUCTION
Much of the research done in the area of automated generation of graphics has been focused on designing individual presentations (e.g., [11, 16]) or planning sequences of static presentations (e.g., [5, 18]). In contrast, designing coherent *visual discourse* has not received the full attention that it deserves. We use the term *visual discourse* to refer to a series of connected visual displays. *Coherent* visual discourse must have smooth transitions between displays, consistent design within and across displays, and visually effective integration of new information into existing displays. Cognitive psychology studies have demonstrated that incoherent visual discourse can greatly impair user performance. The user might be "lost" in a display network [23], confused due to inconsistent design [12], or overwhelmed by information overload caused by a lack of efficient information integration [15].

We present a top-down hierarchical planning approach for automatically designing coherent visual discourse. To achieve coherency, we introduce three effectiveness criteria: *continuity*, *consistency*, and *unity*. *Continuity* guarantees smooth spatial or temporal transitions, *consistency* ensures a consistent design within and across displays, and *unity* addresses how to unify individual components into a coherent whole [14].

In our approach, the design of a visual discourse is a sequence of visual actions. A *visual action* is an encoded graphic design technique (e.g., move an object). We use a hierarchical planner [4] to generate the sequence of actions. Compared to search-based approaches, hierarchical planning achieves computational efficiency by reducing the amount of search needed [24]. It also eases the task of knowledge encoding [20] by reusing the actions that are common to many design tasks. In addition, we adopt a top-down hierarchical decomposition strategy [25] to facilitate the interleaving of planning and execution [20] and to ensure both local and global coherency within a visual discourse.

First, the system sketches a visual discourse design that is complete but too vague to be realized; then it refines the vague parts of the design into more detailed subdesigns until the design has been refined to a sequence of primitive visual actions. During this design process, the design constraints are incrementally specified at each level, and the constraints must be satisfied before the design can be further refined. In contrast, in a bottom-up approach (e.g., [11]), partial designs are generated at the primitive level, and then are composed together to form composite designs. Without considering the design constraints at a higher-level, the partial designs are prone to failure and usually require redesign. In the worst case, satisfying all local design constraints can not even guarantee the satisfaction of the global design constraints.

Following a brief discussion of related work, we present our top-down hierarchical planning approach. We identify various types of visual objects, and then we introduce visual actions that are used as planning operators. Next, we focus on several key design stages to explain how our approach works. Our approach has been used in a computer network management domain to automatically generate visual presentations for different network management tasks. Two major tasks are exploring internal structures of various network entities (e.g., network nodes and links), and monitoring network traffic status (e.g., traffic load on a particular link). We elaborate the design process using an example from the network domain. Finally, we present our conclusions and discuss future improvements.

RELATED WORK

Friedell [6] presents a system for automatically synthesizing graphical objects from object descriptions. He employs a top-down approach in which complex objects are decomposed into simpler subobjects. Structural relations are used to relate objects and their subobjects and specify how the subobjects are oriented with respect to each other. APT [11] and SAGE [16] generate 2D visual presentations of quantitative information. Both systems adopt a bottom-up strategy in which partial designs corresponding to various data relations are planned first; then all partial designs are composed together to construct a complete design based on design constraints. All three systems focus on designing individual presentations. They do not address either design consistency in consecutive displays, or how to integrate new designs into existing ones.

APEX [5] can automatically create pictorial explanations to depict actions in a 3D world. APEX generates a series of pictures to depict a sequence of actions. Both IBIS [18] and WIP [1] use a generate-and-test approach to design a sequence of static visual displays for procedural explanations. They all maintain a certain degree of consistency for related images (e.g., using a consistent view specification when appropriate). They also achieve continuity by arranging the images in a logical order (e.g., the order in which depicted procedures are to be performed). However, they do not explicitly address transition issues from one picture to another in their generation. Moreover, the generate-and-test approach often results in a large search space [2].

ESPLANADE [8] employs a hierarchical planning approach to automatically generate animated presentations using film-making heuristics. It ensures smooth frame-to-frame transitions by exploiting standard film-making techniques such as dissolve, fade in, and fade out. Like IBIS and WIP, it uses realistic images of physical objects and generates presentations with little user interaction. Thus, it does not deal with consistency issues at the symbol encoding level (e.g., mapping object attributes to visual variables) and unity issues such as integrating new information into the current scenario.

APPROACH

In a planning approach, problem-solving operators are applied to various objects to accomplish planning tasks. In the context of visual design, the objects refer to both domain and visual objects. Visual actions are graphical techniques that are used as planning operators to design or manipulate visual objects. First, we introduce visual objects and visual actions. Then we illustrate how the objects and actions are utilized in the design process.

