Cross-modal Interaction using XWeb

Dan R. Olsen Jr., Sean Jefferies, Travis Nielsen, William Moyes, Paul Fredrickson

Computer Science Department, Brigham Young University, Provo, UT

{olsen, jefferie, nielsent, wmoyes, pfred}@cs.byu.edu

ABSTRACT

The XWeb project addresses the problem of interacting with services by means of a variety of interactive platforms. Interactive clients are provided on a variety of hardware/software platforms that can access and XWeb service. Creators of services need not be concerned with interactive techniques or devices. The cross platform problems of a network model of interaction, adaptation to screen size and supporting both speech and visual interfaces in the same model are addressed.

Keywords

Cross-modal interaction, network interaction, screen layout, speech interfaces.

INTRODUCTION

Exponential growth in computing capacity and size of the Internet, as well as exponential decline in costs for a given fixed capacity impose interactive challenges that traditional user interface architectures cannot meet. The exponential growth in capacity produces ever larger repositories of information with ever more diverse content. The exponential decline in cost pushes computation into ever-smaller packages into increasingly many parts of human activity. The prospect of cheap connectivity to virtually everything holds enormous potential for interactive systems.

The service problem

Information and control services will be increasingly diverse ranging from petabyte-sized databases down to microcontrollers in small appliances. This diversity poses two sets of problems. To justify the very large database, it must service a large and diverse user population. Training and supporting software installations for a diverse population is an almost insurmountable task. This problem is exacerbated by the variety of interactive platforms, such as lap tops, cell phones, personal digital assistants, interactive walls and rooms with new platforms yet to be invented.

The converse of the large server problem is the microserver problem. With the advent of significant computation and memory in packages costing less than $50 to $100 the cost

gap between control and information services and their user interfaces becomes quite large. A primary reason for the difficulty in programming VCRs is that the hardware cost of a truly effective user interface would almost double the cost and size of the VCR.

In addition to VCRs there are a variety of other uses for such microservers. Vending machines must be filled by people. If these machines can provide people with remote access to their current state, many visits can be avoided and costs reduced. Instruments in remote locations can be serviced. Large appliances and other devices can be instrumented for diagnostic information that can be remotely accessed. If, however, the user interface must be directly coupled to each such device, both the hardware and training costs will jump significantly.

Pervasive computation cannot succeed if every computational device must be accompanied by its own interactive hardware and software. Diverse populations cannot be served by an architecture that requires a uniquely programmed solution for every combination of service and interactive platform. What is needed is a universal interactive service protocol to which any compliant interactive client can connect and access any service. Decoupling the user interface from the service is the only possible approach.

The user problem

Users are faced with a similar problem to that encountered by the service providers. There is a huge diversity in the set of possible control and information services that a particular user may find useful. Installing a unique piece of software for each desired service is not a good solution. It is already true that for most personal computer users, software occupies more space than any other class of storage. If every new service requires the installation of a new piece of software, the accessibility of that service is sharply diminished. Users have already discovered that new software installation is the most likely reason for failure of their computers.

Installing unique user interfaces for each service also poses a serious learning barrier. The average user cannot master more than a few different user interfaces. If every new information service, appliance, entertainment device, or piece of automation poses a new interface to be learned, the result is an unusable morass. The usability problem no longer lies in the design of a particular interface, but rather in the collective mass of such interfaces. A huge barrier to

pervasive computing is that users are drowning in the diversity.

In addition to the diversity of services that a user may want to access there is also the diversity of situations in which they may want to interact. Consider our example of vending machine servicing. The dispatcher/manager in the home office will have a different set of interactive devices available than the service person in a truck traveling to the next stop. These will be different still from the set of appropriate user interface devices for actually servicing the machine. Each situation imposes its own physical requirements on the set of devices and set of interactive techniques that are possible and effective. It is not reasonable to require each vending machine to support all such possibilities. Neither is it reasonable to force the users in each situation to work with inappropriate user interfaces. A more general solution is required.

An additional diversity of interactive behavior arises when considering people with various disabilities. It is not cost effective for each service to build a unique user interface for each unique set of disabilities. Nor is it likely that such users will be able to learn and/or adapt to a large diversity of interfaces. Again the unique user interface for each service approach fails.

What an individual user needs is a small number of hardware platforms, each with its own user interface that has been tuned to the capabilities of that platform and the class of users/situations for which it is intended. Each such hardware/software platform must be capable of interacting effectively with any service. An architecture where interactive clients are independent from control and information services, can scale to the level of diversity brought about by Moore's Law and the Internet.

Learning from the WWW

The World Wide Web meets most of the issues described above. HTTP/HTML provide a uniform protocol that separates services from their user interfaces. Users need only one piece of browser software for each interactive platform that they use. HTML browsers have been developed for personal digital assistants, cell phones and other devices in addition to the original desktop versions. This one piece of software accesses a huge variety of services.

