Automated Visual Discourse Synthesis: Coherence, Versatility, and Interactivity

Ph.D. Thesis Proposal

Michelle X. Zhou

Department of Computer Science
Columbia University
New York, NY 10027
zhou@cs.columbia.edu
http://www.cs.columbia.edu/~zhou

Technical Report
CUCS-031-97

Abstract

In this proposal, we present comprehensive and systematic approaches of building systems that can automatically generate coherent visual discourse for interactive envirornments. A visual discourse refers to a series of connected visual displays. A coherent visual discourse requires smooth transitions between displays, consistent designs within and among displays, and effective integration of various components. Our research focuses in part on establishing a general framework by abstracting various generation systems and providing a reference model in which a specific system is considered an instantiation of the framework. In other words, any automated graphics generation system must contain a *knowledge base*, an *inference engine*, a *visual realizer* and an *interaction handler.* As a consequence, not only can a general framework serve as a template from which a specific generation system can be instantiated, but the framework also can be used as a base for comparing or evaluating different systems.

We concentrate on the basic issues involved in establishing these four core components. In particular, we identify various knowledge sources and determine effective knowledge representation paradigms in constructing the knowledge base. We emphasize the efficiency, usability, and flexibility issues in modeling the inference engine. We are concerned with portability and parallelization issues in building the visual realizer, and we also take into account interaction capabilities for interactive environments. To demonstrate the generality and comprehensiveness of the framework, we address its application to the design of coherent visual discourse for heterogeneous information in interactive environments. Within such discussions, heterogeneous information refers to both quantitative and qualitative, or static and dynamic information. In addition, we assume that the system aims to support a wide variety of visual techniques, ranging from individual 2D displays to interactive 3D animation sequences.

We describe a system called IMPROVISE (Illustrative Metaphor Production in Reactive Object-oriented VISual Environments) that serves as a proof-of-concept prototype. IMPROVISE is built based on our framework, aiming to automatically generate coherent visual discourse for various application domains in interactive environments. IMPROVISE has been used in two testbed application domains to demonstrate its generality and flexibility. Examples from both domains will be given to illustrate IMPROVISE's generation process and to identify the future research areas.

Contents

CHAPTER 1 Introduction

One picture is worth ten thousand words. — F. R. Barnard

For many years, graphic forms or visual presentations have been used to make information more understandable by humans. However, as the complexity and volume of the information grows, our ability to hand-craft customized visual presentation lags far behind our ability to process the information. A new generation of computational techniques and tools is required to automate the visualization of useful information. These techniques and tools are the subject of the emerging field of automated information visualization or knowledge-based information visualization.

During the past two or three decades, most presentations created by computers were carefully designed and constructed by hand. Systems like the TeX formatting system, give the experienced user very precise control of the layout, but overload the casual user. Considering that not everyone has had training in graphic design and assuming the sole purpose of creating visual presentations is to aid information understanding, a reasonable compromise was found in the creation of style sheets (or template-based presentations). While the majority of the layout styles, color schemes, or fonts were carefully chosen ahead of time by trained professional layout artists, and the content was supplied by the user. However, when the scale of data manipulation, exploration, and inference grows beyond human capacities, this template-based strategy is no longer feasible to cope with a great deal of information in a timely, customized manner. Therefore, researchers are now investigating how to use computer technology to automate the process of designing visual presentations.

Within the past ten years, automated generation of visual presentations has become an active research topic. Systems such as APEX [Feiner, 1985], APT [Mackinlay, 1986], SAGE [Roth and Mattis, 1991], and IBIS [Seligmann, 1993] have been developed to demonstrate the significance and feasibility of some of the research ideas. However, much of the research done in the area of automated graphics generation has been focused on designing individual presentations (e.g., [Mackinlay, 1986; Roth and Mattis, 1991]), or planning sequences of predominantly static presentations (e.g., [Feiner, 1985; Seligmann, 1993]). 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. To remain coherent, a visual discourse must ensure smooth transitions between displays, maintain consistent designs within and among displays, and achieve effective visual unifications among various components.

Complementing the work for automated generation of individual presentations or sequences of static presentations, our research focuses on establishing a comprehensive and systematic paradigm for automatically generating coherent visual discourse. In our work, we propose a general framework for building such systems. We suggest that the framework consist of four essential components: *knowledge base*, *inference engine*, *visual realizer*, and *interaction handler*. More importantly, we emphasize how to *assemble* a working system based on the instantiation of each individual component. Furthermore, a working system IMPROVISE (Illustrative Metaphor Productions in Reactive Object-oriented VISual Environments) has been implemented to demonstrate the generality and feasibility of the established framework.

1.1 Problems

There are two main research issues in building automated generation systems for coherent visual discourse: understanding the cause of incoherence and searching for solutions of ensuring coherence.

Incoherence within visual discourse could arise in several situations. While physical displays are not large enough to accommodate all information at one time, information has to be presented in successive displays. Lack of smooth transitions between displays or inconsistent designs within or among displays will result in incoherent visual discourse. Another problem with incoherent visual presentations is caused by ineffective integration of various information into a coherent whole. Ineffective integration not only results in visual clutter or visual noise, but it also degrades the true value of using visual representations as an information interpretation tool that aids the user to perform tasks.

Visual presentations express information in their own specific channel. We often employ visual presentations to express *and* communicate information to the user and guide him/her to perform tasks. Cognitive psychology studies have demonstrated that incoherent visual discourse can greatly impair user performance. The user might be “lost” in a display network [Woods, 1984], confused due to inconsistent design [Marks, 1991b], or overwhelmed by information overload caused by a lack of efficient information integration [Norman et al., 1986].

Different measurements have been proposed to relate human performance to the effectiveness of visual displays. As a result, guidelines can be formulated to lead effective visual presentation designs. Woods [Woods, 1984] uses *visual momentum* to measure the user's ability to extract and integrate information across displays. Similarly, another measurement *visual scope* is used by Norman et al. [Norman et al., 1986] to evaluate the user's ability to integrate information across a display of multiple windows or screens and grasp the whole of whatever is being presented. While the *visual momentum* is created by providing smooth transitions between displays, *visual scope* is established by matching the surface layout (i.e., visual presentation structures) against the user's cognitive layout. Intuitively, the problem of ensuring visual discourse coherence becomes the problems of identifying the user's cognitive layout for a particular task, and investigating the mapping between a visual representation and such layout. As we search for computational approaches to these problems, we find ourselves entangled in four research areas: *knowledge engineering*, *inference modeling*, *visual realization*, and *interaction handling*.

First, to automatically design visual presentations, a system needs to have certain knowledge that it can reason about. The knowledge base should contain information such as general visual design principles, the specific domain model, and the expected user or situation (e.g., display device, and time constraints) context. In addition, the knowledge must be represented and organized in a form such that it can be efficiently accessed or modified. Thus, *knowledge engineering*, namely, determining knowledge sources and their representation formalism becomes our first problem.

Second, automated graphics generation is a computationally complex task. Nevertheless, the time takes to search and infer a design becomes the computation bottleneck. To make a system applicable, an efficient inference model is needed to reduce the system's response time as it enhances the system's usability. To investigate and develop an efficient inference approach—*inference modeling*, becomes the key in building a practical automated graphic design component.

Third, converting a design into visual displays on the screen is not a trivial problem either. We use the term *visual realization* to refer to the process that actually translates the planned design into human perceivable visual displays on the screen. During this translation process, several issues are involved: specifying the graphic design, translating the design into the target graphics language, and performing the rendering. As automated graphics generation system is computationally costly, the upgradability of an automated generation system becomes one of the important issues in system design. In particular, such systems should be made easily portable and be parallelized to take full advantage of parallel processing and networking technologies. Such portability and parallelism require the design and realization components to be disentangled and independent of each other. Therefore, the capabilities of *visual realization* reaches far beyond the point where it is concerned only with rendering routines and their performance. *Visual realization* needs to deal with issues that interface the two major processes (i.e., design and realization) to enforce the desired system portability and parallelism, and concerns with providing necessary information needed by different components (e.g., design process and rendering process might require information to be expressed in different format).

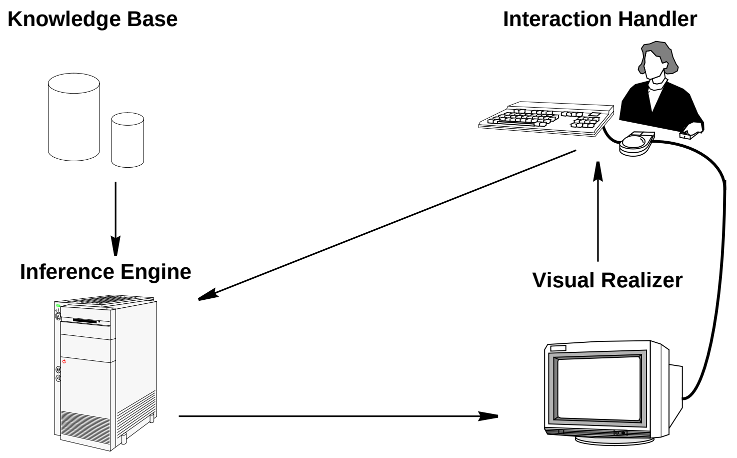


Figure 1-1 Framework for automated graphics generation systems

Finally, as a system is targeted for an interactive environment, allowing certain user interactions becomes one of the desired features. Determining what types of interaction should be allowed and how to handle various user interaction events become the topics of *interaction handling.*

So far, we have introduced four problems/tasks that need to be taken care of before an automated graphics generation system could be assembled. The crux of the matter is that solving the four tasks results in four structural components of automated graphics generation systems. They are knowledge base, inference engine, visual realizer, and interaction handler. Therefore, a general framework for building automated graphics generation systems can be established by making an abstraction of each of these four components. Then each practical system, which is made up of instantiations of each of the four components, would be considered as a specific instance of the framework.

1.2 Approach

The conceptual model for building a system that automatically generates visual discourse is shown in Figure 1-1, which illustrates the flow of a complete presentation generation process. The *inference engine* infers a visual design based on the information stored in the *knowledge base.* Upon completion of the design process, the design is carried out by a *visual realizer.* In the course of the presentation, an interaction handler processes user events. Next, we discuss each of these four components and how to assemble a system that includes them.

Knowledge Base. An automated graphics generation system generates visual representations based on certain information it knows about. How efficient and effective the generation would be depends upon what types of knowledge the system has and how efficiently the knowledge could be utilized. Thus, there are two issues involved in constructing a knowledge base: identifying knowledge sources and determining representation mechanisms. There are at least four types of knowledge required in the process of visual design: domain knowledge, visual design knowledge, situation knowledge, and meta knowledge.

A generation system not only needs to possess certain knowledge, but it should also be able to reason about it based on what it knows. Due to the diverse nature of the knowledge that we are dealing with, our work focuses on finding a combination of different knowledge representation formalisms to facilitate knowledge acquiring, retrieving, and reasoning. The mixed representation paradigm should meet three criteria [Barr and Feigenbaum, 1989]: *understandability*, *modularity*, and *extensibility*.

Understandability refers to the ability to acquire or store knowledge in a form that is natural and straightforward to humans, who serve as the source of knowledge. *Modularity* implies the ability to manipulate a piece of data independently of the rest of the knowledge base. *Extensibility* indicates the easiness to expand the current knowledge base and accommodate new types of information. Based on these three criteria, we employ an object-oriented representation formalism as our primary knowledge representation paradigm, supplemented by procedural and production rule representations.

Inference Engine. To accomplish visual tasks, a graphics generation system should be able to reason about what it knows about, make decisions and infer the visual presentation design. An efficient inference engine inherently employs an effective problem solving method to guide the reasoning process. There are several factors that must be taken into account in choosing or developing such an inference engine.

Efficiency is inarguably one of the most important factors. A generation system should be able to generate effective presentations within a reasonable time frame. *Usability* indicates that whether an inference model is suitable for a particular domain, in our case, the domain of designing visual presentations. No matter how efficient the inference process is, if it can not provide what the application expects, the model is useless for this particular application. *Flexibility* is another factor that needs to be taken into account as it measures how easily the model could be modified or improved to address new tasks or new platforms. Based on these criteria and the analysis of our problem domain, we have decided that a constructive planning approach would be adequate for our application.

In our approach, a visual discourse is a sequence of visual actions. A *visual action* is an encoded graphic design technique (e.g., moving an object). We use a hierarchical planner [Cohen and Feigenbaum, 1989] to generate the sequence of actions. Compared to search-based approaches, hierarchical planning achieves computational efficiency by reducing the amount of search needed [Yang and Tenenberg, 1990]. It also eases the task of knowledge encoding [Wilkins, 1988] by reusing the actions that are common to many design tasks. In addition, we adopt a top-down hierarchical decomposition strategy [Young et al., 1994] to facilitate the interleaving of planning and execution [Wilkins, 1988] and to ensure both local and global coherence within a visual discourse.

Visual Realizer. A complete graphics generation cycle includes planning a design and realizing it. In order to be realized, the design must be represented in a form that can either be understood by a rendering package directly, or be easily translated into the rendering language. It is nonetheless desirable to separate the process of design from the process of realization. In other words, both the design and realization processes should not be affected by the other's representation or functionality. In addition, the separation will contribute towards the desired system portability or parallelism. Bearing these criteria in mind, we decide that an *intermediate language* should be developed to represent the design, and this design should be easily translated into the language understood by the graphics package. To switch to a different graphics package, we only need to write another translator.

Determining which graphics package to use is another challenge task. Ideally, the package should be platform independent. In addition, we often want to deal with graphics objects at a high level of abstraction, but retain the right to access the lowest level primitive functions when necessary. Fortunately, architecture-independent graphics packages have become available, including both high-level object-oriented specifications (e.g., OpenInventor [Wernecke, 1994]) and low-level primitive manipulation functions (e.g., OpenGL [Neider et al., 1993]).

Interaction Handler. At any given time, the user wishes to have control over what s/he is viewing. Thus user interaction is desired in an information visualization system. In our case, the user should be able to explore the information as s/he wishes, or to stop the ongoing visual presentation at any given time and resume it as s/he prefers. Thus, an interactively inviting and interruptible visual discourse should be produced.

While designing interactively attractive visual presentations involves analyzing various interaction metaphors and interaction styles, providing an interruptible visual discourse requires reactive planning [Wilkins et al., 1994]. In order to resume, the system must maintain a presentation execution history to record when the interruption occurred, what actions have been performed and what actions are yet to be carried out. Furthermore, the system replans based on the result of the current user interaction. As the ability to be able to replan in an uncertain environment (e.g., when and what the user is going to do is unknown to the system) is a whole research topic of reactive planning.

Next, we outline our major contributions.

1.3 Contributions

This thesis establishes a framework which captures comprehensive and systematic *approaches* to building systems that automatically design coherent interactive visual discourse for heterogeneous information. To demonstrate the feasibility and generality of these approaches, we have also designed and implemented a working prototype *system* called IMPROVISE that can automatically generate coherent visual discourse in two application domains. From these approaches and the implemented system, we highlight the following contributions*:*

I. Development of approaches for generating *coherent* visual discourse

- ❑ To ensure visual discourse coherence, we have established a set of *expressiveness* and *effectiveness* criteria [Mackinlay, 1986]. Based on these criteria, a coherent visual discourse is expressive enough to allow a user to attend selectively to the local information and perceive all information cohesively as a whole. It will also be effective enough to ensure smooth transitions between displays, maintain design consistency within and among displays, and unify various components into an organic whole.

- To ensure coherence, we design an inferencing process based in part on hierarchical-decomposition partial-order planning algorithms. We incorporate heuristic strategies (e.g., achievability and consistency preferences) into our inferencing process. In this inferencing paradigm, visual discourse is represented as action sequences; and visual design techniques are employed as problem-solving operators. Guided by heuristic strategies and a top-down refinement strategy, the generation process not only ensures the global coherence of the visual discourse, but also improves the design efficiency.

II. Development of approaches for *versatile* visual discourse

- To represent heterogeneous information, we have established a six-dimensional data characterization taxonomy to relate the information characteristics to different visual properties. Based on the relationships between the data properties and the visual attributes, we are able to develop mapping methods that comprehensively and systematically map the data onto different visual objects.
- To represent heterogeneous information, we would like to employ a wide range of visual forms (e.g., graphs, charts, and animations) effectively. We establish a visual hierarchy to categorize various visual representation forms and define their composition principles at different levels of abstraction using their syntax, semantics and pragmatics. In particular, we have developed a visual lexicon, which is a compact and comprehensive representation formalism, to effectively capture the relationships between atomic domain objects and their different visual representations.
- To carry out different visual tasks effectively, we have also developed a visual task hierarchy. Based on this hierarchy, we analyze the characteristics of each type of task by their visual accomplishments and visual implications. As a result, we are able to establish mappings between the high-level presentation intents and the visual tasks, and between the visual tasks and the underlying visual techniques.

III. Development of approaches for generating *interactive* visual discourse

- To construct interactive visual discourse, we incorporate many conventional user-interface interaction metaphors (e.g., buttons and menus) and styles into visual presentation design so that the visual presentations present certain visual affordances to encourage user interaction.
- We are also exploring possible reactive planning approaches so that we can present an interruptible visual discourse. Moreover, such a discourse could also properly respond to a user by modifying the current presentation plan.

IV. Design and Implementation of working system IMPROVISE

- IMPROVISE automatically generates coherent visual discourse to represent heterogeneous information using a variety of visual techniques and visual media, ranging from static 2D graphs to 3D animation sequences.
- IMPROVISE composes interactive visual presentations that enable users to interact with and control the visual discourse.

- ❏ IMPROVISE employs a sophisticated planner (PREVISE) to efficiently create visual presentations and guarantees both local and global cohesiveness as a result of a top-down, gradual refinement design process [Zhou and Feiner, 1997b].
- ❏ IMPROVISE generates visual presentations for two different domains: a computer network management application and a health care application, which demonstrates the modifiability and the extensibility of the system.

1.4 Organization

The rest of the proposal will be organized as follows: Chapter 2 discusses related work in the area of automated graphics generation. Chapter 3 briefly illustrates the framework and its components. Chapter 4 describes IMPROVISE and its components in detail. Finally, Chapter 5 concludes with a discussion of remaining work and our schedule.

CHAPTER 2 Related Work

Discovery consists of seeing what everybody has seen and thinking what nobody has thought. — Albert von Szent-Györgyi

As a growing research area, automated graphics generation has been taking advantage of progressively more sophisticated computational techniques. Researchers have explored various computational paradigms, aiming to establish a comprehensive and systematic treatment of the problem. Among the various paradigms, most of them fall into one of two categories: *parametric graphics synthesis* and *constructive graphics synthesis.*

Parametric graphics synthesis predefines a set of visualization models that are characterized by functionalities and represented by sets of parameters. The whole process involves analyzing the data attributes, matching the data against the visualization models, and instantiating the visual parameters of the selected model to interpret the data. Conversely, *constructive graphics synthesis* is a deductive approach. It reasons about the logical operations that could be used to achieve the high-level communicative intent, and attempts to carry out these logical operations using primitive visual objects. These primitive visual objects serve as building blocks, with which complex visual displays can be constructed. Unlike *parametric graphics synthes*is, the constructive approach designs every visual presentation from scratch by gluing together the most basic visual variables [Bertin, 1983] to form a coherent whole. Based on these two approaches, this chapter enumerates some of the representative research work in automated graphics generation, which are most closely related or most influential to our work.

2.1 Parametric Graphics Synthesis

Parametric graphics synthesis is based on the theory that comprehensive visual model(s) can be built to reflect the user's mental model and effectively convey information to the user [Robertson, 1991]. Moreover, through a careful examination of the visual attributes of the models and the data properties, the mapping between the visual attributes of the models and the data properties can be established in a formal and systematic manner. Most systems that employ this approach are largely identical but with minor differences in how to map their data onto the defined visual models. Nevertheless, these systems can be differentiated by the richness and flexibility of their visual models, as well as their capabilities to combine primitive models into a composite one. In this section, we present several representative systems in a chronological order.

AIPS. As early as in 1981, Zdybel et al. [Zdybel et al., 1981] developed a system that generates visual displays for information in a knowledge base. Templates are first constructed to describe different types of visual displays. Each template includes information such as visual attributes and positional information. The user can also define templates and specify their usage. Generation process is a two-step process that maps information to an appropriate template, and instantiates the template. To establish the mapping, AIPS specifies certain information such as data properties, template properties and their semantics. AIPS automatically matches the information to be presented against a set of templates and selects the most appropriate one. The selected template is instantiated based on its syntax and the information to be presented.

The major drawback of such a system is that as the information type varies, the template types must be augmented to accommodate this new type of information.

Natural Scene Paradigm. Robertson's natural scene paradigm (NSP) [Robertson, 1991] is one of the most sophisticated parametric graphics synthesis systems. Robertson claims that a user can easily interprete the data through the 3D surface structure by glancing the scene. He proposes to map data variables to various features of the natural scene such as surface condition, surface height, and density based on the characteristics of the data. To direct such mapping, he conducts research on three aspects: analyzing the natural properties of various data and interpretation goals of such data, developing a catalogue that includes the capability of each visual variable that can be used in the natural scene, and establishing the mappings between the visual capabilities of the natural scene and the interpretation goals. Here we briefly describe the significance and results in each of these three areas.

Robertson analyzes various data along its dimensionality and types (e.g., ordinal or nominal). He also distinguishes three types of data interpretation intents as *values at point*, *local distribution of values*, and *global distribution of values*. To map data properties to a natural scene surface, Robertson presents a table that identifies the representation capabilities of various natural scene properties based on the data dimensionality and type. For example, the surface height can be used to represent 2D, ordinal, continuous points. Finally, a mapping to correlate the interpretation intents and the representation capabilities of the natural scene is established to generate the graphics.

NSP does not address how to represent non-quantitative information since it focuses on representing scientific or statistical data sets. In contrast, we attempt to find a more comprehensive and systematic treatment for representing heterogeneous data sets, including both quantitative and qualitative information. Moreover, under our general framework, NSP can be considered as an instance of a specific visual design paradigm for particular presentation tasks (e.g., representing quantitative multivariate information).

VISTA. VISTA [Senay and Ignatius, 1994] is a knowledge-based visualization system that suggests various visual techniques for a given data set and allows the user to modify the design interactively. Compared to the systems described above, VISTA offers a much richer set of visual techniques that can be used to effectively encode a wide range of scientific data. Furthermore, a more sophisticated compositional design approach is employed to design a composite visual display.

VISTA's knowledge base contains information that is divided into five categories: data characteristics, visualization vocabulary, primitive visualization techniques, compositional rules and visual perception rules. *Data characteristics* describe certain presentation related data properties such as data types, cardinality, and continuity. Since VISTA focuses on visualizing scientific data, quantitative data set is further distinguished into subcategories including *scalar, vector* and *tensor. Visual vocabulary* defines the most basic building blocks in scientific data visualization techniques—marks and their composition. A mark can be categorized based on its three properties: positional, temporary, and retinal. A set of primitive visual techniques is defined so a structured design paradigm can be applied. However, the primitive visualization techniques are limited to scientific visualization techniques.

To accommodate multidimensional data, VISTA defines a set of compositional rules to combine the primitive visualization techniques together effectively. The composition rules specify the conditions under which two visualization techniques could be combined to form a multidimensional display. The supported compositions include *mark composition, union composition, transparency composition, intersection composition*. *Visual perception rules* define the mapping from a data set onto a set of visualization techniques, based on both expressiveness and effectiveness criteria [Mackinlay, 1986].

VISTA uses a three-step bottom-up design approach: partitioning a large data set into simple sets of data, selecting unused primitive visualization techniques to represent each small data set, and assembling all selected primitive visualization techniques together. Decomposition of a data set into subsets is based on functional decomposition in relational database theory. The selection of the primitive visualization technique for each partition is guided by the visual perception rules. The visual properties of each mark is determined after the selection is done. When each partition is mapped to a primitive visualization technique, those primitive visualization techniques are composed together to form a coherent visual display based on the composition rules.

2.2 Constructive Graphics Synthesis

Instead of trying to fit the data to be presented into a set of predefined visual models, *constructive graphics synthesis* is a more flexible, deductive approach. It applies graphical operations to visual variables based on communicative intent or data properties to construct a visual design from scratch. It initiates the design process by analyzing the communicative goals or data properties, and attempts to select and apply a set of graphical operations to a set of visual variables. Eventually, the most basic visual variables are composed together to form a coherent visual display based on compositional rules. Visual presentations constructed based on this approach range from conventional 2D graphs such as bar graphs, line graphs representing quantitative information to 3D animations depicting real-world objects or actions.

