Pergamon

Comput. & Graphics, Vol. 21, No. 1, pp. 79–87, 1997
© 1997 Elsevier Science Ltd. All rights reserved
Printed in Great Britain
0097-8493/97 $17.00 + 0.00

PII: S0097-8493(96)00072-6

Technical Section

ADDI: A TOOL FOR AUTOMATING THE DESIGN OF VISUAL INTERFACES

M. G. EL-SAID,[1] G. FISCHER,[2] S. A. GAMALEL-DIN[1] and M. ZAKI[1†]

[1]Systems and Computer Engineering Department, Al-Azhar University, Nasr City, Egypt
e-mail: azhar@frcu.eun.eg
[2]Computer Science Department, University of Colorado at Boulder, Boulder, Colorado, U.S.A.

Abstract—Visual interfaces are often poorly designed because the application developers lack graphics design knowledge. Designing effective interfaces requires great effort, especially when many types of graphical objects and relations are involved. In addition, many domain applications require interactive and dynamic graphical interfaces. Existing automatic graphical presentation systems focus on interfaces that convey only static information. Current user interface management systems that facilitate the process of creating interfaces for applications provide no assistance to help developers design the visualization features of interfaces. The aim of this research is to contribute to the design process of interfaces in general and dynamic ones in particular. A model-based framework and prototype tool called ADDI (Automatic Design of Dynamic Interfaces) has been created to help developers design domain-specific visual interfaces. ADDI combines a specification application with various visualization knowledge-base components to automatically generate a visual interface. Therefore, graphical user interface designers, by using ADDI, can select visual properties and hence be able to rapidly and correctly convey information visually according to the task specification in a domain application. Accordingly, high quality and consistent interfaces are generated even when the application developers lack the necessary graphics expertise. ADDI supports visual interface designers in four ways. The first is automatically determining the visual appearance of the interface objects from the Application Model specification. The second is computing the sizes and positions of the visual objects. The third is allowing developers to specify their preferences of designing visual objects. The fourth is aiding them in creating specific interactive presentation functions for end-users to interact with the interface. © 1997 Elsevier Science Ltd

1. INTRODUCTION

Designing effective graphical interfaces to convey information remains a very time consuming and tedious task. The design process becomes even more complex if it is necessary to convey heterogeneous and dynamic information via the graphical objects of the interface. Heterogeneous information implies that graphical objects must be designed to convey more than one type of information in the same interface, for example, numeric and connection types of information. A dynamic graphical object is an object whose appearance may change over time to reflect changes in the domain object's attribute values [1]. The interfaces of many simulation and visualization applications need such graphical objects to display dynamic and heterogeneous information [2].

Two types of systems can be used to assist application developers design graphical interfaces. One type is interface builders, which cover both Graphical User Interface (GUI) toolkits and User Interface Management Systems (UIMSs). Most of these interface builders [3] supply a set of widgets (objects) to help developers design windows, *e.g.* scroll bars, menus, and buttons. No support is given to designing the appearance of domain-specific objects that are used by end-users to perform specific activities in their work. These objects are constructed according to their properties in the application data base. The menu in Fig. 1, for example, could be designed using such tools, but no support is given for the work area of the window. Although some interface tools (*e.g.* [4, 5]) support the primitive components that can be used to design dynamic visual interfaces (*e.g.* rectangle, circle, shading) the process of mapping the domain objects and their properties into corresponding visual properties in the interface is left to the developers. Developers may therefore select poor properties that yield ambiguous interfaces if they do not follow some graphics guidelines [6].

The second type of system is automatic presentation systems [7–10], which use graphics knowledge to automatically generate visual interfaces. These tools are able to assess and design only static presentations, not interfaces that are dynamic and interactive. As interface builders, they are not able to design the graphical objects in the work area of Fig. 1. For instance, the box labeled "CISCO" in Fig. 1 represents a node in a local area network (LAN)

† Author for correspondence.

