Adaptive multimedia interfaces in PolyMestra

Ephraim P. Glinert and G. Bowden Wise

Computer Science Department, Rensselaer Polytechnic Institute,
Troy NY 12180, USA

glinert@cs.rpi.edu, wiseb@cs.rpi.edu

ABSTRACT

An architecture for a new generation of multimedia systems is presented based on the concept of metawidgets, which are collections of alternative representations for information, both within and across sensory modalities, along with user—transparent mechanisms for choosing among them. The proposed architecture allows us to overcome certain drawbacks of today's systems, where the designer typically must assign each component of the display to a specific modality in a fixed and inflexible manner. The design of the PolyMestra environment based on our architecture is next described in detail, with particular emphasis on the layered development approach, core software tools and inter—application communication. Finally, we discuss the current status of the implementation, and outline plans for distribution of the prototype later this year to get user feedback.

Keywords: multimedia systems, multimodal widgets, object oriented frameworks, C++, Standard Template Library (STL)

1. INTRODUCTION

Multimedia systems which include sophisticated audio output capabilities are fast becoming the standard platform for most users, both at work and at home. No longer restricted to a visual medium, these systems allow users to not only see the information presented to them but to hear it as well. Research into so—called virtual reality hints that it may not be long before our repertoire of standard interaction techniques is further augmented to include touch, gestures, voice and 3D sound.

The expanded palette of interaction technologies is attractive, in that they may enable users to communicate with their computers in a more "natural" way. But this additional freedom also presents new challenges to software designers, who must now develop applications which deliver information to end users in the most effective manner possible in a multisensory realm that encompasses text, graphics, speech, nonspeech audio, etc.

Successful concurrent exploitation of several modalities requires careful planning, otherwise some information may not be perceived by the user. Today's multimedia applications typically prescribe the modality in which any given information is presented in a hard coded (predetermined) manner. The drawback of this strategy is that, no matter how much or how well the systems organizer wrestles with the issue of how best to design the output, he/she is fighting a losing battle.

The reason is simple. An inflexible assignment of information to a particular modality by the designer must eventually lead to situations in which the chosen output modality is unacceptable to a given user and/or the circumstances at hand. For example, environmental conditions such as a noisy factory floor can preclude the use of sound. Even disregarding such "external" sources of interference, users often run several applications simultaneously and must absorb information from all of them; one issue to consider in this case is that too much (cumulative) information in a single modality can lead to sensory overload.

Most importantly, from our viewpoint, many users have sensory impairments and cannot interpret information in one or more modalities. Application designers who create systems whose output is rigidly distributed over multiple modalities therefore sharply reduce the potential customer base for their products! A better approach, in our opinion, is for applications to become multimodal at a high (abstract) level so that they are able to display all information by exploiting alternative and/or complementary sensory modalities, as (changes in) the working environment, individual users' needs and preferences, and other factors (both extra— and intra—system) dictate.

This report describes ongoing research, in which we are exploring a new architecture for multimodal systems based upon the preceding ideas. Mestra was a figure in ancient Greek mythology who, in order to escape the slavery into which she had been sold by her father, prayed to Poseidon who loved her and conferred upon her the power of metamorphosis so that whenever she was sold she could escape by changing her form. Our choice of name for our system thus reflects our goal of freeing users of multimedia systems from the confines of inflexible information displays by supporting metamorphosis of information from one modality to another.

In what follows we first present and justify our architecture for the new generation of multimedia systems we envisage. We then discuss in detail the design of the PolyMestra environment for IBM—compatible PCs which is based upon our architecture, and whose implementation is currently in progress. Finally, we outline our plans for distribution of the prototype to get user feedback later in the year.

2. AN ARCHITECTURE FOR MULTIMODAL SYSTEMS DEVELOPMENT

Multimodal applications (i.e., those which must or can present information in two or more modalities) require a very different environment than that now provided by graphical user interfaces (GUIs), because the simultaneous presentation of information in more than one modality requires careful management if all the information is to be understood rather than lost. In this section, we define concepts and present a development methodology for end user applications that are empowered to present information in alternative modalities. However, before presenting our architecture we briefly review pertinent findings from the science of human perception and from cognitive psychology, in order to get a better understanding of how we process information in a multisensory environment.

2.1. Cognitive Aspects of Multimodal Interaction

At any given moment, each of us is performing a variety of tasks. Sometimes, we are able to easily perform two or more tasks at once (e.g., driving a car, listening to music, and drinking a cup of coffee). At other times, we are only able to perform one task at a time because it requires so much concentration (e.g., operating a power saw). How can we explain these apparent limitations and discrepancies in the behavior of the human cognitive system? Psychologists call this mental attention.

The user of a multimodal computing environment is bombarded by a multitude of informational messages, each of which may be presented in a different modality. The user must attend to each message, in order to comprehend it. When there are insufficient attentional resources, information will be lost. If a multimodal interface is to be both usable and efficient, the probability of the user missing a message or not understanding it must be minimized.

To gain a better understanding of how to present information more effectively using several modalities, we examined a number of theories of attention, including: capacity theory, resource theory, confusions theory, and compatibility of proximity. Fracker and Wickens (Fracker and Wickens, 1989) provide an excellent discussion of these ideas; we summarize here just that work which we found most applicable to our research.