To compose visual displays from scratch, not only does the system need to know the particular compositional syntax of the visual techniques, but it also needs to maintain certain criteria to ensure the design quality. Different design criteria have been developed to ensure both visual and perceptual effectiveness. Moreover, various composition methods (e.g., top-down vs. bottom-up) have been employed in the generation process. Due to limited computational capability, earlier graphics generation systems intend to focus on generating 2D static pictures. More recent, generation systems have started to deal with real-time 3D graphics generation, and to incorporate sophisticated dynamic visual cues (e.g., animation). We have chosen several pieces of work that each has its unique features and can represent a whole category of similar systems in its own territory. We characterize those systems by roughly covering the following aspects: composition methods, data properties, visual presentation capabilities, and design criteria.

APT. Mackinlay's APT is one of the earliest automated graphics design systems [Mackinlay, 1986]. APT focuses on using 2D graphs or diagrams to represent relational data. It is also the first system where both expressiveness and effectiveness criteria have been explicitly established and used to guide the automated graphics design. In APT, graphical presentations are viewed as sentences of graphical languages. Expressiveness determines the capability of a graphical language to convey the desired information, while effectiveness signifies the language capability to convey *exactly* the desired information accurately and efficiently. While expressiveness criteria is derived from the language definition, effectiveness criteria is deduced from several factors, such as accuracy ranking and perceptual task ranking of visual variables.

Mackinlay defines a set of primitive graphical languages based on Bertin's visual vocabulary [Bertin, 1983]. Syntactic structures are formulated for five major categories of graphical languages including *position languages*, *retinal-list languages*, *map languages*, *connection languages* and *miscellaneous languages*. Moreover, semantic properties of the primitive languages defines the expressiveness criteria for those languages. Composition principles and composition operators are also specified in APT to unify the primitive languages into a coherent whole. APT uses three types of composition operators: *single-axis composition*, *double-axis composition* and *mark composition*.

APT uses a three-step bottom-up process: partition, selection and composition. Multi-relational data set is first decomposed into simple data sets based on the importance order of the data relations. Then primitive visual techniques are filtered or selected for each partition based on the specified expressiveness and effectiveness criteria. At last, composition operators are applied to compose individual designs into a unified presentation. The composition process determines which composition operator to use and satisfies constraints that are introduced during the composition. Unlike VISTA, APT does not have a set of predefined visual presentation models. Instead, APT defines a set of primitive graphics sentences and compose them together to form higher level, comprehensive visual presentations.

SAGE. Like APT, SAGE [Roth and Mattis, 1991] also generates 2D visual presentations of quantitative information. However, the number or complexity of displays and data types handled by SAGE has gone beyond APT. SAGE can successfully design effective visual displays for extended data sets to accomplish a larger set of tasks. Unlike APT, which designs visual displays completely relying on the data characteristics, information seeking goals are introduced as a factor in SAGE's design process.

Roth and Mattis [Roth and Mattis, 1990] have proposed a systematic data characterization method for quantitative data. They have abstracted presentation related data properties in three categories: features of data objects, properties of data relations among the data objects, and attributes of relationships among relations. In each category, they have carefully analyzed and categorized data properties based on their subtle differences. In addition, they have also developed a vocabulary to characterize information seeking goals that express the user or application's communicative intent.

Similar to APT, SAGE also adopts a bottom-up approach and select the appropriate visual presentation techniques based on both expressiveness and effectiveness criteria. Its design is a three-stage process: selecting a graphical technique, refining the technique, and integrating the primitives to form a coherent design.

Unlike APT or other earlier systems, SAGE first selects a list of desired visual techniques for a particular data set based on the expressiveness criteria; then it orders those candidates by effectiveness criteria and the information seeking goals. The information seeking goal plays an important role in deciding which visual technique to use; for example, to view a trend of car prices, a line graph would be picked over other possible choices. When a visual technique has been chosen, the technique is refined and additional constraints for layout or elements are formulated and propagated to guide the next stage of integration. In the last design stage, SAGE attempts to integrate the individual displays into a single one for all information. A set of synthesis rules are developed to guide the integration. In the process of synthesis, SAGE not only deals with syntactic merging among compatible visual displays, but also considers functional integration; for example, two objects can be merged if they perform the same function and have at least one data set in common.

APT and SAGE are representative systems that can automatically visualize information using 2D displays. They all focus on generating individual displays but do not address issues such as providing transition between displays or integrating new information into existing displays. While much of the work in automated generation is devoted to individual 2D charts or diagrams, a few automated systems, discussed below have been built to depict actions or objects using 2D icons or 3D realistic pictures.

The VIEW System. Friedell [Friedell, 1984] presents a system for automatically synthesizing graphical objects from high-level object specifications. The VIEW System can generate 2D iconic displays to depict information in a conventional database or create 3D realistic scenes to represent real world object descriptions.

The synthesis process in VIEW requires four elements: *object frames*, *synthesis operators*, *synthesis agenda* and *synthesis control*. An *object frame* is a data structure to which the synthesis operators are applied. It includes the basic graphics descriptions and their structural relationships. *Synthesis operators* define how an object could be assembled or modified. The *synthesis agenda* is a collection of sequential tasks that must be performed in the synthesis process. The agenda comprises six steps, which perform a task that either selects or applies a synthesis operator. The application of an operator is described in procedural format. *Synthesis control* oversees the whole process of object synthesis. It is a hierarchical process that operates on three levels: *task selection*, *component selection,* and *operator selection*.

Task selection determines the next task to perform based on the synthesis agenda. Once a task is determined, control is passed to the second level—component selection, which decides which component to process within the current object frame. Once a component is chosen, *operator selection* occurs. The *situation space* is used as the basis for operator selection. The *situation space* is very similar to the *situation information* described in Chapter 1. It stores the knowledge that is pertinent to the synthesis situation such as object information, viewer identity and viewer's task.

In short, the View System synthesizes objects based on the synthesis agenda and guided by the situation space. Friedell uses a top-down approach in which complex objects are decomposed into simpler subobjects. Structural-descriptions are used to relate objects and their subobjects and specify how the subobjects are oriented with respect to each other. Based on the structural descriptions, partial object descriptions are generated for primitive subobjects, and those intermediate descriptions are assembled together bottom-up based on their structural relations to form a complete object description.

APEX. APEX [Feiner, 1985] is one of the earliest systems that automatically generate 3D realistic pictures individually or in series. APEX focuses on automatically creating pictorial explanations to depict actions in a 3D world. APEX can generate an individual picture to signify a single action or a series of pictures to depict a sequence of actions. To effectively depict the actions, APEX makes decisions such as what objects to include, what visual cues should be used and which level of details should be shown. Similar to Friedell's VIEW system, all 3D objects are hierarchically structured as trees of their parts and the interconnections between the parts.

APEX selects objects to be included based on the functional role of the objects and the relationships among the objects (e.g., including an object's parent to provide the context). There are four types of real-world objects: *frame objects*, *landmark objects*, *similar objects* and *supporting objects*. Each type of objects plays an important role in picture synthesis. APEX makes decisions about which objects to include in this order: APEX first determines the *frame objects*, which are the main actors in the resulting pictures and the actions to be depicted are centered around them. Then the system finds a *landmark object*, which becomes a good reference for locating the frame objects. To clearly disambiguate the frame objects from other *similar objects* and provide a base to compare them, APEX also includes those similar objects in the scene. At last, *supporting objects* are included to provide necessary context for the scene. Not only does APEX deal with real-world objects in the synthesis process, but it also makes use of another special type of objects—*meta-objects* (e.g., indication arrows) to depict the dynamic aspects of the actions or refer to the real-world objects. In addition to object inclusion, APEX also reasons about lower-level graphics decisions such as the rendering style and the viewing specifications.

To depict a sequence of actions, APEX utilizes a one-to-one mapping between each action and a picture frame. The system makes decisions about how to customize the sequence of images to avoid unrelated or redundant details. Moreover, to make a series of coherent illustrations, APEX attempts to deal with the continuity issues by taking the previous picture into account while it designs the next one.

IBIS. Like APEX, IBIS [Seligmann, 1993] is an automated graphics generation system that uses 3D realistic pictures to depict actions or procedures. However, IBIS provides a richer set of depiction models. Notably, IBIS can generate composite illustrations that are composed of simple illustrations (e.g., single pictorial display), as well as interactive illustrations that allow certain user interactions such as changing the viewing specification.

The input to IBIS is a set of communicative goals. The knowledge base in IBIS consists of *design rules*, *style rules* and *illustration procedures*. The design process is a two-step deductive process that maps a set of communicative goals to a set of illustration procedures. First, design rules map a set of communicative goals to a set of style strategies. Style strategies specify the visual effects and visual cues for the illustration. Style rules then map the style strategies to illustration procedures. Illustration procedures are lower-level descriptions of graphical techniques that produce the desired visual effects or render the visual cues.

IBIS uses a generate-and-test approach to design a sequence of static visual displays for procedural explanations [Seligmann and Feiner, 1991]. In addition to the *methods* that specify how to achieve a goal, there are *evaluators* that determine whether a goal has been achieved. Thus, while each method suggests a possible solution, an evaluator asserts whether a goal has been achieved already or a solution proposed by the method is adequate or satisfactory.

IBIS has also designed composite illustrations and interaction illustrations [Seligmann and Feiner, 1989]. If a single display becomes inadequate to fulfill the communicative intent, IBIS attempts to use a composite picture. It decomposes the high-level goal into subgoals, and tries to generate a simple picture for each of the subgoals. This process goes on and recursively breaks down a complex goal into simpler subgoals until its subgoals can be achieved using a single picture. Then all simple pictures are composed together to form a composite picture (e.g., a main picture with an inset). To enable user interactions, IBIS allows the user to modify certain properties of the scene (e.g., the viewing specification), and the system is able to redesign the picture and ensure that the satisfied constraints remain intact as the user interacts with it.

To generate coherent pictures, IBIS maintains a certain degree of consistency between related images. For example, it attempts to maintain a consistent viewing specification whenever it is possible, and allow child illustrations to inherit visual properties from their parent in the composite illustration hierarchy. It also achieves continuity by arranging the images in a logical order (e.g., the order in which depicted procedures are to be performed). However, IBIS still mainly focuses on generating a series of static pictures of 3D worlds: it does not deal with design consistency issues at the symbol encoding level (e.g., mapping object attributes to visual variables), or explicitly address transition issues from one picture to another in their generation, or deal with integrating new information into the existing presentations. Moreover, the prioritized goal input and the generate-and-test approach prevents the system from efficiently searching for desired visual techniques and constructing sequences of depict actions effectively [Russell and Norvig, 1995].

ESPLANADE. ESPLANADE [Karp and Feiner, 1990; Karp and Feiner, 1993] employs a hierarchical planning approach to automatically generate animation sequences using film-making heuristics.

The input to ESPLANADE is a script that contains a sequence of actions and a set of goals that specifies the communicative intent. It adopts a top-down hierarchical planning approach to construct animation sequences that illustrate the communicative intent. ESPLANADE plans the animation by following a hierarchical process. First, it plans at the *film level* to decide what the subjects of the animation. Next, the system plans at the *scene* level, at which the scene that are part of film are determined. Below the scene level, ESPLANADE decides the *sequences* that are part of the scene and selects the sequence structure formalism. The lowest level—*shot level* plans the most basic unit in a film, which involves selecting the content, viewing specifications, and transitions. To ensure the smooth transitions, ESPLANADE uses standard film-making strategies to transform from one picture to another. A set of rules are also developed to guide the system to determine which transition technique to apply under certain conditions.

Based on those film-making heuristics, ESPLANDADE incrementally plans the camera placement and movement to automatically generate animations frame by frame. The generated animation fulfills the communicative intent and certain effectiveness criteria (e.g., smooth transition between frames) that have been established by the film production process. However, like APEX and IBIS, it uses realistic images of physical objects and generates presentations. ESPLANDADE also deals with little user interaction and does not deal with consistency issues at the symbol encoding level and unity issues such as integrating new information into the current scenario.

CHAPTER 3 Framework

The meaning of the whole is greater than the sum of the meaning of the parts
—Szlichcinski

As described in Chapter 2, many researchers have developed a number of different system components in their automated graphics generation systems. However, no attempt was ever made to merge various system architectures and abstract the commonalities to establish a general framework, which can systematically and comprehensively guide the development of various generation systems. Without a general framework, individual researchers have to establish their own system framework and define every component from scratch. Without a common ground, it is also difficult to compare and evaluate different systems comprehensively and systematically. Therefore, our research focuses in part on establishing a general framework by abstracting various generation systems and providing a reference model in which a specific system is considered an instantiation of the framework.

There is a wide-spreaded discussion as to what it means for a system to be "intelligent", here we adopt a fairly common and simple definition [Barr and Feigenbaum, 1989; Jackson, 1990]: an intelligent system exhibits its intelligent behavior through performing reasoning over representations of human knowledge, and solving problems by heuristics or approximate algorithms. By this definition, automated graphics generation systems are intelligent systems. The amount of intelligence is determined by the amount and quality of the stored knowledge, the sophistication and efficiency of the inference algorithm, the efficiency and reliability of the rendering component, and the capability of handling user interactions. In other words, any automated graphics generation system must contain a *knowledge base*, an *inference engine*, a *visual realizer* and an *interaction handler.* These structural ingredients together deliver the intelligent behaviors. On the other hand, the abstractions of these four components are clustered together to establish a framework that serves as the reference model for all automated graphics generation systems in interactive environments.

In the rest of this chapter, we briefly describe the general establishment of each component in an automated visual discourse system.

3.1 Knowledge Base

To perform reasoning, an automated graphics generation system needs to have knowledge with which to reason. The knowledge base is more or less the direct encoding of human knowledge. There are two issues involved in constructing a knowledge base [Reichgelt, 1991]: identifying knowledge sources and determining knowledge representation formalisms. Identifying knowledge sources involves deciding the *types* of knowledge that the system must have in order to solve the problems as expected. Determining knowledge representation formalisms, on the other hand, is concerned with how to represent various knowledge sources so that they can be efficiently accessed and modified. In the following section, we briefly describe each type of knowledge and its representation, while Appendix A illustrates knowledge sources and their representation paradigms in detail.

3.1.1 Knowledge Sources

Although there is a great deal of knowledge involved in a graphic design process [Roth and Hefley, 1993; Beshers and Feiner, 1994], the knowledge can be roughly partitioned into four chunks based on their origin: *domain knowledge, visual design knowledge, situation knowledge*, and *meta knowledge*. *Domain knowledge* refers to knowledge about objects in a specific application domain (e.g., computer network management domain) and the relationships among them. *Visual design knowledge* encodes world-applicable visual design rules, along with structural and functional descriptions of various visual forms. To tailor a visual presentation to a specific user or a unique situation, *situation knowledge* is needed to store user-specific information (e.g., the user’s information seeking goal), to model occasions under which the system will be used, and to specify various constraints (e.g., time constraints) that may affect the presentation (e.g., a brief presentation is required). To explicitly reason about the control of an inferring process, *meta knowledge* is needed [Reichgelt, 1991]. Meta knowledge provides knowledge about the other three types of knowledge mentioned above, including extent or origin of the knowledge, and the reliability of the knowledge.

3.1.2 Knowledge Representation Formalism

Different types of knowledge vary significantly in their nature, so should the representation formalisms used to represent them. The ability to represent various types of human knowledge in a natural and useful way is considered one of the fundamental aspects of an intelligent system. The basic criteria used to evaluate a system's knowledge representation component are: efficiency, understandability, modularity, and extensibility [Barr and Feigenbaum, 1989; Reichgelt, 1991].

Efficiency requires that a knowledge representation formalism store information in a space-efficient way and retrieve information in a time-efficient way. *Understandability* requires that the representation formalism organize knowledge to reflect its natural structure in the real world so that the represented knowledge can be easily understood. *Modularity* specifies that the representation should be modular so adding or deleting a piece of information should not affect or have little impact on the rest of knowledge base. *Extensibility* demands that the knowledge representation formalism be easily extended to support new types of knowledge.

Given these four requirements, we have examined a variety of knowledge representation formalisms [Barr and Feigenbaum, 1989; Jackson, 1990; Reichgelt, 1991] and believe that there is no single formalism that can allow us to efficiently capture and access all the knowledge we need. Therefore, we have chosen a hybrid representation formalism that primarily uses a *frame-based* (i.e., *object-oriented*) *formalism,* supplemented by *production rule* and *procedural* representation formalisms.

Object-Oriented Representation

Frame-based representation paradigm is described in [Minsky, 1975; Schank, 1975; Bobrow and Winograd, 1977; Stefik, 1979]). *Object-oriented* representation is a more recent term that evolves from the area of computer programming languages. However, *object-oriented* representation is to a large extent based on frames. Hence, here we regard the two terms "object-oriented" and "frame-based" interchangeable.

Organizing knowledge around objects is a very natural paradigm in which each piece of information is considered as an object, and each object has a set of unique attributes. The relationships among different pieces of information are modeled as relationships among various objects. In automated graphics generation systems, both domain and visual objects can be captured using this representation paradigm. The main advantage of this paradigm is that the natural structure of the knowledge is directly reflected in the knowledge base [Minsky, 1975]. This feature makes the represented knowledge easily understood by a domain expert and result in fewer difficulties in maintaining the knowledge base [Jackson90]. Furthermore, due to the independent description of each object, the knowledge base is also modular. This enables the knowledge base to be easily extended to include new information with little impact on the rest of the knowledge base.

Production Rule Representation

Even though object-oriented representation formalism captures many types of knowledge in a natural way, the formalism becomes awkward and inadequate when it is used to represent so-called *instructional knowledge* [Jackson, 1990]. *Instructional knowledge* is also known as *condition-action* or *situation-action* knowledge. In particular, these are conditional statements about *what* to do *when* certain conditions are satisfied. Furthermore, this type of statements are most frequently used by human experts to explain how they do their job.

As an automated graphics generation system simulates a human graphics designer's decision making process to construct visual presentations, visual design principles or policies naturally fall into the category of instructional knowledge to guide such a process. Thus, we need a representation formalism other than the object-oriented paradigm to capture this type of the knowledge in our knowledge base. The representation formalism that encodes such knowledge through certain pattern regularities (i.e., IF *situation* THEN *action*) and enables the system to perform a pattern-directed inference [Reichgelt, 1991], is known as *production rule* representation formalism.

One of the strongest attractions in this formalism perhaps is the *naturalness* of the representation and the *modularity* of the rules. In our case, visual design principles about *what* visual objects to use (e.g., use a table chart) or design constraints to maintain *when* certain conditions (e.g., need to list the information) are satisfied are naturally encoded into production rules. Furthermore, related rules can be easily grouped together and individual rules can be added, deleted or changed independently. Because rules don't call each other directly, changing one rule won't have direct effects on the other rules.

Procedural Representation

Human knowledge can be divided into two types: *what* the facts or rules are, and *how* to perform certain actions [Reichgelt, 1991]. We have discussed using *object-oriented* and *production-rule* formalisms to represent *what*, and now we describe using *procedural* representation to represent *how*. *Procedural* representation is best when used to capture knowledgeable about how to use certain knowledge, such as finding relevant information, retrieving certain attributes, or executing certain actions [Barr and Feigenbaum, 1989]. Naturally, the knowledge about retrieving certain information of a domain/visual object or executing visual operations, can be encoded as *procedures*.

3.2 Inference Engine

An automated graphics generation system is indeed a *problem-solving* system [Barr and Feigenbaum, 1989]: it *reasons* about the domain and visual information, and *infers* the visual presentation by following visual design principles. As the *knowledge base* provides all the information, *inference engine* performs reasoning and solves particular problems of visual design. To efficiently solve the design problem, the inference engine must be equipped with powerful *problem-solving* [Barr and Feigenbaum, 1989; Rich and Knight, 1991] techniques. Thus, deciding which problem-solving technique to use and how to utilize it in the visual design domain becomes the focus of this section.

3.2.1 Hierarchical Decomposition Partial Order Planning

Let us first examine what a final visual design looks like: the final design will be in the form of a *visual discourse*. Based on the syntactic definition of the *visual discourse* and some of the top level visual objects (Figure 3-1), the design output will be a series of temporally ordered *visual frames*, which in turn can be either *static* or *dynamic*. *Static frames* are

$$S \rightarrow VisualDiscourse$$
$$VisualDiscourse \rightarrow VisualFrame \otimes VisualDiscourse | VisualFrame$$
$$VisualFrame \rightarrow StaticFrame | DynamicFrame$$
$$StaticFrame \rightarrow VisualStructure \oplus StaticFrame | VisualStructure$$
$$DynamicFrame \rightarrow VisualAction$$
$$VisualStructure \rightarrow Chart | Graph | Diagram | \ldots$$

S : Starting symbol

$A \otimes B$: Certain temporal constraints imposed onto *A* and *B*.

$A \oplus B$: Certain spatial constraints imposed onto *A* and *B*

Figure 3-1 Syntactic specification of high-level visual objects

actually combinations of various visual objects, while *dynamic frames* refer to any of the visual actions, which are encoded visual techniques (e.g., *Enlarge*). As any visual action always acts upon certain objects (i.e., the operands), we could view static frames as the operands of a special visual action—*Display*. Thus, the final design is a temporally ordered *visual action sequences*. Such a pattern is reminiscent of the similar result produced by *planning* approaches described in [Wilkins, 1988; Cohen and Feigenbaum, 1989; Russell and Norvig, 1995]: planning is the process of developing a sequence of actions to achieve a goal.

Unlike search-based problem solving approaches, which consider unbroken sequences of actions starting from the initial state and are forced to decide what to do at each state, planning approaches are more flexible and can work on whichever part of the problem that is most likely to be solvable given the current information. By making “important” decisions first, planning approaches reduce the branching factor for future choices and reduce the need to backtrack over arbitrary decisions.

As a consequence, an automated graphics generation system could be viewed as a planning system that can perform actions. In particular, the system models visual design principles as constraints, and employs visual actions as problem-solving operators to either construct new visual objects or transform old visual objects into new ones. The final design is a sequence of visual actions, which consist of both static display acts (e.g., *Display*) and dynamic animation acts (e.g., *Move* and *Scale*). Furthermore, in an interactive visual environment, the system may need to maintain the user interaction context to respond properly to its users and adhere to the human-computer interaction guidelines. A system based on goals and plans is the most flexible since such a system can more easily decompose a complex task into subtasks using the problem-solving operators, detect goal interactions and find the order of the actions. Such plans not only provide the sequence of actions, but they also maintain the history and state of actions as steps are begun and completed. This representation provides much-needed information for history information and facilitates future replanning.

As approaches to planning vary substantially [Cohen and Feigenbaum, 1989; Rich and Knight, 1991], *hierarchical planning* stands out by reducing the amount of search needed to achieve computational efficiency [Yang and Tenenberg, 1990], and by reusing the operators that are common to many tasks to ease knowledge encoding [Wilkins, 1988]. Hierarchical planning in essence generates a hierarchy of representations of a plan in which the highest level is a simplification or abstraction of the plan, and the lowest is a detailed plan, sufficient to solve the problem. In the course of planning, *decomposition* refers to the process of refining the vague parts of plan by replacing higher-level, abstract visual actions using lower-level actions [Knoblock, 1993; Young et al., 1994]. Moreover, as the hierarchical planning could go either way [Weld, 1994], a *top-down* strategy [Young et al., 1994] facilitates the interleaving of planning and execution [Wilkins, 1988] and ensures both local and global coherency within a visual discourse. In contrast, in a bottom-up approach (e.g., [Mackinlay, 1986]), 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. Please refer to Appendix B to see detail planning stages within a hierarchical decomposition partial-order planner.

3.3 Visual Realizer

A *visual realizer* carries out the visual design and communicates it to the user. It needs to understand the meaning of the design and render the design using the underlying graphics language. For the sake of simplicity, from now on, we will use *design process* to refer to the process that uses the *inference engine* and *knowledge base* to actually plans the visual presentation, and *realization process* to refer to the process that calls the *visual realizer* to carry out the planned design.