Visual Objects

There are five types of visual objects: visual primitives, visual unities, visual structures, visual frames, and visual discourse. These form a hierarchy, as shown in Figure 1.

Visual Hierarchy

At the bottom of the hierarchy are *visual primitives,* which are individual visual variables: shape, material, texture, size, position, and orientation [3].

Next come *visual unities*, which are constructed from visual primitives and other visual unities. For example, a red plastic cube is a visual unity that is formed by two visual variables (cuboid shape and red plastic material).

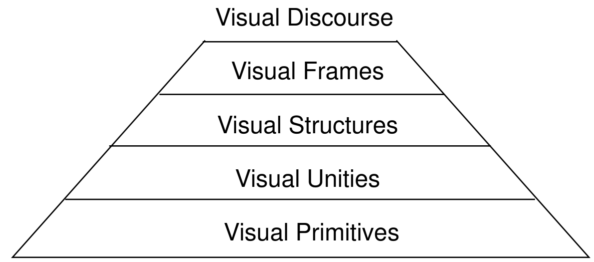

Figure 1. Visual hierarchy

Above the visual unities are *visual structures*. Visual structures spatially organize various visual unities and other visual structures in a specific way to convey abstract concepts or explain how processes work [21]. For example, a 2D line graph that plots height against age is composed of line segments and two orthogonal axes with textual labels. This plot indicates how height changes with the age. Conventional visual structural classification used in cognitive psychology studies [22] identifies three basic structures that fall between text and realistic images: charts, graphs, and diagrams. Based on the visual representation classification proposed by Lohse [10], we have extended our visual structure classification. For example, graphs are further partitioned into scatter plots, bar graphs, etc. A visual structure could also be a cartogram (i.e., a spatial map overlaid with visual objects, such as dot maps that use symbols to show the location of individual objects on the map), which does not quite fit into any of the three basic categories. The visual structures that can be automatically designed by our system are shown in Figure 2.

Visual frames contain single or multiple visual structures. For example, a visual frame representing a computer network's topological structure and its traffic statistics over a time interval could include a cartogram (network nodes and links overlaid on a geographical map) and a textual table chart. The cartogram represents the topological structure, and the table chart conveys the traffic information.

Visual discourse is a cohesive formation of a series of visual frames. Suppose we need to show network traffic statistics over a long time period that is divided into several shorter time intervals. One visual frame can display traffic statistics for one short time interval, while all visual frames are connected to form a visual discourse that shows the network traffic statistics over a long time period.

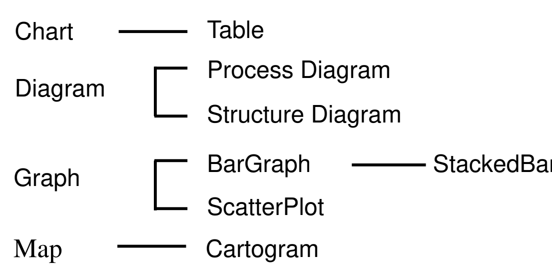

Figure 2. Implemented visual structures

Syntax and Semantics

To automatically compose visual discourse, we must specify both syntax and semantics for visual objects [13]. Syntax specifies how a visual object should be constructed. For example, the syntax for constructing a table chart must indicate the spatial relations among the rows and columns, and how the rows and columns are formed. In contrast, semantics defines the meaning of an object's syntactic features. For example, each division of a pie chart specifies a *part-whole* relationship between the object represented by the division and the object represented by the whole pie.

In the process, syntax is used as both layout constraints (e.g., all elements within a table column must be vertically aligned) and pattern constraints (e.g., a 2D bar graph uses the same line width for both axes) to synthesize a visual object. Semantics, on the other hand, is employed to ensure syntactically valid visual objects are also conceptually correct [9], based on domain information characteristics [27]. Figure 3 is a cartogram that represents a network's topological structure. In this example, semantic constraints make sure that similar objects have similar visual appearance—spheres and cones represent switch and router nodes, respectively, while thin cylinders depict links. In a top-down approach, the task of designing a visual object is refined to designing its components. For example, designing a bar graph is refined to designing its bars, axes and textual labels. Here, the syntax of the visual object is used to limit the possible combinations of its lower-level components. For example, the syntax for a 2D bar graph specifies that the two axes must have similar visual appearance. If the line color and width for one axis is decided, then the same visual properties will be applied to the other. Thus, there is no need to design the visual properties for two axes separately.

Visual Actions

In our system, we encode graphical techniques as visual actions that are used as planning operators.