Most modern theories of attention are based on capacity theory (Knowles, 1963; Moray, 1967), which contends that information related to simultaneous tasks is processed in parallel until the mental load forces a bottleneck to ensue, after which only one task may be performed at a time. However, the precise nature of this “bottleneck” has been a source of conflicting opinion among psychologists.

A number of theories assume that task performance can be related to the task’s demand for processing capacity, commonly referred to as “resources.” When multiple tasks are performed simultaneously they compete with one another for the limited mental resources available, which may ultimately cause performance to deteriorate. These theories have been collectively called resource theories (Kahneman, 1973; Navon and Gopher, 1979; Wickens, 1984; Wickens, 1987).

Early resource theory (Kahneman, 1973) hypothesized that there is a single, undifferentiated pool of resources available to all tasks and mental activities. The available resources can be used for several tasks, provided the demands imposed by the tasks do not overly deplete the finite pool of resources. Attention is envisaged as a resource that can be distributed among different stages of processing, depending upon task demands.

More recent theories, however, suggest that there is more than one pool of resources. Multiple resource theory (Wickens, 1984; Wickens, 1987; Boles and Law, 1992) shares the philosophy that several tasks can be performed concurrently as long as they do not compete for the same resources. Unlike the earlier models, however, all tasks are not assumed to compete for a single, undifferentiable pool of resources. Rather, the hypothesis now is that there are multiple pools of resources within the human processing system which span three orthogonal dimensions (see Fig. 1):

(1) Stage of information processing (encoding, central processing, or response selection).

(2) Modality of input (visual or aural).

(3) Code of information processing (spatial or visual).

Each dimension can be thought of as a separate pool of resources. For the stage—defined resource pools, information is assumed to be processed by humans in stages, going from encoding to central processing to response selection. Tasks in the different stages use different resources for the most part. This implies that humans can encode or centrally process information for one task at the same time they are responding to another.

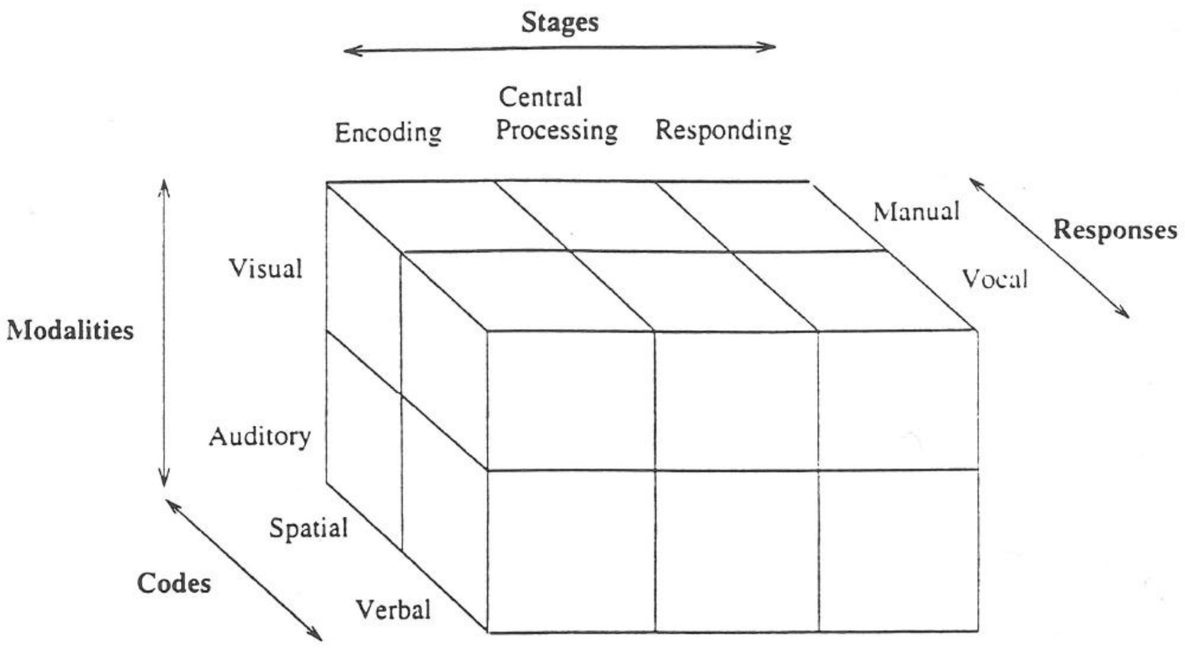

Figure 1: *Dimensions of resources found in the human processing system as suggested by Wickens (Wickens, 1984).*

For the modality—defined resource pools, tasks can be more easily performed when each utilizes a different modality. Thus, humans are able to attend to a visual message and an auditory message at the same time. This is directly relevant to our multimodal architecture, since it suggests that users will be able to comprehend more information when it is divided among several modalities rather than presented in a single modality.

Information can also be presented either visually or spatially. Performance will be better when information for different tasks is presented using different codes rather than all being presented in the same manner. This dimension impacts all stages of processing: encoding (speech vs. graphics); central processing (spatial working memory vs. linguistic working memory); and response (speech output vs. manual output).