As shown in Figure 3-2, there are two possible relationships between a design process and a realization process. One is to embed the realization process inside the design process. In this case, the design process carries out the plan by directly calling the rendering routines as if it understood the rendering language. The other is to completely separate the two processes through using a common language that can be understood by both parties. In other words, the design process would write out the design in the common language, and the realization process would take the intermediate output and realize it using whatever graphics

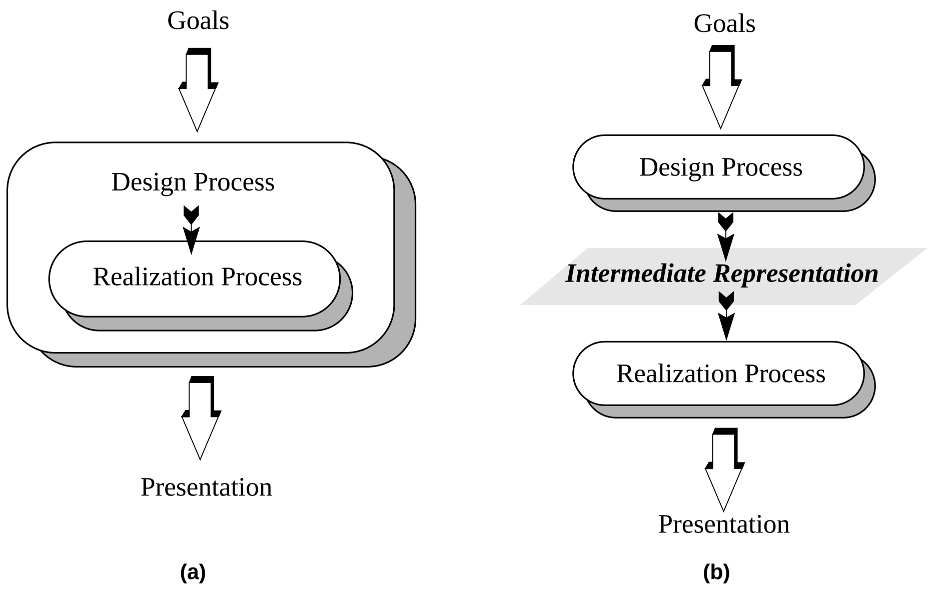


Figure 3-2 Two arrangements between design and realization processes

language it understands. Although the former strategy might be more efficient in some sense by directly calling the realization without producing intermediate output, the latter one would have more long-term advantages. With the advent of advanced multiprocessing and networking technologies, computationally costly processes can be parallelized and distributed onto different processors to reduce computational time. From this point of view, a generation system can be parallelized by distributing the design and the realization process onto different processors. Apparently, the parallelization would not be so easy if one process is completely buried within another (Figure 3-2a).

Ensuring the portability of the renderer is also very important since the system may need to immigrate to progressively more powerful platforms. If the output of the visual design or the renderer is platform-specific, then the *visual realizer* would need to be completely rewritten for a new platform. Therefore, separating the design and realization process and designing platform-independent intermediate language become the two main issues in the *visual realizer.*

Separating design from realization means that the two processes are independent of each other, and they communicate with each other through an intermediate language. In this architecture (Figure 3-2b), design process outputs the entire design specification in a language that is understood by both processes, while the realization process only focuses on translating the intermediate language into the target graphics language. Whereas the two processes can be easily parallelized, adopting a new graphics language or porting it to a new platform seems less difficult either under the assumption that the intermediate language is platform-independent, and expressive enough to contain the high-level descriptions with low-level common graphics primitives [Foley et al., 1990].

Based in part on Open Inventor file format [Wernecke, 1994] and VRML 2.0 [VRML, 1996], we have decided to design the intermediate language to satisfy these criteria: *simplicity*, *comprehensiveness*, and *uniformity*. *Simplicity* refers to the fact that the syntactic structures of the language is straightforward and easily understood. *Comprehensiveness* refers to the language's ability to express both high level structures and low level details. A *uniform* language represents various objects consistently, and it uses the similar structures whenever it is possible.

3.4 Interaction Handler

It is desirable to allow certain user interactions in a visual presentation system as the user always wishes to have certain control over the information accessing process. In the context of automated visual presentation, supporting user interaction means two things: First, the presentation should appear *visually interactive* so it can attract the user to voluntarily participate in the presentation and explore the information as they wish. The other is to give the user certain control over the presentation or presentation design by allowing them to interactively specify their preferences.

To provide the interaction capabilities mentioned above, there are two issues involved: *design* and *control*. The system needs to incorporate *object interactivity* into its *design* to create the *interactive look*; and it also needs to provide the facility to *control* the presentation once the user interaction occurs. Thus, the *interaction handler* in an automated presentation generation system has two duties: modeling user interactivities and supplying this information to the design component, and reactively controlling the realization of the presentation when responding to the user events.

3.4.1 Designing an Interactive Presentation

Modeling user interactivities includes analyzing the user task profiles [Shneiderman, 1992], characterizing information properties, and categorizing interactive metaphors [Foley et al., 1990]. To design interactive presentations, the system needs to know what types of tasks in which the user is most likely to take an active role, and what interaction metaphor to use for representing the specific domain information.

Interaction Tasks

The major tasks requiring user participation are known as *exploration* tasks [Shneiderman, 1992]. When presented with a large amount of information, the user might not be to able to determine at once what s/he exactly needs. Usually, the user would prefer to navigate the displayed presentation by either asking for more information or requesting the same information presented in a different form. Another type of interaction task often seen is the *experimental* task. To learn some uncertain effects of events, the user would like to experience different types of effects that are caused by experimenting various interaction tools (e.g., changing the size of the object).

The type of tasks also determines what type of information could be accessed by the user interactively. Once the user has grasped a piece of information completely through the presentation, it is unlikely that s/he would explore or perform experiments on that information. This leaves the complicated, implied information to the user for further exploration or experimenting.

Interaction Metaphors

To make presentations appear visually interactive, the system needs to employ a set of interaction metaphors such as buttons, hyperlinks, and pull-down menus [Newman and Lamming, 1995; Foley et al., 1990]. These metaphors visually imply the interactivity of the presentation and provide an avenue to engage the user-system communication. In visual design process, interaction metaphors become another type of functional building blocks to lead the user to act upon what s/he sees, e.g., presenting affordances [Gibson, 1977].

Suppose that a composite object "Columbia University" is represented as part of a visual design. If the system simply represents it using a textual label (Figure 3-3a), it is unlikely that the user would pay more attention to it besides acknowledging its existence, let alone explore what is hidden behind it. In contrast, if the system represents it as a button (Figure 3-3b), it not only provokes the user to expect that certain information is hidden behind the but-

Columbia University
Columbia University
(a)
(b)

Figure 3-3 Non-interactive versus interactive representation of same concenpt

ton, and it is likely that the user would press the button and find more information about "Columbia University".

In the context of automated graphics design, the system needs to automatically select the appropriate interaction metaphors and utilize them following certain interaction styles. As interaction styles [Shneiderman, 1992] vary with task profiles and information characteristics, common styles such as *menu selection* and *direct manipulation* [Shneiderman, 1992], could be directly incorporated into visual presentations.

3.4.2 Controlling an Interactive Presentation

Once the system has successfully integrated the interaction metaphor into the presentation, certain user interactions are expected. Based on whether the user activities occur within the *presentation loop*, the user events can be divided into two categories: *isolated user interaction* and *combinatorial user interaction*. *Presentation loop* refers to a time interval during which the system starts to present the information to the user until the presentation is finished. During this time interval, the system might replay the whole presentation more than once at the user's request or might jump back and forth within a range of frames. And we use the term *loop* to emphasize that such a process is not an end-to-finish straight-line process.

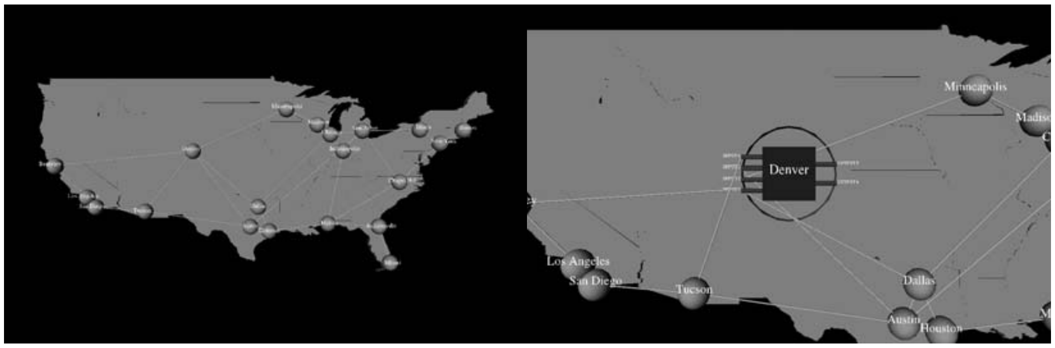
Isolated user interaction occurs outside of the *presentation loop*. The interaction handler does not need other historical information to deal with this type of events. But to guarantee that there is no side effect caused by the action, the interaction handler always attempts to restore the original state right after the action is performed. Unlike an isolated user interaction, *combinatorial user action* does occur within the *presentation loop.* To handle the event properly and continue with the unfinished presentation, the system needs to keep track of the presentation events and user actions.

In this chapter, we have briefly discussed the four core components that are building blocks of an automated graphics generation system: *knowledge base, inference engine, visual realizer,* and *interaction handler.* In the next chapter, we will describe how an prototype system IMPROVISE puts all the theory into action and automatically generates visual presentations for two different domains.

CHAPTER 4 IMPROVISE

The world can only be grasped by action, not by contemplation. . . . The hand is the cutting edge of the mind. — Jacob Bronowski

Based on our general framework, we are developing IMPROVISE (Illustrative Metaphor Production in Reactive Object-oriented VISual Environments) as a proof-of-concept prototype system. To demonstrate the generality and comprehensiveness of the framework, IMPROVISE aims to automatically design coherent visual discourse for heterogeneous information in interactive environments. Within such discussion, heterogeneous information can be quantitative or qualitative, static or dynamic. *Quantitative* information refers to discrete or continuous numerical scales or ranges, while *qualitative* information abstractly describes various aspects of domain objects (e.g., a *tall* person). Whereas the *static* information remains unchanged, *dynamic* information varies with time. We also assume that IMPROVISE intends to support a wide variety of visual techniques, ranging from individual 2D displays to interactive 3D animation sequences. As an instantiation of the general framework, IMPROVISE consists of a *knowledge base*, an *inference engine*, a *visual realizer*, and an *interaction handler*. Each of these four components has been either fully or partially developed. Furthermore, IMRPOVISE has also been used in two testbed application domains to demonstrate its generality and flexibility. One is a computer network management application that visualizes a network's structure and behavior [Crutcher et al., 1995]. The other is a hospital information application that provides a care giver with a multimedia sum-



(a) (b)

Figure 4-1 Exploring the internal structure of a network node

mary of a patient's medical record [Dalal et al., 1996a].

Before we further dissect the structure and examine the functions of IMPROVISE, we briefly present high-level background information about the two application domains, including their major domain tasks and expected audience. Then we present a detailed view of the key design modules of IMPROVISE through its system architecture to show how all four core components can be exploited and coordinated for presentation design. We will briefly explain the function of each module and how the modules collaborate together to assemble a cohesive working system. Following the architecture walkthrough, we will analyze two examples from each of the application domains to demonstrate how the system actually plans the design and comes up with the desired solution. At last, a hypothetical example is given to illustrate what a completed IMPROVISE is capable of in related to the research tasks to be conducted.

4.1 Problem Domains

To demonstrate the generality and flexibility of the framework, we have used IMPROVISE in two different application domains: computer network management and patient medical record summary. In this section, we will describe domain tasks and expected audience.

Computer Network Management

Computer network management is a complex task. Visualizing network structure and behavior could help network researchers or network managers understand aspects of network activities [Crutcher et al., 1995]. Moreover, the notion of using spatial or temporal metaphors enables the targeted audience to directly interpret objects and operations as they directly experience the visual manipulation results in 3D space.

Management Tasks. There are two types of tasks: interactively *exploring* network entity structures and *monitoring* network traffic status. The first type of task involves examining the structure of various network entities (e.g., nodes and links) at different abstraction levels. In particular, we have modeled an ATM network in our experiment. In Figure 4-1(a), the initial view of a network node (i.e., node Denver) could be symbolized by a simple sphere, the second view of the same node could be “exploded” to reveal its internal structure, which might include its port structure and the connectivities of the ports (Figure 4-1b). Gradually exposing the different levels of detail at the user’s request enables the user to organize management information hierarchically but also gives the user freedom to access any part of the information interactively.

Traffic monitoring tasks, on the other hand, are directly linked to a network entity. For example, the user is always interested in learning the traffic status within a link, or the traffic passing through a particular network routing path within a given time interval. Compared to exploring network entities, this type of activity involves more dynamic information since network traffic data varies over time. To simulate real network traffic monitoring, IMPROVISE has been connected to a network emulator [Chan et al., 1995] to obtain dynamically changing network traffic data.

Audience. The expected audience in this domain is network management researchers. A presentation tool helps them to establish network service hierarchy, and develop and verify their network management algorithms [Crutcher et al., 1995].

Patient Medical Record Summary

In a hospital, a number of patient medical records have been entered in a database and made available to care givers through computer access. However, most information is not in a form that the care giver can conveniently access or interpret. Thus, the task of this domain is to effectively organize such information and present them coherently as a multimedia summary [Dalal et al., 1996a]. As the multimedia summary contains coordinated text, speech, and visual presentations, constructing a working system is an collaborative effort among several research areas including knowledge representation, natural language generation and graphics generation. Amid various components in an experiment system [Dalal et al., 1996b], IMPROVISE serves as the graphics generator, which is responsible for deploying appropriate visual media to effectively represent and organize information.

Summary Tasks. To effectively present the patient information summary to the care givers, summary tasks can be approximately divided into three categories based on the purpose of the summary: *overview*, *detail summary*, and *special report*.

Overview is used to brief care givers about patient’s information at a very high level without going into depth about that information. Information details can be brought up at the user’s request (e.g., through user interaction). This type of summary usually is concise but comprehensively covers different aspects of the patient information.

Detail summary describes the patient information at a high level, then drills down to the details of that information. For example, a report could first describe the overall medication that has been given to a patient, then it brings up the specific details such as the drug usage that has been administered (e.g., time and dosage).

Special report focuses on an abnormal or emergency situation. This type of report aims to catch the care giver’s immediate attention so that s/he can act upon the situation properly.

It is common that more than one type of the summary may coexist in a report, as a mixture of summaries is needed to provide a comprehensive coverage of patient information.

Audience. In our application, the system aims to address two types of care givers: nurses

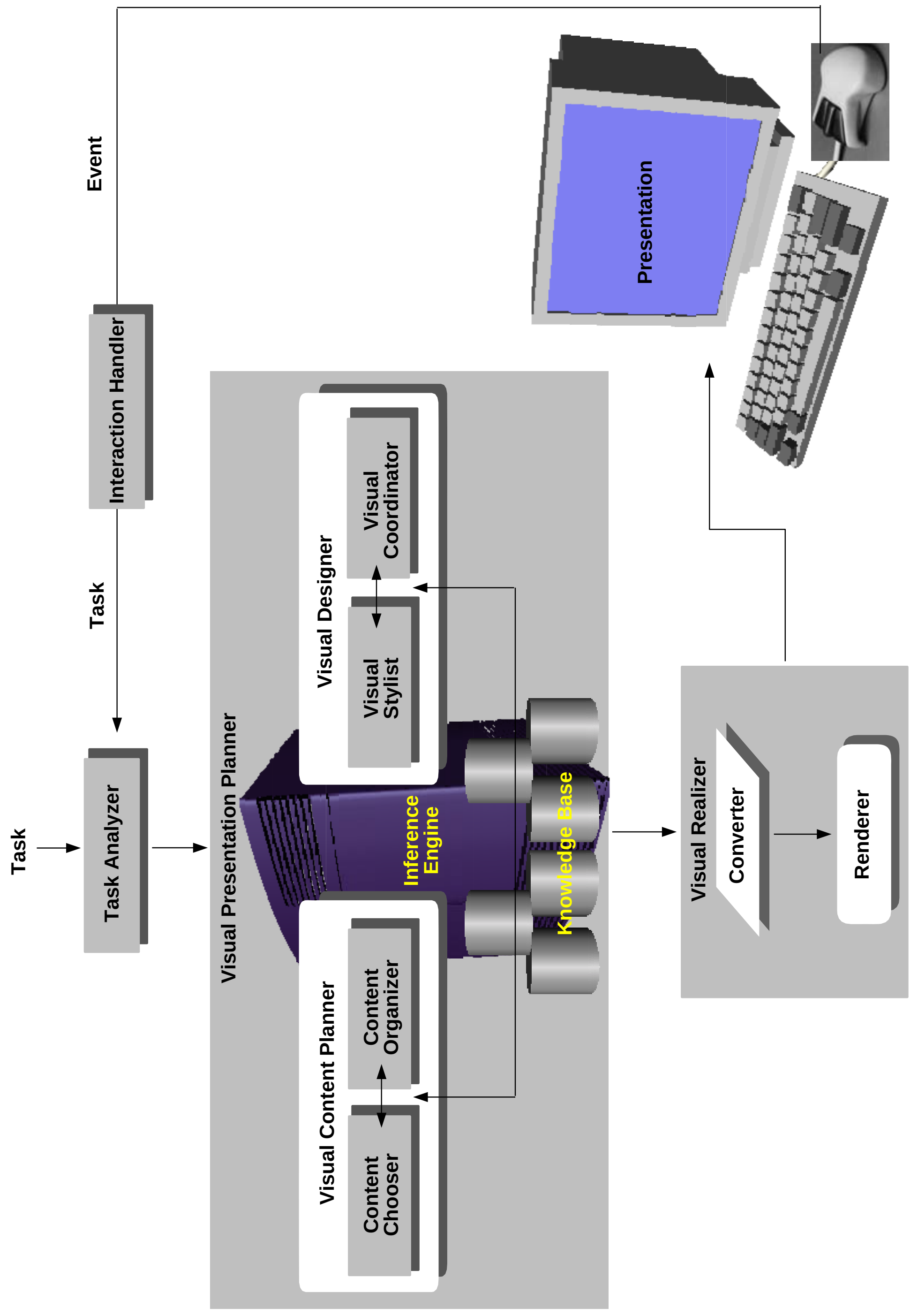


Figure 4-2 IMPROVISE System Architecture

and doctors. Based on their needs, the type of information and the style of the summary may vary. For example, a nurse might prefer to see information arranged spatially related to a patient's physical body, while a doctor may be interested in examining important events arranged along the time line when they occurred.

4.2 System Architecture

As shown in Figure 4-2, IMPROVISE has four main modules: *task analyzer*, *visual presentation planner*, *visual realizer*, and *interaction handler.* A full cycle of presentation design starts with a communicative task (i.e., goal). The *task analyzer* is responsible for interpreting and formulating domain tasks (e.g., examine a network node structure) into corresponding visual tasks [Wehrend and Lewis, 1990]. To accomplish the visual tasks, the *visual presentation planner* starts the design process. The design process is an interleaving process between two submodules: *visual content planner* and *visual designer.* The *visual content planner* selects and organizes the content that needs to be presented, while the *visual designer* makes decisions about what visual cues to use and how to coordinate various visual components into a coherent whole. Once the design is finished, it is written out in an intermediate language and eventually is converted to the target graphics language to be realized. The user then may interact with the generated presentation. The user interaction events are captured and processed by the *interaction handler.* The processed events are formulated as new communicative goals, and the resulting goals are passed to the *task analyzer* where a new design cycle begins. Next, we explain the functionality of each module in more detail.

4.2.1 Task Analyzer

Communicative tasks, also known as communicative acts, describe the communicative intent of the visual presentation to be designed. These high-level communicative intents are usually represented in a form of rhetorical structure (e.g., *Inform* patient record) and do not always explicitly specify or directly reflect the related visual goals that need to be achieved. Visual presentation designing, on the other hand, is a process motivated by fulfilling the required visual goals. For example, a high-level communicative task of informing a patient's information to a nurse could be interpreted as the following combination of visual goals: *clustering* related information spatially around the patient's physical body, *locating* the important information, and *emphasizing* the details of certain information. The visual tasks such as *clustering*, *locating*, and *emphasizing* all refer to various visual effects caused by different visual acts. It is the *task analyzer*'s responsibility to perform task analysis and correlate the communicative tasks with visual goals.

Task Representation

To translate high-level communicative tasks into visual tasks, we represent both types of tasks in a similar structure. Each task is represented in two parts: *act* and *object*. *Act* specifies what type of task it is, while *object* augments the task specification by stressing the types of information to be conveyed. For example, a task representation may specify that the system needs to *inform* the user about a patient's blood replacement therapy instead of his/her lab reports.

We have adopted the taxonomy from natural language discourse or dialog theory [McKeown, 1985; Grosz and Sidner, 1986; Mann and Thompson, 1988] to depict the communicative acts. Conversely, the visual task taxonomy is in part based on [Wehrend and Lewis, 1990] and picture functions suggested by various researchers in cognitive studies (e.g., [Levin et al., 1987; Sutcliffe and Darzentas, 1994]). However, our current task characterization taxonomy is still coarse, we need to refine the taxonomy to describe both communicative and visual tasks more comprehensively and precisely.

Task Analysis

Analyzing communicative tasks and proposing corresponding visual goals is the main responsibility of the *task analyzer.* As visual goals can be viewed as the effects of various visual actions, the task analyzing process becomes a conventional planning process: the task analyzer employs visual actions as planning operators, and makes decision about which visual action(s) to use to accomplish the communicative tasks. The output of this planner is a set of partially ordered visual actions. These high-level visual actions (i.e., visual tasks) are sent to the visual presentation planner for further processing. Although, the task analysis process uses visual actions to fulfill communicative intent, it is different from an actual visual presentation design process. The task analyzer only specifies *what* visual tasks need to be carried out by proposing a set of visual goals instead of dealing with the issue of *how* to use visual techniques to accomplish the visual goals.

While interpreting communicative tasks involves a great deal of domain information, the general-purpose visual tasks are domain independent. As different domain tasks are formulated into various combinations of domain-independent visual tasks, it relieves the *visual presentation planner* from performing a considerable amount of domain-related reasoning. Correlating communicative tasks with corresponding visual goals is not an easy task. It demands careful analysis about both rhetorical structures and visual functions. As part of our future research plan, we will concentrate on one or two types of communicative task (e.g., *inform*) and study the underlying relationships between this task and various visual tasks (e.g., *display* or *emphasize*).

4.2.2 Visual Presentation Planner

The main task of the *visual presentation planner* is to plan the visual presentation to accomplish the visual tasks proposed by the *task analyzer.* The planning task is distributed into two subcomponents: *visual content planner* and *visual designer.* These two subcomponents interact with each other and together they determine the *content* and the *style* of the visual presentation. As a part of one complex planning process, each module performs planning on its own but all modules consistently use one planner—PREVISE (Planning in REactive VISual Environments), which has been developed based in part on [Wilkins, 1988] and [Young et al., 1994].

Visual Content Planner

The *visual content planner* makes decision about what to include in the visual presentation and how to organize them into perceptual groups. In particular, if the information to be presented is very complex, the visual content planner plans to select, partition and group various pieces of information into reasonably sized chunks so that the user will not be overwhelmed due to information overload or an ill-formed presentation. For example, presenting a patient’s information to a nurse involves conveying various types of information such as patient identity, medical history and treatment. How to visually organize all information effectively will have a direct impact on the nurse’s reaction to critical information. In this case, the most important (e.g., abnormal or urgent) information might be grouped together in one frame and presented first, followed by the routine information about the patient. Moreover, to provide or maintain the context of the presentation dialogue, the system also needs to determine what other objects must be included besides the focus object. For example, in the network management domain, while a link is the center of focus, so are two nodes that are directly connected to the link. In other words, if the system decides to enlarge the link, so must it also enlarge the two end nodes proportionally. We will see an example of this situation in Section 4.3.

In IMPROVISE, deciding what objects to include is based on the object-relationships. A more sophisticated object relationship modeling hierarchy can be found in [Feiner, 1985]. As a visual presentation always consists of an array of visual objects, the content of each visual object is decided based on information characteristics and related visual object contents. Therefore, the *visual content planner* by itself cannot make decisions about the visual object content. It needs to communicate with the *visual designer* and gradually refine its decision based on the information provided by the designer. The interleaving of the *visual content planner* and the *visual designer* reveals the nature of the hierarchical planner: the *visual content planner* sketches the high-level organization of the presentation, and the *visual designer* uses such information as a start to plan the presentation. Unless more design details become available, the visual content planner cannot continue to make decisions. After the content planner refines its decision about the content, it passes the information back to the designer so it can refine its own plans. Thus, planning is a constructive process in which a sketch is incrementally refined, and arbitrary decisions are seldom made due to insufficient information.

Visual Designer

The *visual designer* is the component that actually deals with the visual design subject: it determines how to employ visual techniques and how to coordinate various visual components together into a coherent whole. The designing task is decomposed into two subtasks: *visual styling* and *visual coordinating,* carried out by the *stylist* and the *coordinator,* respectively.

Visual Stylist

The *visual stylist* deals with visual design issues, ranging from deciding high-level visual presentation schemes (e.g., a table chart or a bar graph) to low-level symbol encoding (e.g., using red to represent critical information). As illustrated in Chapter 3, visual objects are represented as objects at different levels of abstraction in a hierarchy. Deciding what visual components are at one level becomes the problem of refining the visual object at the level above. For example, deciding to design a 2D bar graph as a *visual structure* is actually a process of using a 2D bar graph to refine a particular component of a certain *visual frame*.

This object refinement process becomes the focus of the visual styling process in which abstract visual object descriptions are refined using more concrete visual specifications.

