Computer Networks and ISDN Systems 29 (1997) 1531 - 1542

COMPUTER NETWORKS ISDN SYSTEMS

Seamless integration of interactive forms into the Web

Andreas Girgensohn[a,*], Alison Lee [b,1]

[a]FX Palo Alto Laboratory, 3400 Hillview Avenue, Palo Alto, CA 94304, USA
[b]NYNEX Science and Technology, 500 Westchester Avenue, White Plains, NY 10604, USA

Abstract

The phenomenal interest and growth of the World Wide Web as an application server has pushed the Web model to its limits. Specifically, the Web offers limited interactivity and versatility as a platform for networked applications. One major challenge for the HCI community is to determine how to improve the human-computer interface for Web-based applications. This paper focuses on a significant Web deficiency – supporting truly interactive and dynamic form-based input. We propose a well-worked form interaction abstraction that alleviates this Web deficiency. We describe how the abstraction is seamlessly integrated into the Web framework by leveraging on the virtues of the Web and fitting within the interaction and usage model of the Web. © 1997 Published by Elsevier Science B.V.

Keywords: Active fields; Automatic layout; Built-in checklist; Formulae; Field procedures; Form-based input; Java; Rapid prototyping; Toolkit; Visibility; Web applications

1. Introduction

Increasingly, the World Wide Web (WWW) is being used to develop and deliver interactive and customizable applications. Many of the initial applications are in the areas of database searches and on-line ordering. Other types of more recent applications include those for collaboration [2,10] and for corporate intranet applications that are replacing proprietary interfaces to applications. Many of these applications push the Web model to its limits, including the user interaction components. Compared to other user interfaces, the Web only offers limited interactivity and versatility. However, its popularity, its improved means of communication, and its emergence as a standard for networked application on the internet and intranet indicate that the Web is here to stay. This means that well-established, user-friendly approaches should be transferred to or integrated with the Web. It is a challenge for HCI researchers to determine how to make Web-based applications more user friendly, interactive, and easy to use [5].

Many applications on the Web require form-based input. Simple forms provide input for search engines, more complex ones are used for on-line ordering and surveys, and one can envision a form for a Web-based tax return. However, forms provided as part of the Hypertext Markup Language (HTML) are not suited for complex data entry with many form fields and dependencies between different fields. Complex forms contain many fields so that a large scrollable area is the result. In many applications, only a small fraction of the fields really has to be filled for any given sit-

* Corresponding author. E-mail: andreasg@pal.xerox.com
[1] E-mail:alee@nynexst.com

PI1 S0169-7552(97)00063-9

uation. Consequently, there are many fields that just distract the user and use up valuable screen real estate. Also, form input should be validated immediately instead of delaying it until the form is submitted.

Scripting languages such as JavaScript [7] used in concert with HTML forms address this problem partially but they do not go far enough. Our form interaction abstraction [3], which drew on work described in the literature (e.g., [6,13,14]) and evolved in a user-centered fashion, addresses similar issues for form-based systems for non-Web use. A Web implementation of the abstraction provides more interactivity and user orientation than current Web-based form systems. This paper describes the seamless integration of our form interaction abstraction into the World Wide Web model. It leverages on the virtues of the WWW and fits well within its interaction and usage model.

This paper starts with a scenario illustrating the shortcomings of HTML forms and how our form interaction abstraction can improve that situation. After a discussion of different approaches to make Web-based forms more interactive, we describe how our approach is seamlessly integrated into the Web framework. The paper concludes by discussing future improvements.

2. A scenario

To motivate the need for more interactive and flexible forms, we present an application for entering meeting notes. This application is part of our design intent system [l], an intranet application, and has been used to enter the notes of weekly meetings into a database so that HTML output can be generated to display the notes. Users can specify time and topic of a meeting, select the attendees, enter a general description and notes of a meeting, and specify action items. The same form can be used later to make changes to the notes (see Fig. 1).