Corporate information providers are rapidly converting to web-based user interfaces because they are freed from the installation and training problems imposed by the application-based UI architecture. Devices ranging from digital cameras to network caching appliances provide an HTTP server as their only user interface. The actual user interface to such devices is found in any standard web browser. This approach is an obvious success story.

However, HTML and HTTP are interactively impoverished. The level of user interface that they provide is equivalent to the old IBM 3270 terminals that were in use two decades ago. The architecture of the WWW is right but the interactivity is insufficient. New initiatives such as WebDAV and WAP address document versioning and cell phone interaction as incremental modifications of HTML/HTTP. Neither takes on the full range of interactive modalities.

XWeb

This paper describes the interactive solutions to these problems that have been developed as part of the XWeb project. XWeb is based on the architecture of the WWW with additional mechanisms for interaction and collaboration. XWeb servers provide responses to the XTP network interaction protocol. Server implementations are completely independent of the interactive platforms that users might use. So far we have demonstrated servers which provide interactive access to directory trees of XML documents, relational databases and home automation devices.

Users work in the XWeb world via interactive clients that are tuned to the interactive capacities of particular interactive platforms. So far we have developed clients for the desk top, for speech only situations, and for pen-based wall displays. We are currently working on clients using only minimal button sets as well as multidisplay interactive rooms. Our strategy is to choose client situations that pose the greatest possible diversity of interaction.

The key to the scalability of the XWeb user interface architecture is that services and clients can independently choose a variety of implementation strategies that are tuned to their particular needs. The vending machine status server can be extremely small in terms of memory and software complexity. Huge information repositories can be very complex with replication architectures and specialized search functions. All such implementations are independent of each other and of the particular mode of interaction that any user might chose, provided that they conform to the XWeb Transport Protocol (XTP). Similar advantages accrue to users in that they can pick a particular client that is suited to their needs, learn only that client, and yet interact with all possible XWeb services.

Successful development of the XWeb interactive architecture depends on solutions to the following problems.

- Defining an interactive protocol for communication between service and client
- Defining user interfaces in a form that is independent of a particular mode of interaction
- Adapting interaction to available input devices ranging from minimal button sets to interactive rooms

- Adapting to variation in screen size and aspect ratio
- Defining interfaces that can be based on speech and audio as well as visual display.

XTP INTERACTION PROTOCOL

The World Wide Web defines the HTTP protocol as the basis for communication between clients and servers. At the heart of this protocol is the GET message that will retrieve data from any HTTP server. Our XTP (XWeb Transport Protocol) uses that same GET method. However, XWeb goes beyond the publish-mostly architecture of the WWW. XWeb is intended to provide full interactivity.

In our earlier work on user interface software architectures we have found that the vast majority of all interactive behavior is to find, browse, and modify information[9,10]. A server is viewed as a tree of data to be searched, retrieved, and modified. We chose trees because of their ability to encode a very general set of data structures. We also chose trees because of the simplicity of generating names for objects in the tree. In this regard we have retained the URL ideas from the WWW. We have avoided graphs or directed acyclic graphs because of naming problems. However, our interfaces do support links from object to object, which essentially provides users with any graph representation of information.

We represent all XWeb data as XML. XML is a quite general mechanism for representing virtually any tree-structured data. Note that XML is the transport representation for data, not necessarily the internal representation. We have implemented servers based on file directory structures, XML files, relational databases and Novell directory services. In the WWW, HTML is used as a facade for a variety of information storage formats. We have done the same, except that we have discarded the notion that all information is a formatted document.

Retrieving information

To XTP the server's data looks like a tree of objects each of which has a tag type, named attributes (with string values) and zero or more child objects. Child objects can be referenced by ID or by index. A URL for XWeb has the form

$$xweb://domainname:port/pathname$$

much like an HTTP URL. As in HTTP the port is optional. Path names consist of the indices or identifiers of the child objects starting from the root of the site. Negative indices count backwards from the last child. So the last child of some object is at index -1. Attributes are referenced by a special $@attname$ syntax. Thus using, path names it is possible to identify any object on the site.

When referencing an object is it important to differentiate between just that object or the entire tree rooted at that object. This is a particular problem when an entire directory of objects is referenced. If the whole object is desired, an entire site may be downloaded, which is rarely appropriate. On the other hand, it is frequently useful to retrieve entire subtrees at once so as to interact with the tree as a whole. We differentiate between references to an entire tree and a simple skeleton description of that tree. If the URL ends in "/" then only the skeleton is requested. For implementation reasons, many types of XWeb services refuse to return anything more than the skeleton (summaries of child objects) rather than the entire subtree. Subobjects are then retrieved individually as needed. Example URLs might be

- xweb://my.site/games/chess/3/@winner
  the winner attribute of the fourth (starts at 0) chess game
- xweb://automate.home /lights/livingroom/
  a skeleton description of the set of lights in the living room
- xweb://automate.home/lights/familyroom/-1
  all the current information about the last set of lights in the family room

Modifying information

All interaction with an XWeb site is defined in terms of changes to that site's data tree. A CHANGE message consists of a URL for the site and subtree to which the change is to be applied. A CHANGE consists of a sequence of editing operations. Each operation contains one or more references to objects or attributes to be modified. Such references are relative to the root of the CHANGE subtree. The editing operations are:

- set an attribute's value
- delete an attribute
- change some child object to a new value
- insert a new child object
- move a subtree to another location
- copy a subtree to another location

This set of editing operations defines all of the ways in which a client may interact with an XWeb service. By composing these operations, any manipulation of a tree can be expressed. Note that these manipulations are far more extensive than those supported by HTML or WML.