Categorization

Visual actions are categorized based on their functionality. There are three types of visual actions: *formational actions*, *transformational actions*, and *camera actions*. *Formational actions* design visual objects from scratch by mapping the information to the visual objects. In contrast, *transformational actions* modify the existing visual objects

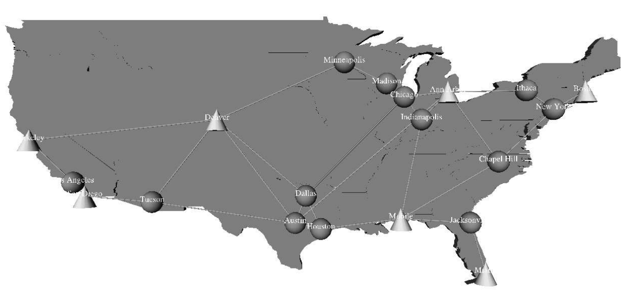

Figure 3. A computer network

in two ways: *appearance transformations* (e.g., *Transfigure* changes an object's material) and *geometry transformations* (e.g., *Scale* modifies an object's size). While the first two types of actions act upon the domain and visual objects, *camera actions* involve *viewing specification* (e.g., *Set* camera position or orientation) and *viewing modification* (e.g., camera *Zoom*). Table 1 shows the set of *primitive* visual actions currently identified and implemented in our system.

Formation Action	Transformation Action	Camera Action
MapProperty	Align	Dolly
MapGeometry	CutAway	Pan
MkBarGraph	FishEyeView	Rotate
MkCartogram	Highlight	Seek
MkStructDiag	Move	Set
MkTableChart	Open	Zoom
ShowInset	Scale	
ShowMultiView	Transfigure	

Table 1. Primitive visual actions

Representation

Every action is represented in this format:

```
(defclass Action (is-a Operator)
        (operands)
        (preconditions)
        (postconditions)
        (parameters)
        (preference))
```

Each action has a set of *operands* that it acts upon. All *preconditions* must be true before an action can be executed and all *postconditions* become true after the action is executed. Each action is also associated with a set of *parameters*. For example, *Scale* has the parameter *scale factor*, while *Move* has parameters such as *source* and *destination*. Preference indicates how preferable the action is in the current design context. How the preference of an action is determined will be described in the following section.

A visual action is either *primitive* or *composite*. Primitive visual actions, such as *Enlarge* (i.e., *Scale* action with *scale factor* greater than *1*), can be executed by the rendering component directly. In contrast, composite actions are represented by a set of partial plans, which will eventually be decomposed into primitive actions. Each partial plan is represented using a decomposition schema [25]. Hence, each composite action is associated with a set of decomposition schemata. A decomposition schema specifies subactions that are used to accomplish the composite action, and partial orders and causal relations among the subactions. A composite action is represented as:

```
(defclass CompositeAction (is-a Action)
        (operands)
        (preconditions)
        (postconditions)
        (parameters)
        (decompositionSchemata)
        (decompositionPreferences)
```

```
(currentDecomposition)
(preference) )
```

In addition to the slots specified in a primitive action, a composite action has three extra slots: *decompositionSchemata*, *decompositionPreferences* (discussed later), and *currentDecomposition*. For example, the composite operator *RevealInternal* has two decomposition schemata as shown in Figure 4.

In *DecompositionSchema1, RevealInternal* could be decomposed into three subactions: *Focus* on ?objX, *DesignVisObject* (see next section), and *Open* ?objX (to expose its internal structures). Moreover, the *partialOrders* limit the order of the three actions to be applied: both *Focus* and *DesignVisObject* must precede *Open*. Similarly, *DecompositionSchema2* states that the action *RevealInternal* can also be achieved by another combination of three actions. In this decomposition, *ShowInset* replaces *Open* to display ?objX and its internals in an inset window. We will discuss how a decomposition schema is selected later.

There are two distinct advantages to having both primitive and composite actions. First, composite actions can be used by the hierarchical planner at a high level to sketch an abstract design without worrying about computationally overwhelming details. For example, at a high level, the design task of “comparing the traffic load of two network nodes” could be solved by a very abstract plan that includes two composite actions: *Focus* on the two nodes and *Show* the traffic load within these two nodes. At this level, the plan is not concerned with how to focus on the nodes or how to show the traffic load. A more detailed plan will be constructed later through decomposition. Second, only a small set of primitive actions need to be implemented in the rendering component. Primitive actions can be reused in a wide range of combinations to specify different composite actions. For example, *Enlarge* is used as one of the subactions for the composite action *Focus* to achieve the focusing effect by enlarging the object. The same action is also used in the composite action *Distinguish* to distinguish an object by increasing its size.

```
(DecompositionSchema1
    (subactions
        (Subaction1 (Focus ?objX))
        (Subaction2 (DesignVisObject (internal ?objX)))
        (Subaction3 (Open ?objX)))
    (partialOrders
        (< Subaction1 Subaction3)
        (< Subaction2 Subaction3)))

 (DecompositionSchema2
    (subactions
        (Subaction1 (Focus ?objX))
        (Subaction2 (DesignVisObject (internal ?objX)))
        (Subaction3 (ShowInset (internal ?objX) ?objX)))
   (partialOrders
        (< Subaction1 Subaction3)
        (< Subaction2 Subaction3)))

(RevealInternal (is-a Action)
      (operand ?objX)
      (decompositionSchemata
            DecompositionSchema1
            DecompositionSchema2)
      (. . .))
```

