Declarative Models of Presentation

Pablo Castells

Universidad Autonoma de Madrid

Cantoblanco, 28049

Madrid, Spain

Tel: +34-1 397-3973

castells@lola.iic.uam.es

Pedro Szekely

Information Sciences Institute

University of Southern California

4676 Admiralty Way, #1001

Marina del Rey, CA 90292

Tel: (310) 822-1511

szekely@isi.edu

Ewald Salcher

Institute of Computer Graphics,

University of Technology, Graz,

Muenzgrabenstrasse 11/II/V

A-8010, Graz, Austria

Tel: +43 316 695810

salcher@icg.tu-graz.ac.at

ABSTRACT

Current interface development tools cannot be used to specify complex displays without resorting to programming using a toolkit or graphics package. Interface builders and multi-media authoring tools only support the construction of static displays where the components of the display are known at design time (e.g., buttons, menus). This paper describes a presentation modeling system where complex displays of dynamically changing data can be modeled declaratively. The system incorporates principles of graphic design such as guides and grids, supports constraint-based layout and automatic update when data changes, has facilities for easily specifying the layout of collections of data, and has facilities for making displays sensitive to the characteristics of the data being presented and the presentation context (e.g., amount of space available). Finally, the models are designed to be amenable to interactive specification and specification using demonstrational techniques.

KEYWORDS: model-based user interfaces, user interface design techniques, user interface development tools, graphic design.

INTRODUCTION

A large portion of interface design effort involves specifying the displays for an application. The displays of most applications consist of a static part and a dynamic part. The static part consists of menus, toolbars, and dialogue boxes with a variety of menus and buttons. Typically, the set of static components remains fixed while the application executes, except for simple state changes (e.g., graying out, becoming invisible). The dynamic part consists of more free-form displays combining text and graphics. The dynamic part typically displays application-specific data that the application generates at run-time, or that users construct interactively. Display components are created, destroyed, modified and laid out while the application executes. Displays must be updated when data changes.

Current tools for interactively specifying displays, such as interface builders and multi-media authoring tools [6] provide support for specifying the static part of a display, but provide little or no support for specifying the dynamic parts. At most, they provide the ability to use predefined components such as tables and graphs, but these components often lack the flexibility needed in many applications (see Figure 1 through Figure 3).

Even for the static parts, support is not adequate. Interface builders provide little support for the way graphic designers work. The layout facilities are patterned after the layout facilities of drawing editors where groups of elements can be left-aligned, right-aligned, etc. Graphic designers often work by defining guides and grids to organize page layouts [3, 15]. For example, Figure 6 shows the grid design for Apple's "Making it Macintosh" displays [1].

This paper describes a declarative language for modeling presentations that addresses many of the shortcomings of current tools. This language is part of the MASTERMIND model-based user interface development environment. The language was designed to meet the following goals:

1. Support both static and dynamic displays.

2. Incorporate principles of graphic design such as grids and guides.

3. Support automatic display update when data changes, or the presentation context changes (e.g. amount of screen space available).

4. Support for using display components for input operations (e.g., sophisticated ways to select objects on the screen).

5. Amenable to interactive specification.

Our approach is a declarative language for modeling displays with only a few primitives that can be composed in many different ways. The language has iteration and conditional constructs, and facilities to connect displays to application data in order to support goal 1. It uses guides, grids and constraints as the basic constructs to specify layouts to support goal 2. The declarative nature of the language is the key to supporting goals 3, 4 and 5 because declarative languages are good for supporting tools that reason about the specification. The run-time support system reasons about the specification to figure out how to update displays (goal 3), and to figure out what is displayed where in order to support input (goal 4).

The declarativeness of the language also facilitates constructing tools that reason about the display specifications. Example tools include critics which detect design problems, advisors and automatic generation tools that refine partially specified designs or generate designs based on the data to be displayed. Also, the language constructs can be visualized graphically to support interactive specification of displays.