By focusing the client/server interaction on data manipulation rather than event propagation, we achieve a level of independence between client and service that is not possible otherwise. The X windows system provided for distributed user interfaces by propagating input events and drawing commands across the network. This, however, bound the interface to the originally concieved style and set of input devices. Adapting such interfaces to a different modality such as speech and audio [8] is quite cumbersome. Remote interaction in terms of events is also

quite sensitive to network latency. It is unacceptable if each input event must make a complete round trip to the application before the user gets any feedback. Using replicated data, the user can proceed with most interactions without confirmation from the service. This provides for rapid local feedback and interaction across the network.

PLATFORM INDEPENDENT INTERFACES

A key problem is for the creator of an information service to define interfaces that will be effective on a variety of interactive platforms. Our approach to this is to define XViews that are general XML descriptions of an interaction, which do not specify the interactive techniques to be used. The primary purposes of an XView are to

- select the data elements that are to be included in the user interface,
- map those data elements to specific interactors and ranges of possible values,
- provide resources for interactors to use in implementing their user interfaces.

When an XWeb client initiates an interaction, it uses a two part URL.

DataURL::ViewURL

The DataURL is a reference to some subtree of some XWeb site. The ViewURL is a reference to an XView specification that defines the interaction with the data. Any missing portions of the ViewURL are supplied from the fully specified DataURL. This simple referencing mechanism allows for multiple views of data items as well as the application of a view specification to numerous data items. We plan to further abbreviate this specification to allow servers to inform clients about appropriate default ViewURLs for a particular data item. This would allow clients to specify only the DataURL.

Interactors

The heart of the XView specification is the interactors or widgets. Note that our use of interactor should not be confused with the terminology introduced by Brad Myers[7]. The purpose of an interactor is to specify the possible types of values that a data item can have. It is also the purpose of an interactor to transform an internally encoded data value into an external representation that is appropriate for users. Interactor specifications do not dictate input events, interactive techniques, or layouts. An interactor description encodes what the desired information is, leaving specific mechanisms of how the users perceive and specify new values up to the various client implementations. This is key to our goal of platform independence.

Interactors fall into two categories, atomic and aggregate. The set of interactors that we have provided is similar to the widget set that one might expect in a user interface tool kit except that we have specified them at a higher semantic level.

Atomic interactors

Most tool kits define their basic set of atomic widgets around specific, generally useful interactive techniques. The criteria for choices are to provide flexible composition of widgets to meet most needs. Our design is focused on frequently used semantic concepts. The implementation is not radically different, but it makes significant differences in terms of separating the user interface from the service and preserving platform independence. Our currently implemented set of atomic interactors are

- Numbers with multilevel units and unit conversions
- Dates
- Times
- Enumeration of finite choices
- Text (single or multiline)
- Links to other data and/or views

A normal user interface tool kit would provide check boxes, radio buttons, combo boxes, labels, buttons and scroll bars. However, each of these implies an interactive technique, which limits the choices for interactive client implementations. The semantic goal of choosing from a finite set is the same whether radio buttons, combo boxes, menus, function keys or speech are used. We define the enumeration interactor and leave the specific implementation to the client. The choice of interactive technique depends very much on the interactive devices available and the available presentation resources. We do not provide buttons because they do not model any change to a piece of information. Many clients use buttons but as a means rather than a semantic goal.

We specifically identified dates and times as special interactors because these semantic values occur with a high frequency among applications. By specifying how a date or time is encoded in the data, we leave the client to develop an effective set of interactive techniques to manipulate that information. By stepping up to a higher semantic level, we provide specific client implementations with more interactive flexibility.

Our number specification is quite sophisticated in that it provides an abstraction for hierarchical units as well as conversions among systems of units. For example, lengths can be specified in feet and inches as well as meters and centimeters. Only linear unit conversions are supported. Units of almost any type can be supported. By supporting units we have semantically packaged a concept that would require several widgets in most implementations. By retaining the semantic whole, we increase the flexibility of interactive choices for various client implementations.

We do not provide labels in our interactor set. Each interactor carries with it its own descriptive information. Our approach is that each interactor can have a variety of information resources associated with it. The client

implementation can then choose which of them to download and how they should be presented. Minimally each interactor must have a name. Beyond simple names, icons (of various sizes), abbreviations, synonyms, and recorded sounds can be added. Client implementations choose from among these resources when presenting the interactor to the user. Many use label widgets for the names. Speech clients, for example do not. The choice is in the implementation. Explanation and help texts can also be added to any interactor for presentation to the user. We do not use any resource inheritance mechanism such as in X Windows or Cascading Style Sheets [13]. Specifying an interactor description and then applying it to many data objects fills the need without the complexities of inheritance.