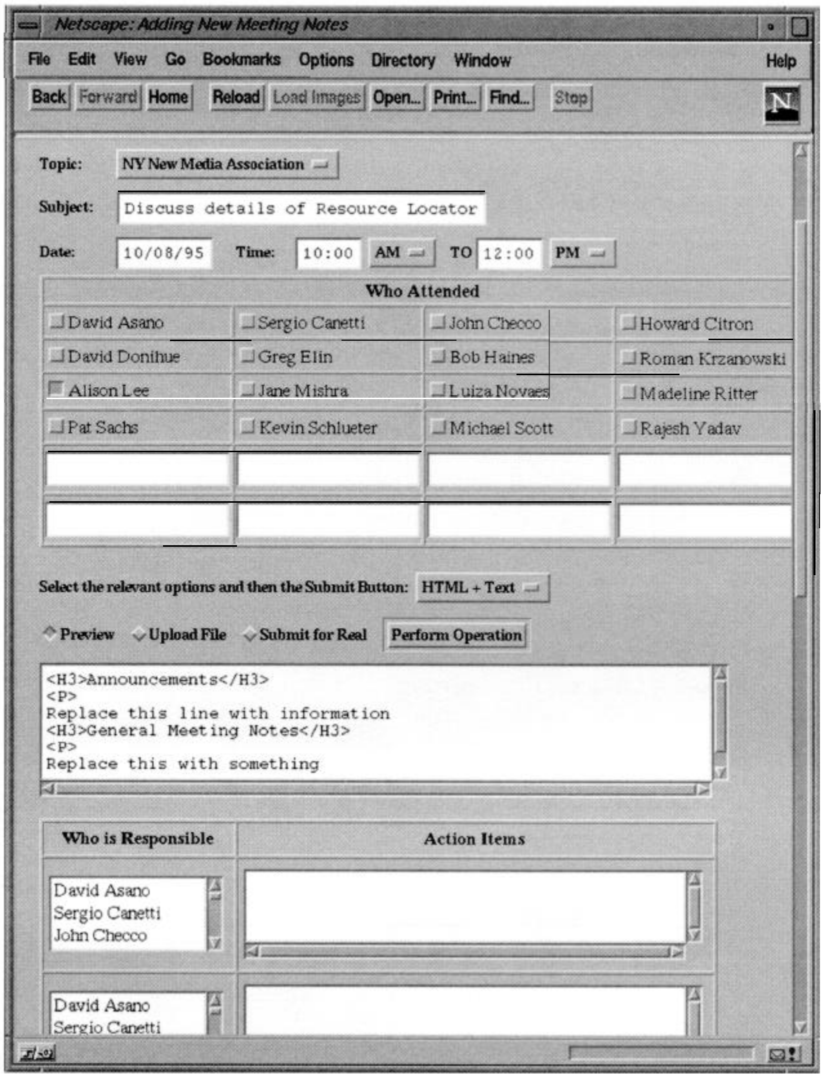
This form, built in HTML, has several disadvantages. First, much screen real estate is used to include additional fields so that the names of guest attendees and action items can be entered. Room for several of these fields has to be provided because there is no mechanism to hide all but one of the unfilled fields. Second, this form contains many fields and navigating to different parts of the form is both difficult and unwieldy. Similarly, it is difficult to obtain a holistic view of the form to determine, for instance, which fields are unfilled or contain errors. Third, all regular attendees are listed as candidates “responsible for action items” rather than just the people that actually attended the meeting. Finally, it is not possible to include the names of guest attendees of the meeting in the list of candidates “responsible for action items” when their names are entered in the attendees list.

The Dynamic Forms version of the same form (see Fig. 2) addresses all of the problems discussed above. First, one field for entering guest attendees is shown and another field appears as soon as text is entered in the first field (see Fig. 3). Similarly, fields for additional action items only become visible after an entry is used. Second, related fields are grouped into sections that can be expanded and collapsed, similar to an outline mode of a word processor (see Fig. 3). This enables the user to bring sections of interest handily within view — by collapsing earlier sections in the form — without having to scroll blindly to the relevant fields. Third, fields which require input are marked in red and change color as soon as they contain the correct input (see field “Subject” in Fig. 2). Consequently, a built-in colored checklist is revealed when the user collapses the contents of subsections to guide the user in the form-filling task and to make sure that no required step is overlooked. This feature in combination with displaying only necessary fields help make the overall structure of the form-filling task easier to see and navigate. Finally, changes to the attendee fields cause the list of eligible people “responsible for action items” to change so that only attendees, including guests, appear in the list.

All these adaptations happen on the client side without accessing the server so that delays are minimized. Additions and deletions of fields, changes in the visibility of fields and changes to the form widget items do not cause a complete refresh of the page and disorientation is avoided.

3. Existing Web forms

There are many criteria that a good form toolkit for the Web should fulfill. Most importantly, forms


Netscape: Adding New Meeting Notes
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Load Images Open... Print... Find... Stop
Perform Operation

Topic: NY New Media Association

Subject: Discuss details of Resource Locator

Date: 10/08/95 Time: 10:00 AM TO 12:00 PM

Who Attended			
David Asano	Sergio Canetti	John Checco	Howard Citron
David Donihue	Greg Elin	Bob Haines	Roman Krzanowski
✓ Alison Lee	Jane Mishra	Luiza Novaes	Madeline Ritter
Pat Sachs	Kevin Schlueter	Michael Scott	Rajesh Yadav

Select the relevant options and then the Submit Button: HTML + Text

● Preview ◇ Upload File ◇ Submit for Real

```
<H3>Announcements</H3>
<P>
Replace this line with information
<H3>General Meeting Notes</H3>
<P>
Replace this with something
```