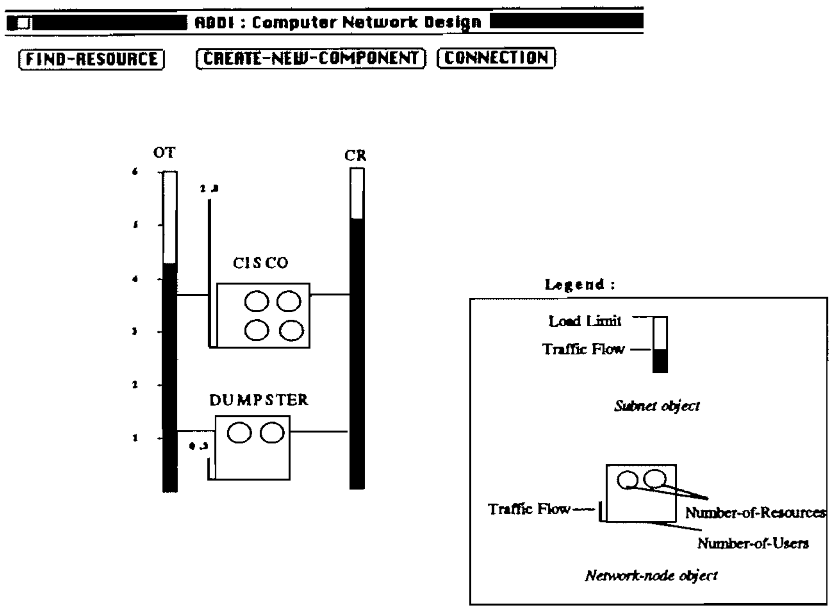


Fig. 1. A user interface used in a local area network application.

and is attached to a line that represents the amount of traffic flow that passes through the node.

In summary, neither type of system is able to support the design of dynamic visual interfaces for domains such as LANs. Designing a visual interface such as that shown in Fig. 1 entails two main tasks. The first is to determine the appearance of the graphical objects that make up the interface as well as how those objects would respond dynamically to changes in the state of the application object. The second task is to attach the semantic (behavior) to interface objects so that their dynamic characteristics could be described. To achieve the second task we must describe the computational and behavioral relations between graphical objects in terms of their domain's semantics.

A system called Automatic Design of Dynamic Interfaces (ADDI) has been created to address shortcomings in the types of systems described above. ADDI generates visual interfaces from a data model in which the domain is formally specified, and from design principles that are kept in the system's knowledge base. The notion of event, which is part of the application model, has been introduced to describe the ways that end-users will interact with the interfaces produced. This will allow one to automatically produce effective visual interfaces that are both dynamic and heterogeneous.

2. CONCEPTUAL FRAMEWORK

Automating the process of generating visual interfaces from the Application Model specification can be achieved by building bridges that fill the gap between the interface and that specification. The framework developed in this research bridges this gap through two main conceptual parts, the Application Model and the Presentation Manager (PM). The Application Model part covers the Data Model and Task Specification Component. Figure 2 illustrates these components along with the roles of ADDI, developers and end-users. Developers define the Data Model of a domain, which covers application object hierarchy, object properties, and events. In most applications, neither all application objects nor their properties are required to be displayed at once in the interface. The role of the Task Specification Component is to let end-users define a subset of Data Model entities that must be visualized to serve the task at hand. In other words, it simply determines which properties and objects as well as events in the Data Model are required to be visualized in the interface. Application specifications are internally represented as predicates called Application Model Entities (AMEs).

The PM visually presents the application elements selected by the Task Specification Component considering visualization preference determined by a Preferences Component. The PM consists of five components: Preference Component, Data Preprocessor, Visual Planner, Layout Manager, and Picture Generator. The Data Preprocessor paves the way to the activities that are performed by other parts of the PM by deducing both the comparison logical relations among a group of application properties and their appearance behavior (*i.e.* whether an application property has a dynamic or static behavior).