By only requiring a name with all other resources being optional, a minimal client capability is established. We see future interactive clients with very limited capacities. If, however, such clients can manage a network connection and express the basic interactor values they can provide full interactive functionality. We see this as important to the development of small wearable client devices that are unobtrusive but fully capable. An example might be an infrared client with a minimal button set that is built into a wristwatch. In our worldview of interaction, the ability to scale down is as important as scaling up.

Aggregate interactors

At present we have implemented two aggregation interactors, which can assemble smaller pieces into a larger whole. These are groups and lists. A group is simply a finite collection of interactors, which together have some logical meaning. A group has descriptive resources and a set of child interactors. There is very limited layout information associated with a group other than that the group should be logically presented together. The geometric layout issues are addressed later in this paper. A hierarchy of groups forms the fundamental structuring mechanism for an XView. This structure is key to managing interactions where there is limited presentation capacity such as small screens or audio-only interfaces.

A group also carries with it a summary descriptor. This is a pattern, which is used to assemble a brief textual description of the data contents of the group. In audio and small screen situations, this brief summary is invaluable in conserving presentation resources. The assembly of summaries is discussed later in the section on tree mapping.

Ordered multi-column lists are our primary mechanism for handling arbitrarily large sets of data. Our implementation provides for client-side sorting on any of the columns. Most of our list views present summary information in a few columns with a link interactor that leads to a full view of the selected object.

Future interactors

There are various ways in which XWeb's current interactor set falls short. There are no interactors that can manipulate images or sound. These media data types are important and must be included. It is very important that an interactor be provided for selection from a finite, but arbitrarily large set of choices. An example would be selecting a name from a phone book. There are many interactive techniques that have been developed which rely on such enumerated sets to provide focus to fuzzy or ambiguous inputs. Among them are speech recognition, text entry via phone pads[5], handwriting recognition, and pen-based typing [6]. XWeb needs such an interactor.

The group and the list are an insufficient set of aggregate interactors. The list cannot handle very, very large data sets. An interactor that can handle sparse traversals of large data is required. We also feel that an interactor that can manage spatial relationships found in diagrams, schematics, and maps is required.

In this first cut at the XWeb architecture we have focused on the underlying substrate and the relationships between clients on various platforms and various servers. Having laid down the fundamental architecture and explored the issues of cross-platform interaction, there is now greater justification for more general high level interactors. Such abstractions are not justified in the traditional architecture, where the user interface is tightly bound to the information, because the effort to use the abstraction can become as complicated as programming the solution by hand. However, when only a very few interactors can be programmed into each and every client these higher level abstractions become much more important.

Tree remapping

Defining an XView is fundamentally a process of mapping fragments of the data tree onto the tree of interactors that constitutes the user interface. Essentially the process must integrate content (the data) with interactive presentation. This is very similar to the goals of the eXtensible Stylesheet Language (XSL) [14]. XSL is focused on transforming data into a suitable presentation. This causes several problems. The first is that XSL is far more general purpose in terms of mapping arbitrary XML trees into other arbitrary trees. The second problem is that XSL mappings are many to one. In an interactive setting we need one to one mappings. Not only must the data be presented, but also when the user changes the presentation, that change must be transformed back into a change on the original data. In the case of a many to one mapping the reverse transformation is undetermined. In addition, it is hard for designers to predict the reverse behavior of a series of pattern matching rules. Our final constraint is that the mapping algorithm must be small so as not to impose code size problems on small interactive platforms.

A second approach to defining the relationship between interactors and their data is to provide a programmatic connection such as JavaScript. Our problem with this solution is that designers must explicitly implement both directions of the transformation. This is further complicated by the fact that interaction is based on change record generation. Our last problem with explicit client-side programming of the interface is that it leads to platform specific interfaces. A declarative relationship between interactor and data provides more latitude for implementation by interactive clients.
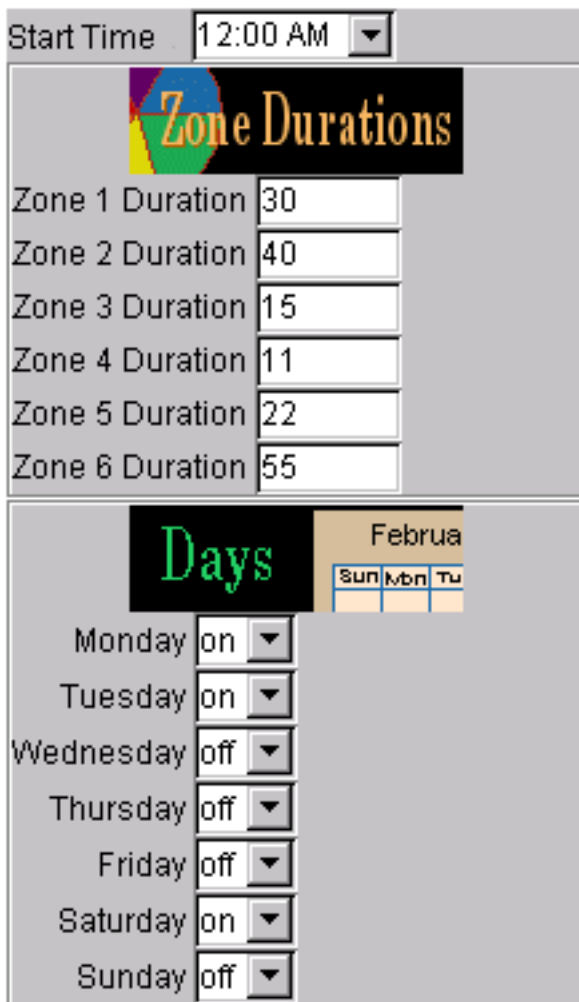