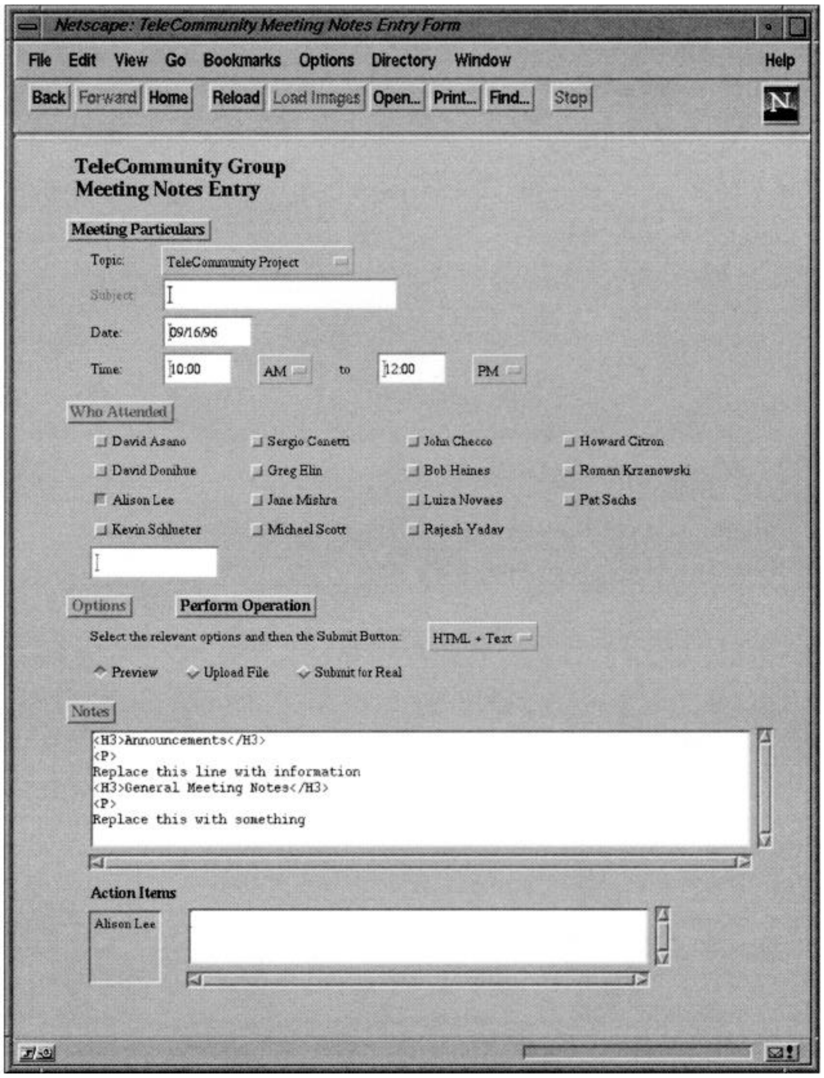
Who is Responsible	Action Items
David Asano Sergio Canetti John Checco	
David Asano Sergio Canetti	

Fig. 1. HTML forms design.

created for the Web need to be easy to use and powerful enough to manage complex input tasks. HCI principles can be helpful in evaluating form systems along those lines.

One major advantage of the Web is that it is relatively easy to create Web pages. This is desirable for forms as well; thorough knowledge of a programming language should not be a requirement for creating forms. In addition, support for rapid prototyping is important so that it is easy for form developers to involve their users. Another development question is whether form elements and their properties can be represented in a descriptive language or whether one has to write a script to generate form elements. This distinction is similar to the one made between object-oriented and procedural languages. The former is better for situations with natural objects as it is the case with form elements. Finally, it is beneficial if a form can be filled out on any platform using any browser. Proprietary approaches that work only for a small fraction of the Web users should be avoided.

Table 1 compares the existing Web form approaches and Dynamic Forms.


Netscape: TeleCommunity Meeting Notes Entry Form
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Load Images Open... Print... Find... Stop
Meeting Particulars
Who Attended
Options
Perform Operation
Notes

TeleCommunity Group
Meeting Notes Entry

Topic: TeleCommunity Project

Subject:

Date: 09/16/96

Time: 10:00 AM to 12:00 PM

☐ David Asano	☐ Sergio Canetti	☐ John Checco	☐ Howard Citron
☐ David Donihue	☐ Greg Elin	☐ Bob Heines	☐ Roman Krzanowski
☑ Alison Lee	☐ Jane Mishra	☐ Luiza Novaes	☐ Pat Sachs
☐ Kevin Schlueter	☐ Michael Scott	☐ Rajesh Yadav	

Select the relevant options and then the Submit Button: HTML + Text

◆ Preview ◇ Upload File ◇ Submit for Real

```
<H3>Announcements</H3>
<P>
Replace this line with information
<H3>General Meeting Notes</H3>
<P>
Replace this with something
```