In the case of designing new presentation by transforming existing visual objects, visual styling becomes the process that searches for primitive visual actions to fulfill the specified visual goals. In either way, a top-down hierarchical planner suits such a process well and it provides all desired information as the planning ends (e.g., the sequence of visual actions to transform the current presentation into a new one).

Evidently, an effective visual presentation rarely communicates various information by just using a single presentation scheme (e.g., a table chart). Moreover, by carefully examining the structure of various visual presentations [Tufte, 1983; Tufte, 1990], we discover high-level presentation schemes are exploited as a schematic way of organizing information visually. Each high-level visual scheme is recursively defined and various visual objects are incorporated as its structural components. For example, an organization chart could contain icons or photos as its components to represent personnel in the chart, or it can use a graph plotting each person's achievements with his/her picture. As one single type of visual object cannot make a comprehensive visual communication, an effective visual presentation always integrates various visual objects (e.g., the table chart organization of multiple bar graphs). Hence, we can recursively apply a limited number of basic visual objects to compose complex visual presentations.

As described in Chapter 3, using a hierarchical planning approach enables the visual stylist to focus on selecting the visual techniques for the current tasks at a high level rather than wondering about the details of the visual techniques. Decisions about the low level details of each visual technique gradually appear on the agenda as the design process progresses. Eventually, all details of the visual techniques are finalized as the planning process moves toward completion.

So far we have defined three types of high-level visual objects: *table chart*, *structure diagram*, and *cartogram*. Although we have studied and defined the syntactic features of these visual representations extensively, their semantic and pragmatic features are somehow weakly defined. While we are developing a formal specification scheme to describe the semantic and pragmatic features of various visual objects, we are also extending the catalogue of high-level visual representation; at least, including the *time chart* representation.

Visual Coordinator

The *visual coordinator* coordinates various visual components or visual actions. Coordinating visual components involves laying out the components effectively. In the case of underconstrained visual parameters, the coordinator is also responsible for instantiating the unconstrained values while taking into account the characteristics of the whole presentation. For example, a patient's name could be represented as the title of a presentation in a text string, but the text font size may not be strictly specified except for a readability constraint. By considering the style of the overall presentation, the coordinator might decide to use one font size instead of another. To automatically layout the components or supply numerical values to parameters, IMPROVISE needs design knowledge and a numerical constraint solver. We have decided to use STM (Snap-Together Mathematics) [Gleicher, 1994] to help layout the components and enforce certain numerical constraints. However, we do not yet have the communication interface that connects the STM and our visual coordinator. As part of our plan, we will define an interface so that constraints specified in the visual coordinator can be solved by STM.

As the output of a partial-order planner, underconstrained visual actions may not be fully specified or completely ordered. Thus, the visual coordinator is responsible for instantiating parameters of visual actions or imposing certain temporal orders among them before their execution. The order of the visual actions can be determined based on cinematic theory to bring out the maximum effect of an animated presentations. For example, some actions might be performed together to enforce certain effect, while others are not. Some parameters, such as how long an action should last, also need to be decided before the action can be executed. All this decision making process requires extensive amount of knowledge from human graphic designers or film makers, we will continue to consult with our experts and enrich our visual design knowledge.

Like the *visual content planner*, the *visual stylist* and the *visual coordinator* subcomponents are also interleaved in the design process, although the *visual coordinator* usually starts at the later planning stage since it plans the low-level details of a presentation.

Once the design phase is done, the design is passed to the visual realizer to be transformed into visual displays on the screen.

4.2.3 Visual Realizer

The *converter* performs a two-phase translation: the design is first written out in intermediate language, then the specification in intermediate language is translated to the descriptions in the target graphics language. At this point, the design is passed to the *renderer* to be rendered.

Rendering a design includes two parts: displaying a new visual design and dynamically animating the visual actions. Currently, IMPROVISE's *renderer* is implemented using Open Inventor, an object-oriented 3D interactive graphics toolkit [Wernecke, 1994]. We encode all

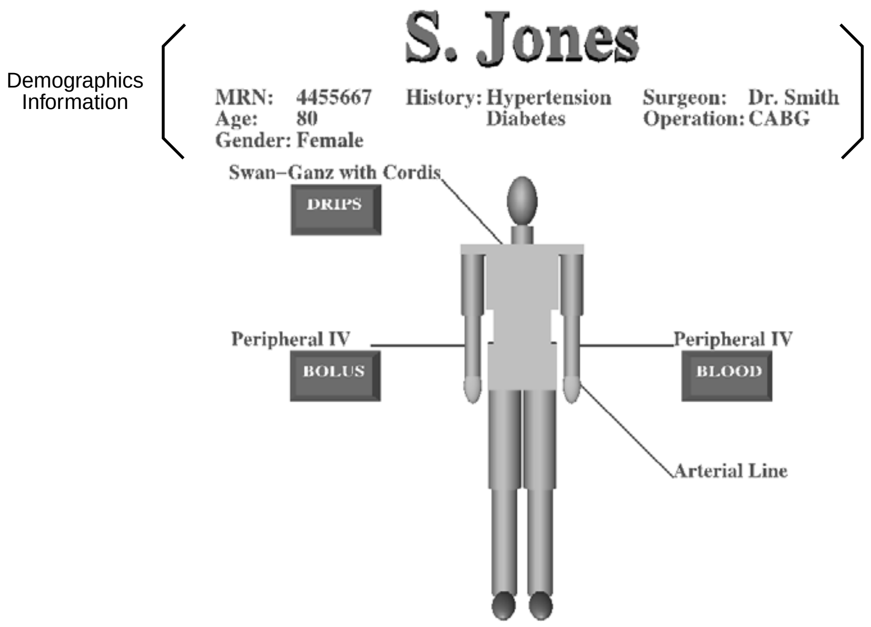


Figure 4-3 Automated generated presentation of a patient information for a nurse

primitive visual representations and visual actions as Inventor nodes by deriving them from the existing Inventor primitives. In particular, visual representations are derived from Inventor's *shape* node, and visual action nodes are derived from the topmost Inventor node hierarchy, augmented with temporal constraints. To execute the visual actions in sequence, a plan agent checks the temporal constraints for each action node, and forms an action queue to execute the action in the specified order. For static displays, temporal constraints are also used to regulate how long a scenario would remain on the screen.

Currently, the converter is only partially implemented, the intermediate language needs to be refined and the translation processes needs to be completed.

4.3 Examples

In this section, we will see three examples, one from our network management domain, two from the medical domain. The first two examples emphasize how IMPROVISE designs a desired visual presentation, while the third example illustrates what current IMPROVISE can *not* design, but should be able to accomplish it when it is complete. All examples are used for illustrative purpose and have been greatly simplified.

4.3.1 Medical Domain Example

Our first example comes from the medical application domain. The task is to present a patient's information to a nurse. As the ultimate presentation contains coordinated speech, text, and graphics, here we focus on the process during which IMPROVISE behaves as a graphics generator in a collaborative multimedia system [Dalal et al., 1996b]. Within a complete presentation, a large amount of information needs to be conveyed to a nurse (Figure 4-3). In our example, we focus on how the system plans to present the patient's demographics information (as indicated in Figure 4-3) to the nurse. As we mentioned earlier, IMPROVISE cannot automatically compute the spatial layout for this example due to the missing interface between STM and IMPROVISE.

A patient's demographics information is a composite data object, including data items such as *name*, *age*, and *operation*. In IMPROVISE, we represent the demographics information and some of its components (e.g., *mrn* and *age*) as follows:

```
(patient-4455667-demographics
    (is-a CONCEPT)
    (type composite)
    (components patient-4455667-name
            patient-4455667-mrn
            patient-4455667-age
            patient-4455667-gender . . . )
    (sense LIST)
    (role IDENTIFYING)
    . . .
)
```

```
(patient-4455667-mrn
    (is-a ATTRIBUTE)
    (type atomic)
    (value 4455667) . . . )

(patient-4455667-age
    (is-a ATTRIBUTE)
    (type ATOMIC)
    (value
       (patient-4455667-age-value
          (is-a MEASUREMENT)
          (unit "years")
          (value 80))) . . . )
```

In this case, IMPROVISE plans a design based in part on information characteristics. It first determines that a table chart could be used to fulfill the goal of conveying the demo-

```
  # Visual lexicon entry for ATTRIBUTE sense1
  . . .
  (concept-pattern (?x (type ATTRIBUTE)))
  (senses
     (sense1
       (syntax (category VISUAL-STRUCTURE)
         (subcategory TABLE-CHART)
         (media GRAPHICS-MODEL))
       (semantics (role IDENTIFY)
         (scope SMALL-ATTRIBUTE-VALUE)
         (sense LIST))
       (lexeme (TABLE-CHART
         (cells (VISUAL-UNITY (geometry
           (TEXT
             (string (get-attrName ?x)))))
             (VISUAL-UNITY (geometry
           (TEXT
             (string (get-attrVal ?x)))))
) . . . )
```

```
# Visual lexicon entry for ATTRIBUTE sense2
 . . .
 (concept-pattern (?x (type ATTRIBUTE))
 (senses (sense1 . . . )
   (sense2
     (syntax (category VISUAL-STRUCUTRE)
       (subcategory TABLE-CHART)
       (media GRAPHICS-MODEL))
     (semantics (role IDENTIFY)
       (scope LARGE-ATTRIBUTE-VALUE)
       (sense LIST))
     (lexeme (TABLE-CHART
       (cells (VISUAL-UNITY (geometry
             (TEXT
               (string (get-attrName ?x)))))
           (VISUAL-UNITY (geometry
             (OPTION-BUTTON
               (string (get-attrVal ?x))))))
   ))
```

Figure 4-4 Two senses for ATTRIBUTE object and their visual representations

graphics information. A table chart is chosen over other types of visual representation scheme because demographics information is a composite object and needs to be represented in LIST sense, in which all its components should be listed as textual entries. Moreover, IMPROVISE also decides to use the patient's name as the title of the table chart so that the name can serve as the identification of the patient information.

Once IMPROVISE decides to use a TABLE-CHART, it constructs an abstract table chart, in which each table cell is undefined. Based on the syntactic constraints of TABLE-CHART, the system further refines the design of the abstract table chart by designing its table cells. First, IMPROVISE needs to decide what information should be grouped together to fit into one cell. The grouping information may also be provided by a high-level general content planner [Dalal et al., 1996b]. In our case, based on the domain knowledge, three groups will be formed: *age*, *gender*, and *mrn* in one group as general identification information, *medical history* by itself in one, and *surgeon* and *operation* in the third group to describe the operation. In other words, the abstract table chart is refined as having three cells, each of which conveys part of the demographics information.

As each group is still a composite data object (e.g., group1 has three data items), its representation has not yet been determined. Based on the reason similar to why a table chart is selected to represent demographics information, table charts again are used to represent each

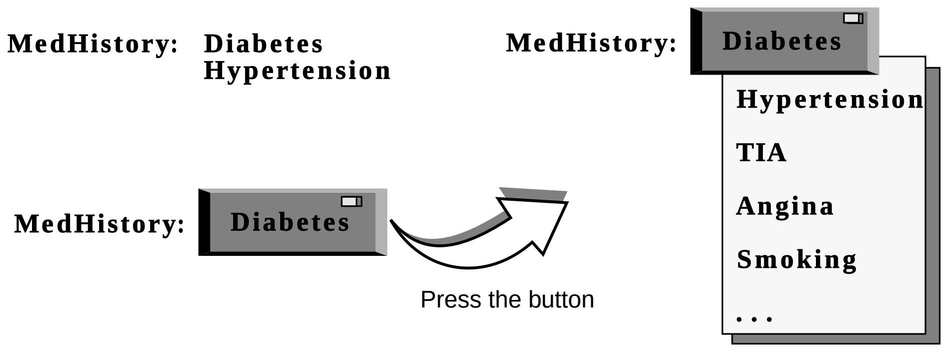


Figure 4-5 Visual representations for ATTRIBUTE object

of the three groups. In other words, each table cell itself in the abstract table chart will be represented as a table chart. Continuing with the same top-down refinement, each constituent is eventually refined to the point where IMPROVISE needs to determine the representation of each individual item (e.g., *age*) in the chart.

At this point, IMPROVISE will search through the visual lexicon to find possible representations for each individual item as each data item is already an atomic data object. To avoid constructing every visual presentation from scratch, visual lexicon prestores the possible visual representations for various *atomic* objects. Instantiations of the *visual lexeme* could be directly used as building blocks in a design. For example, the visual lexicon entry for an atomic ATTRIBUTE object such as *age* is shown in Figure 4-4.

As specified in Figure 4-4, an ATTRIBUTE object can be represented in one of two forms shown in Figure 4-5. In sense1 the object is simply represented in an attribute-value pair format, with both attribute name and value represented as simple text strings. However, in sense2, the attribute values are represented as entries in an option button, which indicates that the current attribute has a large number of values (e.g., more than 3 values). The first attribute value (e.g., *diabetes*) is displayed as the button label and is always visible, while the rest are hidden and arranged as entries in a pull-down menu. That may be brought up by pressing the button (Figure 4-5). In our case, IMPROVISE chooses the first sense since the ATTRIBUTE objects involved are either single valued (e.g., *age* and *gender*) or double valued (e.g., *medical history*).

In summary, a tree-like structure diagram in Figure 4-6 can be used to illustrate the top-down hierarchical planning process conducted by IMPROVISE so far: at a high level, IMPROVISE refines the abstract table to a three-cell table. Each table cell in turn is a table chart itself. Moreover, each sub-table is made up of the visual representations of individual data items in that group. In this example, each individual item happens to be represented by a simple table chart again. The recursive relationships among various levels of table chart representations can be indicated by annotating the automatically generated display as shown in Figure 4-7.

Notice that nothing has been said about the layout of the table charts, nor the spacing among the table chart cells. The determination of this type of parameter is carried out based on various syntactic, semantic and pragmatic constraints. For example, one constraint might state that the spacing between the cells in a table chart should be larger than the same directional spacing between the sub-table cells (e.g., spacing *x* should shorter than spacing *y* in

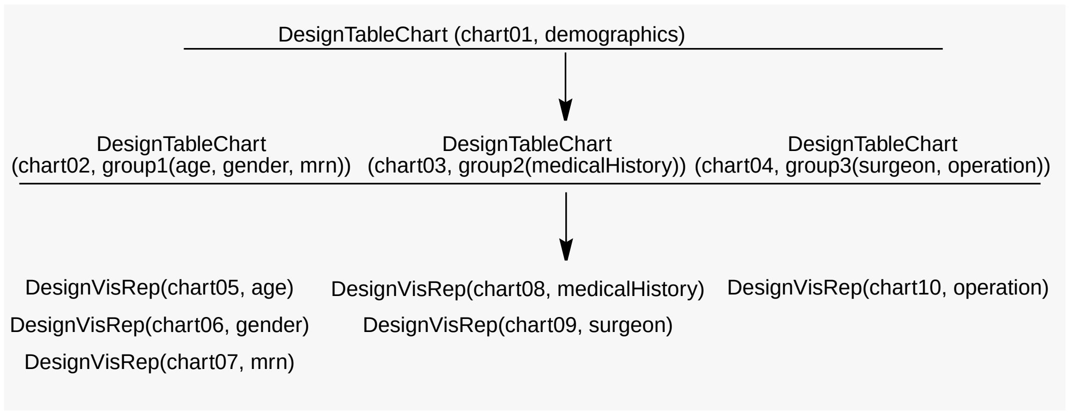

DesignTableChart (chart01, demographics)
DesignTableChart (chart02, group1(age, gender, mrn))
DesignTableChart (chart03, group2(medicalHistory))
DesignTableChart (chart04, group3(surgeon, operation))
DesignVisRep(chart05, age)
DesignVisRep(chart06, gender)
DesignVisRep(chart07, mrn)
DesignVisRep(chart08, medicalHistory)
DesignVisRep(chart09, surgeon)
DesignVisRep(chart10, operation)

Figure 4-6 Planning levels for displaying demographics information

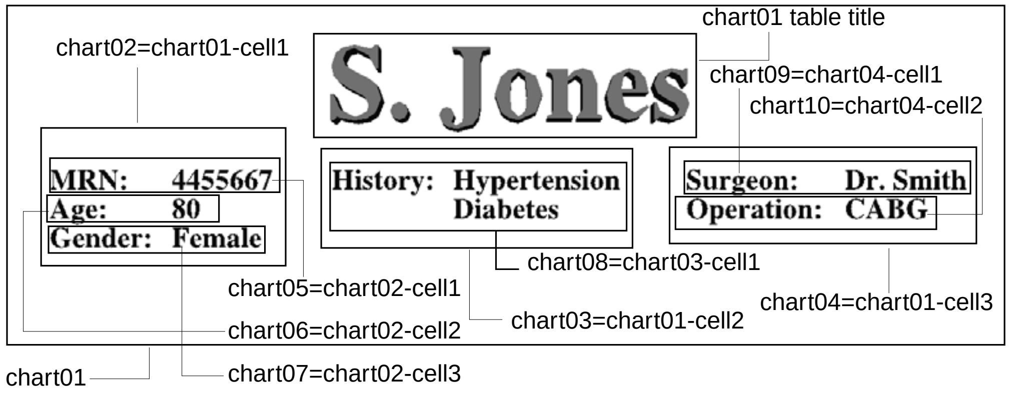


Figure 4-7 Automated generated display annotated by its structural representation

Figure 4-8). Although this constraint appears to be one of the syntactic constraints used in table chart construction, it is not hard to justify the semantic rationale behind this constraint: domain objects having closer relationships should be placed closer to each other to justify *gestalt* grouping theory. In our example, the object *age* is represented by its attribute name Age and its value 80. The relationship between the attribute name and its own value is considered an *intra-relationship* within a concept, and it is certainly closer than any *inter-relationship* existing outside of the concept (e.g., between concept *age* and *medical history*).

Eventually all parameters such as font size, font color are determined, and a complete design for conveying a patient's demographics information is rendered as part of the complete presentation as shown in Figure 4-3.

4.3.2 Network Domain Example

Having just demonstrated how IMPROVISE composes a visual presentation from scratch, here we show how IMPROVISE designs new visual presentations by transforming an existing presentation. This example comes from one of the tasks in the computer network management application. 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. Here, we describe how a visual discourse is planned to fulfill the task of revealing the internals of a selected link.

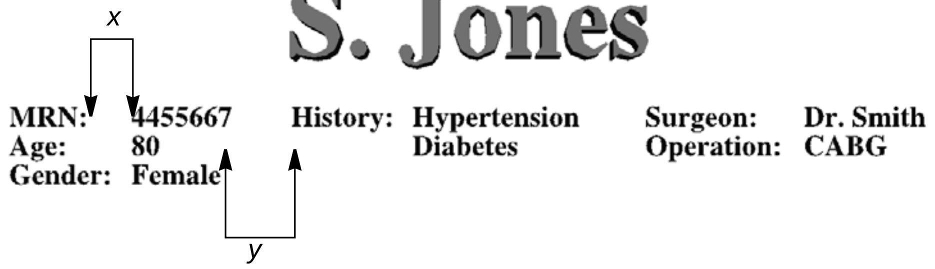


Figure 4-8 Spacing between different table charts

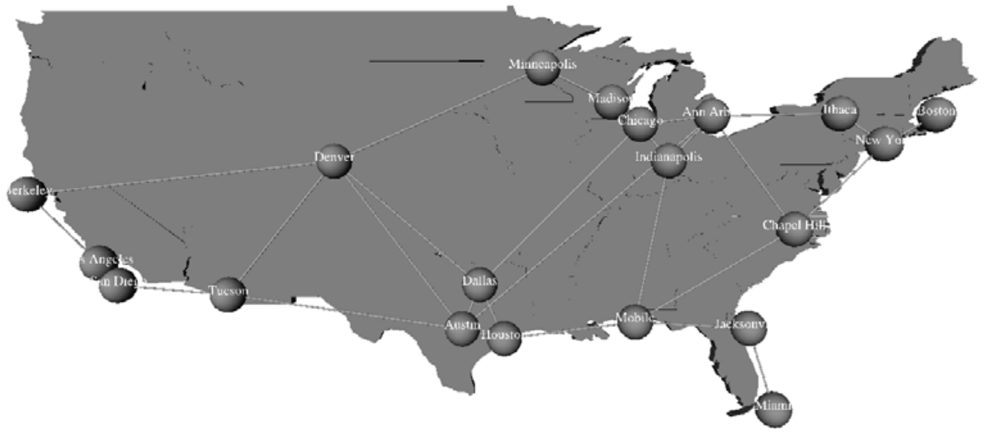


Figure 4-9 Visual representation of a computer network

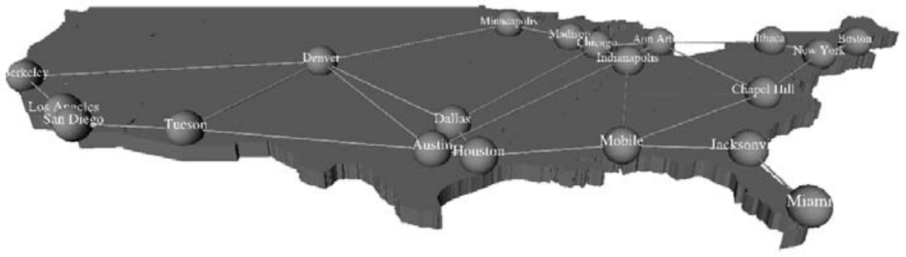
Suppose that a computer network is displayed on the screen as shown in Figure 4-9. The user has selected link23, the link between Austin(node1), and Tucson (node5). The system formulates the communicative task as: (Task (ShowInternal link23)).

The planner seeks visual actions or high-level visual tasks that have an effect 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 shown in Figure A-9. In our example, the selected link's bounding box is too big to satisfy the preference condition for DecompositionSchema2. 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).

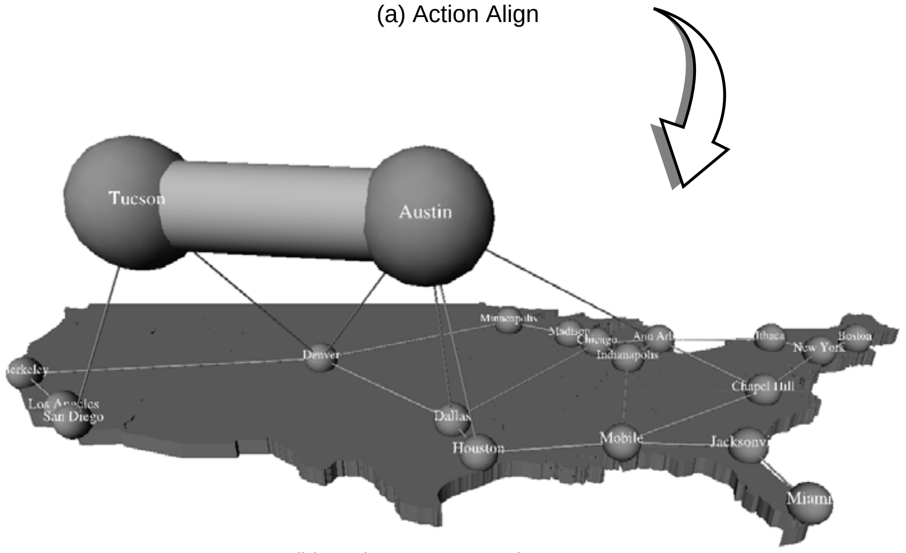
Among the three subactions, only action *Open* is a primitive action; the other two are composite actions and need to be further decomposed. *Focus* is decomposed into *Extract* (extract link23 from the rest of the network). This decision is made based on the fact that a link is likely to intersect with 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. The first is concerned with designing the visual representations of the internal objects, while the second takes care of building visual displays for representing the attributive relationships between the internal objects and their capacities. The process of building the visual displays for the internal objects and their capacities is similar to the process described in the previous example. Actions of designing visual presentations for composite objects (e.g., a set of virtual path segments) are decomposed to subactions that determine the visual presentations for their components (e.g., individual virtual path segments). At the lowest level, the process uses the visual lexicon to decide the visual representations for atomic objects. In our network domain, one visual word specifies the shape (cylinder) for 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 from scratch.

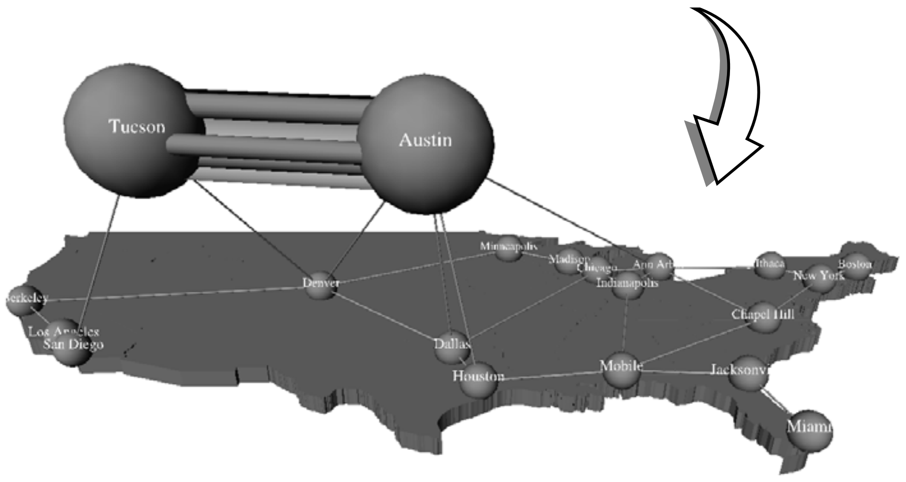
At this point in planning, let us assume all preconditions but one are satisfied and no



(a) Action Align



(b) Action Move + Enlarge + . . .