Figure 1 - Sample Interactors

Example

Assume that the following XML fragment represents the data for a home automation system that controls sprinklers.

```
<sprinklers startTime="00:00">
    <days ID="days" sun="0" mon="1" tue="1"
          wed="0" thu="0" fri="0" sat="1"/>
    <zone time="30"/>          <zone time="40"/>
    <zone time="15"/>          <zone time="11"/>
    <zone time="22"/>          <zone time="55"/>
</sprinklers>
```

What is desired is a user interface like that in figure 1.

Specifying the view

The view starts with a root interactor that is almost always a group interactor. The groups form a hierarchy of interactors. As each interactor is instantiated there is a binding created between that interaction description and a corresponding data object. Every interactor has a loc attribute, which specifies a path from its parent's data object to its own data object. The loc may be empty, which indicates that the interactor references the same object as its parent, or it may be arbitrarily long, reaching deep into the data object hierarchy to reference a specific object.

The skeleton view for this interaction is shown in the following XML

```
<xview rootID="timer">
    <group ID="timer" loc="" >
        <time loc="" . . .> <name text="Start Time"/>
            . . . . . .
        </time>
        <group loc=""> <name text="Zone durations"/>
            <number loc="0/@time". . .> . . . . </number>
            <number loc="1/@time". . .> . . . . </number>
            . . . . . .
        </group>
        <group loc="days"> <name text="Days"/>
            <enum loc="@mon">
                <name text="Monday"/> . . . .
            </enum>
            <enum loc="@tue">
                <name text="Tuesday"/> . . . .
            </enum>
            . . . . . .
        </group>
    </group>
</xview>
```

The root view is the group with the ID of timer. This is mapped to the root object of our data, which is the object that encloses the <sprinklers> data object. This group contains a time and two other groups. Since the start time is found as an attribute of the <sprinkler> object, its loc is empty. Since all of the information for the Days group is found in the object whose ID="days", its loc is "days". Note that the second <number> in the Zone durations group has a loc of "1/@time", which indexes the <sprinkler>'s children (zero relative) and selects the time attribute as the container for this number. This recursive selection via path names is very easy to implement and easily supports the required two-way mapping between user and data.

Data extraction composition patterns

Some of the interactors have composite data values. For example a <time> consists of hours, minutes, seconds and

AM/PM. A <date> has the year, month, day of the month, and day of the week. This information can appear in several formats and can be encoded in the data in various ways. There are also multiple ways in which the information can be presented to a user. In addition, there are the group summaries that are composed of data fragments from data being edited. We handle all of these mappings by means of a simple text matching and composition algorithm.

All atomic data values in the data tree are text strings. A data value such as a date may be represented in several such strings (separate for day, month and year), or encoded into one string. The information presented to the user for manipulation is generally in one string. An extraction/composition pattern is a string with variable name references embedded in it. For example the pattern to extract the start time from the <sprinklers> is

"$(hour_24):$(minutes)"

All characters up to the colon are placed in the variable *hour_24* and the ones after the colon are extracted into the variable *minutes*. On many interactors the variables are just named placeholders. In date and time there is a fixed set each with a special meaning. The userFormat attribute on the <time> specifies how the time is to be presented to the user. In this case the format is:

"$(hour_12):$(minutes) $(AMPM)"

In transforming information from the data to the user interface the process is to match the pattern against the data, filling variables as the extraction match proceedings. Then any variable calculations are done. In the case of <time>, the *hour_12* (twelve hour clock) and *AMPM* values must be computed from the *hour_24* variable. Once all variables are computed then the user data is composed using the user pattern. When the user edits the time, the reverse process is done. In the case of <time>, <date> and <group> summaries, the data can be in many places. The descriptor specifies all the places which can be sources for the data. Before variables are computed, all part strings are matched, to extract variable values from all of the places where they might be stored. The algorithm is simple, but provides for quite general encodings and two way mappings of data.