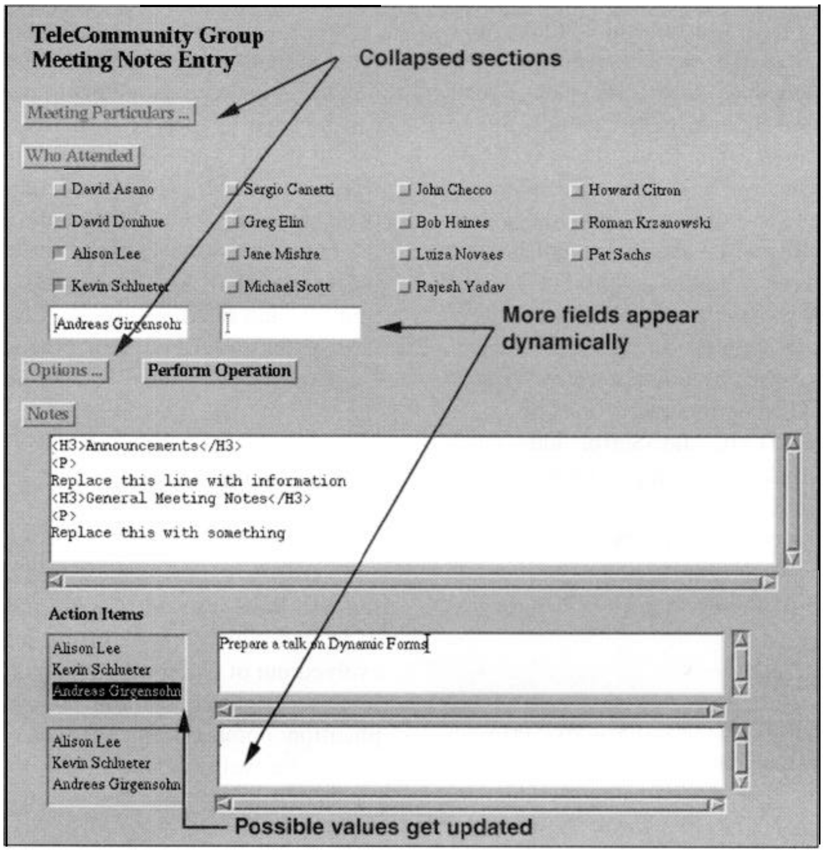
Action Items

Alison Lee

Fig. 2 Web dynamic forms design.

Table 1
Comparison of form features

Criteria	HTML forms	JavaScript	Tcl/Tk	Java widgets	Dynamic forms
Available in all browsers	yes	no	no	soon	soon
Dynamic visibility of widgets	generate new form (refresh)	generate new form (refresh)	yes	yes	simple expression
Dynamic layout of widgets	generate new form (refresh)	generate new form (refresh)	yes	yes	automatic
Input verification	no	programmable	programmable	programmable	simple expression
Computation of field values	no	programmable	programmable	programmable	formula
Integration with HTML	yes	yes	some	some	much
Descriptive representation	yes	no	no	no	yes
Automatic repetition of fields	no	no	no	no	yes


TeleCommunity Group
Meeting Notes Entry

Meeting Particulars ...

Who Attended

David Asano	Sergio Canetti	John Checco	Howard Citron
David Donihue	Greg Elin	Bob Haines	Roman Krzanowski
Alison Lee	Jane Mishra	Luiza Novaes	Pat Sachs
Kevin Schlueter	Michael Scott	Rajesh Yadav	

Andreas Girgensohn

Options ... Perform Operation

Notes

```
<H3>Announcements</H3>
<P>
Replace this line with information
<H3>General Meeting Notes</H3>
<P>
Replace this with something
```

Action Items

Alison Lee
Kevin Schlueter
Andreas Girgensohn

Prepare a talk on Dynamic Forms

Alison Lee
Kevin Schlueter
Andreas Girgensohn

Fig. 3. Dynamic changes

3.1. HTML forms

HTML forms provide user interface elements such as text entry fields, check boxes, radio buttons, list boxes, and pull-down menus. The language for creating form elements is very descriptive, i.e., field properties such as the name and the initial value can be specified. HTML forms are standard for all browsers so that full portability is guaranteed. The layout is mostly automatic, just like for the rest of HTML, and has limitations, e.g., widgets are placed in the normal text flow. The layout can be improved by using tables. The model of filling out a form is completely batch-oriented. Users do not get any feedback until they submit the form to the server.

Creating a form does not require any more computer literacy than creating an HTML document. However, processing the form values on the server requires programming. This is common to all Web-based form approaches. Using standard scripts to perform common functions would alleviate the programming effort.

Adding new HTML tags to HTML forms would make HTML overly complex and would decrease the number of browsers that could display such forms. Languages that can define new form widgets and behaviors themselves, such as Java or Tcl/Tk, have an advantage here. As long as a browser supports the language at all, more advanced forms can be processed.

3.2. Forms with JavaScript

Netscape Navigator provides JavaScript as a scripting language for making the browser more interactive. Currently, other browsers do not support JavaScript. This should change soon, however, because JavaScript is being reviewed as a standard. By adding JavaScript, HTML forms can become more interactive. Event handlers with JavaScript statements can be attached to form elements. Events that can be processed include field changes, mouse clicks, and mouse movements. Using such event handlers, field input can be checked on the client side and field values can be computed from the values of other fields. However, JavaScript cannot hide or show fields nor can it change the position of widgets on the screen. It is possible in JavaScript to regenerate the whole form with the appropriate changes on the fly but that causes a complete refresh of the browser window. This option requires code to be written to generate HTML. JavaScript adds much complexity to the task of creating a form. Dependencies between fields cannot be represented in a fashion similar to a spreadsheet, but instead have to be scripted in a programming language that can access properties of field objects.

3.3. Tcl/Tk form widgets

Thistlewaite and Ball [12] describe a Web browser that can present forms with an attached Tcl script [9]. The initial form is described using the standard HTML form statements but the form elements are Tk widgets that can be manipulated by the script. In response to user input, widgets can be removed or inserted. Unfortunately, there is no higher level representation for the additional widgets. Instead, Tcl/Tk code for widget creation has to be written. The additions to the initial HTML form can only be used in the SurfIt browser developed by Thistlewaite and Ball that is written in Tcl/Tk. Standard browsers such as Netscape Navigator and Microsoft Internet Explorer cannot process such forms.