2.2. Metawidgets and Representations: A Substrate for Multimodality

Typically, applications which run under today's GUIs are developed using libraries of widgets which represent familiar user interface elements such as windows, icons, and buttons. However, because they are constrained a priori to the visual modality due to their graphical nature, widgets are too low level to be of use directly by a multimodal application, which must communicate with the environment in which it lives at a higher level of abstraction in order to interact with the user in any of several modalities in a manner transparent to the applications programmer.

Metawidgets, proposed by Glinert and Blattner (Blattner, Glinert, Jorge and Ormsby, 1992; Glinert and Blattner, 1993), are multimodal widgets which represent some informational abstraction of the application that is to be presented to the user. Each metawidget contains a repertoire of representations, each of which may be in a different sensory modality (or combination of modalities), as well as methods for selecting among them. By taking into account the user's preferences, along with relevant extra—and intra—system factors, the selection mechanism can determine which representation is optimal for presentation when the metawidget is displayed.

From a functional point of view, metawidgets must be able to select a suitable representation, display it to the user, compute the cognitive resources this representation will consume, and communicate with the underlying core software. Many of these operations, if not all, will be performed the same way for every metawidget, indicating that much of the metawidget code may be reused.

2.3. Modalities and Cognitive Resources

Users often run several applications simultaneously, and must absorb information from all of them. Too much (cumulative) information in a single modality leading to sensory overload is one potential problem to be overcome. Cross—modality conflicts, as when the visual dominance (Wickens, 1992) of the human perceptual system causes an auditory stimulus to be ignored, are another.

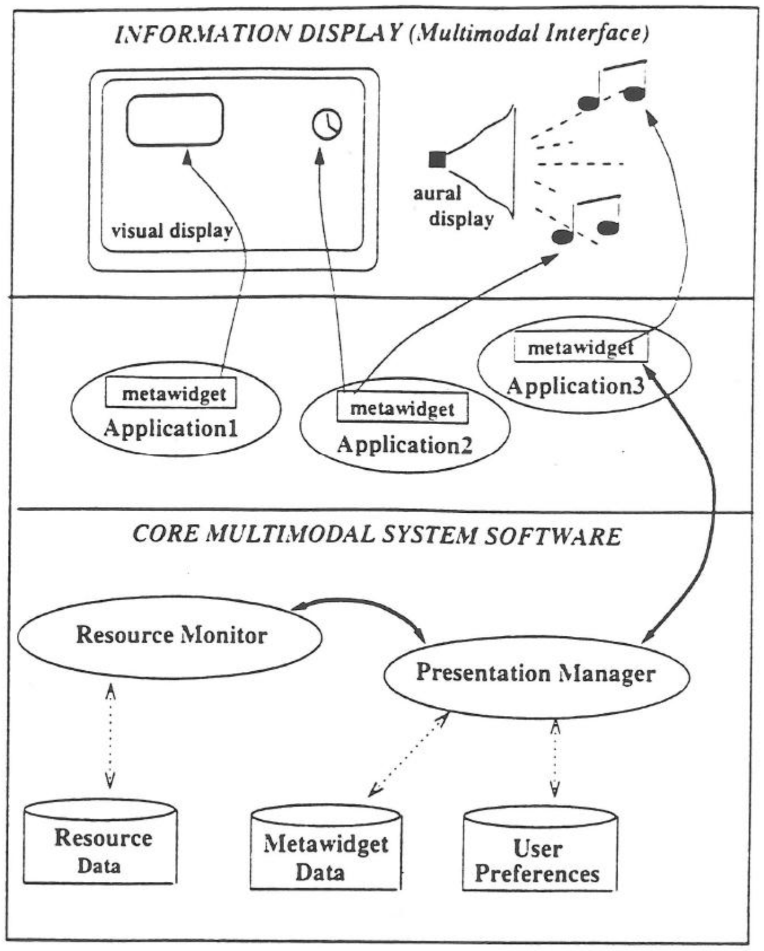

Figure 2: *Run time view of a multimodal environment. Thin lines indicate the representations associated with a metawidget. Thick lines show inter—application communication paths. Dotted lines show internal data paths. Although each metawidget will have a similar communication path to the Presentation Manager, only one is shown in the figure for clarity's sake.*

To eliminate or at least alleviate such phenomena, the demands on the user's cognitive system must be periodically monitored by the computing environment. We define a cognitive resource as some aspect of the user's perceptual or attentional processes that is to be monitored for this purpose.

Each representation of a metawidget resides in one or more modalities. Each modality of interest will have a set of cognitive resources defined for it, along with heuristics for determining the contribution of each such resource to the user's total cognitive load. If a representation resides in more than one modality, it will consume resources from all of them. It is the responsibility of the underlying core software to monitor which representations are currently active, measure their contribution to the overall cognitive load, and take action where appropriate to prevent the user from becoming overloaded.

In our architecture, data on currently active applications, their metawidgets and associated representations are collected and assessed by a pair of core system tools which run in the background concurrently with the user's applications (cf. Fig. 2):

- Resource Monitor. This tool is responsible for determining the contribution to the total cognitive load by each application running in the environment, on the basis of its consumption of cognitive resources relevant to the user.
- Presentation Manager. This tool handles the presentation of information so that the user does not become overloaded. The cognitive load, user preferences, and other system data are all weighed to determine whether some information needs to be transformed into a representation in an alternative modality, and if so which modality is the optimal choice for this user at this time.