Obviously the tree and data transformation capabilities of an XView are not sufficient for the most general transformations. We delegate such transformations to the server implementation. The data tree that a server presents is only a faÇade over the real service. The server implementation can program any encoding or transformation it desires. The purpose of the XView transformations is to support multiple views of the data for various purposes. Real computation is reserved for the server implementations.

ADAPTING TO AVAILABLE INPUT DEVICES

The primary mechanisms for adaptation of interfaces to multiple interactive devices are 1) the general semantic nature of the interactors, 2) multiple client implementations, and 3) a variety of descriptive resources attached to interactor descriptions. The interactors are purposely designed to capture the nature of a specific data type rather than an interactive technique for manipulating that type. In our <sprinklers>, on and off for a particular day are represented as 1 and 0. Any interactive technique that chooses between 1 and 0 is acceptable. It can be two radio buttons, a check box, a checked menu item, two different buttons on a pager or two different spoken words. It is up to the client implementation to choose a technique based on its own interactive capabilities. The interactor only encodes the range of possible values.

Our vision for XWeb is that every interactive platform will have its own XWeb client implementation. Each client will have its own interactive techniques. We believe (as has occurred with the WWW) that once a user learns the behavior of a particular client, that knowledge will transfer to all XWeb applications accessed through that client. We think this will have particular value for speech interfaces that do not have the advantage of having most information displayed simultaneously and no displayed prompt information. We believe that developing skill with the client will greatly facilitate learning the spoken interface for new applications. This learning transfer should also be valuable for minimal button clients. Investing learning time in a minimal client that is highly mobile but somewhat awkward is more beneficial if that learning can be transferred to many applications.

To push our understanding of cross-modal interaction we have focused on three specific client implementations. These are a standard desktop client, a pen-based wall display client, and a speech-only client. We have implemented the full interactor set on each of these clients. For the desktop client we made the obvious choices for interactor implementations drawn from the Swing toolkit. For the pen-based display we are faced with the lack of a keyboard. All interactors that could use text input were provided with pen input using Graffitti. In addition, since many of our interactors manipulate ordered values, we provided flicking gestures to increment or decrement digits, days of the week, months etc. Such flicking of any value allows for quite rapid modification of these values using easily learned gestures. Once the flicking technique is learned it works on any application. The speech client was somewhat more problematic. Its particular issues are discussed in a later section. By leaving the interactive techniques up to the client, effective interactions can be

developed for each platform that all manipulate the same semantic information using the same CHANGE messages.

To augment the capabilities of the various clients we allow interactors to be decorated with a variety of informational resources. Every interactor can have explanation text that can form the basis for user help. Icons of various sizes can be provided for screen-based clients. Abbreviations of names can be provided for text-only clients with limited displays. Synonyms are provided for speech clients as well as recorded sounds for use by audio display clients. These additional resources allow XView designers to enrich an interface without constraining it. The clients are free to choose which resources are of value and download only the ones that would be helpful for that particular client.

SCREEN LAYOUT

A problem that arises when designing interfaces that must adapt to a variety of interactive platforms is screen layout. There are three major issues involved in this question. They are 1) determining minimal visible size, 2) dealing with the widely varying size of screens and 3) handling the variation in aspect ratios. In one respect, we can ignore the problem and let each client implementation deal with it. The layout is primarily driven by the user interface structure imposed by the group and list trees. Each client can display that structure in any form that they choose.

Providing a general layout solution has three main advantages. The first is that the general solution can be reused to simplify development of new interactive clients. The second is that a general solution has advantages for large display clients that may want to shrink displays down to very small windows. The small window problem is isomorphic to the small screen problem. The final advantage is that XView designers working on large displays can adjust window sizes to understand how their designs will work on more limited screens.

The first problem is to determine the minimal visible sized object. This cannot really be represented as a fixed number of pixels because pixel size varies so radically among displays. A fixed number of pixels is also not adequate when dealing with displays that will be used from varying distances. For example, wall displays can be used by someone with a pen standing at the display or by people sitting in chairs some distance from the display. The minimum visible size may vary dynamically in such cases. Our solution to this is that minimum visible size is determined by the smallest readable font size. Font metrics on most graphics systems will convert a font size into a pixel height. The pixel height can then be used internally. Using the minimum font size most widgets immediately can compute their own desired sizes. For icons we use the minimum pixel height as the minimum size for icons. This immediately eliminates icons that are too small. Use of larger icons can be based on available screen space.

There have been a variety of layout schemes proposed for dealing with variable sized windows. One of the earliest was by Cardelli [2]. The Higgins system [3] proposed alignment enhancements. Various other constraint systems have also been used to adapt screen layouts[12]. All of these mechanisms provide adaptability but only within a limited range of possible layouts. Any geometry-only solution cannot bridge the gap between a 2000x2000 wall sized display and the small size of a PDA. The most serious difficulty is change of aspect ratio. The geometric relationships among objects in a portrait style display are very different from a landscape display. Simply transposing X and Y is also not an adequate solution because the aspect ratios of text cannot be equivalently transposed.