Based on that mapping result the Visual Planner

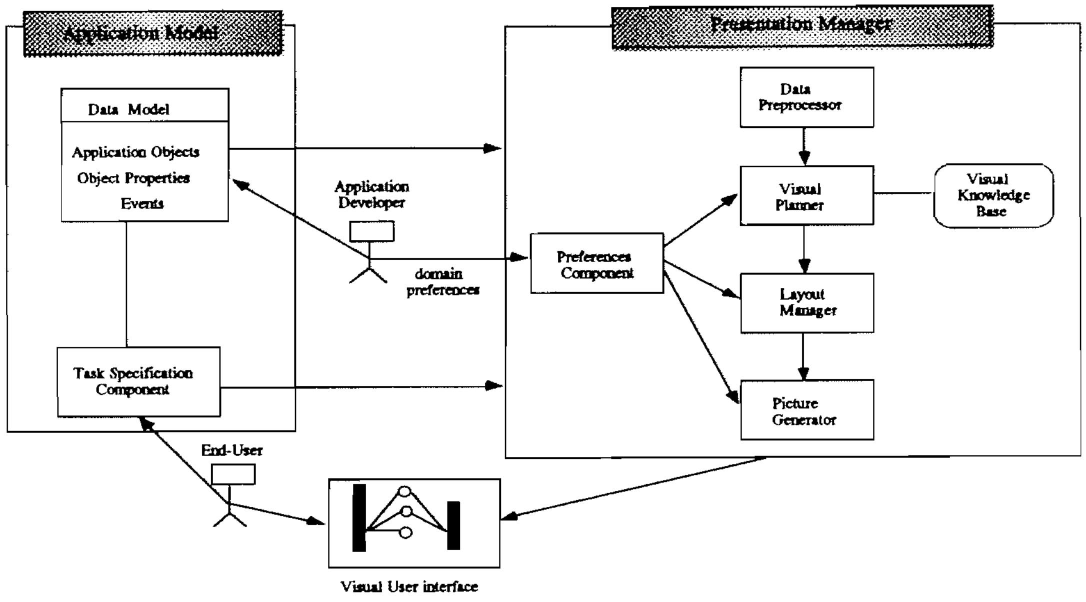


Fig. 2. Overview of ADDI architecture.

(VP) determines other graphical specification such as the spatial relation between these primitives. Since the graphical specification is different than application specification, a different representation than AMEs is needed. The Visual Entities (VEnts) which are internally represented as frames are used for that purpose. The slots of these entities which are manipulated by the different parts of the PM capture all the graphical specification in the interface that is used to present an application specification.

The output of VP is fed directly into the Layout Manager, which computes the positions of the interface graphical objects that are captured as slots of the VEnts. The Layout Manager satisfies the spatial arrangement constraints of the interface objects, such as connection and containment. At this point, the visual features and the positions of interface objects are determined, and the Layout Manager forwards this information into the Picture Generator to instantiate these specifications into actual images on the screen. This generator is a module that accepts the high-level graphical specification described in VEnts and generates images that match precisely the contents of VEnts.

Having been presented with the required information determined by the Task Specification Component, the end-user would be able to manipulate interface objects through Events. These events, which are created by application developers, are used to interact with the underlying application. Their semantic feedback is visualized on the interface, creating a dynamic behavior. The programming effort of events is divided into two parts. One part is to prepare the information to be presented in the interface and the other part is concerned with the interface visualization issues. Developers are supported in programming the second part by delegating the visualization issues of the interface to the PM.

It is almost impossible to totally automate the design of visual user interfaces that are suitable for all application domains, so the Preferences Component is used by developers to override automatic decisions. Via this component, developers can change any visual primitive or spatial layout relation that is used to present information in the interface. The developer can go back and forth between the Preferences Component and the generation of an interface until a satisfactory presentation is achieved.

Through a detailed scenario, the following sections describe the roles of the different elements of the system's architecture and knowledge representation as shown in Fig. 2. This scenario illustrates the roles of the end-users, the application developers, ADDI subsystem modules, and the knowledge bases.

3. EXAMPLE SCENARIO: LOCAL AREA NETWORKS (LANS)

Before we demonstrate how the different components of ADDI work to achieve the overall objective of the interface visualization, an example has been introduced here to explain ADDI's components. This example describes how to create visual interfaces for a LAN design environment. The example shows how network administrators benefit from visual interfaces to enhance their ability to simulate modifications to networks, such as adding new components (*e.g.* workstations) into the network or reassigning the roles of servers and clients, thus enabling them to explore design alternatives. An important goal is to observe changes in traffic flow and server dependencies, which are dynamic in nature to get a better design alternative.

Table 1. Task properties

Object Type	Object Scope	Application Property
Subnet	CR, OT	Traffic flow, load limit, name, connection
Network node	CISCO, DUMPSTER	Traffic flow, name, number-of-services, number-of-users

The task of adding a new component to a subnet in the network depends on the component's physical location and its technical specification. Network administrators may use a form-based interface in ADDI to specify the task by which the system determines an initial interface. End-users (network administrators) specify the task by entering the object types followed by the objects' names and eventually the required properties for these objects (see Table 1). Object class types, which represent the network components in this application, are subnet, network node, and resource. Network administrators may select from more than twenty different properties to specify their tasks. Ultimately, this task specification will cause ADDI to generate the interface shown in Fig. 1.

4. THE ARCHITECTURE OF ADDI

4.1. *Application model*

The application model, which consists of a data model and interaction events, describes the computational and behavioral relations of the domain objects and their properties (attributes). Figure 3 shows the menu used by an application developer to define a data model for the LAN domain. The radio buttons beside each entry declare the options available to the developer. Each network property required to be displayed in the interface has to be defined through the entries in this menu. ADDI compiles these entries into domain entities and keeps them in the knowledge base. According to the specification of the task, the system extracts the corresponding DEnts and sends them to the VP to be presented visually. Figure 3 shows a part of the corresponding domain entities for the information specified in Table 1.