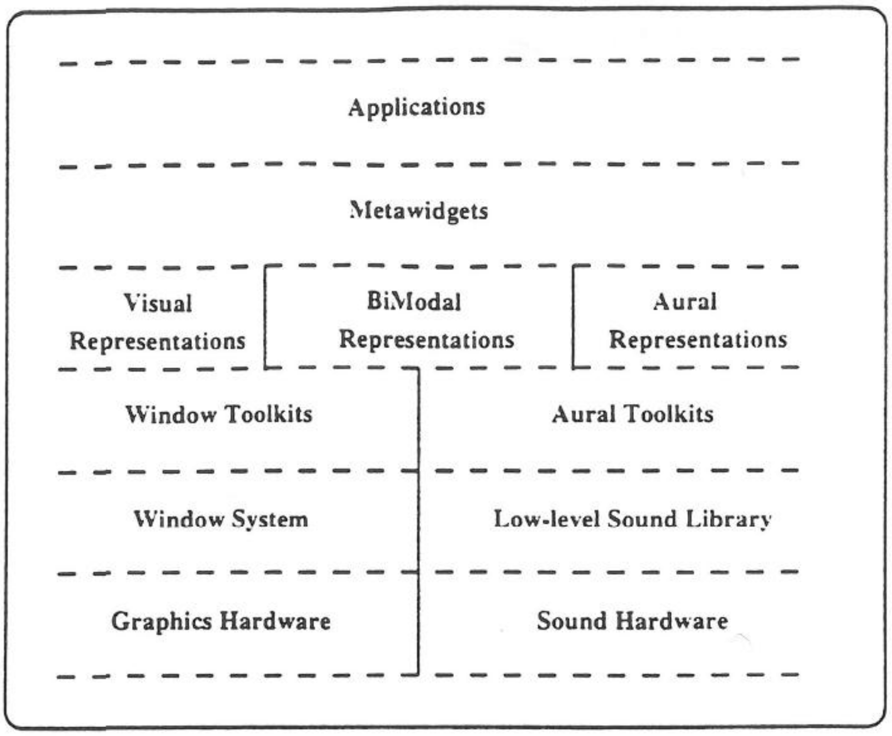

Figure 3: *A layered approach to multimodal systems development.*

2.4. A Layered Development Approach

Metawidgets, with their associated representations, provide natural building blocks for developing multimodal applications. Thus, at the heart of our architecture for multimedia computing environments lies a layered multimodal framework, as shown in Fig. 3. The representations of a metawidget are developed using toolkits containing widgets in a particular modality. Visual representations, for example, can be constructed using a toolkit of widgets for the underlying window system. Similarly, aural representations can be developed from audio toolkits that utilize the underlying sound hardware.

3. POLYMESTRA: A MULTIMODAL ENVIRONMENT FOR MICROSOFT WINDOWS

PolyMestra is a multimodal environment based upon our architecture currently under development for IBM—compatible PCs. PolyMestra provides application designers with a C++ framework for building multimodal applications for the Microsoft Windows 3.x platform. In this section, we discuss the implementation of PolyMestra in detail, with particular focus on some of the challenges encountered and our solutions for overcoming them.

3.1. Metawidgets and Representations

One of the early crucial decisions was how to associate a metawidget with its representations. What is needed is a mechanism that allows the current representation of a metawidget to change, without the metawidget having to be aware of the precise representation chosen (the metawidget should only be concerned with whether the selected representation is in an acceptable format to, and does not overload, the user).

Four basic assumptions drove the metawidget—representation design:

(1) Each metawidget must have one or more representations.

(2) These representations may be in the same or (preferably) different modalities.

(3) The representation to be displayed by a metawidget is selectable at run time, based on user preferences as well as extra—and intra—system information.

(4) The representation of a metawidget may change (metamorphose) while it is on display.

These assumptions imply that the coupling between metawidgets and their representations does not depend on the content of the information being displayed or its modality of presentation. This suggests that the mechanism employed to associate a metawidget with its representation can be common to all metawidgets.

In C++, such a mechanism is easily implemented by having each metawidget contain a base class pointer to its representation. By deriving all representations from a common base representation class (called BaseRepresentation in PolyMestra), the metawidget can point to an instance of any one of the derived classes. Furthermore, this approach ensures that all representations for a metawidget share the same interface and reuse common functionality.

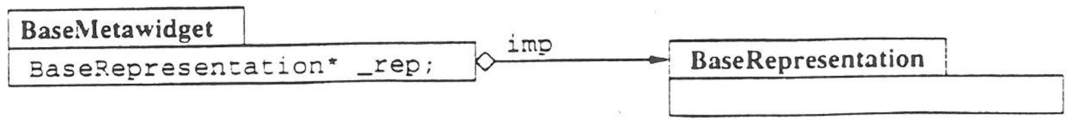

Figure 4: *A metawidget in PolyMestra contains a pointer, or handle, to its representation. The arrowhead line marked 'imp' with a diamond at its tail indicates that metawidgets are implemented by one or more representations.*

Once an optimal representation for a metawidget is chosen, the base class pointer is updated to point to that representation. Whenever a metawidget is told to display/hide itself, to calculate its resources requirements, or to send these requirements to the core software, the operation is forwarded to the active representation (via virtual functions).