The rest of the paper is organized as follows. The next section shows example displays to illustrate the range of presentations that can be modeled in MASTERMIND, and to illustrate the modeling constructs. The following sections describe the modeling constructs, related work and conclusions.

EXAMPLE DISPLAYS

The following figures show examples of displays modeled in MASTERMIND. We will use these examples to illustrate the shortcomings of current approaches for specifying presentations, and to illustrate the power and simplicity of our approach to specifying presentations.
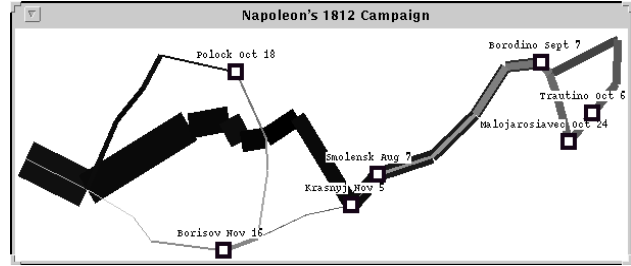

Figure 1. Napoleon's march to Moscow.

Figure 1 is the famous Minard chart showing Napoleon's march to Moscow. The thickness of the line encodes the number of troops in Napoleon's army, the line darkness encodes the temperature (darker is hotter). The squares and labels indicate places where battles took place. The input data consists of two lists of records. One containing information about latitude, longitude, number of troops, and temperature, and the other list containing records of the time and places where battles took place.

This figure is an example of a custom designed display that cannot be produced by any charting program. Sage [8, 9] can automatically generate this chart from the relational data, but it requires that each tuple provide the two end-points of each line. In MASTERMIND this display can be modeled independently of the format in which the data comes in (list of points or list of intervals).
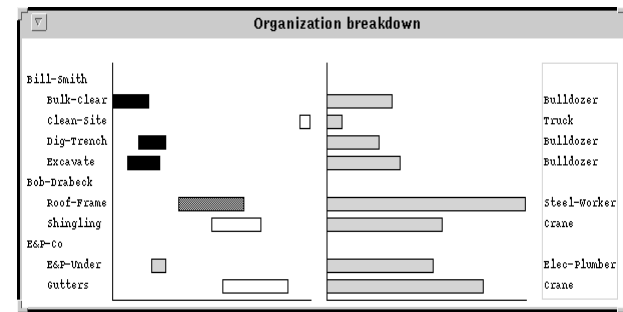

Figure 2. Complex bar-chart

Figure 2 shows a composite bar-chart. The input data is a list of three records one for each person. The record for each person itself contains a list of records about the activities that the person is managing. This chart cannot be produced by charting programs, but can be produced by Sage. The difficulty in generating this display is that it consists of two charts put side by side in a coordinated way, and each chart itself is a hierarchical composition of an outline display (the data for each person) with a chart (the activities managed by each person).
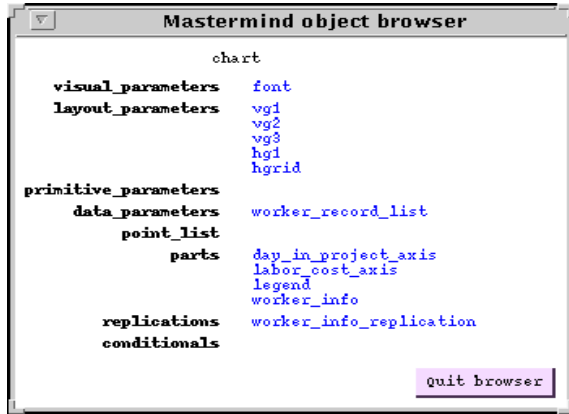
Figure 3. Browser

Figure 3 shows a browser to view objects in the MASTERMIND models. The figure illustrates the use of conditional presentations: if attributes have a single value, only a string is presented, but if they have multiple values, the values are shown as a column of labels.