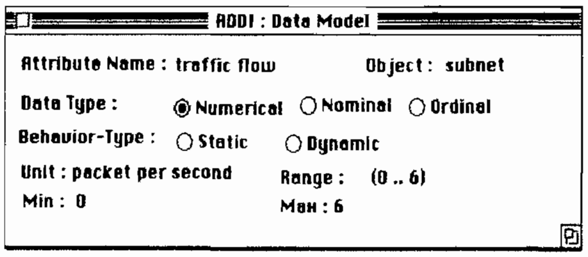
The data model which is used to define an application property is depicted in Fig. 4. Its entities, which are not application specific, must be specified for each application property that might appear in the interface. A Visual Knowledge Base is developed to support different possible combinations of these entities. Each combination defines different characteristics of a property that constrain its appearance in the interface.

4.2. *Presentation Manager*

The structure of the PM is designed so that it automates the construction of the interface from a high-level application description to that kept in the Application Model. Each component performs different activities required to develop the interface. Figure 5 shows how the representation of the initial application specification, represented as AMEs, is transformed by ADDI's components until the interface is generated.

The input of the PM comes from the Task Specification Component and contains the instances' names of each application object class, the application properties, and the required events for that task. These specifications are cross-referenced to their Application Model description, which is represented as AMEs by the Data Preprocessor (DP). The DP paves the way for the other PM's components by preparing some data characteristics of that application specification, which helps these components perform their activities. These characteristics compose the logical comparison relations and the behavioral appearance of each application property. Finally, the DP sorts the properties according to a priority scheme.

The VP sends the application properties to the Visual Knowledge Base (VKB) for mapping them into visual primitives. The result of the VKB is saved in 'mapping-predicates' which contain each application element and its corresponding visual primitive(s) in the interface. The VP instantiates a new structure, called the Visual Entities, to capture the detail implementation of the interface. The VP performs


ADDI : Data Model

Attribute Name : traffic flow   Object : subnet

Data Type : [✓ Numerical] Nominal Ordinal

Behavior-Type : Static Dynamic

Unit : packet per second   Range : (0 .. 6)

Min : 0   Max : 6

Fig. 3. Part of data model definition for a LAN domain.

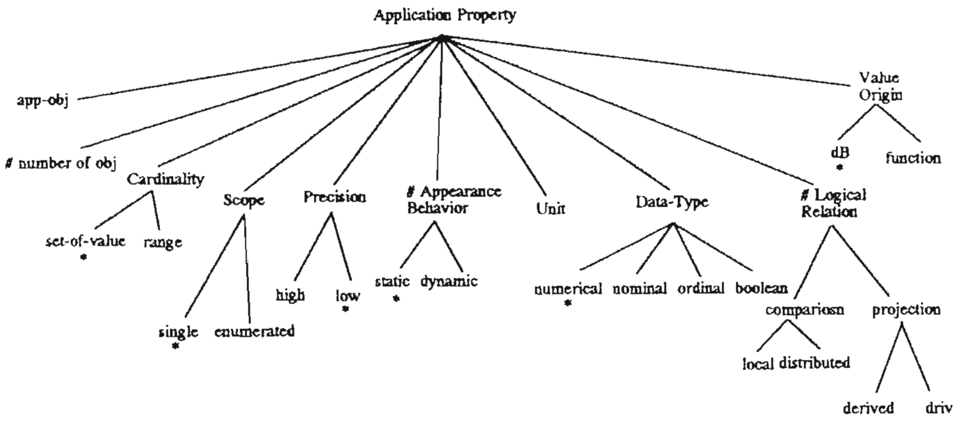


Fig. 4. The data model entities used to define application properties.

various activities to fill out the structure of the entities. The key activities are the computation of graphical values for each selected visual property and the determination of the size of interface objects based on their graphical values.

Based on the sizes of interface objects, the Layout Manager computes the positions of interface objects using two different layout algorithms. The Render Component generates the physical images on the screen based on the values stored in the Visual Entities.

The Preference Component is used to override automatic decisions that violate graphical conventions specific for an application domain and to design more effective visual presentation. The developer can go back and forth between the Preferences Component and the generation of an interface until a satisfactory presentation is achieved.

4.3. *The Visual Knowledge Base*

The design principles used by ADDI to construct visual interfaces are formalized in what is called the Visual Knowledge Base (VKB).