The intrinsic size layout pioneered in T_EX, brought to user interfaces by InterViews[4], and popularized by Java is the most promising for our purposes. The intrinsic size approach allows each interactor to report on its desires and then provides various mechanisms for compositing interactors to allocate space based on the desired sizes. The normal intrinsic size algorithms suffer from 1) inability to deal with aspect ratio variation, and 2) continuous rather than discrete allocation of space. An additional important issue, which the T_EX algorithm will handle, is the use of interactive focus to manage screen space.
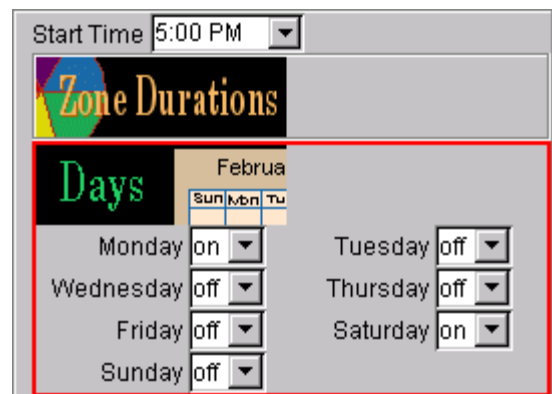

Figure 2 - Reformatted Layout

Accommodating differing aspect ratios

The intrinsic size layout algorithm involves two passes. The first requests from each widget, its minimum, desired and maximum size in both X and Y. Groups then use this information to report their own size needs. Once the total window size and the needs of child widgets are known, space is allocated for each child widget. This proceeds recursively down the widget tree. The problem arises when one considers a group such as the Days in Figure 1. The needs of this group are area needs, rather than specific vertical and horizontal values. This group could be half as tall if there were room for two columns, as in Figure 2. This group could also be completely horizontal.

Our approach, which works well in practice, handles the computation of width and height independently. Each group uses the following algorithm for layout of its children.

```
Ask each child their min, preferred and max width
Based on maximum preferred width, calculate
        the number of columns of children widgets
Assign each child to a column
Adjust column widths based on the preferred widths of
        the widgets actually assigned to that column
Ask each child widget how tall it would like to be
        (min, preferred, max) given the width of it's
        assigned column
Assign row heights based on widget height requests
```

The key modification here is that a widget is not asked for its height until a tentative width has been assigned. The Days group for example would use the tentative width to decide how many columns it should use and then report a height based on the number of columns it has room for.

Exploiting focus to allocate screen space

The primary means for controlling screen utilization is the maintenance of a current interactive focus (which widget is currently selected). Any group that is selected or contains the selected widget will report as its minimum width and height, sufficient space to show the minimums for all of its children. Any group that is not selected will report as its minimum sufficient space for only its name or icon. The interactive effect is that if space is limited, groups open and close themselves when selected or not, as is shown in Figure 2. Regardless of interactive focus, groups report preferred sizes that will show all of their children.

It is frequently the case that there is enough room for more than one group to be open at a time. The normal intrinsic size approach is to allocate minimum space and the give each widget a proportion of its preferred space. This is a continuous sort of "share the wealth" strategy. Our strategy is a discrete "all or nothing first". For each group we maintain a list of most recently selected children. If more than minimum space is available we give widgets all of their preferred space working down from the most recently selected until all of the space is gone. This produces the interactive effect of having widgets open or closed based on how recently they were selected. If there is still space remaining, we assign what there is to the next widget on the recently selected list. This widget does what it can with the additional space.

Another problem that can occur is that the selected child does not receive enough screen space to satisfy its minimum requirements. If this occurs, we increment that widget's preferred width by the size of one minimum column and then reallocate space using the algorithm described earlier. This has the effect of allowing the selected child to use more horizontal space to accommodate its needs. We continue giving more horizontal space to the selected object until it has all of the available width. At that point we are forced to scroll.

If there is a lot of screen space, these strategies produce very stable and generally pleasing layouts. If there is minimal screen space, the layout moves and changes as the focus changes, thus adapting to user needs.

AUDIO/VISUAL CROSSOVER

Our final challenge is to deal with both audio-based and visually oriented interaction in the same architecture. This is somewhat complicated by the lack of "speech widgets," in earlier work. Most of the software architecture work associated with speech is focused on natural language. The assumption of much of the speech research is that the primary advantage of speech is that it supports natural language. We believe that the key advantages to speech is its ability to scale down to very small physical form factors and to support hands-free or eyes free interaction. Natural language systems have not yet shown the breadth of generality needed for an XWeb client and so far are too big to fit onto small computing platforms.

For our first XWeb speech client, we chose a structured, widget-oriented approach rather than natural language. We wanted to reflect the same interface structure that users will experience on their desktops and PDAs. Learnability of speech interfaces is a critical problem and any transfer from other modalities should help.

Our strategy, as with other XWeb clients, is to have the speech interface language fixed, with each XView filling in the blanks. Thus once a user learns the client's interactive behavior, that skill should transfer from application to application. This fixed language approach to speech interfaces is supported by findings in HyperSpeech [1] and VoiceNotes [11]. The XWeb speech client has many of the same goals as Mercator[8]. The key advantage is that instead of reverse engineering the interface structure from artifacts in the X Windows protocol, the interface structure is explicitly represented in the XView descriptors.