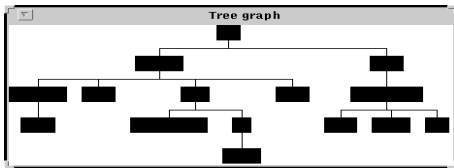

Figure 5. MASTERMIND Architecture

Figure 4. Tree layout

Figure 4 shows a tree structure where the width of each node is dependent on the information contained in that node. This figure is interesting because it shows that MASTERMIND supports recursively defined models.

ARCHITECTURE

To specify a display in MASTERMIND, developers build a model that specifies the structure and graphical components of the display, how the components are connected to application data, the visual appearance of each component, and how the components are laid out. Figure 5 shows the architecture of the MASTERMIND's presentation generation system.

The declarative model is translated into an executable representation called *component prototypes*, before it is given to the *run-time system* to generate and manage the displays at run-time. The component prototypes contain essentially the same information as the declarative model, except that the declarative information is translated into constraints that the run-time system can execute directly [4, 5]. To generate the displays, the run-time system takes the application data to be displayed and makes instances of the component prototypes, yielding a tree of graphical components, which is drawn to generate the displays that appear on the end-user's screen.
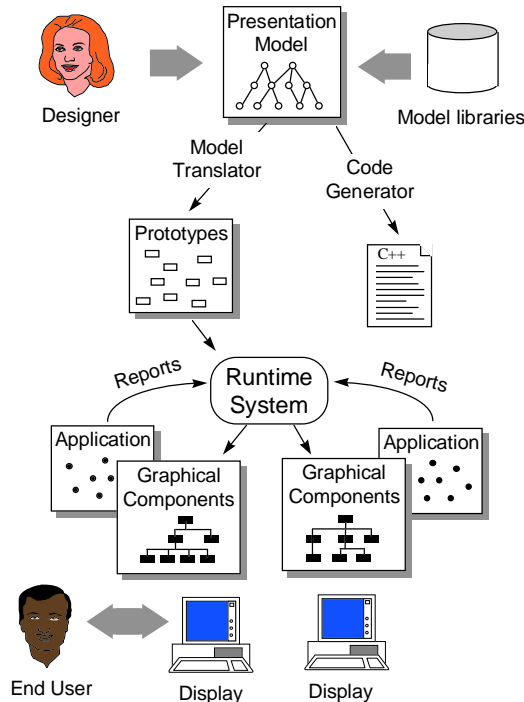
The run-time system updates the trees of graphical components in response to *change reports*. Applications are expected to generate reports when they change data that might be displayed. MASTERMIND also generates reports when the size of windows changes, or when the user interactively requests that the display format be changed. When reports arrive at the run-time system, they cause attributes of the component trees to be invalidated, which causes constraints to be invalidated. In the simplest case, re-computing the constraints yields new values for the component attributes that specify the visual appearance or layout of the graphical components. In the more complex cases, such as when the number of data elements to be displayed changes, re-computing the constraints causes components of the tree to be added or deleted.

The loose relationship between models and trees of graphical components gives MASTERMIND the flexibility to construct displays where the number of graphical components, the kinds of components, and their arrangement in the tree is actually determined at run-time while the application is executing.

In contrast, interface builders and multimedia authoring tools have a one to one mapping between the tree of graphical components that is generated at run-time and the specifications that developers provide. Developers essentially specify at design-time the graphical component trees to be generated at run-time, eliminating the flexibility required to generate dynamic displays. This is the reason why developers using these tools also need to use a scripting language, or a general purpose programming language to construct such dynamic displays.

The following sections describe the main elements of the MASTERMIND presentation modeling language.

PRESENTATION MODEL
In MASTERMIND a display is modeled as a composition of objects called *presentations*. Each presentation is specified as a refinement of another presentation, called its *archetype*. The MASTERMIND library contains a wide variety of presentations (e.g., window, group, button, rectangle) which can be used as archetypes of new presentations. In addition to the archetype, each presentation contains the following information:

Parameters. Each parameter has attributes to represent the type of the parameter value, and optionally, the value. The value can be either a constant, or an expression that computes a value in terms of the values of other parameters. Expressions often call application procedures.