Figure 6 shows the relationship between the Visual

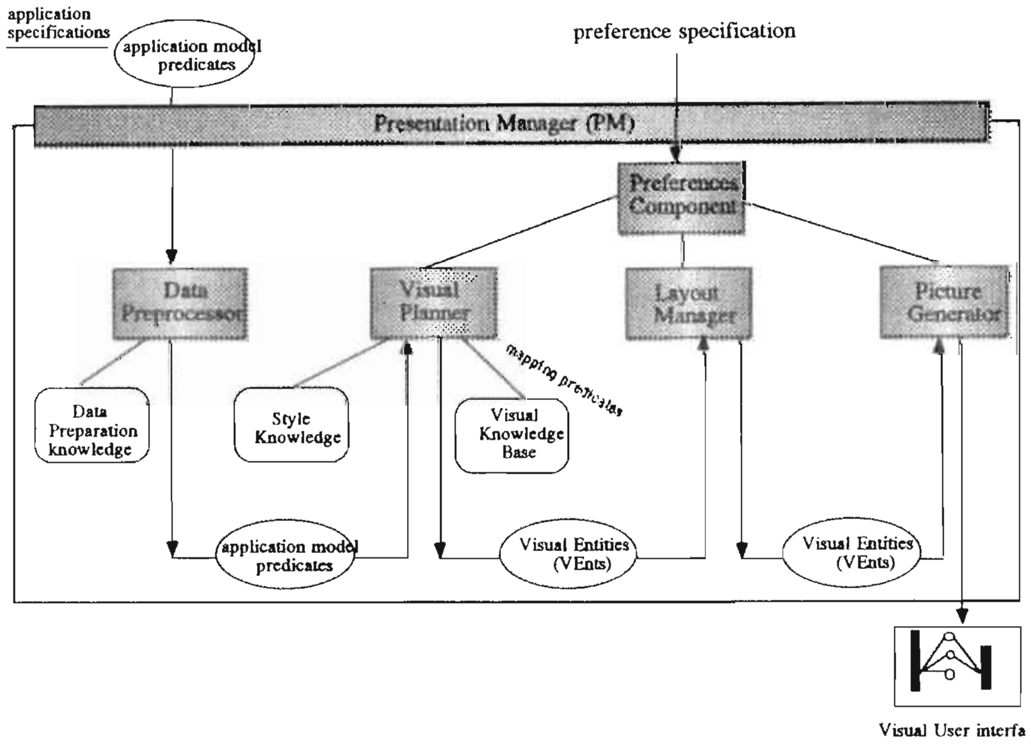


Fig. 5. The PM components and the knowledge they manipulate.

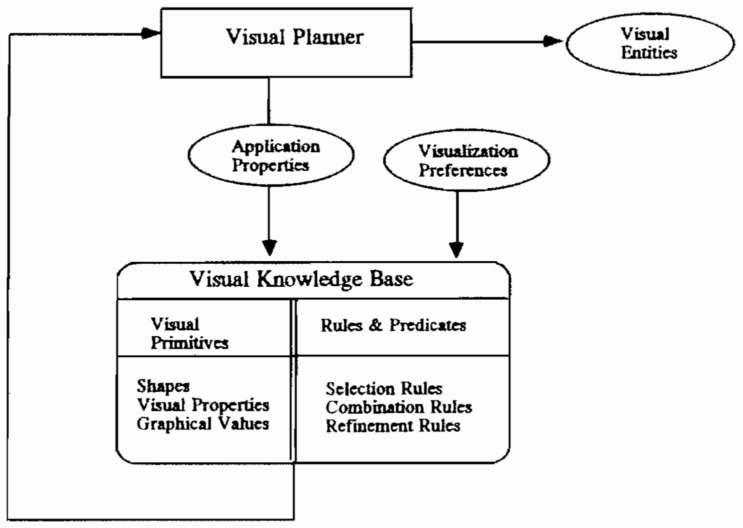


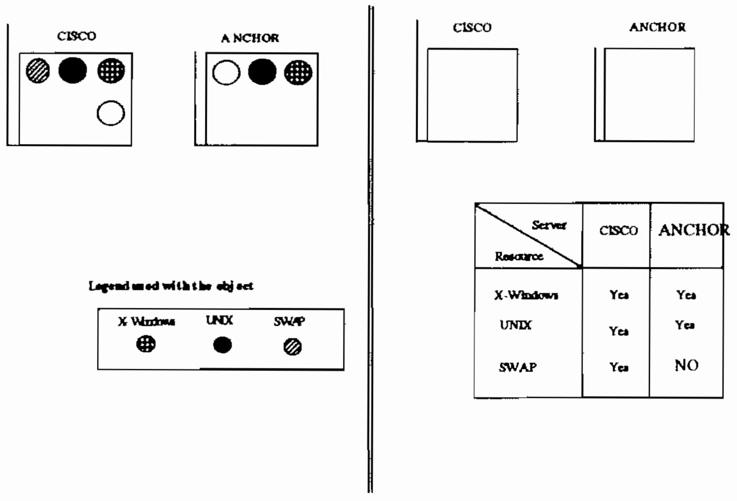
Fig. 6. The relationship of the VKB with the Visual Planner.