Figure 4. Decomposition of action RevealInternal

Effect

Visual actions are also used to achieve continuity. In a visual discourse, maintaining continuity is interpreted as providing smooth transitions between individual displays.

Smooth transitions are achieved by animated visual actions. Animated visual actions are actions with temporal constraints:

```
(defclass AnimatedAction (is-a Action)
        (operands)
        (. . .)
        (startTime)
        (stopTime))
```

A temporal constraint is represented as a time interval that is specified by its start and stop times.

A *geometric transition* is a smooth transformation from one geometric configuration to another. It is achieved by animated geometric transformational actions (e.g., *Move* or *Align*). These actions interpolate between the old and new geometric values (e.g., position) over time to achieve a smooth transition. A *visual appearance transition*, on the other hand, is a smooth transformation from one appearance to another (e.g., gradually making an object transparent). It is worth pointing out that the interpolation usually is not linear. We use conventional animation techniques such as slow-in, and slow-out to achieve appealing animation effects.

In a visual discourse, the current display might be replaced by a completely different display. To maintain aesthetic continuity [26], we adopt conventional film transition techniques such as dissolve, fade in, and fade out (as used in [8]) to achieve smooth transitions between displays.

In the following section, we explain in detail how a design task is accomplished using the visual hierarchy and visual actions.

Design Stages

The input to our system is a design task. The hierarchical planner starts to plan based on the current input. A planning cycle oscillates among four steps: selecting an action to accomplish a task, decomposing a composite action, resolving conflicts, and checking whether the planning process is finished. Here we focus on two key steps: action selection and action decomposition. These two steps together determine how a design is constructed and how the search space is pruned.

Action Selection

One of the major steps in planning is to find an action that can accomplish the current task. The planner compares the design task with the postconditions of visual actions. If one of an action’s postconditions matches the task, then the action becomes a candidate to accomplish the current task. A candidate action could be either a brand new action that has never been instantiated before, or a used action that was instantiated earlier for accomplishing another task. Usually, there is more than one candidate action. Nondeterministically choosing one of them might result in later planning failure. Therefore, several heuristic preference constraints

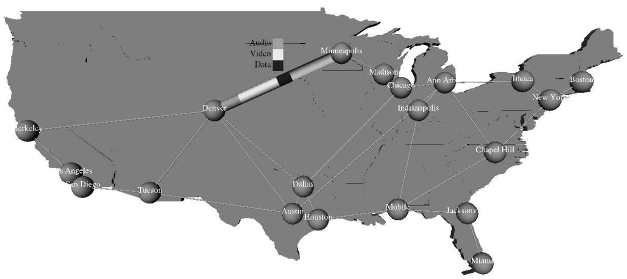

Figure 5. Traffic status in opened physical link

are employed to rank the candidate actions. The higher the action is ranked, the more preferable it is.

Currently five types of preference constraints are used in our system: *achievability*, *consistency*, *unity*, *sharing,* and *availability.*

Achievability ranks the actions based on the fact that some actions will lead to a more effective design than others for a particular task. For example, *DesignLineGraph* is ranked higher than *DesignBarGraph* if the task is to compare different sets of numerical data emphasizing a trend over time [7]. Achievability is specified in the action's postconditions. The following representation shows that the *DesignLineGraph* has higher rank than the *DesignBarGraph* when the postcondition is *ShowTrend*:

```
(DesignLineGraph (is-a CompositeAction)
    (. . .)
    (postconditions (postcondition1 (ShowTrend (rank 10)))) )

(DesignBarGraph (is-a CompositeAction)
    (. . .)
    (postconditions (postcondition1 (ShowTrend (rank 5)))) )
```

Consistency asserts that if an action was used before to accomplish a similar task in a similar situation, then it is preferable. The consistency preference not only applies to the action selection process, but also affects action instantiation. For example, assume that the current user dialogue is shown in Figure 5, which uses colors to encode different types of network traffic in a link. Suppose the user wants to examine the traffic in a different link. Unless there is a good reason to do otherwise, not only should the color encoding action be used, but the action should also be instantiated to use the same color code. To accomplish this, we use a history list to record the dialogue's tasks and solutions.