3.4. Java and plug-in form widgets

Netscape Navigator has a plug-in mechanism to support additional functionality. These plug-ins have to be developed for every software and hardware platform to be supported. A plug-in manages a rectangular area of an HTML document similar to an inline image. Plug-ins such as Macromedia Shockwave and Tcl/Tk provide widgets that can make forms much more interactive. Java also has a widget toolkit so that Java applets can present forms as well.

While Java and plug-ins do not provide complete form systems, the widgets can be used as building blocks. A dedicated area inside an HTML document can be used to present Java widgets or widgets native to the plug-in. One issue is integration with the rest of the HTML document (e.g., for selecting fonts or for loading new documents). Another issue is the communication with the server after a form has been completed. Also, there is usually no representation for field relationships, so that such relationships have to be described in a procedural fashion with languages such as Java, Tcl, or Lingo.

4. Web dynamic forms

Problems with form toolkits available for the Web and positive experiences with Dynamic Forms [3] for non-Web user interfaces made a Web version of Dynamic Forms look promising. Dynamic Forms evolved out of experiences with designing and building user-centered systems in several different application domains. The goal for the Web version was to be well-integrated with the techniques used on the Web without losing its ease of development and use, and its support for rapid prototyping. Java was chosen as the implementation language because of its widespread availability in Web browsers across many platforms, its integration into HTML documents, its support for interactivity, and its ability to communicate with the Web server. Dynamic Forms designers are shielded from much of the Java complexities and they can take advantage of Dynamic Forms' sophisticated presentation algorithms. In most situations the use of Dynamic Forms requires very little Java programming, but there is a gradual increase in complexity for situations in which more Java features are desirable.

4.1. Dynamic forms features

The Web version of Dynamic Forms has all the features of the earlier version described in more detail in [3]. It supports developers in creating a single, dynamic, scrollable form using a form description language. The virtual form is structured into sections and subsections so that effective organization and navigation of the information are possible. The object-oriented, textual, and interpretive nature of the language allow developers to incorporate user suggestions concerning changes to a dynamic form quickly and with minimal tools. Automatic field layout, dynamic visibility of fields, nested forms, built-in checklists, active fields, selection fields, arrays of fields, field procedures and formulae are features of Dynamic Forms.

These features provide enhancements over conventional form-based interfaces. The automatic layout of fields plays a key role for many of the features. It can handle changes to the visibility of fields, changes to the font size, and even changes to the height of text fields as a result of the insertion or deletion of line feeds. The visibility of fields is dynamically recomputed so that only fields relevant to the current task are shown and the space taken by the unnecessary fields can be reused. Related to this is the notion of nested forms in which section and subsection headings can be used to collapse and expand sets of fields similar to an outline mode of a word processor. Built-in checklists guide the users through tasks and make sure that nothing is overlooked. Active fields cause the recomputation of other fields and they can initiate side-effects such as the display of dialog windows. They are created using either spreadsheet-like formulae or field procedures that trigger functions on field entry, exit, or change. Selection fields provide an easy way to present users with choices and they take care of all button placements. Arrays of fields are useful for describing repetitions of fields.

Fig. 3 shows how these features are used in the meeting notes application. Some of the sections are collapsed (indicated by "...") to make more room for the parts of the form that still need to be filled out. Fields appear dynamically to accept guests and action items as soon as the other fields in the same category have been filled. Changes to the attendees immediately cause the list of people eligible to be responsible for action items to be recomputed. The color of section headings changes to green as soon as all the required fields in that section have been filled.

4.2. Dynamic forms representation

All fields in a form store string values that are presented and changed through interface elements such as buttons and entry fields. Relationships between fields can be expressed as formulae and field procedures. Viewing Dynamic Forms in this manner makes them similar to structured spreadsheets. Information that is entered in one field may be used in calculations and displayed in other fields. Formulae and expressions controlling aspects such as visibility use a C-like syntax, and can call functions defined by the application developer. Field procedures are triggered by events such as field changes, and provide an interface to the rest of the application. All fields are named and can be referred to by their names within expressions.

The following example defines a text entry field that has an input mask to restrict input to a proper date format. To further restrict input beyond the capabilities of an input mask, a boolean expression checks for a valid date. The field belongs to the section with the heading field "meeting-info". It contains the same value as the field "date_2" (via cyclic formulae). A field procedure is called whenever the meeting-date field loses the input focus. Layout instructions keep the field on the same line as the previous field and align it with other fields in the same group of fields.

```
{EntryField meeting_date,
  Prompt "Date",
  InputMask "99/99/99",
  Width 8,
  Flags {NONEWLINE, GROUPALIGN},
  DependsOn meeting_info,
  Requirement isDate (this),
  Formula date_2,
  Fprocs {computeHeading}};
```

This method of describing and changing fields is made easier by an automatic layout mechanism. When a field is inserted in the middle of existing fields, the mechanism automatically changes the layout of the fields. This is similar to systems such as PICASSO [11] which takes a description of the user interface and places widgets automatically. The Dynamic Forms layout mechanism is not only used for creating a window but also for changing layouts that become necessary after user input causes changes in the visibility of fields. Font size and font type changes are also handled easily as the position of every field is recomputed using the new sizes of the fields.