Planner and the VKB to map application properties into visual primitives. The Visual Planner sends a group of properties into the VKB. Based on their Application Model specification, the VKB maps them into visual primitives. The knowledge of the VKB is represented through two components indicated in the figure to achieve this mapping. The first component contains the visual primitives, which are the elementary units used to compose the visual interface. The second component contains the rules used to synthesize the visual appearance of the interface objects. The synthesis process is encoded via three different sets of rules by which the primitives are selected, combined, and refined. The result of the mapping process goes back into the Visual Planner to instantiate visual entities for the mapped application properties.

4.4. *Design principles used to create ADDI's VKB*

Based on the capability of the human visual system, the following factors summarize the principles that are taken into consideration by ADDI for guiding the design of dynamic visual interfaces. The rest of this section describes how these factors are addressed by the VKB rules.

- Locality factor [11]: Larkin and Simon demonstrated that the satisfaction of the locality principle would allow viewers to comprehend visually presented information more rapidly and accurately. This principle aims at reducing the eye search in the interface by combining visual primitives to present several related application properties in one graphical object. For instance, in the computer network interface of Fig. 7 (left), a box presents a network node, while the filling marks encode information about the resources a node has. The eye, reading this figure, can infer all information by looking at only one spatial location in the interface. On the other hand, Fig. 7 (right), ignoring the locality principle, presents the available resources in a table which has a different spatial location than the boxes; hence, the eye must search between different locations in the presentation to read the needed information.
- Expressiveness and effectiveness criteria [12, 13]: If more than one visual primitive can convey the same information, the role of effectiveness criteria is to select the most rapid and accurate one to convey the information. The ADDI's VKB has its own expressiveness criteria by which it selects visual primitives that are capable of clearly expressing desired information. To illustrate, consider the example of Fig. 8. The traffic flow of a network node may be close or equal to zero under certain circumstances, such as a server breakdown. If the height of a box were to be used to represent traffic flow, the box's border could vanish as in Fig. 8(a). Therefore, the presentation in this figure does not satisfy the expressiveness criteria. Observations indicated that network administrators had problems in interpreting the shape (box) in that figure as well as its relationship with the circles. The containment relationship is lost, in this example, since the height vanishes. Two possible solutions here: either the contents of the box or the height of the box is replaced by different visual primitives. Since it is hard to graphically present visual property which uses the area of a shape with different properties, the VKB adapts the second solution. It uses the composite shape technique to augment a new shape so that its dynamic change does not affect the expressiveness of the interface object. The presentation of Fig. 8(b) maintains the box shape and the containment relationship and, hence, satisfying the expressiveness criteria.



Server / Resource	CISCO	ANCHOR
X-Windows	Yes	Yes
UNIX	Yes	Yes
SWAP	Yes	NO

Fig. 7. Two different presentations of the same information.

- Logical operations: Ignoring the logical relation of application properties would lead to uncompleted mapping between the properties and visual primitives. Not only should each application property be mapped into visual primitive(s) but their logical relations should be considered during the mapping process.
- Dynamic behavior: During the selection of visual primitives the dynamic behavior of the application properties is assessed. Based on this assessment, those that can more effectively display the dynamic appearance of the properties are selected.

5. THE VISUALIZATION SYNTHESIZING PROCESS

The synthesis process which embodies the above described design principles is broken into three processes to select, combine, and refine the visual primitives. These processes shown in Fig. 9 are captured as rules of the VKB. The selection rules are responsible for selecting a visual property and its shape for an application property. The combination rules define the constraints that must be satisfied to accept a visual property and a shape which are chosen by the selection rules. The refinement rules assess the impact of the dynamic behavior on the interface objects and enhance the aesthetic features of the interface.