Unity states that if an action leads to a design that can be better integrated with the existing design than other actions, then this action is preferable. Figure 5 shows a network with traffic load information in a link. The stacked bars inside the link represent the traffic load. A regular bar graph can also be used to display the traffic information. However, in this example, the design task is to show the link's traffic status without comparing different links. Therefore, the oriented stacked bar graph conveys the needed information and can be better integrated with the selected link than a bar graph. Thus, *DesignStackBar* is preferred over *DesignBarGraph*.

Within the same planning process and concerned with the same object(s), *sharing* states that if one of several competing actions is *used* and others are not, then the used action is more preferable since it can be shared by the previous and current task [17]. For example, if the current subtask is *Distinguish objectX*, both *Enlarge objectX* and *Highlight objectX* are among the choices. But suppose *Enlarge objectX* is instantiated earlier in the same planning process to accomplish another subtask *Identify objectX*. In that case, *Enlarge* is a better candidate than the unused action *Highlight* since it can be shared by both subtasks *Distinguish objectX* and *Identify objectX*.

Availability specifies that an action is more preferable if it has more satisfied preconditions than another action. For example, *Enlarge* and *Highlight* could be both used to distinguish an object. If two of the preconditions of *Enlarge* are already satisfied, and none of the preconditions in *Highlight* has been satisfied yet, then *Enlarge* is preferred to *Highlight*. At one extreme, under an availability preference, actions without any preconditions are always preferable to actions that require preconditions.

Preference constraints not only improve performance by avoiding backtracking, but they also ensure the satisfaction of certain effectiveness criteria (e.g., consistency, unity). Moreover, preferences themselves can be ranked based on their importance when multiple preferences can be applied at the same time. The current ranking in a descending order is: achievability, consistency, unity, sharing and availability. Our ranking is based on the fact that ensuring the selection of an effective design (achievability) is most important. Next come consistency and unity criteria to support overall cohesiveness. Finally, sharing and availability preference complement the design process as performance tuning factors (e.g., availability will favor actions with fewer unsatisfied preconditions). The preference slot will hold the computed ranking based on preference constraints.

Once an action is selected, it will be processed. If it is a composite action, it will be decomposed using one of its decomposition schemata. If it is a primitive action, all its preconditions are checked to see whether they are satisfied. Next, we focus on action decomposition.

Action Decomposition

A composite action must be decomposed into subactions before a complete plan can be constructed. Just as arbitrarily choosing an action might result in later plan failure, randomly selecting a decomposition schema in action decomposition could also cause plan failure. Hence, we attach a set of decomposition preference constraints to a composite action to guide the decomposition. We use two types of preference constraints: *achievability* and *consistency.* Similar to the achievability preference in action selection, *achievability* states that under *conditionX*, a decomposition schema is preferred since it leads to a more effective design. *Consistency* preference aims to maintain design consistency by favoring a decomposition schema that was used in previous dialogues.

The two types of preferences themselves can also be ordered. Our system gives priority to the achievability preference. Recall that *RevealInternal* has two decomposition schemata (Figure 4). Its decomposition preferences are specified as:

```
(RevealInternal (is-a Action)
    (operand ?objX)
    (. . .)
    (decompositionSchemata
        DecompositionSchema1
        DecompositionSchema2)
    (decompositionPreferences
        (preference1
            (condition (and
                (HighIntersection ?objX)
                (SmallBoundingBoxSize ?objX))
            (preferred DecompositionSchema2))
        (default
            (preferred DecompositionSchema1))))
```

Preference1 asserts that if the operand ?objX is likely to intersect with other objects and it has a small 2D bounding box (the size of the inset window created by *ShowInset* is proportional to the bounding box), then *DecompositionSchema2* should be used. Here, "small" is interpreted to mean that either dimension of the bounding box does not span more than 1/3 of the full display window. In a network application, some nodes are very likely to intersect with other nodes nearby especially when they get enlarged. In this case, the second decomposition schema should be selected.