5. Web dynamic forms implementation

Web Dynamic Forms is implemented in Java [4]. It shields form application developers from much of the complexities of Java by providing a simple field description language that is compiled into Java code. The Dynamic Forms layout is much more suited for forms and adapted to users' tasks than the layout managers provided with the Java environment.

5.1. Implementation details

Web Dynamic Forms is implemented as a set of Java classes, one class for each type of field (see Table 2). This object-oriented design supports the introduction of new field types by adding new classes to the hierarchy. A protocol of Java methods describes each field class so that new classes can inherit some of the methods and redefine others. Each form in a browser window or in a pop-up window is represented as an object as well (class *DF_Package).* Global properties of a form such as the default font for each field can be manipulated at run time via this object.

The form description language is a textual language that describes fields including their types, properties, and relationships. Properties can have simple string values such as the prompt of a field or they can contain expressions with a C-like syntax, for example, to describe the visibility condition of a field. Fields are named and their values can be accessed in expressions just like variables.

A precompiler converts the form description language to Java code. This approach was chosen to reduce the overhead during the loading of the applet containing a form. The precompiler parses the input and precomputes relationships between fields. Specifically, the sources for formulae and the fields used in visibility conditions are computed by the precompiler so that it is clear which other fields are affected by the change of a field. This part needs to be efficient because changes are checked after every key stroke. Field procedures used in the form are included as functions in a Java class defined for the field procedures for the form. These functions must be registered and compiled with the form application.

The precompiler generates a Java class that represents a form. Expressions such as visibility conditions are converted to Java code. The example below shows how a form represented by the class *MtgForm* can be used in a Java applet (see also the main window in Fig. 4).

```
public class MtgNotes extends Applet {
  DF_Panel dp = null;
  public void init () {
    GeneratedForm form = new MtgForm ();
    DF_Package pkg = form.createForm ();
    setBackground (Color.red);
    dp = new DF_Panel();
    add (dp);
    dp.setBackground  (Color.white);
    dp.setPackage (pkg);
  }
}
```

Table 2
Dynamic forms classes

Class	Purpose
EntryField	A standard text-entry field
MultilineEntryField	A text-entry field with several lines of text
ReadOnlyText	Field for displaying text that cannot be changed by the user (but the program can change it)
MultilineReadOnlyText	ReadOnlyText that can have multiple lines of text
RadioButtons	At most one value can be selected with radio buttons
CheckBoxes	Any number of values can be selected with check boxes
ListBox	A pull down menu
PushButton	A standard push button
Heading	Push button for opening and closing sections
Prompt	Displays a prompt (label) without an associated value
InvisibleField	Is never visible and contains values that can be used by other fields
Array	Repetition of a set of fields

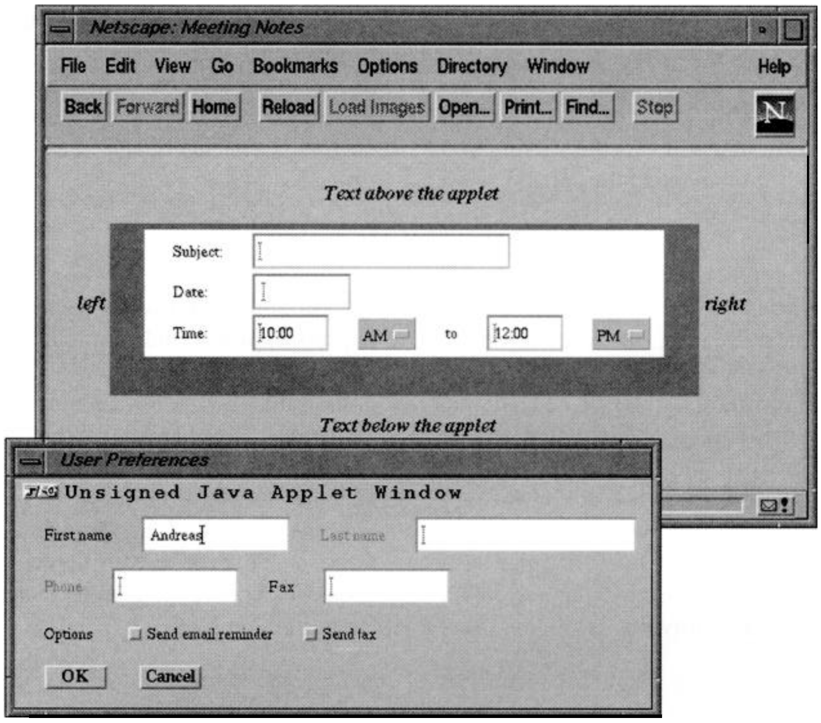

Netscape: Meeting Notes
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Load Images Open... Print... Find... Stop
Text above the applet
left
right
Subject:
Date:
Time: 10:00 AM to 12:00 PM
Text below the applet
User Preferences
Unsigned Java Applet Window
First name Andreas
Last name
Phone
Fax
Options Send email reminder Send fax
OK Cancel

Fig. 4. Windows with forms.