(c) Action Open

Figure 4-10 Series frames for the network domain example

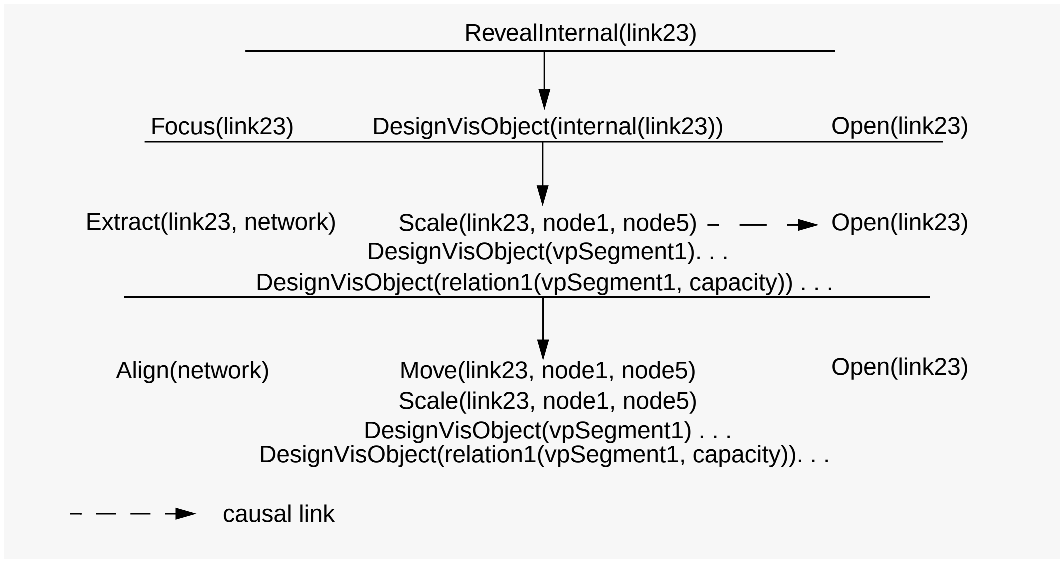

RevealInternal(link23)
Focus(link23)
DesignVisObject(internal(link23))
Open(link23)
Extract(link23, network)
Scale(link23, node1, node5)
Open(link23)
DesignVisObject(vpSegment1). . .
DesignVisObject(relation1(vpSegment1, capacity)) . . .
Align(network)
Move(link23, node1, node5)
Open(link23)
Scale(link23, node1, node5)
DesignVisObject(vpSegment1) . . .
DesignVisObject(relation1(vpSegment1, capacity)). . .
causal link

Figure 4-11 Planning levels for the network domain example

conflicts need to be resolved. The 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 4-10a), while *Move* moves the link and its two end nodes away from the rest of the network (Figure 4-10b).

Figure 4-11 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*, and *Scale* 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 (Figure 4-10b) in this example. The screen shots for several keyframes resulted in the design can be seen in Figure 4-10(a)-(c).

4.3.3 A Hypothetical Example

In this section, we demonstrate what IMPROVISE can not design at current stage, but it would be able to accomplish in the future. Through an example, we illustrate the current missing parts of IMPROVISE, and indicate the related future research tasks yet to be carried out.

We hypothesize an example in the medical domain to inform the nurse about a patient’s information. However, this example significantly differs from the previous patient example in terms of its complexity of the required visual design. As shown in Figure 4-12, we need to show the physical body of patient Baker and his medical information arranged around the body. In this case, IMPROVISE needs to show both front and back view of the body. IMPROVISE currently can manage to reveal various aspects of the body *sequentially* by employing transformation actions such as *Rotate*. However, this is not the case. In this exam-

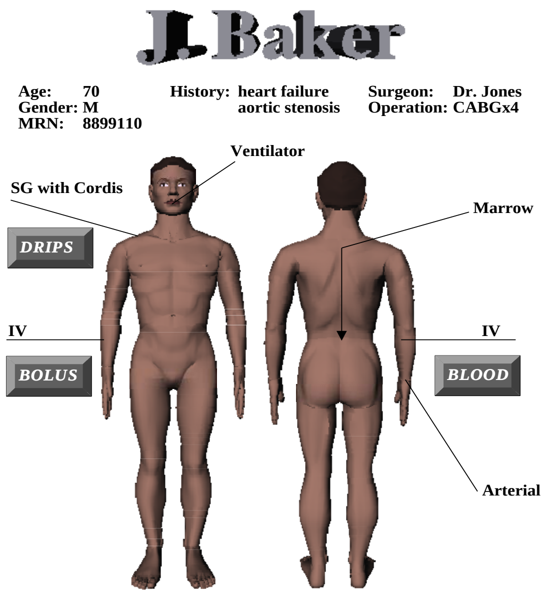


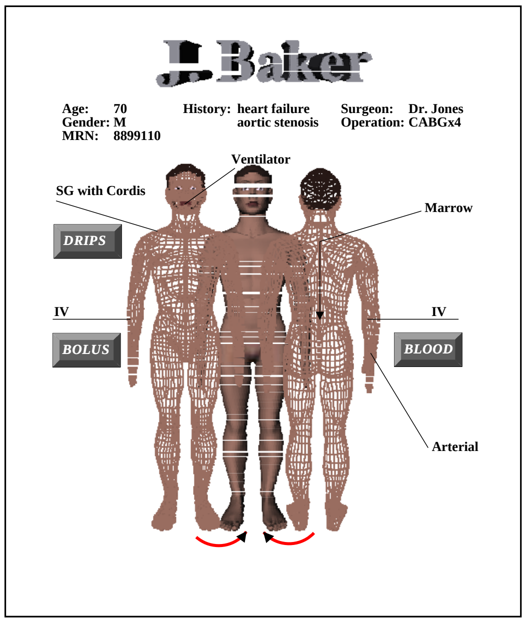
Figure 4-12 A hypothetical example in medical domain

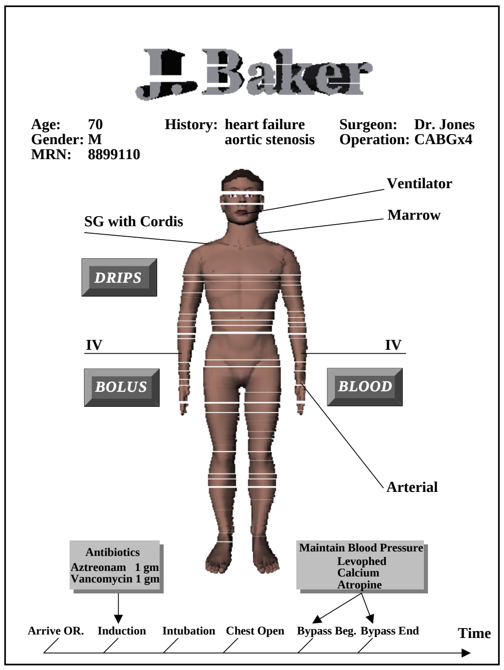
ple, IMPROVISE needs to show both views of the body *simutaneously* since nurses prefer to see all the information layed out *at once*. This poses a problem in the current planning paradigm since IMPROVISE only knows *one* patient Baker and *one* physical body of his. To generate multiple presentations for one object, IMPROVISE needs to know that the visual representation of the patient can be *cloned* with different foci of interest. The special actions, such as *Clone*, need to be developed to handle these complicated situations. Not only should IMPROVISE be able to clone the visual representation of an object, but also need to *merge* an object's various visual presentations into one. In our example, at certain point (Figure 4-13a) IMPROVISE might need to combine the two views together to produce *one* coherent presentation. Obviously, a complete version of IMPROVISE should also automatically determine the spatial layout of the picture using STM, while current IMPROVISE cannot.

As shown in Figure 4-13(b), IMPROVISE chooses to use a TIME-CHART to convey the drug usage. It indicates the time and type of the drug being administered. Currently, IMPROVISE has no knowledge about how to design a TIME-CHART. We need include TIME-CHART definition in our visual object catalogue.

Ideally, IMPROVISE should allow the user to interrupt the ongoing presentation at anytime; for example, the user might stop the presentation and press the drip button to obtain more information. However, this requires research in the area of reactive planning and adds another future research topic on our to-do list.

Based on the framework, we have described IMPROVISE, a prototype system that can automatically generate coherent visual discourse in two application domains. We have discussed the basic functionality and the missing parts of IMPROVISE in detail. From the system architecture, it is easy to discover how the four core components illustrated in the





(a) Combining various representations (b) Using a time chart to describe drugs

Figure 4-13 IMPROVISE should create various representations

general framework are coordinated to work together and provide the backbone of this system. We also presented two very different examples to demonstrate the flexibility and extensibility of IMPROVISE. Although the core components of IMPROVISE have been built, a complete working system requires further work in a few areas (e.g., handle user request during the presentation) as indicated by the last example. This leads us to the next chapter where we will summarize these future tasks and the approximate time frame, during which the desired features would be completed.

CHAPTER 5 Proposed Plan

If we knew what we were doing and what we are going to do it wouldn’t be research. — Unknown

While major components of IMPROVISE are in place, there are several areas that still need to be developed or improved.

5.1 Task Characterization

A formal and comprehensive task characterization scheme should include both presentation task (i.e., communicative goals) and visual task characterization. While we have established a taxonomy for classifying and studying various visual tasks, we still need to refine further on characterizing presentation tasks and their relations to visual tasks. As the analysis of communicative tasks largely depends on the application domain, we will put our main focus on refining the current visual task hierarchy and studying the relationships between visual tasks and low-level visual techniques.

5.1.1 Visual Task Characterization

Although we have established a visual task taxonomy, some of the classifications need to be further refined. The relationships between these tasks and the low-level visual techniques need to be carefully analyzed and summarized. Out of the fifteen types of visual tasks, we will concentrate on a subset of them including *Associate*, *Emphasize*, *Reveal*, and *Encode* since these visual tasks are used most often in visual presentations.

5.1.2 Representative Presentation Task Study

Researchers from different areas (e.g., [McKeown, 1985; Mann and Thompson, 1988]) have proposed various characterization taxonomies to categorize presentation tasks (communicative tasks). However, since much of this research has focused on speech acts instead of visual acts, the proposed taxonomy needs to be analyzed and relationships between the general-purpose communicative tasks and visual tasks need to be established in visual communication environments. In addition, we could also enrich these presentation tasks by employing the task-related knowledge structures introduced in [Johnson et al., 1988]. For the purpose of our application domains, we will only concentrate on the presentation task *Inform* and analyze its decompositions and its visual interpretations using corresponding visual tasks.

5.2 Visual Language Formal Specification

Various visual objects (e.g., visual structure and visual frame) have been defined and used in IMPROVISE's design process. However, some of the design rules have not been defined as rigorously as they could be. To take advantage of the existing formal specification methods, various visual object specifications, in particular their semantics and pragmatics will be modified and described using a formal specification [Rich and Knight, 1991; Russell and Norvig, 1995]. We believe that a formal specification could aid IMRPOVISE in planning a design more systematically and in justifying its decisions.

This work will be based on the current visual object specification. The resulting formal specification is a refined version of the current one and will be written out in a formal specification language.

To refine the current specification and establish a more precise evaluation of current design principles, we need to conduct further research in integrating professional visual design and film-making theory into our system. As an addition to our current visual object catalogue, we would also like to implement a time-chart like visual structure to demonstrate how to integrate new visual knowledge into the current system.

5.3 Constraint Specification

To automatically layout graphic elements, IMPROVISE needs to specify various numerical constraints at different levels of abstraction, and have them solved by a constraint solver. We have integrated STM (Snap-Together Mathematics) [Gleicher, 1994] into IMPROVISE as our numerical constraint solver. However, we need to determine how much preprocessing is necessary to provide a good starting point for STM.

5.4 Visual Action Hierarchy Extension

Within one visual presentation, it is common that different aspects of the same object are required to be conveyed. Suppose we need to describe two features of a car: the trunk and the interior. This would not be a problem if we could display the features of the car *sequentially*. We can show the interior of the car first, then smoothly change the camera view of the car and turn it to the car's trunk. However, this will pose a problem in a planning process if the two features needs to be displayed *simultaneously* since the system only knows *one* car, which usually has *one* visual appearance at one time. In this particular problem, the system needs to know that the visual representation of an object can be *cloned* with different foci of interest. The special actions, such as *Clone*, will be used to handle complicated situations. These actions are to be defined and implemented. In particular, three extremely useful actions are:

Clone. Make copies of a visual object, but all copies have the same identity. This type of action is suitable to show various aspects of the *same* object.

Separate. Make copies of a visual object, but each individual copy has its own identity. This type of action is used to describe different objects that have certain common aspects. The main purpose of this action is to separate individuals from a generic representation. For example, if we use a human model to represent a generic concept of patient, then multiple copies of the model could be used or transformed to represent individual patient as the general patient model is separated into different patient models.

Merge. *Merge* is the opposite of *Separate*. Various visual objects with certain common attributes could be collapsed together to represent the abstraction of such a group.

5.5 Reactive Planning

As we mentioned in Chapter 3 and 4, user interaction should be allowed during the presentation to enhance the controllability of the presentation style. To respond to user events during the presentation and modify the current presentation accordingly is the area of research known as reactive planning [Wilkins et al., 1994]. As reactive planning is an open problem itself, we will determine an appropriate reactive planning strategy to use and incorporate it into the current planning algorithm so that limited user interactions could be handled properly: allowing user to stop the ongoing presentation and resume the presentation where it is left off.

5.6 Time Table

Figure 5-1 is the proposed timeline for the future tasks. As most of the tasks are independent of each other, we choose to start some of the simpler tasks (e.g., task characterization,

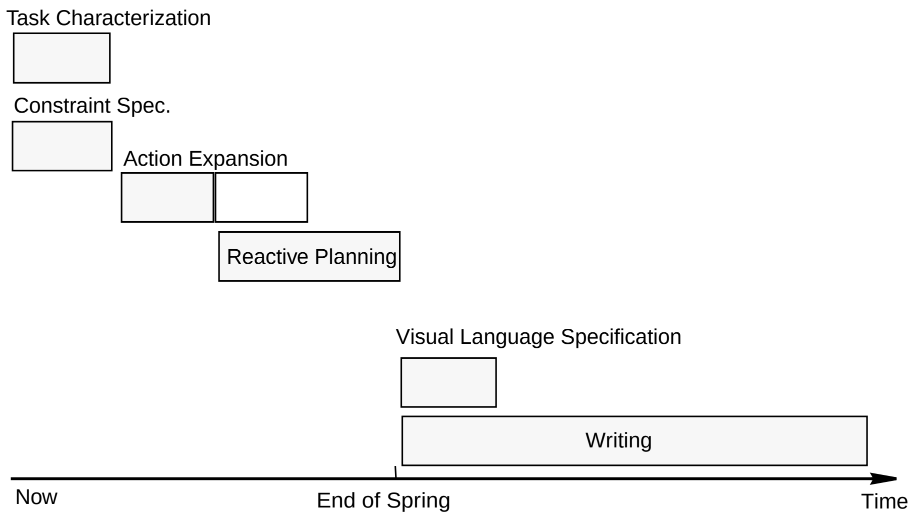


Figure 5-1 Proposed schedule

and constraint specification) first so that IMPROVISE is capable of performing basic graphic design (e.g., layout graphic elements). Those tasks (e.g., action expansion, and reactive planning) that require a considerable amount of research and will enhance IMPROVISE's design capabilities substantially are scheduled after IMPROVISE's basic functions are achieved. The following time table records the estimated time frame for each of the tasks mentioned above.

Task	**Estimated Time**
Task Characterization	1/2 month
Visual Language Formal Specification	1 months
Constraint Specification	1/2 months
Extend Visual Action Hierarchy	1-2 months
Reactive Planning	2 months
Writing	6 months

Table 5-1 Estimated Time Table

Bibliography

Arens, Y., Hovy, E., and Vossers, M. (1993). The knowledge underlying multimedia presentations. In Maybury, M., editor, *Intelligent Multimedia Interfaces*, chapter 12, pages 280–306. AAAI Press/The MIT Press, Menlo Park, CA.

Barr, A. and Feigenbaum, E., editors (1989). *The Handbook of Artificial Intelligence*, volume 1, chapter II. Search. Addison-Wesley, Reading, MA.

Bertin, J. (1983). *Semiology of Graphics*. Univ. of Wisconsin Press, Madison, WI. (trans. by W.J. Berg).

Beshers, C. and Feiner, S. (1994). Automated design of data visualizations. In Rosenblum, L., Earnshaw, R., Encarnacao, J., Hagen, H., Kaufman, A., Kilmenko, S., Nielson, G., Post, F., and Thalmann., D., editors, *Scientific Visualization: Advances and Challenges*, chapter 6, pages 87–102. IEEE Computer Society Press, London.

Bobrow, D. and Winograd, T. (1977). An overview of KRL, another perspective. *Cognitive Science*, 3:29–42.

Casner, S. (1991). A task-analytic approach to the automated design of graphic presentations. *ACM Trans. on Graphics*, 10(2):111–151.

Chan, M., Pacifici, G., and Stadler, R. (1995). Managing real-time services in multimedia networks using dynamic visualization and high-level controls. In *Proc. ACM Multimedia '95*, San Francisco, CA. ACM.

Chang, B. and Unger, D. (1993). Animation: From cartoons to the user interface. In *Proc. UIST '93*, pages 45–55, Atlanta, GA. ACM SIGGRAPH and SIGHCI.

Cleveland, W. and McGill, R. (1984). Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79:531–554.

Cleveland, W. and McGill, R. (1985). Graphical perception and graphical methods for analyzing scientific data. *Science*, 229:828–833.

Cohen, P. and Feigenbaum, E., editors (1989). *The Handbook of Artificial Intelligence*, volume 3, chapter XV. Planning and Problem Solving. Addison-Wesley, Reading, MA.

Crutcher, L., Lazar, A., Feiner, S., and Zhou, M. (1995). Managing networks through a virtual world. *IEEE Parallel and Distributed Technology*, 3(2):4–13.

Dalal, M., Feiner, S., McKeown, K., Jordan, D., Allen, B., and alSafadi, Y. (1996a). MAGIC: An experimental system for generating multimedia briefings about post-bypass patient status. In *Proc. 1996 AMIA Annual Fall Symp*, pages 684–688, Washington, DC.

Dalal, M., Feiner, S., McKeown, K., Pan, S., Zhou, M., Hoellerer, T., Shaw, J., Feng, Y., and Fromer, J. (1996b). Negotiation for automated generation of temporal multimedia presentations. In *Proc. ACM Multimedia '96*, pages 55–64, Boston, MA.

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

Foley, J., van Dam, A., Feiner, S., and Hughes, J. (1990). *Computer Graphics: Principles and Practice, Second Edition*. Addison-Wesley Publishing Company, Reading, MA.

Friedell, M. (1983). *Automatic Graphics Environment Synthesis*. PhD thesis, Case Western eserve University.

Friedell, M. (1984). Automatic synethsis of graphical object description. *Computer Graphics: (Proc. ACM*

SIGGRAPH'84), 18(3):53–62.

Gibson, J. (1977). Notes on direct perception and indirect apprehension. In Reed, E. and Jones, R., editors, *Reasons for Realism: Selected Essays of James J. Gibson*, pages 289–293. Lawrence Erlbaum Associates, Hillsdale, NJ.

Gleicher, M. (1994). *A Differential Approach to Graphical Interaction*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891.

Goldsmith, E. (1984). *Research Into Illustration: An Approach and A Review.* Cambridge University Press, Cambridge.

Grosz, B. and Sidner, C. (1986). Attention, intensions, and the structure of discourse. *Journal of ACL*, 2(3):175–204.

Jackson, P. (1990). *Introduction to Expert Systems*. Addison-Wesley, Wokingham, England, 2 edition.

Johnson, P., Johnson, H., Waddington, R., and Shouls, A. (1988). Task-related knowledge structures: Analysis, modelling, and application. In Jones, D. and Winder, R., editors, *People and Computers IV, Proc. of the Fourth Conference of the British Computer Society Human-Computer Interaction Specialist Group*, pages 35–61. Cambridge University Press, Cambridge.

Karp, P. and Feiner, S. (1990). Issues in the automated generation of animated presentations. In *Proceedings of Graphics Interface '90*, pages 39–48.

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

Keller, P. R. and Keller, M. M. (1993). *Visual Cues: Practical Data Visualization*. IEEE Computer Society Press and IEEE Press.

Knoblock, C. (1993). *Generating Abstraction Hierarchies—An Automated Approach to Reducing Search in Planning*. Kluwer Academic Publishers, Boston, MA.

Kosslyn, S. (1989). Understanding charts and graphs. *Applied Cognitive Psychology*, 3:185–226.

KRSL (1993). *Knowledge Representation Specification Language Reference Manual*. DARPA/Rome Laboratory Planning and Scheduling Initiative Knowledge Representation and Architecture Issue Working Group. Version 2.0.2.

Levin, J., Anglin, G., and Carney, R. (1987). On empirically validating functions of pictures in prose. In Willows, D. and Houghton, H., editors, *The Psychology of Illustration: Basic Research*, volume 1, chapter 2, pages 51–86. Springer-Verlag, New York.

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

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

Mann, W. and Thompson, S. (1988). Rhetorical structure theory: Towards a functional theory of text organization. In *TEXT*, volume B, pages 243–281. Springer.

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

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

McKeown, K. (1985). *Text Generation*. Cambridge University Press, London.

Minsky, M. (1975). A framework for representing knowledge. In Winston, P., editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, NY.

Mullet, K. and Sano, D. (1995). *Designing Visual Interfaces*. SunSoft Press, Mountain View, CA.

Neal, J. and Shapiro, S. (1991). Intelligent multimedia interface technology. In Sullivan, J. and Tyler, S., editors, *Intelligent User Interfaces*, chapter 2, pages 11–43. ACM Press/Addison Wesley, New York, NY/ Reading, MA.

Neider, J., Davis, T., and Woo, M. (1993). *OpenGL Programming Guide*. Addison-Wesley, Reading, MA.

Newman, W. and Lamming, M. (1995). *Interactive System Design*. Addison-Wesley Publishing Company, Reading, MA.

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

Pomerantz, J. (1981). Perceptual organization in information processing. In Kubovy, M. and Pomerantz, J., editors, *Perceptual Organization*, chapter 6, pages 141–180. Lawrence Erlbaum Associates, Hillsdale, NJ.

Reichgelt, H. (1991). *Knowledge Representation: An AI perspective*. Ablex Publishing Corp., Norwood, NJ.

Rich, E. and Knight, K. (1991). *Artificial Intelligence*. McGraw-Hill, Inc., second edition.

Robertson, P. (1991). A methodology for choosing data representations. *IEEE Computer Graphics and Applications*, 11(3):56–67.

Roth, S. and Hefley, W. (1993). Intelligent multimedia presentation systems: Research and principles. In Maybury, M., editor, *Intelligent Multimedia Interfaces*, chapter 1, pages 13–58. AAAI Press/The MIT Press, Menlo Park, CA.

Roth, S. F. and Mattis, J. (1990). Data characterization for intelligent graphics presentation. In *Proc. of SIGHCI '90*, pages 193–200, New Orleans, LA. ACM/SIGCHI.

Roth, S. F. and Mattis, J. (1991). Automating the presentation of information. In *Proc. IEEE Conf. on AI Applications*, pages 90–97.

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

Schank, R. (1975). *Conceptual Information Processing*. North Holland, Amsterdam.

Seligmann, D. (1993). *A Visual Language for Automated 3D Graphics Generation*. PhD thesis, Columbia University, New York, NY.

Seligmann, D. and Feiner, S. (1989). Specifying composite illustrations with communicative goals. In *Proc. UIST '89*, pages 1–9, Williamsburg, VA. ACM.

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

Senay, H. and Ignatius, E. (1994). A knowledge-based system for visualization design. *IEEE Computer Graphics and Applications*, 14(6):36–47.

Shneiderman, B. (1992). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Publishing Company, Reading, MA, second edition.

Stefik, M. (1979). An examination of a frame-structured representation system. In *Proc. IJCAI '79*, volume 6, pages 845–852.