In our top-down approach, an abstract design is sketched at a high level using composite visual actions. Thus, action decomposition can also be considered as design refinement. Through action decomposition, a rough design is refined. Moreover, design consistency is ensured through the refinement. Suppose we want to design a network diagram to represent a computer network. At a high level, action *DesignNetworkDiagram* is selected. It is specified as:

```
(DesignNetworkDiagram (is-a CompositeAction)
        (operand DiagramX)
        (. . .))
```

DiagramX is an instantiation of the network diagram class:

```
(DiagramX (is-a NetworkDiagram)
        (nodes)
        (links)
        (. . .))
```

At this stage, the representations for nodes and links are still empty. Later, *DesignNetworkDiagram* is decomposed into a set of *DesignVisObject* actions for nodes and links:

```
(DesignVisObject (is-a CompositeAction)
        (operands nodeX)
        (. . .))

(DesignVisObject (is-a CompositeAction)
        (operands linkX)
        (. . .))
```

Both *DesignVisObject* actions design visual representations for each node and link. Suppose one material should be consistently applied to all nodes. Then each node will have a similar representation:

```
(NodeX (is-a Node)
        (id)
        (. . .)
        (representation NodeRep))

(NodeRep
        (is-a VisualObject)
        (material ?NODE_MATERIAL)
        (. . .))
```

At this point, DiagramX is refined accordingly:

```
(DiagramX (is-a NetworkDiagram)
        (nodes Node1Rep Node2Rep . . .)
        (links)
        (. . .)
        (variable-list (?NODE_MATERIAL (binding NONE)))
```

Here, the variable ?NODE_MATERIAL serves as a global variable in the network diagram design. If ?NODE_MATERIAL in one of the node is instantiated later to some actual material value, then all nodes will have the same material through the variable binding. The ability to use variables is an important feature in a planner [20]. Using variables in abstract visual presentations ensures consistent design within a display, and also defers some detailed decisions at the high level to avoid costly redesigns. In this example, deciding which color to use is really not important at this level—the link representation has not been designed yet and the system knows nothing about the background color either. An arbitrary early decision will likely lead to a failure. For example, if the system arbitrarily assigns the color white for the nodes, it might realize later that white is inappropriate due to a white background.

Conflict Resolution

Before a plan can be further refined, all conflicts at this level must be resolved. Conflict resolution involves detecting potential conflicts (e.g., one action's postcondition might deny another action's precondition), formulating new constraints to avoid potential conflicts, and ensuring that the constraints are likely to be satisfiable. Currently, we use a straightforward approach to solve simple constraints: we exhaustively check each constraint in the constraint network. A more sophisticated constraint solver could be used to replace the current one in the future.

Completion Checking

This step checks if a plan is finished. A plan is finished if and only if:

1. All preconditions for every action are satisfied,
2. there is no composite action in the plan, and
3. there is no conflict in the plan.

Any precondition that has not yet been satisfied becomes a subtask that needs to be achieved before a complete plan can be formed.

EXAMPLE

A network link contains a set of *virtual path segments*. Each segment has attributes such as capacity and utilization. To reveal the internals of a link is to display all virtual path segments inside the link. In addition, those segments are distinguished by their capacities. In this section, we describe how a visual discourse is planned to fulfill the task of revealing the internals of a selected link. This is a much simplified version of the actual planning process, which fires 210 rules.

The user has selected link23 (the link between nodes Austin and Tucson). The system formulates the task as:

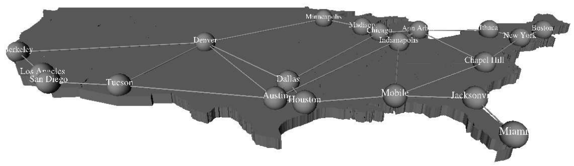

Figure 6. Aligned network representation

(Task (ShowInternal link23))

The planner seeks actions that have a postcondition that matches the task description. In this case, the planner selects the composite action *RevealInternal*. Next, *RevealInternal* is decomposed using one of its decomposition schemata. In our example, the selected link's bounding box is too big to satisfy the preference condition for *Decomposition-Schema2*. Therefore, the default decomposition schema is used. Thus, *RevealInternal* is decomposed into three subactions: *Focus* (focus on link23), *DesignVisObject* (design visual representation for internal objects of link23), and *Open* (open link23).

Among the three subactions, only action *Open* is a primitive action; the other two are composite actions and need to be further decomposed. *Focus* is decomposed into *Extract* (extract link23 from the rest of the network). This decision is made based on the fact that a link is likely to intersect with other objects and *Extract* separates intersected objects or prevents the potential intersections while achieving focusing. Focusing by *Enlarge* increases the intersection possibility, and focusing by *Highlight* does not fix or prevent any intersection.