Similarly, all metawidgets are derived from a common base class BaseMetawidget, so that every metawidget also shares the same interface and reuses functionality common to all metawidgets. BaseMetawidget contains a data member, namely a pointer of type BaseRepresentation to hold the currently selected representation. In addition to methods for selecting a representation for a metawidget, the operations provided by every representation are also implemented in BaseMetawidget except these methods simply forward the operation to the metawidget's representation through the base class pointer.

Pointers used in this way are commonly called handles (Coplien, 1992; Murray, 1993). Handles provide the architecture that enables metawidgets to utilize multiple representations, to delegate operations to the current representation, and to change the representation at run time. Handles may also be used to hide implementation details, to minimize the impact of changes during development, to determine the type of an object from the context in which it is constructed, and to create objects when an object's size is unknown.

Fig. 4 shows how the representation of a metawidget is implemented as a pointer to a BaseRepresentation. The object oriented class diagrams found in this figure and the next are patterned after Rumbaugh's Object Modeling Technique (OMT) (Rumbaugh, Blaha, Premerlani, Eddy and Lorenson, 1991). Classes are drawn in rectangular boxes; arrowhead lines are used to denote relationships between classes. Simple arrowhead lines indicate inheritance, while arrowhead lines with a diamond at the tail denote aggregation. Pseudocode for class methods is sometimes shown in a dashed box connected by a dashed line to the operation it represents.

3.2. The Representation Repository

Another design issue concerned how to manage the collection of available representations for each metawidget. Not only will each metawidget have more than one representation, but a particular representation may be useful to more than one metawidget! Complicating the matter further was our desire to make it possible for developers of a multimodal application to easily incorporate additional representations into a metawidget's existing palette of alternatives, without breaking existing application code.

Given that the representations for a particular metawidget are cataloged in some way, a mechanism is also needed for a metawidget to obtain a list of these representations at run time so that one of them may be chosen. This same list of representations can also be consulted to select a new representation during metawidget metamorphosis.

In PolyMestra, representations for a particular informational concept share a common base class, from which all representations in the collection are derived. A metawidget is associated with a particular collection of representations by identifying itself with a base representation class. Each metawidget contains a virtual method, BaseRepName(), which returns the name of the metawidget's base representation class.

Fig. 5 demonstrates how a metawidget class for text strings, MTextString, is associated with a collection of text strings derived from a common base class BaseTextString. In this example, four different representations of a text string are derived from BaseTextString: two visual representations VTextLabel and VTextButton, one aural representation ATextAnnounce, and one bimodal representation, BTextView. By specifying its base representation name as BaseTextString, metawidget MTextString may present itself using any of the four representations derived from BaseTextString.

As the reader may already have noticed, we have adopted a naming convention for class names in the PolyMestra framework. The prefix of each class name indicates its category: "M" for metawidget classes;

"Base" for base representation classes; "V" for visual representations; "A" for aural representations; and "B" for bimodal representations. These naming conventions are employed in several of our figures.

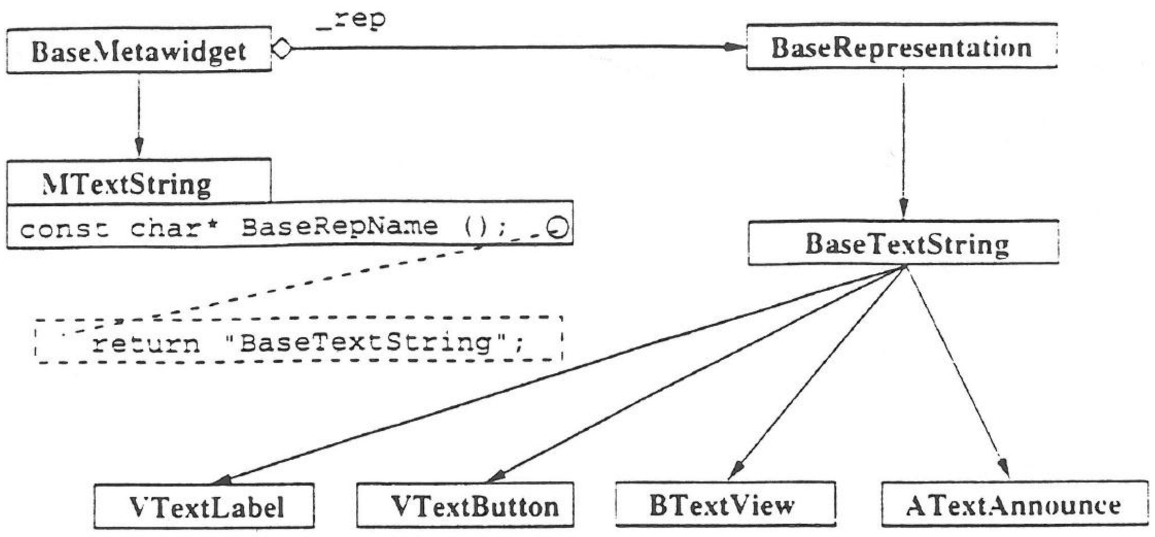

Figure 5: *Representations of a metawidget are associated by the name of the representation's base class.*

Whenever a metawidget selects a representation, its base representation name is used to query the representation repository for a list of possible representations. The list is then examined to select and create a representation for the metawidget. But how does an instance of a representation get created using this list? What kind of object is stored in the representation repository?