Sutcliffe, A. and Darzentas, J. (1994). Use of visual media in explanation. In Tauber, M., Mahling, D., and Arefi, F., editors, *Cognitive Aspects of Visual Languages and Visual Interfaces*, chapter 2, pages 105–132. Elsevier Science B.V., Amsterdam.

Treisman, A. (1982). Perceptual grouping and attention in visual search for features and for objects. *Journal of Experimental Psychology: Human Perception and Performance*, 8(2):194–214.

Tufte, E. R. (1983). *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT.

Tufte, E. R. (1990). *Envisioning Information*. Graphics Press, Cheshire, CT.

VRML (1996). *The Virtual Reality Modeling Language Specification*. The VRML Architecture Group. Version 2.0.

Wehrend, R. and Lewis, C. (1990). A problem-oriented classification of visualization techniques. In *Proceedings of the First IEEE Conference on Visualization: Visualization 90*, pages 139–143. IEEE, Los Alamitos, CA.

Weld, D. (1994). An introduction to least commitment planning. *AI Magazine*.

Wernecke, J. (1994). *The Inventor Mentor: Programming Object-Oriented 3D graphics with Open Inventor.* Addison Wesley, Reading, MA.

Wilkins, D. (1988). *Practical Planning: Extending Classical AI Paradigm*. Morgan Kaufmann, San Mateo, CA.

Wilkins, D., Myers, K., Lowrance, J., and L., W. (1994). Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 6:197–227.

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

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

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

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

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

Zdybel, F., Greenfeld, N., Yonke, M., and Gibbons, J. (1981). An information presentation system. In *Proceedings of IJCAI '81*, pages 978–984, Vancouver, Canada.

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

Zhou, M. and Feiner, S. (1996). Data characterization for automatically visualizing heterogeneous information. In *Proc. IEEE InfoVis'96*, pages 13–20, San Francisco, CA.

Zhou, M. and Feiner, S. (1997a). The representation and use of a visual lexicon for automated graphics generation. In *Proc. IJCAI'97*, Nagoya, Japan. To appear.

Zhou, M. and Feiner, S. (1997b). Top-down hierarchical planning of coherent visual discourse. In *Proc. IUI'97*, pages 129–136, Orlando, FL.

APPENDIX A Knowledge Base

Here, we describe the detail content of various knowledge sources and how they are represented in the knowledge base.

A.1 Knowledge Sources

As mentioned in Chapter 3, there are at least four types of knowledge that are needed to construct automated graphics generation systems. They are: domain knowledge, visual design knowledge, situation knowledge, and meta knowledge.

A.1.1 Domain Knowledge

Domain knowledge refers to information that needs to be presented visually. Such information could vary a great deal (e.g., quantitative vs. qualitative, or static vs. dynamic). Nevertheless, there are certain presentation-related properties that are common to all domain information. Rather than enumerate the individual properties (e.g., a person's age) associated with a specific piece of information (e.g., a person), we concentrate on the presentation-related properties that are common to all information. The task of understanding which information properties are related to presentation design and how they are related is called *data characterization* [Roth and Mattis, 1990]. We have defined a systematic and comprehensive data characterization method in which a data-analysis taxonomy is developed to characterize heterogeneous information [Zhou and Feiner, 1996]. Based on this taxonomy, we discuss how domain information is organized and represented to reflect the presentation-related properties and facilitate the design process.

Although many researchers (e.g., [Roth and Hefley, 1993]) have realized that data characterization is an important part of automated graphics design, other work on data characterization either focuses on a particular type of information (e.g., quantitative data characterization in [Mackinlay, 1986; Roth and Mattis, 1990]) or simply establishes a conceptual taxonomy (e.g., [Wehrend and Lewis, 1990; Arens et al., 1993]). In contrast, our data characterization taxonomy not only comprehensively captures properties of heterogeneous information, but has also been used in a practical system to demonstrate its usability and coverage (see Chapter 4).

As general human knowledge is quite complex in structure and abundant in quantity, we could not possibly develop a hierarchy to exhaust every type of information or every property of information. Instead, we have developed a taxonomy that is flexible enough to easily accommodate new types of the information, or new properties of information. Our taxonomy describes data characteristics along six dimensions: *data type*, *data domain*, *data attributes*, *data relations*, *data role*, and *data sense*. *Data type* distinguishes a piece of information by whether or not it is divisible into smaller pieces. *Data domain* categorizes information in a semantic taxonomy; for example, whether or not it is a physical entity. *Data attributes* differentiate data based on properties such as an object's shape, and *data relations* specify connections among data. The last two dimensions characterize data pragmatically based on user information-seeking goals (*data role*) and user visual-interpretation preferences (*data sense*). Based on this taxonomy, we describe what is stored in every piece of domain information along with their unique domain-specific properties.

Next, we describe each dimension in detail.

Data Types

There are two basic data types: *atomic* and *composite*.

Atomic. An *atomic* object is the most basic, indivisible data unit.

Composite. A *composite* object is a combination of atomic or composite objects. Composite objects can be further classified as *sets* and *structures*, based on the structural relations among their parts. A *set* is a composite object in which each component is considered independent of each other. In contrast, in a *structure,* components are related to each other.

To store the *type* information helps to control the design process since the object type usually signifies whether a particular design is done (see more in chapter 4). For example, atomic objects usually appear at the leaves of the design tree and signals that the design process should stop. Moreover, knowledge of the different characteristics of composite objects can facilitate the construction of displays to represent them. Usually, the preferred visual representation for the structural relations among objects has structural properties based on those of the data. For example, a set might be presented using a table, but typically not a flow chart. In contrast, a structure might be presented by a network diagram.

Data Domain

To construct an effective visual presentation, it is often important to take into account the data's domain, or the general ontology, which usually is used to categorize a broad range of objects and relations [KRSL, 1993; Russell and Norvig, 1995]. Roth and Mattis [Roth and Mattis, 1990] characterize data with regard to quantitative domains such as time, space, temperature, and mass; for example, to allow temperature to be preferentially displayed along a vertical axis. We generalize this classification to accommodate a broader range of information. Each object belongs to one of the following top-level domains, which can themselves be further subdivided: *entity*, *concept*, *measurement* and *event.*

Entity. An *entity* is an object that exists independently and has a unique identity. An entity may correspond to a physical object in the real world (e.g., a car) or may be depicted as a physical object (e.g., a dragon).

Concept. A *concept* is an abstract idea that either exists independently by itself or must be attached to other objects. Unlike entities, concepts are not physical objects. For example, the abstract notion of "age" is independent of any object, while a particular person's "age" is a concept that is attached to that person to specify how old the person is. The quantitative domains identified by Roth and Mattis would be among the subclasses of concept.

Measurement. A *measurement* is a numeric or non-numeric value with or without its unit of measure. A measurement cannot exist by itself, but must be related to a concept, such as weight. For example, we can use "120 lbs." to specify a person's weight *quantitatively*, or "good" to specify a car's performance *qualitatively.*

Event. *Event* is a composite object that describes something happened at given location or time. Event could be further distinguished by the cause of the event: *natural event* and *activity. Natural event* is driven by the natural law and without human as its direct participant, for example, a raining event. In contrast, *activity* usually refers to human actions—or event conducted by human beings, for example, a surgical operation.

Data Attributes

Although, at first glance, information and its visual representations may appear to differ greatly from one application domain to another, there are basic connections between the nature of the information (its attributes) and the presentation style. Therefore, presentation-related properties can be abstracted for all types of information. Some properties are shared by different types of data, while others are unique to a particular data type. First, we describe the data attributes that are common to all data types: *form*, *material*, *location*, *transience*, and *importance.*

Form. *Form* is related to shape and may be one of*: shaped*, *shapeless*, or *none.* A *shaped* form refers to a category of objects that have solid, physical shapes in the real world, such as a person or a car. A *shapeless* object might have a shape under certain circumstances, but its shape changes so often or is so abstract that it is hard to depict the object's precise shape. An example of a shapeless object is water. All conceptual objects use *none* as their form value.

Material. *Material* refers to the possibly complex set of appearance variables that determine how an object interacts with light. Material models can range from simple RGB color through sophisticated physical illumination models. Material can often be used effectively to encode other nonvisual attributes of the data.

Location. While the *form* and *material* attributes determine the possible visual appearance for the data object, *location* affects the graphic design decision as to where and when to display the object. There are two types of locations: *spatial* and *temporal.* Spatial location is either represented by absolute or relative quantitative coordinates (e.g., numeric 2D or 3D coordinates) or by absolute or relative qualitative location (e.g., “in front of” or “near”). Similarly, temporal location can be represented by absolute or relative, quantitative or qualitative, values.

Transience and *importance* are used to describe the intrinsic properties of the information mentioned in [Arens et al., 1993]. These are especially useful for grouping or partitioning information when a large amount of information is to be conveyed.

Transience. Objects can be either *dynamic* or *static*. Dynamic information varies over time, while static information remains constant. This expresses the same notion as *live* or *dead* [Arens et al., 1993].

Importance. The fact that some information is more important than others allows us to select different graphical techniques to visually group or partition information. Importance can be characterized using several distinctions (e.g., *urgency* vs. *routine* [Arens et al., 1993], *abnormality* vs. *normality*).

Other attributes are tied directly to composite objects only: *ordering*, *scalability*, and *continuity*. We adopt the attribute categorizations discussed in [Roth and Mattis, 1990] and [Arens et al., 1993], and extend them to address a wider range of information.

Ordering. There are two possible orderings among components of a composite object: *quantitative* and *qualitative*. Quantitative ordering includes both numeric ordering and non-numeric ordering, which correspond to the *quantitative* and *ordinal* notions mentioned in [Mackinlay, 1986]. In our characterization, numeric ordering can be further distinguished as *quantitative individual* or *quantitative range*. For example, the dosages for a set of drugs form a quantitative individual set; in contrast, the price ranges for a set of drugs form a quantitative range set. Non-numeric ordering for orderable qualitative values is called *ordinal* (e.g., a range from “poor” to “excellent”). In contrast, qualitative ordering is not an inherent ordering. Instead, it refers to a logical ordering that is imposed on the objects by the domain application. To avoid introducing new terminology to distinguish qualitative orderings, we overload the terms used in identifying quantitative ordering. We call qualitative sets *nominal* if they are non-ordered (e.g., a list of names) and *ordinal* if they have a logical ordering among their components.

How quantitative ordering can automatically determine graphic design decisions has been carefully examined in both [Mackinlay, 1986] and [Roth and Mattis, 1990]. Qualitative ordering can also influence the spatial and temporal order of an information presentation [Zhou and Feiner, 1996].

Scalability. Roth and Mattis [Roth and Mattis, 1990] point out an important distinction between quantitative data sets. They went beyond the ordering attribute and use *coordinates* and *amounts* to differentiate two types of data: *coordinates* specify points in some domain (e.g., a start date for a project), while *amounts* signify the quantitative measurement used to describe a particular attribute of an object (e.g., the price in dollars of a car). See [Roth and Mattis, 1990] for the usefulness of this distinction.

Continuity. As proposed by Arens, Hovy, and Vossers [Arens et al., 1993], a quantitative data set can be further distinguished by *discreteness* vs. *continuity.* For example, a set of coordinates can be either continuous (e.g., the set of longitude-latitude pairs on a map), or discrete (e.g., the set of dates in a year). On the other hand, a set of amounts can also be continuous or discrete. For example, the number of days in the months of the year is a discrete set of amounts, while the temperature in degrees over the course of a day is a set of continuous amounts.

Data Relations

Characterization along the dimension of data relations captures various types of relationships among objects. We extend the scope of the data relations (e.g., functional dependency) mentioned in [Roth and Mattis, 1990] by including semantic relations among data, including *constituent*, *attributive*, and *enumerative* relationships.

Functional Dependency. Functional dependency was first used by Mackinlay [Mackinlay, 1986] to capture the one-to-one mapping relationship from one domain set to another. For example, every day has an average temperature. This type of relation is required by certain graphical techniques (e.g., a line graph) but not by others (e.g., a network diagram) [Mackinlay, 1986, Roth and Mattis, 1990].

Constituency. Constituency corresponds to the *has-part* relation and can be further partitioned into *physical* constituency and *conceptual* constituency. Physical constituency either indicates the physical components of a physical object (e.g., a network node has several ports among its physical constituents) or refers to conceptual components that can be treated like physical ones (e.g., virtual links). Conceptual constituency, on the other hand, implies that a concept is conveyed by a set of sub-concepts (e.g., patient demographics is conveyed by age, gender, and other sub-concepts). This distinction plays an important role in the process of generating a comprehensive visual display for the information [Zhou and Feiner, 1996].

Attribute. In an attributive relation, some objects describe certain aspects (attributes) of other objects. Furthermore, these relations can be effectively encoded by the corresponding image’s visual attributes, such as its shape, size, or color.

Enumeration. *Enumeration* includes both *isa* and *instance* relations [Rich and Knight, 1991]. Thus, an enumerative relation shows class inclusion (e.g., pitbull *isa* dog) or class membership (e.g., Spot is an *instance* of dog).

All the relations discussed above usually do not exist independently from each other. More often, one or more types of relations coexist in one piece of information. On the other hand, when the request is to display several relations together, it is necessary to take into account the interconnections between those relations in order to integrate all related information in one presentation (see more in chapter 4).

Attributes of Data Relations

We use a straightforward combination of *dimensionality* [Arens et al., 1993], along with *coverage*, *cardinality*, and *uniqueness* [Roth and Mattis, 1990], to describe the attributes of a relation. In a dynamic application domain, such as network management, the properties of data relations can change over time. For example, normally there are three types of traffic in a particular kind of network link (its cardinality value is fixed-multivalued); however, at some times there might be only one or two types of traffic in certain of these links. Such observations lead to more design requirements: either we must be able to predict future changes in advance to make the design generic enough to cope with different expected situations, or we need to redesign the visual presentation on-the-fly when the current one no longer effectively conveys the new information.

Although *data role* and *data sense* can be considered as part of *situation information*, they are also directly tied to a specific domain object so we address them with the domain information.

Data Role

To generate effective presentations, not only do we need to know the data's *intrinsic properties* (e.g., data type or data attributes), but we also need to identify the information-presentation tasks. Casner [Casner, 1991] discusses how user tasks can affect the presentation design, while Roth and Mattis [Roth and Mattis, 1990] also analyze data properties based on the user's or application's information-seeking goals. We use the *data role* dimension to characterize the functional role each piece of information plays in a visual presentation context. In other words, data role specifies particular information-seeking tasks. It could be given by a user during interaction, or could be expressed by domain experts during knowledge acquisition.

We adopt the taxonomy of visualization goals proposed by Wehrend and Lewis [Wehrend and Lewis, 1990], which is a superset of the display functions listed in [Roth and Mattis, 1990]. These visual tasks include: *categorization* (categorizing information), *clustering* (grouping information), *identification* (revealing information identity), *distinguishing* (displaying the difference among information), *comparison* (comparing one type of information to another), *association* (associating one type of information with another), *ranking* (comparing all information in a particular order), *correlation* (correlating one type of information to another), and *distribution* (partitioning information).

Data Sense

To construct an effective visual presentation, the user or application domain visual preference should also be taken into account. For example, one way to distinguish the elements of a small set of objects is to use a different color for each. However, such a presentation would not be effective for a color-blind user. Presentation preferences might also be a function of a specific application domain, perhaps because users in that domain have been trained to use a particular presentation style (e.g., table chart vs. bar graph).

This notion of visual preference leads to our last characterization dimension, *data sense*. The word "sense" can be used to refer to one of a set of meanings for a word or phrase, as in a dictionary. For example, the word "person" can mean a "human being" in one sense, and a "human body" in another. We have adopted this meaning of "sense" to coin the term "data sense," which signifies a preferred way to present the data visually.

Like data role, data sense can be specified by an end user to indicate her individual preference, or captured in the process of knowledge acquisition. Alternatively, data sense might also be inferred, based on other data characteristics. Data sense could be extended to include

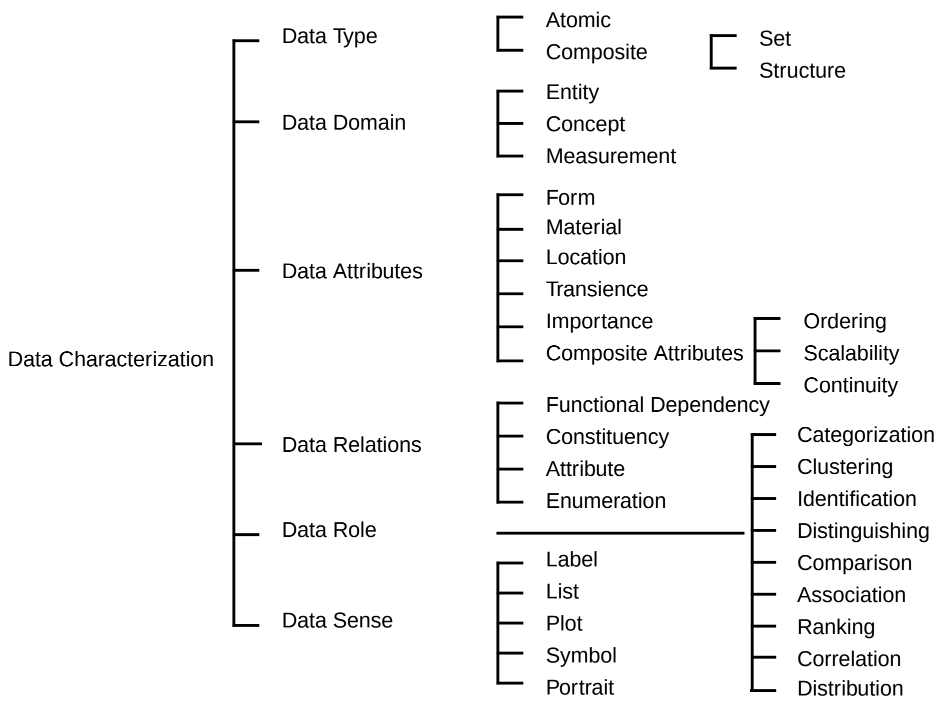


Figure A-1 Data characterization taxonomy

a wide variety of low-level visual preferences (e.g., color or shape). Thus far, however, we have limited data sense only to the set of five high-level presentation style preferences described below: *label*, *list*, *plot*, *symbol*, and *portrait*.

Label. One way to display information is through a textual label (e.g., a network node may be represented by the generic string “node” or the specific string corresponding to the node’s name). A label can be enriched by adding graphics, such as underscoring or a button shape, to convey additional information.

List. In contrast to the label sense, the *list* sense states that a composite object should be displayed in a tabular form in which all its components are listed textually as table entries. As in the *label* sense, both physical and conceptual objects can be represented in the *list* sense. Furthermore, in the *list* sense, all components of a composite object are either listed as an attribute-value pair or as a textual string (e.g., an entry in an indentation chart).

Plot. Both *plot* sense and *list* sense indicate that there is a composite object and that all its components need to be represented. Generally, visual displays in plot sense are schematic depictions, which refer to the typical genres of conventional diagrams or graphs. Quantitative information can usually be effectively expressed in *plot* sense. The research of [Mackinlay, 1986] and [Roth and Mattis, 1990] is mostly concerned with how to generate appropriate visual presentations automatically in *plot* sense.

Symbol. Representing a piece of information in *symbol* sense involves the use of a concrete shape object to symbolize the information. For example, a thermometer icon with its scale can be used to symbolize the temperature concept. Both physical and conceptual objects can be represented in symbol sense. We may use a much simplified (abstracted) version of a real image to symbolize a physical object (e.g., a network node may be symbolized as a sphere) or to represent a concept by implying the special connection between the real object and the concept (e.g., the thermometer and the temperature).

Portrait. Compared to symbol sense, visual representations require much more detail and precision when the information needs to be displayed as a realistic *portrait*. Such precise and detailed visual representation is necessary for people to carry out certain tasks, such as the design of a product's physical appearance.

We have presented the complete presentation-related data taxonomy in detail. Needless to say, to provide comprehensive information about the domain, some domain-specific data properties are also needed. For example, a person might have a physical attribute "age", while a car perhaps is described through its attributes "price" or "performance". How all the information is collapsed together and efficiently represented in the knowledge base is further discussed in the following section.

A.1.2 Visual Design Knowledge

To design various visual presentations for a wide variety of information, the system must be equipped with certain knowledge about the graphic design itself, as well as the characteristics of available visual forms. We refer to this type of knowledge as *visual design information*. In particular, there are two types of *visual design information* based on the roles they play during the design process (Figure A-2): *visual objects* and *visual techniques*. *Visual objects* are syntactic, semantic and pragmatic descriptions of various visualization forms, ranging from the most basic visual variables [Bertin, 1983] to complex visual representations such as animated sequences of 3D pictures [Lohse et al., 1994]. The design process can be viewed as a production process in which either visual objects are assembled from scratch or existing visual objects are modified and transformed to form new visual objects. While we consider *visual objects* as the products of this production process, *visual techniques*, on the other hand, are the tools used to assemble or shape these products. Visual techniques are also known as *visual actions* which serve as problem-solving operators in our visual design domain.

Visual Objects

Although numerous visual forms have been carefully analyzed or studied by various

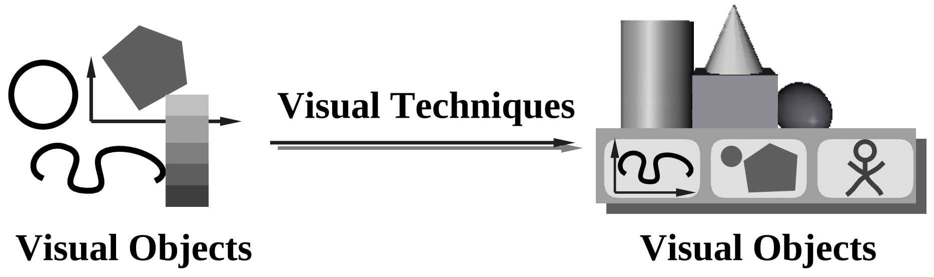


Figure A-2 Visual design information

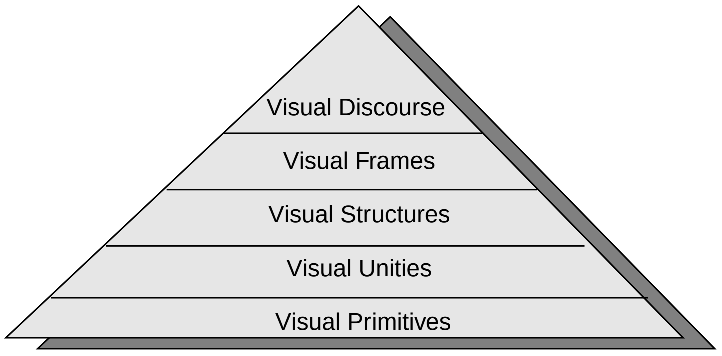


Figure A-3 Visual hierarchy

researchers (e.g., [Winn and Holliday, 1982; Tufte, 1983; Kosslyn, 1989; Tufte, 1990; Keller and Keller, 1993]) in different contexts, most studies are limited to qualitative analysis of various visual forms based on their communicative capabilities. Apparently, to automate the graphics generation process, such qualitative, functional analysis of various visual forms becomes inadequate. As visual communication takes place at three levels [Goldsmith, 1984], visual object synthesis also involves three levels: syntax, semantics and pragmatics. In other words, we not only need to understand functional characteristics (*semantics*) of visual objects, we also need to learn their structural features (*syntax*), as well as their communicative capabilities in fulfilling a particular perceptual task for a particular user or application (*pragmatics*). Based on their syntactic, semantic and pragmatic characteristics, visual objects can be divided into five types: *visual primitives*, *visual unities*, *visual structures*, *visual frames*, and *visual discourse*. As the upper level visual objects are recursively constructed using the objects below them, they form a visual hierarchy as shown in Figure A-3.

Visual Primitives. At the bottom of the hierarchy are *visual primitives*, which are individual visual variables: shape, material, texture, size, position, and orientation [Bertin, 1983]. Visual primitives are the most basic building blocks that participate in visual object synthesis. By themselves, individual visual primitives can hardly communicate any information or be used to fulfill a particular information-seeking goal. In other words, visual primitives have no fixed meaning apart from a visual context.

Visual Unities. 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

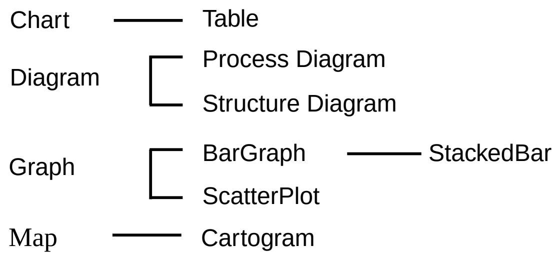


Figure A-4 Implemented visual structures visual variables (cuboid shape and red plastic material). According to the syntactic definition, a visual unity must have a *shape*, including both geometric shape or text. In other words, a visual unity is recognizable as an object representation (e.g., a red cube). Thus, visual unities could be used to convey certain information even though the conveyed information might be ambiguous or incomplete.