The main window in Fig. 4 shows how a form is placed inside a Java applet and how the Java applet fits inside an HTML document. Applets and panels are containers for other widgets including panels. A form does not have to be within an applet but can be placed inside a panel instead. The panel interacts with the Java layout manager and adjusts its size to the form presented in it. In Fig. 4, the applet has a dark background and the panel containing the form is white.

Fig. 4 also shows that a pop-up dialog window can contain a form. The following code fragment shows how such a window could be opened, for example, from a field procedure:

```
DF_Package pkg = DF_Package.getPackage
                      ("preferences");
pkg.windowDialog ();
```

Such pop-up windows are useful to switch context in a dialog with the user or, for example, to set preferences.

5.2. Alternative implementation platforms

Alternatives to Java were considered as an implementation platform for Dynamic Forms. Because of the limited versatility of JavaScript and the limited availability of browsers supporting Tcl/Tk, a plug-in interpreting a form description document was the only viable alternative. Developing a plug-in to interpret the form description language was considered but rejected for several reasons:

- *Use.* Plug-ins currently work only with the Netscape Navigator and to *some* degree with the Microsoft Internet Explorer, so the user population is limited.
- *Development.* Plug-ins have to be developed for every platform to be supported, and they must be installed on client workstations.
- *Expressiveness.* A plug-in severely restricts the expressive power of Dynamic Forms. Specifically, computations would be restricted to the expression language and could not use the native language (C++ or Java) for more complex operations. This would remove the nice Dynamic

Forms feature of a gradual increase in complexity for different kinds of operations.

Another consideration was the question of when to process the form description language. One option was to let the Java applet do all the parsing and processing. However, implementing the parser in Java would have increased dramatically the size of the Java classes to be loaded and the initial processing time when starting a form applet. The alternative chosen was a preprocessor that converts a field description file to Java source code for creating the necessary form objects. This preprocessor uses the Lex/Yacc-based parser implemented for the C++ version of Dynamic Forms. The preprocessor also precomputes all the field dependencies so that the form applet can be started even more quickly. The drawback of this approach is a slightly higher effort for making changes because the preprocessor and the Java compiler have to be used but the overhead for each change is only a few minutes so that rapid prototyping is still possible.

6. Integration with Web framework

Implementing Web Dynamic Forms in Java allows us to take full advantage of the integration of Java applets in HTML documents. Several other considerations related to the seamless integration were required including the integration with Java's layout mechanisms and the use of Java's network communication and multithreading capabilities. These are described in the following sections.

6.1. Java applets in HTML

Java applets occupy a rectangular area inside an HTML document similar to inline images. This means that a form presented in an applet can be surrounded by HTML text and thus integrated into an HTML document (see Fig. 4). Netscape provides an interface for Java applets to access and manipulate the surrounding HTML document that will hopefully be adopted by other browser vendors. This interface called LiveConnect [8] enables Java code to execute JavaScript statements that in turn can access and manipulate document properties. The most important feature is the ability to load a new document that replaces the current document with the included Java applet. This feature makes it possible for the form to disappear after is has been filled out. LiveConnect also enables JavaScript to access Java methods and variables so that scripts embedded in an HTML document can manipulate a Java applet and a form presented in it.

6.2. Pop-up dialog windows

Java applets delivered over the Web also support pop-up windows, and Dynamic Forms can be used to include a form in such a window in order to have a pop-up dialog window. Dynamic Forms supports the Java layout mechanism so that other Java interface components can be included in the same window and the window can adapt its size to the space requirements of the form. Pop-up dialogs have proven to be useful in non-Web applications and the Web does not support them well. Dynamic Forms makes a contribution here.

6.3. Form submission

Once a form is completed, its values can be sent to a server CGI script in the same format as the one an HTML form uses. Dynamic Forms offers this behavior as a property of a push button so that compatibility with existing form-based applications remains, and as a simple means to communicate with the server. However, as part of a Java applet, forms are not restricted to this. At any time, a field procedure triggered by a field change or other events can connect to the Web server, for example, to submit a subset of the fields or to perform additional verifications. Such communication can either use the standard HTTP protocol or an application-specific, more light-weight protocol that connects to another network socket application on the server host through a different port. Such a connection could be kept open to overcome the statelessness of the Web architecture, and to enable the server to send notifications back to the client.

6.4. Background communication with the server

As already discussed earlier, input verification can be performed on the client side without having to interact with the server. However, if server access is required for verification, the Dynamic Forms architecture can still verify input before the whole form is submitted to the server. Java's support of multiple threads together with its network communication capabilities are useful here. If for example, a field entry needs to be verified against a database on the server, a hidden field can be added to the form to store the result of the verification. A background thread can contact the server and change the hidden field once it gets a reply. The hidden field can be used in the checklist expression of the field to be verified so that the field can be highlighted immediately if the verification fails. This approach has the advantage over the traditional Web approach of allowing users to keep on filling out other fields while the verification is pending. Also, with the Dynamic Forms approach, users do not have to wait until the whole form is submitted before seeing the result of the verification.