What is needed is an object from which other object instances can be created. The list of representations obtained from the repository contains objects from which instances of the particular representation types can be created. Object oriented languages may be classified on how they relate classes and objects. Single hierarchy languages, like Smalltalk and Self, treat classes and objects as essentially the same. This allows some objects to be able to create other object instances. These special "factory" objects are called exemplars (Coplien, 1992).

C++, which treats classes and objects as distinct entities, is a so—called dual hierarchy language. Classes are similar to abstract data types. Classes are fixed at compile time, and objects do not exist until run time. The compile time intensive nature of C++ furthermore dictates that objects may be created only through declarations using built in types or classes, or through the use of the operator new. C++ does not have an exemplar facility analogous to that available in single hierarchy languages.

Classes in C++, however, can be imbued with exemplar facilities so that object instances may be created from another object via a virtual function call. The virtual function simply returns a pointer to the newly created object, usually of type equal to the base class from which the exemplar class has been derived. In PolyMestra, each distinct representation has an exemplar object from which representations of that type may be created. By making exemplar objects static, only one exemplar is created and placed in the repository for each type of representation.

In the example representation hierarchy for BaseTextStrings shown in Fig. 5, there will be an exemplar for each of the four types of possible string representations. An exemplar object for VTextButton creates an instance of VTextButton, but returns a pointer of type BaseTextString to it. In C++, this is possible because a base class pointer may point to any instance of a class that is derived from the base class.

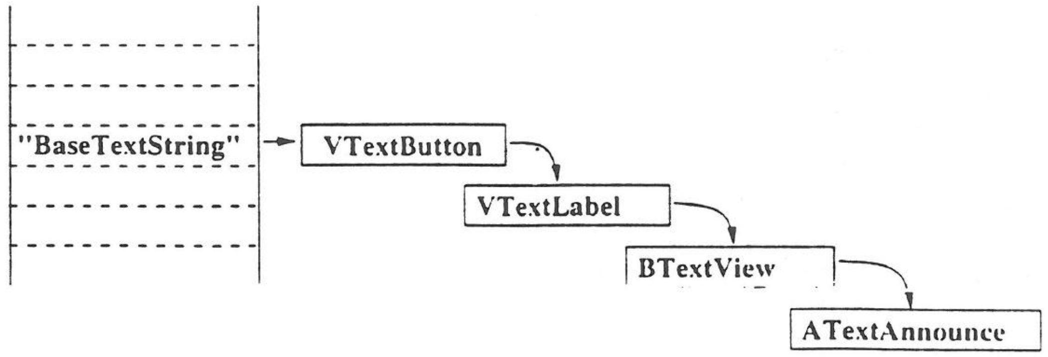

Figure 6: *The repository contains lists of exemplars for each collection of representations, indexed by the base representation name.*

The representation repository contains a list of exemplars indexed by the base representation class name. Fig. 6 depicts how the text string representations are stored in the repository. When a designer creates a new representation, a static exemplar instance of the representation is created which automatically registers itself with the repository. Because the representation exemplars are static, we are guaranteed that the repository entry will be created before any metawidget can use the representation in an application.

3.3. Modalities and Cognitive Resources

A key goal was to exploit the insights into the human cognitive—perceptual system previously discussed in Section to develop mechanisms for measuring the user's cognitive load as information is presented by the multimodal environment. These mechanisms could then be employed by the PolyMestra run time tools to present information in such a way that the user can assimilate it all. Using multiple resource theory and others (Wickens, 1987; Wickens, 1992; Boles and Law, 1992) as a guide, we devised the following assumptions about cognitive resources:

(1) Each modality has a fixed, predetermined set of cognitive resources.

(2) Cognitive resources may be unique to a particular modality or shared across more than one modality.

(3) Representations consume cognitive resources from each modality in which they reside.

Our original design involved the use of separate classes for each modality of interest in the system. Each modality class would then be responsible for measuring its own resources. The drawback of this design was that it required representations to inherit not only from BaseRepresentation but also from a specific modality class. We were concerned that the resulting class hierarchy was too cumbersome, and would cause developers to not want to use our framework.