5.1. *The selection rules*

The selection rules are used to achieve the following steps in the synthesis process: (i) select shape(s) for each application object; (ii) select a visual property according to a selected shape and determine its type, the allowable graphical values, and the scope; (iii) prevent the selection of a visual primitive that is mapped to another application property. The VKB checks the preferences associated with each visual primitive prior to apply the selection rules to select the primitives. These preferences are set by either application developers, through the Preference Component, or the rules of the perceptual organization features.

Each application object is mapped into one or more shapes. If the application object of the property is not assigned to a graphical shape, a shape is randomly selected from the simple list of shapes.

5.2. *The combination rules*

Prior to assigning the visual primitives that are generated by the selection rules, the effect of combining them into previously selected primitives has to be scrutinized. In other words, these generated primitives work as input to the constraints which accepts or rejects them. These constraints that are used to prevent any ambiguity might arise due to an amalgamation of visual primitives. The constraints

CISCO
CISCO

Fig. 8. The impact of dynamic behavior.

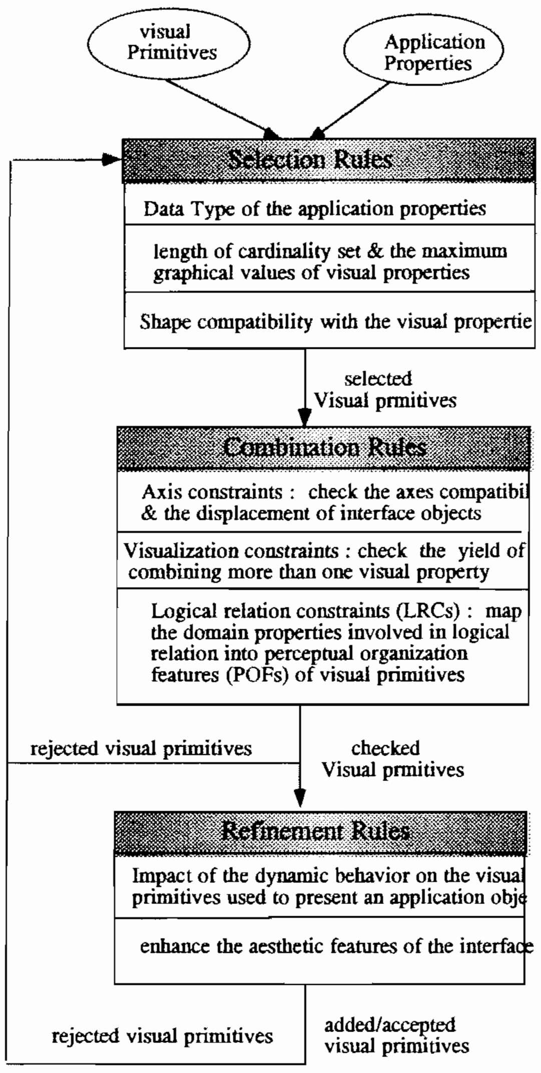

visual Primitives
Application Properties
Selection Rules
Data Type of the application properties
length of cardinality set & the maximum graphical values of visual properties
Shape compatibility with the visual propertie
selected Visual prmitives
Combination Rules
Axis constraints : check the axes compatibi & the displacement of interface objects
Visualization constraints : check the yield of combining more than one visual property
Logical relation constraints (LRCs) : map the domain properties involved in logical relation into perceptual organization features (POFs) of visual primitives
rejected visual primitives
checked Visual prmitives
Refinement Rules
Impact of the dynamic behavior on the visual primitives used to present an application obje
enhance the aesthetic features of the interface
rejected visual primitives
added/accepted visual primitives

Fig. 9. Visualization synthesizing process.

are classified, according to their appearance effect on the interface, into three categories: axis, visualization, and logical relation constraints.

5.3. *The refinement rules*

The last set of rules applied by the VKB to accomplish the mapping process is the refinement rules which are applied right after the selection and combination sets of rules. The refinement rules consider two presentation issues in designing the interface objects. First, the impact of the dynamic behavior on the objects is assessed. This may reject the result of the other two sets of rules and replace some visual primitives by new ones to satisfy the expressiveness of the dynamic properties. Second, more than one visual primitive for the same application property are selected to increase its aesthetic feature.

During the selection process it is not possible to predict the following visual primitives that are to be used for the same interface object and hence it is hard to assess the dynamic behavior of the currently selected property on the entire object. The set of dynamic behavior rules (part of the refinement rules) assess the impact of the previously selected properties, having a dynamic nature, on their graphical objects.

6. CONCLUSION

Automatic generation of the design of visual interfaces based on high-level Application Model specifications makes it easier to enforce and maintain consistency among all interfaces of the same application domain which is achieved because the same graphical principles are guaranteed to apply. When these visualization principles are well tested and widely accepted, the resulting interface is even better.