In our clients we see three basic dialog tasks that the spoken interface must satisfy. They are traversal of the interactor tree, editing of values, and obtaining help. We have defined standard interactive syntax to dealing with each of these.

Tree traversal

As described earlier, each XView is a tree of standard interactors. Each interactor has a set of resources that can be used when presenting itself to the user. In the speech client the key pieces of interactor information are its name, any synonyms, and its data value. These can be augmented by audio clips to overcome the vagaries of today's text to speech systems. Incremental traversal of the tree is handled

by the four commands Enter, Exit, Next and Previous. Enter and Exit are used to enter groups or lists. Next and Previous are used to traverse the children of a group or list. With these four commands a user can go anywhere in the interactor tree. Enter when applied to a Link interactor will follow that Link to a new view and Exit will return from that Link.

However, one of the advantages of speech is referencing things by name rather than laboriously scrolling. Naming any sibling of the current focus object will immediately move to that object. XView designers can provide synonyms for names that are recognized as equivalent to the name. However, the name is always what is spoken back by the client.

Our early experience has shown that the notion of trees and traversal of trees is the hardest concept for users to grasp when working with the speech client. To resolve this we are looking at a more expanded scope for names than just sibling interactors.

Value editing

Editing in atomic interactors is always done with the Set command followed by a spoken expression of the new value. Each interactor type has its own syntax for expressing values. That syntax is standard for all instances of that type. Any syntactic form spoken by an interactor can also be spoken by the user to specify a new value. This facilitates learning by listening.

Lists are somewhat special in that they have additional commands for cut, copy and paste, as well as commands for skipping forward or backward in larger steps than single items. Again the commands are standard for all list instances.

Help

Key to learning a new XWeb speech application are the standard help commands. The Describe command will speak the name and current value of whatever interactor has the focus. In the case of groups, the summary value is spoken. The Explain command will speak a textual description of the purpose and syntax of the current interactor. This information is found in the XView resources for the interactor. "What can I say" always speaks a list of the available commands and what they are for. "Where am I" will speak the sequence of interactor names from the root of the interactor tree down to the current focus interactor. With these four commands the user can learn the interface. In fact, the user need only know "What can I say" to be able to learn everything else.

SUMMARY

The XWeb client/server model can adapt to a wide variety of interactive platforms. Our prototype has specifically shown its effectiveness on desktops, speech only and wall-sized platforms. This architecture forms the basis for an interactive substrate on which a wide variety of applications can be delivered without getting bogged down in the combinatoric explosion of all possible platforms with all possible services.

REFERENCES

[1] Arons, B. "Hyperspeech: Navigating in Speech-only Hypermedia," Hypertext '91, (1991).

[2] Cardelli, L. "Building User Interfaces by Direct Manipulation," ACM SIGGRAPH Symposium on User Interface Software, (1988)

[3] Hudson, S., and King, R. "Semantic Feedback in the Higgens UIMS," IEEE Transactions ofn Software Engineering, 14(8), (August 1988)

[4] Linton, M. A., Vlissides, J. M, and Calder, P. R., "Composing User Interfaces with InterViews," IEEE Computer, 22(2), (February 1989).

[5] Marx, M., and Schmandt, C., "Putting People First: Specifying Proper Names in Speech Interfaces," ACM Symposium on User Interface Software and Technology (UIST 94), (1994).

[6] Masui, T., "An Efficient Text Input Method for Pen-based Computers", Human Factors in Computing Systems (CHI 98), (1998)

[7] Myers, B. A., "A New Model for Handling Input," ACM Transactions on Information Systems, 8, 3 (July 1990)

[8] Mynatt, E. D., and Edwards, W. K., "Mapping GUIs to auditory interfaces," Proceedings of the Fifth Annual ACM symposium on User Interface Software and Technology , (1992).

[9] Olsen, D. R., "A Browse / Edit Model for User Interface Management" Graphics Interface '88. (June 1988)

[10] Olsen, D. R., "A Programming Language Basis for User Interface Management," Human Factors in Computing Systems (CHI '89), (1989).

[11] Stifelman, L. J., Arons, B., Schmandt, C., and Hulteen, E. A., "VoiceNotes: A Speech Interfaces for a Hand-Held Voice Notetaker," Human Factors in Computing Systems (INTERCHI 93), (1993).

[12] Vander Zanden, B., and Myers, B., "Demonstrational and Constraint-based Techniques for Pictorially Specifying Application Objects and Behaviors," ACM Transactions on Computer-Human Interaction 2(4), (December 1995).

[13] World Wide Web Consortium, "Cascading Style Sheets," http://www.w3.org/Style/CSS/, (2000)

[14] World Wide Web Consortium, "Extensible Stylesheet Language (XSL)," http://www.w3.org/Style/XSL, (2000)