7. Conclusions

Web Dynamic Forms brings to the Web arena several important forms features such as support for complex form designs, dynamic field visibility and layout, early validation of field values, computation of field values, and repetition of fields. Web Dynamic Forms makes several contributions along two HCI fronts. First, the forms interaction abstraction is a significant HCI contribution in forms interaction for the World Wide Web. The design and development of the abstraction has evolved as a result of (1) much user input when it was used in several application domains and (2) application of HCI principles and integration of previous lessons learned. Second, the Web version of the abstraction has evolved also as we seamlessly integrated the abstraction into the interaction and usage model of the Web.

Specifically, Web Dynamic Forms

- exist inline with other HTML objects,
- appear as pop-up dialog windows,
- communicate in the background with Web servers,
- submit values to a server in a manner compatible with HTML forms, and
- can be made available to all browsers.

New ground will be broken as a result of the widespread interest in the Web and in delivering Web-based applications for many different purposes. HCI can make substantial contributions to enhancing the interactivity and versatility of the Web. Web Dynamic Forms addresses the Web's limitations with form input, interactivity in forms, and its ability to handle complex forms. It has been achieved through its seamless integration within the framework while taking advantage of the technologies and virtues of the Web.

One can envision an even tighter integration than the current Web technology allows for. This could be achieved by making improvements to existing standards for form input along the lines exemplified by the features of Web Dynamic Forms. Hopefully, standards bodies will make such changes to further empower designers and users.

References

[1] M.E. Atwood, B. Burns, D. Gairing, A. Girgensohn, A. Lee, T. Turner, S. Alteras-Webb, and B. Zimmermann, Facilitating communication in software development, in: *Symposium on Designing Interactive Systems, New* York, ACM, pp. 65-73, 1995

[2] A. Girgensohn, A. Lee, and K. Schlueter, Experiences in developing collaborative applications using the World Wide Web 'Shell", in: *Hypertext'96: 7th ACM Conference on Hypertext, New* York, ACM, pp. 246-255, 1996

[3] A. Girgensohn, B. Zimmermann, A. Lee, B. Bums, and M.E. Atwood, Dynamic forms: an enhanced interaction abstraction based on forms, in: *Proc. INTERACT'95: 5th IFIP Conference on Human-Computer Interaction,* London, Chapman and Hall, pp. 362-367, 1995

[4] J. Gosling, B. Joy, and G. Steele, *The Java™ Language Specification,* Addison Wesley, Reading, MA, 1996

[5] K. Instone, Report on the "HCI and the Web" Workshop at CHI 96, *SIGCHI Bulletin,* 28(4): 42-45, 1996

[6] R. Jeffries and J. Rosenberg, Comparing a form-based and a language-based user interface for instructing a mail program, in: *Human Factors in Computing Systems and Graphics Interface, CHI+GI'87 Conference Proceedings,* Toronto, Canada, ACM, pp. 261-266, 1987

[7] Netscape, JavaScript Authoring Guide, http://www.netscape.com/eng/mozilla/Gold/handbook/javascript/, 1996

[8] Netscape, LiveConnect Communication, http://www.netscape.com/eng/mozilla/3.0/handbook/javascript/moja.html, 1996

[9] J.K. Ousterhout, Tcl and the Tk Toolkit, in: B.W. Kernighan (Ed.), *Professional Computing Series,* Addison Wesley, Reading, MA, 1994

[10] J. Rice, A. Farquhar, P. Piernot, and T. Gruber, Using the

Web instead of a window system, in: *Human Factors in Computing Systems, CHI 96 Conference Proceedings, New* York, ACM, pp. 103-110, 1996

[11] L.A. Rowe, J.A. Konstan, B.C. Smith, S. Seitz, and C. Liu, The PICASSO application framework, in: *UIST'91 Conference Proceedings,* Hilton Head, SC, pp. 95-105, 1991

[12] P. Thistlewaite and S. Ball, Active FORMs, in: *5th International World Wide Web Conference,* http://www5conf.inria.fr/fich-html/papers/P40/Overview.html, 1996

[13] D. Tsichritzis, Form management, *Communications of the ACM,* 25(7): 453-478, 1982

[14] M.M. Zloof, Office-by-Example: a business language that unifies data and word processing and electronic mail, *IBM Systems Journal,* 21: 272-304, 1992

Andreas Girgensohn received his Ph.D. degree in Computer Science from the University of Colorado at Boulder in 1992. His Ph.D. research focussed on supporting end-user modifiability in knowledge-based design environments. From 1992 to 1996 he worked at NYNEX Science & Technology on task-oriented user interface design and development, on support for software developers, and on tools for improving communication and collaboration using the World Wide Web Shell and Lotus Notes. He is currently a research scientist at FX Palo Alto Laboratory, where he works on software agents and web-based applications.

Alison Lee received her Ph.D. degree in Computer Science from the University of Toronto in 1992. Her Ph.D. research is a series of studies investigating the prospects for history-based, user-support tools. Since 1992, she has worked at NYNEX Science & Technology on user interface design and evaluation, on tools and methodologies to improve communication and collaboration amongst distributed work groups, and on tools to support developers of Web applications and services.