Nonetheless, the major purpose of a visual unity is to be embedded into other types of visual objects (e.g., a rectangle incorporated as a bar in a bar graph, as described below). For the sake of simplicity and feasibility, we do not always construct complex visual unities from scratch. Instead, we prestore many visual unities in the knowledge base, and use them as building blocks for composing more complex visual displays. A collection of such prestored visual unities is called a *visual lexicon* (we will further discuss the representation of the visual lexicon in the section on knowledge representation).

Visual unities are not just limited to graphical objects, they also include static images and video clips. Both images and video clips are regarded as prestored visual unities and can also participate in higher level visual object synthesis.

Visual Structures. 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 [Winn, 1987]. 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 [Winn and Holliday, 1982] identifies three basic structures that fall between text and realistic images: charts, graphs, and diagrams. Based on the visual representation classification proposed by Lohse [Lohse et al., 1994], 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), which does not quite fit into any of the three basic categories. The common visual structures that we have studied extensively are shown in Figure A-4.

Visual Frames. *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.

Similar to textual document composition, if a visual structure is equivalent to a sentence, then a visual frame is the visual counterpart of a textual paragraph. In addition to orchestrating various visual structures in a single display window, visual frames also concerns arrangements of multiple viewports or windows on one screen.

However, unlike a textual paragraph, a visual frame can be either *static* or *dynamic*. Static visual frames refers to a screenful display that does not change progressively, dynamic frames, on the other hand, indicate certain movements along the time line. In essence, static frames are just collections of static visual objects, and dynamic frames are animated key frames, which is specified through a visual action (see below) and a set of visual objects that the action acts upon.

Visual Discourse. *Visual discourse* is the ultimate visual form out of our automated graphics generation process. It is a cohesive formation of a series of visual frames. Visual discourse organizes visual frames along the time line: it specifies how long a frame should last and what the transitions should be between the frames.

Visual Actions

As mentioned earlier, to synthesize visual objects, we need a set of tools to either assemble the objects from scratch or transform the existing objects into new ones. This set of tools is known as a set of encoded visual techniques or *visual actions*.

Visual actions are categorized based on their functional roles in visual object synthesis. There are three types of visual actions: *formational actions*, *transformational actions*, and *camera actions.* Although they are expressed in different terms, our categories more or less correspond to those developed by others (e.g., [Friedell, 1983]).

Formational Actions. *Formational actions* design visual objects from scratch by mapping the properties of information to the visual attributes of selected visual objects. There are three types of mapping actions: *geometry mapping actions*, *property mapping actions* and *structural mapping functions*. *Geometry mapping actions* determine the location, size and the orientation of a visual object, while *property mapping actions* take care of the other visual properties such as color, and shape. As both geometry mapping and property mapping actions are specified at multiple levels of abstraction (i.e., different visual objects have different mapping functions), *structural mapping actions* specifies *abstract* structural ingredients of a particular visual objects (e.g., *MakeTableChart*). Speaking of abstractness, it means that structural mapping actions only indicate that certain types of visual objects can participate in the construction, but does not specify the details of the expected visual ingredients.

Transformational Actions. Unlike *formational actions* which assemble visual objects from scratch, *transformational actions* modify the existing visual objects so that they can be used in the new visual presentations. Transformational actions modify visual objects in two ways: *appearance transformations* (e.g., *Transfigure* changes an object's material) and *geometry transformations* (e.g., *Scale* modifies an object's size).

Note that such transformational actions are also specified at multiple levels of abstraction, both the actual execution and the effect of the action differs as the visual objects varies. For example, changing the appearance of a table chart could be quite different from modifying the visual appearance of a bar graph.

Camera Actions. 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*).

Animated Actions

As mentioned earlier, visual actions are also used as connectors between visual frames 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. 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 [Chang and Unger, 1993].

Formation Action	**Transformation Action**	**Camera Action**
MapProperty	Align	Dolly
MapGeometry	CutAway	Pan
MakeVisualUnity	FishEyeView	Rotate
ShowInset	Highlight	Seek
ShowMultiView	Move	Set
	Open	Zoom
	Scale	
	Transfigure	

Table A-1 Primitive visual actions

In a visual discourse, the current display might be replaced by a completely different display. To maintain aesthetic continuity [Zettl, 1990], we use a set of special actions that are adopted from conventional film transition techniques such as dissolve, fade in, and fade out (as used in [Karp and Feiner, 1993]) to achieve smooth transitions between displays.

Formation Action	**Transformation Action**	**Camera Action**
DesignVisObject	Emphasize	
MakeTableChart	Focus	
MakeStructDiag	RevealInternal	
Show		

Table A-2 Composite visual actions

Composite Actions

A visual action is either *primitive* or *composite*. Primitive visual actions, such as *Enlarge*, 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 [Young et al., 1994]. 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. Table A-1 and Table A-2 shows the sets of *primitive* and *composite* visual actions we have implemented in IMPROVISE.

There are two distinct advantages to having both primitive and composite actions. First, composite actions can be used at a high level to sketch an abstract design without worrying about computationally overwhelming details [Zhou and Feiner, 1997b]. 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.

A.1.3 Situation Knowledge

To be able to tailor visual presentations to specific audience or particular situations, the system also needs to be equipped with certain knowledge about the audience (e.g., audience identity or task), the occasion (e.g., the general purpose of the presentation or the time constraint), and the system environment (e.g., the available device such as the display media). A collection of the information that concerns the user, the occasion and the environment is referred as *situation information*. In other words, the effectiveness of visual presentations is also a function of situation information. The more comprehensive the situation information is modeled, the better the visual presentations will be for the modeled user or situation. Other researchers have modeled and utilized certain situation information (e.g., Friedell's situation space [Friedell, 1983] and Mackinlay's display categories [Mackinlay, 1986]) to index the knowledge base and facilitate the selection of synthesis operations. However, the modeled situation information usually is very limited. As modeling the user or the environment is a sizeable task itself, we select to analyze and model several types of key information. A collection of such information not only directly affects the presentation design, but it also models the presentation situation more systematically and comprehensively than any other situation model does.

Although situation information comes from a wide variety of sources, we partition them into three categories: *audience*, *occasion*, and *environment.* These roughly correspond to the information categories that a speaker is expected to know before s/he gives a verbal presentation [Speaking].To make an effective presentation, whether is verbal or visual, the situation information plays an important role in design, as well as delivery.

Audience. The audience is the ultimate judge who will justify how effective a presentation is. Knowing the audience better helps customize the presentation to serve the audience's interest. Currently, we focus on two types of audience information: *audience identity* (e.g., a nurse or a physician in a medical domain) and *audience preference*. *Audience preference* is further characterized by analyzing audience knowledge (e.g., audience presumed knowledge and audience beliefs), audience information seeking goals (e.g., [Roth and Mattis, 1990; Wehrend and Lewis, 1990; Casner, 1991]), and audience visual preference (e.g., [Zhou and Feiner, 1996]).

Occasion. *Occasion information* specifies the particular circumstances under which the presentation is being given. It includes *presentation type*, *mood*, *location*, *criticality,* and *timing*.

Presentation type indicates the main purpose of using a visual presentation. Much research has been conducted in the area of cognitive psychology to characterize various picture functions (e.g., [Levin et al., 1987]) or classify the rhetorical structures of images (e.g., [Sutcliffe and Darzentas, 1994]). Combining classification taxonomies, we have adopted the four top-level visual presentation types: *decoration*, *illustration*, *augmentation*, and *communication*.

- *Decorative presentations* appear to be purely created for aesthetic illustration. Such presentations hardly convey any information within the visual displays.
- *Illustrative presentations* is also known as *interpretive* or *transformational* presentations. These presentations take advantage of human's highly developed visual perceptibility to clarify difficult or abstract concepts through visual organization.
- As certain information is better or easier conveyed by visual media, *augmentative* presentations are meant to complement other media to provide more information to the audience. As the purpose of visual presentations is to *supplement* other medium presentations (e.g., text or speech), augmentative visual displays always play a supporting role in multimedia presentations.
- *Communicative presentations* directly deliver the information to the user through their visual displays. Without accompanying or being accompanied by any other type of media, the sole purpose of communicative presentations is to communicate the information to the audience.

Most our research focuses on automatically generating visual presentations that either *augment* other media to comprehensively convey information or *communicate* information directly to the user through effective visual displays.

Mood describes the nature of the presentation or the atmosphere of the visual presentation (e.g., *formal* vs. *informal*). Mood information could help the system to choose an appropriate style of illustration. For example, a presentation set for a *formal* mood may have a different set of requirements (e.g., using the times font vs. a cartoon font) than it does in an *informal* mode. As most of our presentations are generated for formal occasions, we do not go very far to model mood information. Nonetheless, we have provided an avenue along which other researchers could go further to extend the situation information model.

Location information indicates where the visual presentation takes place. Like *mood* information, this type of information also affects the design in many ways since different location might put different requirements on the presentation. For example, in a healthcare application domain, a presentation made for doctors at their offices might differ greatly from those made for them in the operating room.

Criticality and timing information are another two factors that can significantly influence the content and the delivering style of the final visual presentation. *Criticality* indicates whether presenting certain information is critical for the intended audience, while *timing* information stresses that either the presentation should be ready within a certain time frame or should last for a certain time period.

Environment. Since the visual presentations always need to be designed on certain computers or to be presented on certain displays, there is another type of information that specifies the existing physical system environment under which the presentation is being planned and realized. This type of information is referred as *environmental information*. Currently, we have considered two types of environmental information: one is the *platform* on which the whole system runs, the other is the *display* on which the visual presentation appears.

Platform information is tightly coupled with the efficiency of the system. The more powerful the platform is, the more efficiently the system can plan and realize the visual presentation. *Platform information* also further constrains the visual and interactive capabilities of the system. A platform's efficiency determines the rendering quality and the interactive capability of the system. For example, a SGI Onyx system with a fast graphics card capable of hardware texture mapping enables the user to interact with high quality presentation easily, and it also leaves an option for the system to use texture as a visual cue without sacrificing much interactivity. Platform information naturally could be characterized through its main features: *cpu*, *graphics board*, and *accessories* (e.g., texture mapping board).

While *platform information* affects the efficiency of a generation system, *display information* determines whether the on-screen visual displays are suitable for a particular viewing medium. For example, visual highlighting achieved by placing a red text on a yellow background might be fine for a color display, but it may appear so faint on a black-and-white monitor that the effect of highlighting basically diminishes. Evidently, we could use t features such as *color*, *resolution*, and *size* to characterize a type of display.

All three types of knowledge discussed above are equally important. A lack of any type of knowledge could result in ineffective visual designs that cannot meet the needs of a specific audience or comply with a particular situation. Since knowledge acquisition could be expanded as a whole research topic in knowledge engineering, we emphasize identifying information that is directly tied to our goal. Nevertheless, our characterization and classification of various information leaves an avenue for further research.

A.1.4 Meta Knowledge

Meta information, also known as *strategic information*, helps the sytem explicitly reason about the control of an inferring process or provide sophisticated explanation facility [Reichgelt, 1991]. As this is a very much open research area, we simplify the meta information modeling involved in automated graphics generation systems. The main purpose of introducing meta information is to explicitly control the design process and make the design process more efficient. There are three types of meta information explicitly represented: *information extent*, *information origin*, and *information reliability.*

Information extent describes the scope of the other three types of knowledge. For example, the system could know that its visual presentations can only represent certain type of information (e.g., quantitative) based on the information extent about the visual design knowledge.

Information origin helps to partition the knowledge base by indicating the origin of the knowledge. This partition aids the system to efficiently extract or reason about the information it needs. For example, we can distinguish the knowledge that is acquired from domain experts from that is derived from the current knowledge base. Assume that we have acquired the “raw” design knowledge from graphic design experts. In the meantime, task-oriented design patterns generated by the system are also stored in the knowledge base. Suppose that the system needs to design a visual presentation for another task. Instead of designing the new visual presentation from scratch by completely relying on *raw* graphic primitives (e.g., uninstantiated chart specifications), the system might consult the stored task-oriented design patterns first to see whether a minor modification could be used to accomplish the new task. As we can see, *information origin* helps to index knowledge types and partition the knowledge (e.g., raw knowledge vs. completed design) so the system can employ the knowledge it needs.

Information reliability indicates how reliable a piece of information is. For example, if

```
(defclass ENTITY
   (is-a DOMAIN-OBJECT)
   (slot type)
   (slot visual-attributes)
   (slot domain-attributes)
   (slot relations)
   (slot components)
   (slot visual-rep))
```

(a)

```
(defclass VISUAL-PROPERTY
   (is-a PROPERTY)
      (slot form)
      (slot material)
      (slot location)
      . . .
      (slot role)
      (slot senses))
```

(b)

```
(definstance Smith of PERSON
   (type ATOMIC)
   (visual-attributes
       (form SHAPED)
       (material SKIN-COLOR) . . .)
   (domain-attributes
       (age 80)
       (gender MALE) . . .) . . . )
```

(c)

Figure A-5 Domain object classes and instance

the knowledge base contains information such as "table charts could be used to represent quantitative information", then this information could be tagged with a reliability factor, 0.7. This means that the statement given above is correct 70% of the time. *Information reliability* is used to guide the system to make highly reliable decisions. Moreover, the system could use this information to explain why such a decision is made and how reliable the decision is.

A.2 Knowledge Representation

As described in Chapter 3, three knowledge representation paradigms have been used to store various knowledge in automated graphics generation systems. Next, we follow these three representation paradigms to examine how a specific piece of knowledge is stored.

A.2.1 Object-oriented Knowledge Representation

Domain Information Representation

Each piece of domain information is considered as a domain object. Domain objects are partitioned based on their domain types. As described earlier, there are four top-level domain types: *entity*, *concept*, *measurement*, and *event*, which become the top-level object *classes*. The data dimensions (e.g., type and attributes) we used earlier to characterize domain data become *slots* in a *class*. The slot components is used to record all subobjects that are constituents of a *composite* object, while the slot visual-rep is used to record the constructed visual representation of the object (Figure A-5a).

To facilitate manipulating different types of attributes or relations, we consider each type of *attribute* or *relation* as an object *class* itself [KRSL, 1993]. Thus, we organize different types of attributes and relations (e.g., visual attributes vs. domain-specific attributes) into different PROPERTY or RELATION classes. For example, the general, presentation-related attributes (e.g., *form*, *material*, and *location*) of the domain objects described earlier are represented as VISUAL-PROPERTY class (Figure A-5b).

A specific domain object therefore is an *instance* (i.e., *member*) of one of the *classes*, for example, Smith is represented as an instance of class PERSON, which in turn is a *subclass* of ENTITY. In this example (Figure A-5c), instance Smith inherits slot values from both ENTITY (e.g., slot form) and PERSON (e.g., slot material). .

Visual Design Information

Similar to domain objects, both *visual objects* and *visual actions* discussed earlier are also represented using object-oriented formalism.

Visual Objects

Naturally, various visual objects (e.g., *visual frames* or *visual structures*) are represented as *classes* based on their definitions. The more general the definition is, the higher the visual object is placed in the hierarchy (e.g., Figure A-6(a)-(b)). Their syntactic components become the *slots* in their class descriptions (Figure A-6(b)).

Even though the frame-like representation of a visual object appears like a template definition, there are *substantial* differences between them. A template is a *completely* specified parametrized description, of which all parameters are *fully* constrained in terms of their values. Moreover, the semantics of internal relationships among various parameters usually are not explicitly specified or made known to the user. And the usage of a template is quite straightforward: instantiating a template by supplying all the parameter values as specified. Suppose that we have a bar graph template, in which parameters such as the location of the bar (*x*), and the height of the bar (*y*) are specified in terms of their value types and ranges. Using a template is like making a function call to retrieve the result by supplying the values of each parameter.

In contrast, a visual object specification is more like a *partial plan,* whose parameters and the parameter relationships are only *partially* specified. And such a definition is fully recursive. Because of its partiality and recursiveness, the process of using an visual object is a planning process rather than that of making a function call. Moreover, to determine unspecified relationships and attributes of the visual object, the semantics of its parameters and their relationships must be explicitly specified. For example, each bar in a bar graph could itself be a visual structure such as a stacked bar graph. To design a *complex* bar graph like this, we need to specify relationships between its bars, and the relationships inside each bar. As another example, Figure A-6(c) describes a table chart instance Table1, which is recursively defined in terms of other table chart instances in its cells slot. A visual object is built up by recursively refining the descriptions of its structural components [Zhou and Feiner, 1997a]. In other words, a completed visual object has specific values for all its slots, while an unfinished object has at least one slot value that is vague or unspecified.

```
(defclass VISUAL-STRUCTURE
      (is-a VISUAL-OBJECT)
      (slot placeHolder)
      (slot holderPlacement)
      . . . )
```

(a)

```
(defclass TABLE-CHART
      (is-a VISUAL-STRUCTURE)
      (slot alignment)
      (slot spacing)
      (slot heading)
      . . .
      (slot cells)
      )
```

(b)

```
(definstance Table1
    of TABLE-CHART)
   (placeHolder INSET_WINDOW)
   (holderPlacement (0, 0))
   . . .
   (alignment ROW)
   (cells
       (Table11
           (alignment COLUMN)
           (celss Table111 . . . ))
       . . . ))
```

(c)

Figure A-6 Visual object classes and instance

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

(a)

```
(defclass ANIMATED-ACTION
    (is-a ACTION)
    (slot startTime)
    (slot stopTime))
```

(b)

```
(defclass MOVE
    (is-a ANIMATED-ACTION)
    (slot source)
    (slot destination))
```

(c)

Figure A-7 Visual action classes

Visual Actions

From an object-oriented point of view, visual actions are special objects that can change the state of other objects. A general visual action is defined in the format shown in Figure A-7a), while a specific type of visual actions is represented as a subclass of ACTION (Figure A-7b). In this case, ANIMATED-ACTION is a type of actions with temporal constraints. A temporal constraint is represented as a time interval that is specified by its start and stop times. This type of action is usually used to achieve smooth transitions between individual displays.

As show in Figure A-7, 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. As shown in Figure A-7(c), MOVE has the parameter source and destination. Preference indicates how preferable the action is in the current design context.

As mentioned earlier, a visual action is either *primitive* or *composite*. A composite action is represented by a set of partial plans, which will eventually be decomposed into primitive actions. Each partial plan is represented using a decomposition schema [Young et al., 1994]. 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 Figure A-8(a), while a decomposition schema is defined in Figure A-8(b).

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 A-9. Action *RevealInternal* is used to reveal the internal structure of an object (e.g., reveal a car's internal structure to show its engine condition).

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.

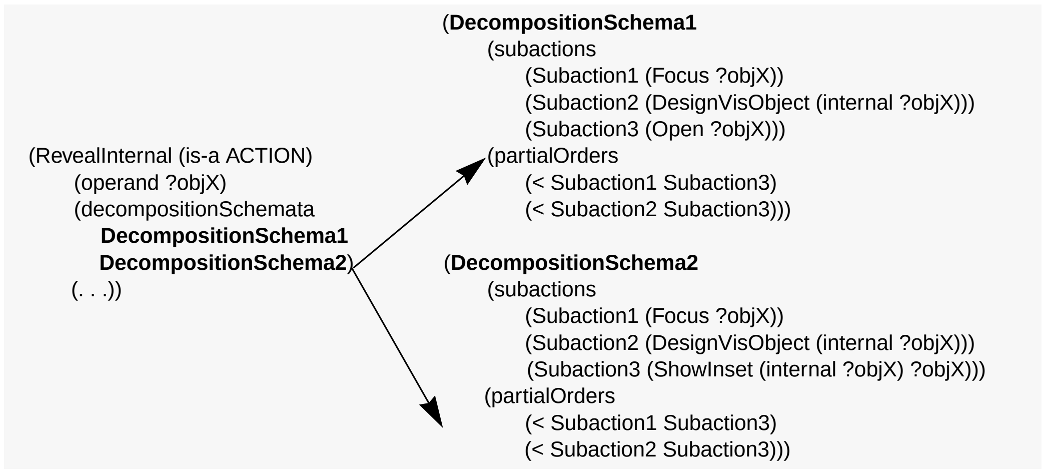
```
(defclass COMPOSITE-ACTION
    (is-a ACTION)
    (slot decompositionSchemata)
    (slot decompositionPreferences)
    (slot currentDecomposition))
```

(a)

```
(defclass DECOMPOSITION-SCHEMA
    (is-a OBJECT)
    (slot subactions)
    (slot partialOrders)
    (slot causalLinks))
```

(b)

Figure A-8 Composite visual action and its decomposition



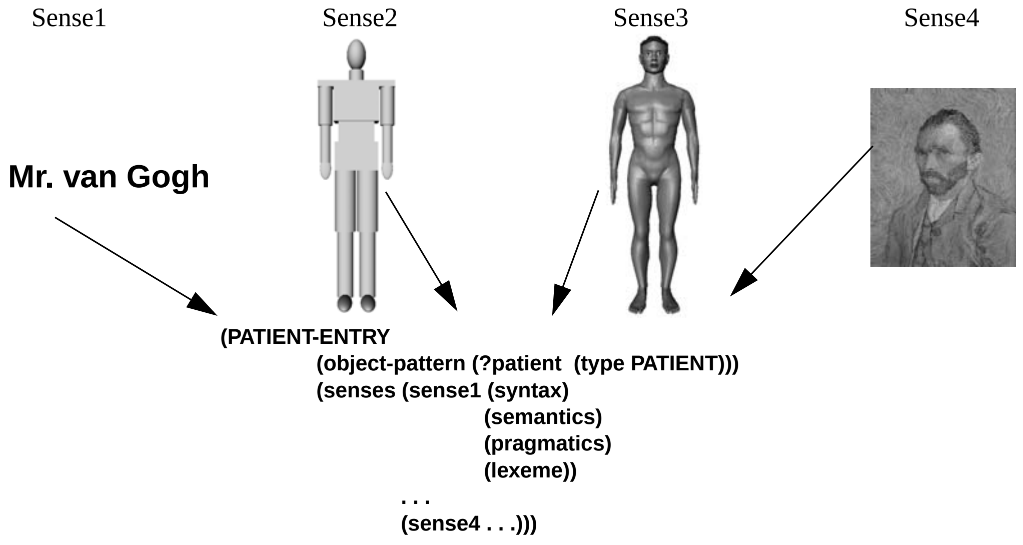
```
                              (DecompositionSchema1
                                  (subactions
                                      (Subaction1 (Focus ?objX))
                                      (Subaction2 (DesignVisObject (internal ?objX)))
                                      (Subaction3 (Open ?objX)))
(RevealInternal (is-a ACTION)     (partialOrders
    (operand ?objX)                   (< Subaction1 Subaction3)
    (decompositionSchemata            (< Subaction2 Subaction3)))
       DecompositionSchema1
       DecompositionSchema2)      (DecompositionSchema2
    (. . .))                          (subactions
                                      (Subaction1 (Focus ?objX))
                                      (Subaction2 (DesignVisObject (internal ?objX)))
                                      (Subaction3 (ShowInset (internal ?objX) ?objX)))
                                  (partialOrders
                                      (< Subaction1 Subaction3)
                                      (< Subaction2 Subaction3)))
```

Figure A-9 Action RevealInternal and its decomposition

Visual Lexicon

As briefly described earlier, the visual lexicon is a collection of primitive visual objects. Each visual lexical item is also known as a *visual word*. Unlike other types of visual objects, visual words are *completely* specified and are used as building blocks to form other visual objects. In this sense, the representation of a visual word is much like a *template*.

A visual word is always associated with a single domain object. However, each domain object can be represented by one or more visual words. In a visual lexicon, we say that a domain object has multiple *senses*. We use an object-pattern that describes a set of domain objects which share the same domain type (e.g., PATIENT) or certain attributes (e.g., their age greater than 20) to index the visual lexical entry. Thus, each object-pattern is associated with one or more sense, and each sense (visual word) has explicit syntax, semantics and pragmatics in the context of graphic design. In addition, a graphical expression, called a lexeme, is also stored. In the following sections, we describe each type of the information stored in a visual lexicon entry.



```
(PATIENT-ENTRY
        (object-pattern (?patient (type PATIENT)))
        (senses (sense1 (syntax)
                        (semantics)
                        (pragmatics)
                        (lexeme))
                . . .
                (sense4 . . .)))
```

Figure A-10 Multiple senses for patient entry

Object Pattern. To map an object onto a visual word, first we need to know the object pattern including the type, attributes, and other object characteristics [Zhou and Feiner, 1996]. As shown in Figure A-10, variable ?patient stands for an instance of PATIENT. More complex patterns can also be represented through the pattern description keywords. Currently, we have implemented two keywords: type and attribute. Type specifies the object domain, while attribute restricts the objects to a subset that satisfies the specified attributive constraints. For example, to represent any PATIENT whose age must be greater than 20, we use:

```
(object-pattern (?patient (type PATIENT) (attribute (> :age 20))))
```

Note that the name of each attribute (e.g., age) is preceded by a ":".