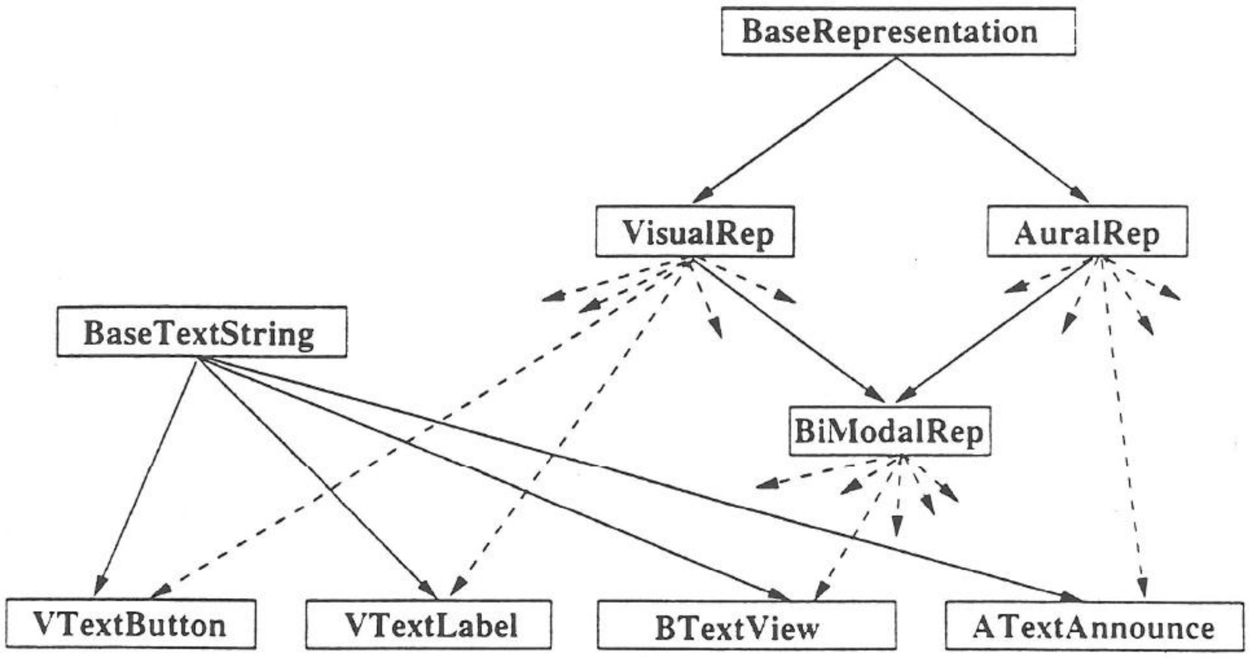

Figure 7: *The hierarchy that results when modality characteristics are absorbed by representation classes. All lines denote inheritance. The dashed lines are used to show inheritance from the representation classes to actual representations in each modality.*

Instead of defining separate classes for the various modalities, we therefore chose to build cognitive resource functionality into the representations. The generic BaseRepresentation class contains virtual functions for calculating cognitive resources, but since no modality has yet been chosen, these methods do nothing. Modality specific representation classes are then derived from BaseRepresentation. Fig. 7 shows the class hierarchy that results when classes VisualRep, AuralRep and BiModalRep are added to the hierarchy.

3.4. Inter—application Communication

In order for the core software tools to monitor the load imposed by client applications, some form of inter—application communication is needed. PolyMestra uses Microsoft Window`s Dynamic Data Exchange Management Library (DDEML) for this purpose.

The Presentation Manager serves as the locus of communication for all metawidgets. When a metawidget is first created, it registers with the Presentation Manager. The Presentation Manager saves the registration information to maintain an internal database of active metawidgets. Later, when a representation must be selected for presentation to the user, the appropriate metawidget queries the Presentation Manager to determine if addition of this representation to the information display would overload the user. If the Presentation Manager decides it would not, the currently selected representation may be presented, otherwise, the metawidget must resubmit its query after selecting an alternative representation from among those available to it.

The Presentation Manager does not maintain information regarding the user's current cognitive load. The Resource Monitor is responsible for collecting the relevant data and computing this value when required, taking into account cross—modality and within—modality effects. By encapsulating knowledge relating to the calculation of cognitive load inside the Resource Monitor, separate from the Presentation Manager, it becomes possible to easily modify/update this part of the system when necessary without otherwise impacting system operation.