DesignVisObject is refined into two actions: *MapProperty* and *MapGeometry*. The former is concerned with the visual appearance of the internal objects, while the latter takes care of the geometric features (e.g., position and size). At this point in planning, let us assume all preconditions but one are satisfied and no conflicts need to be resolved. The

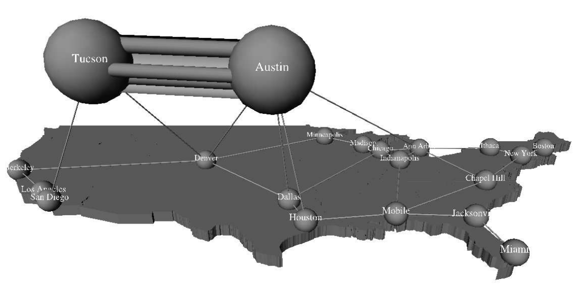

Figure 7. Opened link and its internals

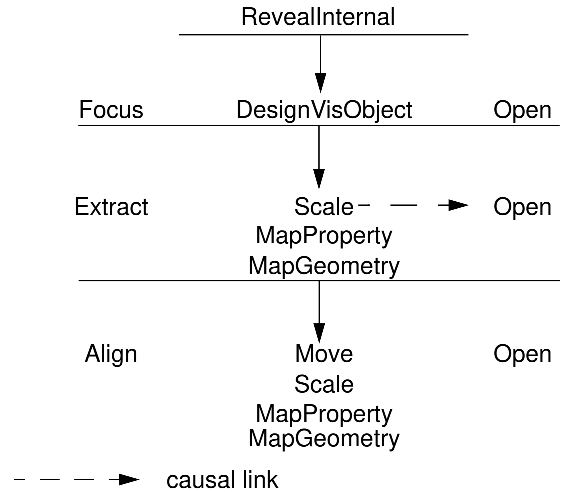

Figure 8. Planning levels

only precondition that is not true at this stage comes from the action *Open*. One precondition for *Open* requires that the object be large enough so that the opening effect is recognizable. *Scale* is selected to establish the causal link for this precondition of *Open*.

At this stage, *Extract* is the only non-primitive action. It is then replaced by *Align* and *Move*. *Align* rotates the whole network so it is laid down (Figure 6), while *Move* moves the link and its two end nodes away from the rest of the network (Figure 7).

We have built a *visual lexicon* that maps domain objects to *visual words*. Each visual word gives a partial or complete visual description for a particular type of object. Moreover, one type of object might correspond to more than one visual word (e.g., a human can be mapped to a stick figure or a realistic image of a person). In the network domain, one visual word specifies the shape (cylinder) for all connection objects. In this example, the system knows that the internal objects within a link are one type of connection. Thus, the shape for the internal objects does not need to be designed. The visual lexicon provides a foundation on which more complex designs can be built so that it is not always necessary to design everything from scratch.

Figure 8 shows all planning levels. Each level is a complete plan, and is a refined version of the level above it. At each level, each action precedes the action to its right (e.g., *Focus* precedes *DesignVisObject*), while all actions in the same column are not ordered (e.g., *Move, Scale*, *MapProperty*, and *MapGeometry* at the last level). In the final plan, a complete sequence of actions is not necessarily a linear sequence. For example, actions *Move* and *Scale* are executed simultaneously in this example.

IMPLEMENTATION

Our system is written in C++ and the CLIPS production system language [19]. We have implemented the hierarchical-decomposition partial-order planner using CLIPS, based in part on the research of [20, 25]. We use the SGI Open Inventor 3D graphics toolkit to implement the rendering component. It takes approximately 2.8 seconds to plan the whole design for the example mentioned in the previous section, running on a 250 MHZ R4400 SGI Indigo2 Maximum Impact.

CONCLUSIONS AND FUTURE WORK

We are building a system that can automatically generate a connected sequence of visual displays to form a coherent visual discourse. In addition to the effectiveness criteria used for designing individual presentations, we introduce three effectiveness criteria to ensure visual cohesiveness: continuity, consistency, and unity.

We use a top-down hierarchical planning design approach. This approach is computationally efficient compared to a search-based or non-hierarchical planning approach [24]. It eases knowledge-encoding through reuse of primitive actions. More importantly, its top-down design assures global coherence, and facilitates interleaving of planning and execution [20].

Currently, we are working on several ways to improve the current system so that it can automatically design timely, coherent visual discourse in an interactive environment. The unity criterion is still fuzzy, and it needs to be refined so that it is computationally measurable (currently it is specified as heuristic rules). We are also working on interruptible visual discourse. This is essential in an interactive environment in which the user may want to stop or resume the ongoing discourse at any time. In addition, we are studying how to interleave planning and execution to reduce response time, which is important in an interactive environment.

ACKNOWLEDGMENTS

The research described in this overview is supported in part by DARPA Contract DAAL01-94-K-0119, the Columbia University Center for Advanced Technology in High Performance Computing and Communications in Healthcare (funded by the New York State Science and Technology Foundation), the Columbia Center for Telecommunications Research under NSF Grant ECD-88-11111, and ONR Contract N00014-94-1-0564.