MASTERMIND distinguishes between three kinds of parameters: v*isual parameters*, which specify the visual appearance of graphical components, such as color, linestyle, etc. *Layout parameters*, which contain information to specify the layout. Layout parameters must be of type Guide, Grid or Magnitude, which are the building blocks for specifying layout. *Data parameters*, which specify the application data to be displayed in a presentation.

Parts. A possibly empty list of parts. Each part is itself a presentation, so it can itself have parameters, parts, replications and conditionals. The parts define the hierarchical decomposition of the display.

Replications. For each part that might be replicated there is a set of attributes that specifies how to generate the replicas (e.g. with respect to a sequence of data elements stored in a data parameter).

Conditionals. A set of rules that specify adjustments to a presentation, and the conditions under which the adjustments are appropriate.

The Napoleon display can be used to illustrate the main constructs for defining presentations. The syntax shown here is just for the purpose of describing the model in this paper. Developers are expected to construct the models using interactive tools, so the textual syntax that MASTERMIND provides is designed to be easy for the tools to generate and to read, and thus is not necessarily easy for people to read.

```
1   Napoleon_Window : Window
2   data_parameters
3     march_data : sequence<Data_Info>
4     battle_data : sequence<Battle_Info>
5   parts
6     segment : Line
7       replicate_for [march] in march_data
8       start_index : 2
9     battle : Group
10      replicate_for [battle] in battle_data
11      parts
12        city : Rectangle
13        name : Label
14        date : Label
```

The Napoleon march display has two data parameters that specify the application data to be presented. The data for the march (march_data) is a sequence of application objects of type Data_Info. The battle data is a sequence of Battle_Info objects. The display consists of two replicated parts called segment and battle. The battle part itself has three parts corresponding to the little square and the two labels. The replicate_for statements specify that the part should be replicated. The symbol in square brackets is the name of a data parameter used to store the data sequence element assigned to each part replica. The start_index specifies that replicas should be generated starting from the second data point (there are N points and N-1 line segments). The details of how the replication and the layout is defined will be described in the next sections.

Since the part hierarchy is not used to define layout, but only the components of a window, the fact that MASTERMIND uses a purely hierarchical decomposition of displays does not impose restrictions on expressive power. As Figures 1 to 4 illustrate, even displays with fairly complex layouts can be specified in MASTERMIND. The reason is that the layout information is separate from the hierarchical decomposition.

Replications
As illustrated in Figures 1 to 4, there is a large class of displays that show variable amounts of application data. The MASTERMIND replication mechanism is designed to make it easy to specify such displays. The replication

mechanism is tied to the part decomposition of a display. Developers decompose a display into parts according to the kinds of components that can appear in the display, and then specify that some of those parts might appear multiple times. The idea is to allow developers to specify how one replica of a part should be displayed, and then to easily augment the model to say that the part should appear multiple times. This strategy should be easy for developers to understand, and amenable to be specified using interactive tools and demonstrational techniques.

The layout of a set of replicated parts can be specified using two strategies called the *reference* strategy and the *anchor/generic* strategy. The reference strategy uses another object as a reference for laying out the parts. The most common reference is a grid, in which case the elements are placed in the grid (the different ways to use grids are explained in the layout section). A different set of replicated parts can also be used as a reference. In this case each of the parts being laid out will be placed in reference to a corresponding part in the reference set. Developers must provide constraints to specify how each part is placed with respect to its reference (e.g., placed below the reference). This facility makes it easy to, for example, add decorations to the parts in that sequence (e.g., a line between each of the components).

The anchor/generic strategy involves specifying where the first element of the replication should be placed, and then specifying the placement of each of the other items with respect to the previous one (more details in the section about layout).