The main objective of this research was to identify the knowledge, graphical functions, and rules used to automate the dynamic visual user interfaces and to base on them a prototype of a CASE tool that aids the interface design. The main contribution of this work is spotted in:

- A framework that unifies different presentation components needed for automatically designing visual interfaces based on the Application Model of a domain.
- A model of the properties and attributes of visualizing application domains (Application Models) which captures the characteristics and properties of the application objects and which is used to define the dynamic behavior of the interface through the event mechanism.
- A knowledge-base containing graphical design principles and guidelines, as identified by the experts, of designing expressive and effective dynamic interfaces based on the continuously evolving Application Model over time.

ADDI is implemented in Macintosh Common Lisp and runs on Macintosh Computers. It has been applied for three different application domains: local area networks, elevator control systems, and an organization accounting model. Using ADDI's framework, it has been demonstrated that application designers can now focus only on the non-graphical aspects of the application while ADDI deals with the graphical design aspects.

A future extension of ADDI is to accommodate a critiquing component in its conceptual framework. In its current design, the impact of the developer's preferences on the interface design is not evaluated. ADDI accepts these preferences whether or not they violate graphical principles. This critiquing component, which takes advantage of the knowledge hosted by ADDI, is to analyze the developers' input and to provide them with the appropriate feedback whenever a graphical design principle is violated.

REFERENCES

1. Hsia, Y.-T. and Ambler, A. L., The use of iconic display functions in visual programming. In *Visual Languages and Visual Programming*, ed. S.-K. Chang. Plenum Press, NY, 1990, pp. 143–158.
2. Szekely, P., Luo, P. and Neches, R., Beyond interface builders: model-based interface tools, human factors in computing systems. In *INTERCHI'93 Conference Proceedings*, 1993, pp. 383–390.
3. Myers, B. A., Giuse, D. A., Dannenberg, R. B., Zanden, B. V., Kosbie, D. S., Pervin, E., Mickish, A. and Marchal, P., Garnet: comprehensive support for graphical highly-interactive user interfaces. *IEEE Computer*, 1990, **23**, 71–85.
4. Singh, G., Kok, C. H. and Ngan, T. Y., Druid: a system for demonstrational rapid user interface development, In *UIST'90 Conference Proceedings. Third Annual Symposium on User Interface Software and Technology*, New York, 1990, pp. 167–177.
5. DeSoi, J., Lively, W. and Sheppard, S., Graphical specification of user interfaces with behavior abstraction. In *Human Factors in Computing Systems. CHI'89 Conference Proceedings*, 1989, pp. 139–144.
6. Johnson, J., Selectors: going beyond user-interface widgets. In *Human Factors in Computing Systems. CHI'92 Conference Proceedings*, Monterey, CA, 1992, pp. 195–202.
7. Hsia, Y.-T. and Ambler, A. L., The use of iconic display functions in visual programming. In *Visual Languages and Visual Programming*, ed. S.-K. Chang. Plenum Press, NY, 1990, pp. 143–158.
8. Casner, S., A task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics*, 1991, **10**, 111–151.
9. MacKinlay, J., Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 1986, **5**, 110–141.
10. Marks, J., A formal specification scheme for network diagrams that facilitates automated design. *Visual Languages and Computing*, 1991, **2**, 395–414.
11. Larkin, J. H. and Simon, H. A. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 1987, **11**, 65–99.
12. Mackinlay, J., Automatic design of graphical presentations. Ph.D. Dissertation, CS-86-1138, Stanford University, 1986.
13. MacKinlay, J., Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 1986, **5**, 110–141.
14. Marks, J., Automating the design of network diagrams. Ph.D. Dissertation, Harvard University, 1991.
15. Harel, D., On visual formalisms. *Communications of the ACM*, 1988, **31**, 514–530.
16. Roth, S. F. and Mattis, J., Data characterization for intelligent graphics presentation. In *Human Factors in Computing Systems. CHI'90 Conference Proceedings*, Seattle, WA, 1990, pp. 193–200.
17. Fischer, G., Lemke, A. C., Mastaglio, T. and Morch, A., Using critics to empower users. In *Human Factors in Computing Systems. CHI'90 Conference Proceedings*, Seattle, WA, 1990, pp. 337–347.
18. Fischer, G., Nakakoji, K., Ostwald, J., Stahl, G. and Sumner, T., Embedding critics in design environments. *The Knowledge Engineering Review Journal*, 1993.