REFERENCES

[1] E. Andre, W. Finkler, W. Graf, T. Rist, and A. Schauder. The design of illustrated documents as a planning task. In M. Maybury, editor, *Intelligent Multimedia Interfaces*, chapter 4, pages 94–116. AAAI Press/The MIT Press, Menlo Park, CA, 1993.

[2] A. Barr and E. Feigenbaum, editors. *The Handbook of Artificial Intelligence*, volume 1, chapter II. Search. Addison-Wesley, Reading, MA, 1982.

[3] J. Bertin. *Semiology of Graphics*. University of Wisconsin Press, Madison, WI, 1983. (translated by W.J. Berg).

[4] P. Cohen and E. Feigenbaum, editors. *The Handbook of Artificial Intelligence*, volume 3, chapter XV. Planning and Problem Solving. Addison-Wesley, Reading, MA, 1982.

[5] S. Feiner. APEX: An experiment in the automated creation of pictorial explanations. *IEEE Computer Graphics and Applications*, 5(11):29–37, 1985.

[6] M. Friedell. Automatic synethsis of graphical object description. *Computer Graphics: ACM SIGGRAPH'84*, 18(3):53–62, July 1984.

[7] E. Goldsmith. *Research Into Illustration: An Approach and A Review*. Cambridge University Press, Cambridge, 1984.

[8] P. Karp and S. Feiner. Automated presentation planning of animation using task decomposition with heuristic reasoning. In *Proceedings of Graphics Interface '93*, pages 118–127, 1993.

[9] S. Kosslyn. Understanding charts and graphs. *Applied Cognitive Psychology*, 3:185–226, 1989.

[10] G. Lohse, K. Biolsi, and H. Rueter. A classification of visual representations. *Communications of the ACM*, 37(12):36–49, 1994.

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

[12] J. Marks. Discourse coherence and consistent design for informational graphics. In *Working Notes from the AAAI Workshop on Intelligent Multimedia Interfaces, Ninth NCAI*, 1991.

[13] J. Marks. A formal specification scheme for network diagrams that facilitates automated design. *Journal of Visual Languages and Computing*, 2(4):395–414, 1991.

[14] K. Mullet and D. Sano. *Designing Visual Interfaces*. SunSoft Press, Mountain View, CA, 1995.

[15] K. Norman, L. Weldon, and B. Shneiderman. Cognitive layouts of windows and multiple screens for user interfaces. *International Journal of Man-Machine Studies*, 25:229–248, 1986.

[16] S. F. Roth and J. Mattis. Automating the presentation of information. In *Proceedings of the IEEE Conference on AI Applications*, pages 90–97, 1991.

[17] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ 07632, 1995.

[18] D. Seligmann and S. Feiner. Automated generation of intent-based 3D illustrations. *Computer Graphics*, 25(4):123–132, July 1991.

[19] Software Technology Branch, Lyndon B. Johnson Space Center. *CLIPS Reference Manual*, June 1993. CLIPS Version 6.0, JSC-25012.

[20] D. Wilkins. *Practical Planning: Extending Classical AI Paradigm*. Morgan Kaufmann, San Mateo, CA, 1988.

[21] W. Winn. Charts, graphs, and diagrams in educational materials. In D. Willows and H. Houghton, editors, *The Psychology of Illustration: Basic Research*, volume 1, chapter 5, pages 152–198. Springer-Verlag, New York, 1987.

[22] W. Winn and W. Holliday. Design principles for diagrams and charts. In D. Jonassen, editor, *The Technology of Text*, volume 1, pages 277–299. Educational Technology Publications, Englewood Cliffs, NJ, 1982.

[23] D. Woods. Visual momentum: A concept to improve the cognitive coupling of person and computer. *International Journal of Man-Machine Studies*, 21:229–244, 1984.

[24] Q. Yang and J. Tenenberg. Abtweak: Abstracting a nonlinear, least commitment planner. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 204–209. AAAI, 1990.

[25] R. Young, M. Pollack, and J. Moore. Decomposition and causality in partial-order planning. In *2nd Int. Conf. on AI Planning Systems: AIPS-94*, pages 188–193. Chicago, IL, June 1994.

[26] H. Zettl. *Sight Sound Motion: Applied Media Aesthetics*. Wadsworth Publishing Company, Belmont, CA, second edition, 1990.

[27] M. Zhou and S. Feiner. Data characterization for automatically visualizing heterogeneous information. In *Proceedings of InfoVis'96, IEEE Symp. on Information Visualization*, San Francisco, October 1996.