In general, the run-time behavior of a replication specification is as follows (some of the details may vary depending on the attributes of the replication specification). MASTERMIND first computes the sequence of data elements to be displayed. For each element in the data sequence MASTERMIND creates a replica of the part (i.e., its corresponding graphical component) and stores the data item in the parameter specified by the replicate_for statement. This allows each replica to access the data item that it displays. By default MASTERMIND first generates all the replicas and then lays them out according to either the reference or anchor/generic strategy. If the reference method is used for layout MASTERMIND stops generating replicas when the data sequence runs out, or the associated reference is exhausted (i.e., runs out of grid lines, or the referenced replication runs out). There is also an on_demand keyword to override the default replication behavior. When on_demand is true, replicas will be generated only as needed by other components

(e.g., another replication that uses this replication as a reference).

MASTERMIND supports automatic display update when data sequences change (elements added or deleted), when the allocated display area changes size, or when replicas change size.

Figures 1 to 4 illustrate the many different ways in which replications can be used. As explained above, the Napoleon march display uses two replications to generate the line segments and the battle information. Since this display shows geographical data, the layout of the replicas is based on the latitude and longitude of each data point, and so it uses expressions to compute the locations of the points based on the application data.

When combined with the conditionals facility, developers can specify adjustments to the presentation of each data item depending on the characteristics of the item or its index (e.g., elements of even index have a darker background).

Conditionals

Conditionals are the mechanism to specify displays whose structure, layout and visual appearance depends dynamically on the data to be presented, on the space available, and on other characteristics of the display.

MASTERMIND supports a conditional presentation mechanism that allows developers to specify rules (condition/action pairs) that describe how to adjust a presentation when certain conditions are true (e.g., if only 10 grid lines are available, hide parts A and B). Each presentation can have a set of conditionals. Each conditional consists of a sequence of rules that specify how the presentation should be adjusted. When generating a presentation, MASTERMIND will visit each set of rules, and apply the first rule in each sequence whose condition is true.

The conditions are expressions that test properties of the data (e.g., whether a number is in a certain range), or that test properties of the display (e.g., whether a graphical component fits in a grid cell).

The actions contribute features incrementally to the graphical component being created. Actions are specified as partially filled in presentation specifications, thus giving conditionals the power to modify any aspect of a presentation. An action can even override the archetype of a part and add more parts. Typically, actions just specify values for visual parameters.

Conditionals are sensitive to changes in both application and presentation context. While evaluating conditionals, MASTERMIND records all the conditions tested while generating a graphical component. If the application data or the characteristics of the display change MASTERMIND can undo the actions for the conditions that are no longer true, and apply the actions for the conditions that became true. This results in the display being updated according to the conditions that are currently true.

MASTERMIND's conditional presentation is not as sophisticated as the presentation planning components of systems like Sage [8, 9] and APT [2], which can do backtracking, and thus plan presentations more intelligently. MASTERMIND's mechanism is designed to let developers pre-specify the kinds of adjustments that should occur at run-time, and it is simple to understand, easy to control and efficient to implement.

Connecting Displays to Application Data
Connecting displays to application data involves both providing a way for the interface to access application data, and providing a way for the application to announce changes to data that should cause the displays to update.

Accessing application data is simple: presentations can access application data by calling application procedures from within expressions used in the various elements of a model (e.g., expressions to specify the value of a parameter, or the condition of a conditional presentation).

Since applications do not know which of its objects are displayed and how, applications cannot directly trigger display updates. MASTERMIND uses a broadcast mechanism to let applications announce changes to data. This mechanism is based on two constructs: *reports*, which the application sends to MASTERMIND, and *report patterns*, which the presentation components use to select from the incoming reports those that should cause the particular presentation to update.

A *report* is a structured object with fields for specifying the kind of report (change, add, remove, delete), a logical identifier, which is an arbitrary symbol, a reference to the object that was changed, and uninterpreted fields for additional information. A report pattern is essentially a partially filled in report structure. MASTERMIND provides a language for the application to declare what kinds of reports it can produce at run-time, and an API that allows applications to send reports to MASTERMIND. Typically, the body of the application procedures needs to be modified to include calls to the routines that send reports to MASTERMIND.