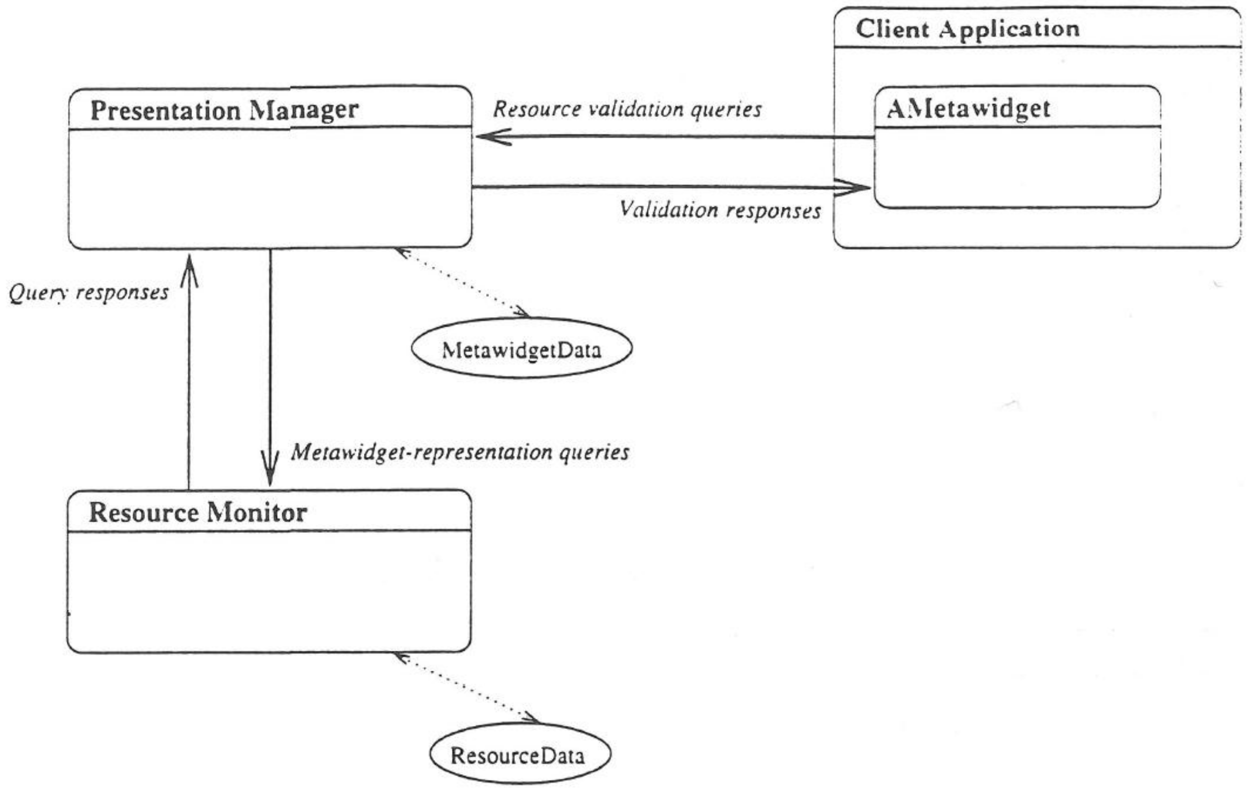

Figure 8: *Inter—application communication in PolyMestra. Solid lines indicate communication paths and dotted lines indicate internal data paths.*

Each query that the Presentation Manager receives from a metawidget includes the anticipated incremental cognitive resource consumption for the representation whose display is proposed. These values are passed by the Presentation Manager (using the DDEML) to the Resource Monitor, which uses them to determine what the new cognitive load on the user would be if this representation were in fact displayed. Fig. 8 shows the PolyMestra communication protocol in detail.

4. CURRENT STATUS

Implementation of PolyMestra is well under way in C++ using the Standard Template Library (STL) extensions (Stepanov and Lee, 1995; Nelson, 1995). Our visual representations are being developed using Borland's Object Windows Library (better known as OWL) and we are developing a toolkit of aural widgets on top of Microsoft Window's Multimedia Control Interface. The system currently consists of over 15,000 lines of code (excluding comment lines). We expect to have the first working prototype, including utilities for web browsing and e—mail, available for distribution to alpha test sites by late summer. We eagerly seek applications programmers and users who are willing to experiment with our system and provide feedback about their experiences, which will then enable us to refine our design and implementation. Interested parties are invited to contact the second author for more information.

Acknowledgement: This research was supported, in part, by the National Science Foundation under contracts IRI--9213823 and CCR--9527151.

5. REFERENCES

M. M. Blattner, E. P. Glinert, J. A. Jorge and G. R. Ormsby (1992). Metawidgets: Towards a theory of multimodal interface design. In Proceedings COMPSAC'92, Chicago, September 22--25, pp. 115--120. IEEE Computer Society Press, Los Alamitos, CA.

D. B. Boles and M. B. Law (1992). Orthogonal lateralized processes have orthogonal-attentional resources, In The Annual Meeting of the Psychonomic Society. St. Louis, November 13.

J. O. Coplien (1992). Advanced C++ Programming Styles and Idioms. Addison Wesley Publishing Co., Reading, MA.

M. L. Fracker and C. D. Wickens (1989). Resources, confusions, and compatibility in dual—axis tracking: Displays, controls, and dynamics. Journal of Experimental Psychology: Human Perception and Performance, 15(1), pp. 80--96.

E. P. Glinert and M. M. Blattner (1993). Programming the multimedia interface. In Proceedings First ACM International Conference on Multimedia (MULTIMEDIA'93), Anaheim, August 2--6, pp. 189--197. ACM Press, New York City, NY.

D. Kahneman (1973). Attention and Effort. Prentice Hall, Englewood Cliffs, NJ.

W. B. Knowles (1963). Operator loading tasks. Human Factors, 5, pp. 151--161.

N. Moray (1967). Where is attention limited? A survey and a model. Acta Psychologica, 27, pp. 84--92.

R. B. Murray (1993). C++ Strategies and Tactics. Addison Wesley Publishing Co., Reading, MA.

D. Navon and D. Gopher (1979). On the economy of the human processing system. Psychological Review, 86, pp. 214--255.

M. Nelson (1995). C++ Programmers Guide to the Standard Template Library. IDG Books, Foster City, CA.

J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson (1991). Object Oriented Modeling and Design, Prentice Hall, Englewood Cliffs, NJ.

A. Stepanov and M. Lee (1995). The standard template library (ANSI/ISO document). Available electronically from ftp://butler.hpl.hp.com/stl/stl.zip.

C. D. Wickens (1984). Processing resources in attention. In Varieties of Attention (D. R. Davies, Editor), pp. 63--258. Academic Press, New York City, NY.

C. D. Wickens (1987). Attention. In Human Factors Psychology (P. A. Hancock, Editor), pp. 29--80. North Holland, Amsterdam.

C. D. Wickens (1992). Engineering Psychology and Human Performance. Harper Collins, New York City, NY.