Syntax. Syntax describes the structure or pattern of a visual word. In particular, category, subcategory, and media together specify the syntactic features under slot syntax (Figure A-11 and Figure A-12).

Category provides the type of a visual word based on the visual hierarchy described in [Zhou and Feiner, 1996], while subcategory further classifies the type information. For example, a visual word is an instance of a VISUAL-STRUCTURE. More specifically, it is in the TABLE-CHART subcategory. Although we could go even deeper to further distinguish the visual word types, we have found the two-level hierarchy to be adequate thus far.

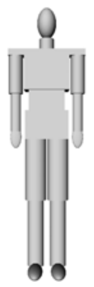
IMPROVISE is designed to deal with a wide variety of visual presentation forms (i.e., visual media formats). We allow four media format: *graphical model*, *graphical file*, *image*, and *movie*.

A *graphical model* describes a visual object in a particular graphics language. In graphical model mode, all graphical expressions are explicitly written out as a lexeme. However, in *graphical file* mode the lexeme refers to the name of a file that contains all the graphical expressions. For example, a human model might be expressed in Inventor [Wernecke, 1994] file format as:

```
Human{
    Head { . . .}
    Body {
        Chest { . . .}
        Waist { . . . } . . . }
```

Mr. van Gogh

```
#Text
(sense1
  (syntax
    (category VISUAL-UNITY)
    (subcategory TEXT)
    (media Graphical-Model)))
  (semantics
    (role Identification)
    (scope Naming)
    (sense LABEL)))
  (pragmatics
    (domainInfo . . .)
    (hardware
       (platform SGI | PC)
       (cpu R4400 | Pentium))
    (display Color | GrayScale)
    (performance FAST)) . . .)
```



```
# Simple body model
(sense2
  (syntax
    (category VISUAL-UNITY)
    (subcategory S-SHAPE)
    (media Graphical-File)))
  (semantics
    (role Locate)
    (scope BodyPosition)
    (sense SYMBOL)))
  (pragmatics
    (domainInfo . . .)
    (hardware
       (platform SGI | PC)
       (cpu SGI 4400 | PPro))
    (display Color | GrayScale)
    (performance FAST)) . . .)
```

Figure A-11 Visual word representations for patient

```
# Complex body model
(sense3
  (syntax
    (category VISUAL-UNITY)
    (subcategory C-SHAPE)
    (media Graphical-File)))
  (semantics
    (role Locate)
    (scope BodyPosition)
    (sense SYMBOL)))
  (pragmatics
    (domainInfo . . .)
    (hardware
          (platform SGI) . . .)
    (display Color | GrayScale)
    (performance MEDIUM))
    . . . )
```

```
# Image
(sense4
  (syntax
    (category VISUAL-UNITY)
    (media Image TIFF)))
  (semantics
    (role Identification)
    (scope Appearance)
    (sense PORTRAIT)))
  (pragmatics
    (domainInfo . . .)
    (hardware
       (platform SGI | PC) . . . )
    (display Color | GrayScale)
    (performance MEDIUM)) . .
    .. . . )
```

Figure A-12 Visual word representations for patient

}

Alternatively, this description can be kept in a file named “human.iv” in graphical file mode. *Image* mode signifies that the current visual object is an image file. Optionally, it can be followed by an argument to indicate the image format as shown in sense4 of Figure A-12. The last media category, *movie*, indicates the current visual object is a video clip. Similar to image, it can also be further qualified by appending a type argument (e.g., MPEG1).

The advantage of providing media information is to simplify the process of handling multiple media formats and to make the system more general and extensible. Usually, we specify a set of abstract visual operations (e.g., *Scale* and *Move*) at a high level without worrying about the implementation details for different media. At a lower level, based on the provided media information, the abstract operations are realized by media-specific procedures. Hiding the details of abstract visual operations makes it easier to extend the system. Consider the abstract visual operation *Scale*. *Scale* transforms the modeling matrix for a graphical model, which can change the dimensions of an image. To support a new media format, we only need to write a set of procedures to perform the *Scale* operation specifically for the new medium without affecting the rest of the knowledge base.

Semantics. While syntax focuses on describing the structure or pattern of a visual word, semantics abstracts the meaning of the syntactic features, summarizes the thematic roles of a visual word, and identifies the scope of its role. Figure A-11 and Figure A-12 also summarize the semantic features for the PATIENT-ENTRY under slot semantics.

Semantic sense specifies the abstraction of syntactic features of a visual word, while semantic role indicates what a visual word is capable of when it participates in a visual presentation. Moreover, semantic scope elaborates how well a visual word can perform in the specified role. In Figure A-11, sense1 is a label that can only identify the patient by naming, but sense4 in Figure A-12 is a portrait that can identify the physical appearance of the patient.

Pragmatics. In language or image understanding, pragmatic features ususa;;y refer to the roles or influences that the language or image possesses to help the use understand distinguishing features and appropriate contexts [Goldsmith, 1984]. To facilitate visual design, we have extended the connotation of the pragmatic features of a visual word. Pragmatics not only includes information that is directly related to the characteristics of the user or context, such as the user’s identity, expectations, and application type, but also take into account factors that describe the relationships between the word and the physical design and execution environments, such as the heardware requirements (e.g., display type). They are *domain information*, *hardware requirements*, and *performance estimation*. Figure A-11 and Figure A-12 list the pragmatics of each word in PATIENT-ENTRY, with domain information omitted since they all share the same one:

```
(domainInfo
        (appType MEDICAL)
        (audienceType NURSE | DOCTOR))
```

Specifically, the pragmatics features of sense1 in Figure A-11 can be read as: the text representation of a patient’s name is suitable for both nurses or doctors in a medical application. Moreover, such a text can be rendered on a SGI or PC with color or gray-scale display. The rendering speed of such a text string is fast (e.g., under a milliseconds).

Although the pragmatic features involve domain-specific information, we can consolidate all the domain-specific information by expressing them in a disjunctive form. For example, if we want to use the PATIENT-ENTRY for both medical and military logistics applications, we have this:

```
(domainInfo
   (info1 (appType MEDICAL)
          (audienceType NURSE | DOCTOR))
   (info2 (appType LOGISTICS)
         (audienceType MEDIC | DOCTOR)))
```

Lexeme. In a visual lexicon, each lexeme is a graphical representation of a visual word. Each type of representation (*graphical model*, *graphical file*, *image*, and *movie*) is actually a parametrized template. For example, the lexeme of sense1 in PATIENT-ENTRY is:

```
(lexeme (VISUAL-UNITY (geometry
                                  (Text2 (string (get-name ?patient))))))
```

This says that the visual word is an instance of VISUAL-UNITY with a geometry of 2D text, which in turn requires that the string be the name of a patient. Embedding procedures in the representation can be very useful. It eases information encoding by abstracting the common

Syntactic Descriptions

Slots	Value	Description
Category	VISUAL-STRUCTURE	VISUAL-UNITY
Subcategory	TABLE-CHART	IMAGE
	TIME-CHART	VIDEO
	BAR-GRAPH	TEXT
	LINE-GRAPH	2D-SHAPE
	PIE-GRAPH	3D-SHAPE
	. . .	. . .
Medium	GRAPHICS-MODEL	GRAPHICS-MODEL
	GRAPHICS-FILE	GRAPHICS-FILE
		IMAGE
		[TIFF | RGB | GIF | PICT]
		VIDEO
		[MPEG1 | SGI | QTIME]

Semantic Descriptions

Slots	Value Description
Sense	LABEL
	LIST
	PLOT
	SYMBOL
	PORTRAIT
Role	CLUSTER
	IDENTIFY
	LOCATE
	DISTINGUISH
	. . .
Scope	PROPOSITION
	SPATIAL-RELATION
	CONCEPT-FUNCTION
	. . .

Figure A-13

features of a visual word and expressing them in a procedural format. Assume that every patient's picture is stored as an image in the knowledge base, and is named by the patient's ID with a suffix ".tif". Instead of explicitly listing all the patients and the file names of their pictures in the visual lexicon, we can have a single expression:

```
(lexeme (Image (fileName (string-cat (get-id ?patient) ".tif"))))
```

This states that the file name can be constructed by concatenating the patient's ID and the suffix ".tif".

To be consistent with the overall constructive planning approach, the visual lexicon stores only visual words that represent atomic objects. Visual representations for composite objects are built from scratch by piecing together their component presentations. Figure summarizes all possible values that can be used to describe the syntax and semantics of a visual word. However it is worth noting that some of the values are correlated; for example, if syntactic category takes visual-unity, then its subcategory can never be a table-chart. Theoretically, we could encode complex visual representations such as time-chart in a visual word. To retain the flexibility and extensibility of constructive graphics synthesis, we usually construct complex visual presentations from scratch while keeping already-made simple visual representations in the visual lexicon.

A.2.2 Production Rule Representation

The purpose of utilizing various visual design principles (rules) is to assure an *expressive* and *effective* design (e.g., [Mackinlay, 1986; Senay and Ignatius, 1994]). To maintain the expressiveness and effectiveness criteria therefore becomes the principles of visual design, and such principles can be further partitioned into different categories, each of which is specifically used to ensure a particular design criterion. Here, we list the general format of the rules under each category. In a real system, these *abstract* rules should be refined using the available information (e.g., domain information) and formulated based on the general format introduced here.

Expressiveness Rules

Expressiveness rules are concerned with the ability to express the desired information *comprehensively, distinctively, generally,* and *specifically.* In regard to the *comprehensiveness* and *distinctiveness* of visual presentations, we refer to the ability to present *all* related information, as well as to emphasize the *specific* attributes of the information. While *generality* regards the ability to describe high-level groupings or abstractions [Mullet and Sano, 1995], *discreteness* describes the ability to present certain information or certain information characteristics. *Generality* and *discreteness* together ensure the information to be expressed in a way, which allows the user to selectively attend to certain piece of information, as well as perceive all the information as a coherent whole.

Integrity Rules

To express desired information, visual presentations must possess an *integrity* that reflects the true meaning of the information. Therefore, this type of rules stresses that:

```
IF certain information characteristics presents, and
   there is a visual object can represent this characteristics
THEN only this visual object can be used to express the desired information integrally.
```

The mapping between the information characteristics and the visual object types are in part based on the pragmatic information of visual objects [Goldsmith, 1984] and the visual variable classification described by Bertin [Bertin, 1983]. For example, quantitative information can *not* be expressed using *shape* variable, nor the *orientation*.

Comprehensiveness Rules

Comprehensiveness refers to the expressive ability that all related information or information characteristics can be communicated to the user. In other words, this type of rules states that:

```
IF certain information needs to be communicated to the user,
THEN all its related information should be presented to the user as well.
```

For example, in a medical domain, presenting drug usage of a particular patient perhaps requires displaying all names of the drugs, along with their dosage and the time when the drugs were applied. Moreover, the resulted display is capable of conveying all the desired information cohesively.

Distinctiveness Rules

One of the desired features of visual presentations is to present information *distinctively* enough so that the information can be readily identifiable. *Distinctiveness* asserts that:

```
IF a specific piece of information could be used to identify the information and
   a component of a visual object can be used to make the information identifiable
THEN use the component to encode the specific information.
```

This type of rules delivers the ability to describe *specific* attributes of the information to maintain its visual and conceptual identity. Let’s use the drug displaying example described above again. Not only can we display the time when each drug is applied, but we also can group the drugs and identify them by their types or effects. For example, the first group of drugs could be antibiotics, while second group of drugs might be the drugs to maintain the patient’s blood pressure. Such specific features that make a group of drugs identifiable should be expressed in the presentation as well. In our example, if a table is used to describe a group of drugs, the headings of the table chart could fulfill such a role by indicating the type of drugs.

Generality Rules

In many cases, the generated visual displays represents a whole category of information instead of individual instance of that category. *Generality* describes the ability to express higher-level groupings or abstractions. The rules ensure design generality therefore stress that:

```
IF a class of information needs to be presented, and
  a particular visual object is a better abstraction of the class than others,
THEN use this visual object.
```

For example, a sphere could be used as an abstract representation for various types of computer network nodes, while a detailed description of a computer might not be as well recognized as a network node by different audience, even though each computer network node is actually a computer or functioning through a computer.

Discreteness Rules

As an automated graphics generation system is capable of presenting a class of information abstractly, it should also be able to “break” the information into parts so that the user can access the information selectively. *Discreteness* rules are made to take care of this type of business:

IF the visual object can represent a piece of composite information,
THEN the components of visual object represent the components of the information.

This type of rules ensures that individual component of the information to be presented is accessible through the components of the visual objects. Assuming that a bar chart is used to represent a set of data, while each piece of the data can accessed through the bars in the bar chart.

Effectiveness Rules

While expressiveness criteria ensures that the desired information is correctly presented, effectiveness criteria determines whether the visual displays *accurately* and *clearly* convey the information as intended. Not only do effective visual presentations convey information accurately and clearly, but they also should be *appropriate* for the task, and the targeted user. Moreover, effective visual displays must possess a perceptual *immediacy* that allows the information to be perceived effortlessly. As our goal is to generate coherent visual discourse, to remain coherent, the visual presentations must ensure visual *consistency* within or across displays, maintain visual *continuity* between displays, and achieve maximum *unification* among different visual components within display.

Accuracy Rules

Accuracy rules concern about whether the desired information could be *accurately* communicated to the user using the specific type of visual objects. This criterion is especially important when the desired information is quantitative. Through their experiments, Cleveland and McGill have discussed the accuracy ranks of various visual cues regarding to quantitative perceptual tasks [Cleveland and McGill, 1984; Cleveland and McGill, 1985]. For example, using *position* to encode quantitative information would be more accurate than using *length*. Thus, a general rule in this category will be:

IF a visual object can convey the information more accurately than others,
THEN use this visual object.

Clarity Rules

While many measurements can be taken to judge whether a visual presentation *clearly* communicates the information to the user as intended, here we focus on the rules that are applied to enforce a visual display to *clearly* reflect the perceptual structure of the information. As pointed out by a number of researchers (e.g., [Pomerantz, 1981]; [Treisman, 1982]) people intend to group or relate information together based on their characteristics while they process or interpret the information. Based on *gestalt* theory, information can be visually organized in certain ways (e.g., by proximity or similarity) to reflect the perceptual structure of the information. One type of rules considers how to utilize various types of visual cues to visually group information together. These rules are primarily based on *gestalt* effects:

Grouping by proximity

IF object *A* is more related to object *B* than any other object is,
THEN visualObject *V1* representing *A* should be placed visually closer to visualObject *V2* representing *B* than any other visual object is.

Grouping by similarity

IF object *A* is similar to object *B*,
THEN visualObject *V1* representing *A* should be similar to visualObject *V2* representing *B*.

Grouping by continuity

IF object *A* is an continuition of object *B*,
THEN visualObject *V1* representing *A* should be displayed as an continuation to visualObject *V2* representing *B*.

Not only should visual presentations exhibit necessary information grouping, but they also need to establish a *hierarchy* of importance for information groups [Mullet and Sano, 1995]. And such a hierarchy can be used to structure the visual elements to reflect their relationships. For example, a hierarchical menu will reflect the *parent-child* relationship between the menu items and their submenus. Rules in this category concern how to use *scale*, *contrast* and *proportion* to order the groups of information into a hierarchy that corresponds to the intended reading sequence. For example, the user should be able to distinguish urgent information from routine information through their visual appearances so that they can selectively attend to the most important information first. Therefore, these rules stress:

IF information group *A* should be distinguished from information group *B*, *and*
different values of visual variable *V* can be used to fulfill the task,
THEN show group *A* and group *B* in different *V* values.

Appropriateness Rules

Successful visual presentations are required to be *appropriate* for the specific task or the targeted user to meet their inherent needs [Goldsmith, 1984; Mullet and Sano, 1995]. For example, it is *inappropriate* to present all textual information in English to a user who does not even know English language. This type of rules asserts that certain visual cues should be used to convey certain information based on the application or the user needs:

IF the user/application needs information to be presented in certain ways, and
particular visual cues can represent the information in the way they want,
THEN use these visual cues.

Immediacy Rules

One way of judging the efficiency of a visual display is to evaluate whether a visual display can be perceived by the user effortlessly and involuntarily [Mullet95]. Perceptual *immediacy* is used to measure such visual effectiveness. As many visual properties may contribute to the quality of perceptual immediacy, here we focus on the immediacy caused by *simplicity*, *directness*, and *sharpness*:

Simplicity

IF both visual object *A* and *B* can be used to encode information *I*, *and*
visual object *A* is simpler than *B*,
THEN object *A* should be used.

By all means, relative *simplicity* of different visual objects can be measured by the complexity of their visual structure (e.g., a 2D bar chart is simpler than a 3D bar chart), or by the number of their components (e.g., color a map using four colors is simpler than using five colors, assuming the color is only used to distinguish one region from another).

Directness

IF both visual object *A* and *B* can be used to encode information *I*, *and*
visual object *A* is more direct than *B*,
THEN object *A* should be used.

As we aim to communicate the information *efficiently* to the user, the more explicitly the visual displays convey the information, the easier the user could perceive the information as intended. Directness is measured by how a particular visual objects can be used to convey the desired information in a straightforward or explicit manner. For example, if the goal is to compare the sales performance for a car dealer for two particular months, a line graph with explicitly marked net gain or lose is much more direct than a bar chart that just plots the sales separately for these two months.

Sharpness

IF both visual object *A* and *B* can be used to encode information *I*, *and*
  visual object *A* can represent information sharper than *B*,
THEN object *A* should be used.

The *sharpness* could be measured by judging whether the desired information can be easily recognized from its background or separated from the information surrounding it. For example, a red object on a red background appears less *sharper* than a red object on a white background.

Consistency Rules

Effective visual presentations require the consistent design within or among displays [Marks, 1991a; Neal and Shapiro, 1991]. *Consistency* rules ensure that the same information or the same user tasks should be handled consistently. For example, encoding a computer network node using a red sphere in one frame, and then depicting the same node using a red square in another frame would be considered an *inconsistent* design. Similarly, if the system decides to use a visual technique or a combination of techniques to solve a visual task, unless there is a good reason to do so, otherwise such decision should be maintained and applied to accomplish the same task. As a consequence, consistent rules maintain design consistency at different levels:

Symbol encoding level

IF a piece of information was previously visually encoded using visual symbol *S*,
THEN the same piece of information should be encoded in *S* too.

Task solving level

IF a task *T* was previously solved in a way described by *W*,
THEN the same task *T* should be solved in *W* too.

Continuity Rules

As *visual momentum* [Woods, 1984] could prevent the user from getting lost in an information network, effective visual presentations must maintain *continuity* between different displays to gain visual momentum. In the context of visual discourse, *continuity* means providing smooth transitions between different visual frames. To complement the animated visual actions, film-making techniques (e.g., dissolve, fade in, and fade out) are also used to provide different types of transition. We have adopted these film-making rules directly from [Karp and Feiner, 1993].

Unity Rules

It is common that a visual representation needs to encode a significant amount of information at one time, let alone new information might need to be incrementally integrated into existing presentation. To avoid information overload or confusion, *unity* rules are designed to address how to efficiently combine all types of information into an integrated whole, or how to efficiently integrate new information in the existing displays.

Efficiently harness every visual component

IF more than one visual tasks needs to be accomplished, and
  visual object *A* could satisfy more visual tasks than the others could,
THEN use *A*.

Suppose that we need to show both the structure and the temperature distribution of the airplane. The same airplane body could be used to show the structure and serve as a base for displaying temperature distribution information. In other words, an effective visual presentation is also *versatile*.

Efficiently integrate new information into existing displays
IF visual object *A* can be better integrated with current display than the others can
THEN use *A*.

The easiness of the integration is determined by the nature of the current visual objects. It can be measured by the amount of space the visual object acclaims, or by the amount of the interference introduced by the new visual object (e.g., how much current visual display needs to be amended to suit the new visual object).

So far we have listed substantial number of visual design principles involved in automatically generating expressive and effective visual presentations. Although we have attempted to establish a comprehensive categorization of various design rules in an orthogonal manner, understanding human perceptual capabilities is not a trivial task. Therefore, the categories introduced here do not necessarily cover an exhaustive set of rules, nor are they perfectly orthogonal to each other. For example, both *clarity* and *immediacy* might both refer to a same rule that using good contrast to *clearly* express information grouping and reinforce a perceptual *immediacy*. Nevertheless, this does not appear to be a problem. Intuitively, it is common that one design rule might be able to achieve more than one expressiveness or effectiveness criterion as one visual expression could be used for a number of different tasks.

Intuitively, when various rules are applied at the same time, conflict design decisions may arise. In case of conflicts, we need to order various rules based on the specific situations (e.g., the application type, or the communicative goals). For example, if the overall goal is to view the trend of a car dealer's sales performance over time, the rules emphasizing comprehensiveness might be favored over those focusing on distinctiveness in case these two types of rules result in a conflict design.

APPENDIX B Planning Stages

We have briefly described how to employ a hierarchical planning approach and its main features (top-down decomposition, partial-order) to automatically generate visual presentations. Now, we look more closely at the main planning stages that involve in constructing an effective visual presentation. More concrete and complete examples will be given in the next chapter to illustrate how this approach actually works.

The input to our system is a design task. The hierarchical planner starts to plan based on the given 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.

B.1 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 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.* These preference constraints not only can be used to guide the action selection, but also can be employed in other decision making process, for example, visual word selection or action decomposition.

Achievability

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 [Goldsmith, 1984]. 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

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 B-1, 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

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 B-1 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*.

Sharing

Within the same planning process and concerned with the same object(s), *sharing* states

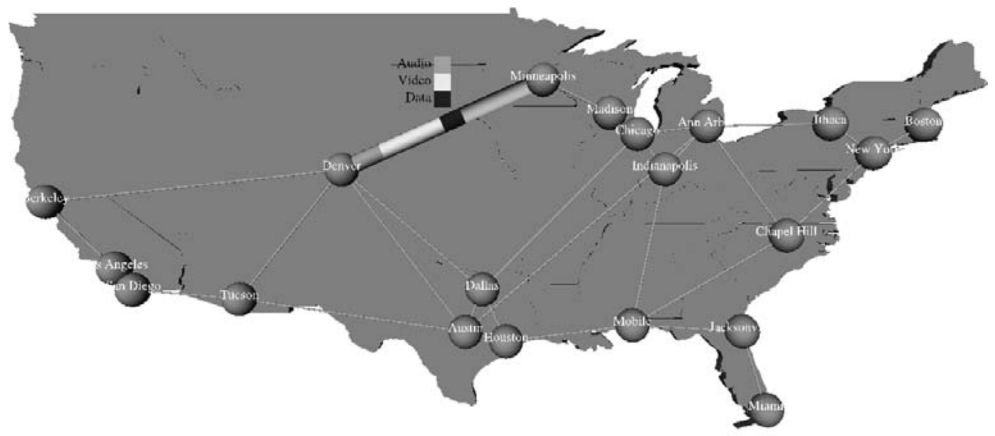


Figure B-1 Traffic status in the selected link

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 [Russell and Norvig, 1995]. 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

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

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

Figure B-2 Decomposition preference for composite action RevealInternal

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.

B.2 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 A-9). Its decomposition preferences are specified in Figure B-2.

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. The class of *DesignNetworkDiagram* and one of its instances are represented as:

```
(DesignNetworkDiagram (is-a CompositeAction)        (DiagramX (is-a NetworkDiagram)
        (operand DiagramX)                                      (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)        (DesignVisObject (is-a CompositeAction)
        (operands nodeX)                               (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)                       (NodeRep
        (id)                                     (is-a VisualObject)
        (. . .)                                  (material ?node_material)
        (representation NodeRep))        (. . .))
```

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 [Wilkins, 1988]. 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.

B.3 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.

B.4 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.

To model the *inference engine*, we have described a top-down hierarchical planning approach as the problem-solving approach for automated graphics generation systems. The major advantage of this approach is its computational efficiency. Compared to a search-based or non-hierarchical planning approach [Yang and Tenenberg, 1990], it achieves efficiency by reducing the search space and deferring decisions to avoid unnecessary backtracking. In addition, it also eases knowledge-encoding through reuse of primitive actions. More importantly, its top-down design assures global coherence, and facilitates interleaving of planning and execution [Wilkins, 1988].

We believe that planning approach is an efficient way to model the *inference engine* for a general purpose graphics generation system unless there is a better problem-solving approach available. Nevertheless, the *inference engine* could be instantiated in any way that best suits the practical needs of the developers or applications. We are not trying to develop an all-purpose super-efficient planner, instead we emphasize the *needs* (e.g., global coherence and computational efficiency) of the *inference engine* and developing the guidelines (e.g., top-down strategy) on how to meet these needs.