The reports mechanism allows changes to be announced at a logical level. For example, if two elements of an array are swapped, the change should be announced as a "swapping" report rather than as two independent "change" reports. Logical or abstract events are crucial to allow the interface to show the changes to users in a meaningful way. In the array example, it is more appropriate to show an animation of the elements being swapped, rather than simply updating the screen.

Visual Appearance
The main determiner of the visual appearance of a presentation is the archetype (e.g., if the archetype is a rectangle, then the presentation will look like a rectangle). However, each presentation defines a set of parameters to control its visual appearance (e.g., fill and line-style colors of a rectangle). The values of the visual appearance parameters can be set to constants or to expressions that compute a value based on the values of other parameters and of application data.

Layout
Specifying the layout of a presentation ultimately involves specifying numbers for the location and size of all parts of a presentation. MASTERMIND provides high level facilities to specify layouts in much more convenient ways.

The MASTERMIND layout facilities are based on the grid design techniques used in graphic design. Figure 6 shows a good example of how these techniques can be effectively used in practice. The figure shows the screen anatomy of Apple's "Making It Macintosh" displays [1]. There is a guide at the left to align the text items in the left part of the window, and the title at the top of the window. There is a grid to place the text and buttons next to the text, and also to define the spacing between paragraphs.
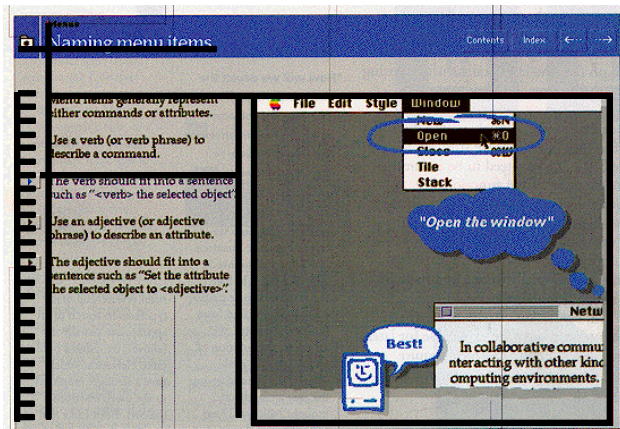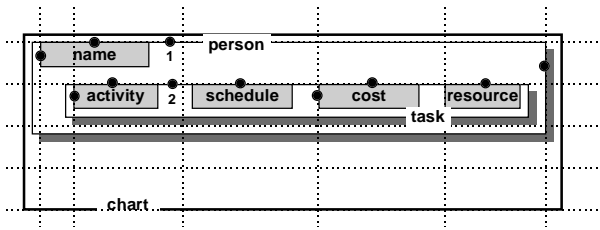

Figure 6. "Making it Macintosh"

Figure 7. Structure of the chart display.

In MASTERMIND layouts are specified using three constructs:

- *Guides*. A guide is a horizontal or vertical line. Its position can be defined using a constant or an expression that computes a value based on the positions of other guides.
- *Grids*. A grid is a set of horizontal or vertical lines. Grids are characterized by four quantities, the number of lines, the separation between the lines, the start and the end positions. A grid is defined by specifying three out of the four quantities.
- *Constraints*. The position of guides and grids is usually defined using an expression that computes a value based on the position of other guides and grids.

MASTERMIND's guides and grids can be used directly to set-up a layout such as the one shown in the "Making it Macintosh" figure. For example, the grid spacing would be defined in terms of the font size using an expression, and the top and bottom of the grid would be defined to match horizontal guides at the top and bottom of the screen.

Interface builders do not provide adequate facilities for doing good page design as illustrated in this example. Some interface builders provide grids, but they are used just for initial placement of the items. It is not possible, for instance to specify the grid size based on font size, so that if the font size is changed at run-time, the design doesn't break apart.

Guides and grids are also crucial for defining the layout of replicated parts. In the simplest case, a replication can be assigned to a grid, meaning that consecutive replicas are placed in consecutive grid-lines. Additional parameters can be used to adjust this basic strategy: it is possible to specify the grid-line index for the first replica, to specify the number of grid-lines to be occupied by each replica, to specify that each replica should go to the next free grid-line, etc. Nested replications can be assigned to a common grid, so the elements are placed sequentially on the common grid.

The chart display (Figure 2) is a good example of the use of replications, grids and guides. Figure 7 is a drawing showing the structure of the chart display, which is defined by a presentation called chart. It has one part called person, which is replicated (as shown by the shadow) with respect to each person for whom data is going to be presented. The person part has itself two parts, one called name, which displays the name of the person (e.g., Bill-Smith), and a replicated part that represents each row of the chart showing the information about each task that the person is in charge of. This part itself has parts corresponding to the task's activity name, schedule, cost and resource usage.

The layout of the parts is specified using guides and grids. The chart defines a horizontal grid that the two replicated parts share as their reference, as indicated by the black dots numbered 1 and 2. When the display is created, each person component will start in a new grid line, and each task created within the person presentation also starts in a new grid line. The vertical guides are used to define the horizontal placement of some of the parts. The black dots indicate guide and grid snappings that define the placement of other components. The horizontal position of the activity, schedule and parts is defined in terms of the vertical guides also, but use more complex expressions to compute the position. For example, the left and right of the schedule depend on the two surrounding guides and the start and end-date for the task.

Recursive presentations such as the tree shown in Figure 4 can also use a grid, where each level of recursion is assigned to the next grid line.

Replications can also be laid out using an anchor/generic mechanism based on conditional presentation. The position of all replicas is specified by specifying the position of the first element (anchor), and by specifying the position of all other elements in terms of the previous one. This is accomplished using two conditional presentations, one that applies to the replica with index one, and one that applies to replicas with index greater than one. The anchor/generic mechanism also allows developers to define more than one generic presentation, say one for the odd-indexed replicas and one for the even-indexed replicas.

The tree display also illustrates the use of the anchor/generic mechanism to specify layouts. The horizontal layout of the children of a node is specified as follows: the anchor is the first child, and its left is specified as the left of the whole sub-tree. The left of all the other

children (generic) is the right of the previous child plus an offset.

RELATED WORK

MASTERMIND's presentation model [13] is based mostly on ideas from HUMANOID [10, 11 and 12], and ITS [14]. MASTERMIND's model of presentation is based in HUMANOID's templates, which also had constructs for replication and conditionals. The main improvement in MASTERMIND is the model of layout based on grids and guides, and the way in which the layout model is integrated with replications and conditionals. Neither HUMANOID or ITS or any other interface construction tool has a model of layout that supports the grid-based design methodology that graphic designers have developed for page design. The integration of the grid-based layout design with the replication construct yields a powerful and simple scheme for laying out collections of graphical components.

MASTERMIND's conditional presentation mechanism has been redesigned to be more similar to the ITS style rules. Like ITS rules, and unlike the HUMANOID conditional construct, the MASTERMIND rules can independently contribute elements to build up a component. Unlike ITS, MASTERMIND records the dependencies of the rule conditions so that if data changes, the rules are re-applied in the appropriate way to yield an updated display.

MASTERMIND borrows the idea of using constraints for propagating values in the component tree from Garnet [4, 5]. MASTERMIND is implemented using Amulet, the C++ version of Garnet. MASTERMIND's models are translated into Amulet objects, and make extensive use of the Amulet constraint system. One can think of MASTERMIND as a declarative layer on top of Amulet, providing convenient constructs (replication, conditionals and grid-based layout) to specify interfaces at a higher level.

In many respects, MASTERMIND is similar to automatic presentation planners such as Sage [8, 9] and APT [2]. MASTERMIND's models of the charts that Sage can generate are very similar to the Sage specifications that Sage uses to render the displays. The advantage of Sage over MASTERMIND is that Sage can generate the specifications automatically based only on the data to be presented. However, Sage can only produce presentations from relational data, where as MASTERMIND can model a much larger variety of displays. In addition, MASTERMIND also models user tasks and dialogue (not described in this paper), so it can model all aspects of a user interface.

CONCLUSIONS

MASTERMIND's few presentation modeling constructs (part decomposition, replications, conditionals and grid-based layout) can be composed in a large variety of ways to specify a large number of displays. The figures provided in this paper show a small sampling of the ways in which these constructs can be combined to construct displays which are not adequately supported with current tools.

MASTERMIND is currently under active development, and the system is not yet complete. We currently have a complete specification of the modeling language [13] and a knowledge representation system to represent the models. The translator that converts models into the component prototypes is not yet implemented, and the run-time system is only partially implemented. We built the models for all displays shown in these paper, and we hand-translated them into the component prototypes which the run-time system uses to generate the displays.

After completing the model translator and the run-time system we will start working on interactive tools for building these models. These tools will be built using MASTERMIND itself. We plan to release the system for general use in the summer of 1996.

REFERENCES

1. L. Alben, J. Faris and H. Saddler. Making It Macintosh: Designing the message when the message is design. Interactions, v1.1, January 1994, pp. 10-22.

2. J. Mackinlay. Automating the Design of Graphical Presentations of Relational Information. ACM Transactions on Graphics, pp. 110-141, April 1986.

3. K. Mullet and D. Sano. Applying Visual Design: Trade Secrets for Elegant Interfaces. Tutorial 2, CHI'94 Conference on Human Factors in Computing Systems, April 24-28, 1994.

4. B. A. Myers, D. Giuse, R. B. Dannenberg, B. Vander Zanden, D. Kosbie, E. Pervin, A. Mickish, and P. Marchal, Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces IEEE Computer, Vol. 23, No. 11, November, 1990.

5. B. A. Myers, et. al. The Garnet Reference Manuals. Technical Report CMU-CS-90-117-R2, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213. May 1992.

6. B. A. Myers. User Interface Software Tools. ACM Transactions on Computer Human Interaction, v2, n1, March 1995, pp. 64-103.

7. R. Neches, J. Foley, P. Szekely, P. Sukaviriya, P. Luo, S. Kovacevic, and S. Hudson. Knowledgeable Development Environments Using Shared Design Models. In Proceedings of the International Workshop on Intelligent User Interfaces, Orlando, Florida, Jan., 1993

8. S. F. Roth and J. Mattis Data Characterization for Intelligent Graphics Presentation, in Proceedings SIGCHI'90 Human Factors in Computing Systems, Seattle, WA, ACM, April, 1990, pp. 193-200

9. S.F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein,. Interactive Graphic Design Using Automatic Presentation Knowledge, in Proceedings of the CHI'94 Conference (Boston, April 1994), ACM Press, pp. 112-117

10. P. Szekely. Template-based mapping of application data to interactive displays. In Proceedings UIST'90. October 1990, pp. 1-9.Szekely 92

11. P. Szekely, P, Luo, and R. Neches. Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. In Proceedings SIGCHI'92. May 1992, pp. 507-515.

12. P. Szekely, P. Luo, and R. Neches. Beyond Interface Builders: Model-Based Interface Tools. In Proceedings of INTERCHI'93 April, 1993, pp. 383-390.

13. P. Szekely and P. Castells Documentation for The MASTERMIND Presentation Ontology. Http://www.isi.edu/isd/Mastermind/Documentation. August 1995.

14. C. Wiecha, W. Bennett, S. Boies, J. Gould and S. Greene. ITS: A Tool For Rapidly Developing Interactive Applications. ACM Transactions on Information Systems 8(3), July 1990. pp. 204-236.

15. R. Williams. The Non-Designer Design Book. Peachpit Press Inc., Berkeley, California, 1994