Spatial Interpretation of Domain Objects Integrated into a Freeform Electronic Whiteboard

Thomas P. Moran, William van Melle, and **Patrick Chiu***

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304

{moran,vanmelle}@parc.xerox.com, chiu@pal.xerox.com

ABSTRACT

Our goal is to provide tools to support working meetings on an electronic whiteboard, called Tivoli. This paper describes how we have integrated structured *domain objects*, which represent the subject matter of meetings, into the freeform whiteboard environment. Domain objects can be tailored to produce meeting tools that are finely tuned to meeting practices. We describe the language for defining domain objects and show examples of meeting tools that have been built with the language. We show that the system can interpret the spatial relationships of domain objects on the whiteboard to encode the meanings of the spatial arrangements, and we describe the computational mechanisms. We discuss some of the design principles for tailoring gestures for domain objects. Finally, we enumerate the techniques we have used to integrate the structured objects into the freeform whiteboard environment.

Keywords: whiteboard metaphor, pen-based systems, freeform interaction, implicit structure, informal systems, recognition-based systems, list structures, meeting support tools, gestural interfaces, user interface design, tailorability, customization

1. INTRODUCTION

Our goal is to provide computational support for working meetings, in which small groups of people collaborate in real time. By "working meeting" we mean groups working with a large set of knowledge materials (assessing, discussing, organizing the materials, negotiating about the materials, creating new materials, etc.). Group interaction in such situations is freewheeling and fluid.

Groups often use whiteboards in meetings to provide a shared visual surface to capture and manipulate their materials. Research in pen-based computational systems has developed a variety of user interaction techniques, mostly focused on personal use on tablets, based on a variety of metaphors: from writing in notebooks [20, 21] to sketching on napkins [3] or onionskin [4]. There is also some research on pen-based interaction with wall-size displays supporting a whiteboard metaphor for meetings (e.g., [15, 19, 22]). There are several products providing the basic whiteboard scribbling functionality, but effective pen-based meeting tools beyond scribbling is still in the research stage.

We have been working for several years on a program of research to provide computational meeting tools based on a whiteboard metaphor. Our idea, in the spirit of [7, 16, 17], is to allow freeform creation and manipulation of materials on the whiteboard; to provide facilities to easily organize and structure the materials as needed by the group; and to provide perceptual mechanisms in the system to make sense of the organized materials.

Our program is based on the LiveBoard [2], a large, shared, pen-based electronic (rear-projected) display. This provides a whiteboard-size interactive display that allows us to experience and experiment with "group-computer interaction" in a way not possible with smaller displays. We have developed a software application, called *Tivoli* [15], that simulates whiteboard functionality on the LiveBoard (or any other display). Tivoli provides basic pen-based scribbling and editing with pen-based gesturing and wiping techniques. In this paper, we use the term "board" to refer to an interactive electronic whiteboard.

We extended Tivoli to allow "implicit structuring" of the material on the board [11]. Material is created on the board in a freeform manner, which means that anything can go anywhere without constraint. However, the user can indicate by certain gestures that the material is to be regarded as temporarily structured in some manner, e.g., as handwritten text. Tivoli will then apply editing operations in accordance with the conventions of that structure, such as opening and closing space where needed when moving words around.

*Current Address: Patrick Chiu, FX Palo Alto Laboratory, 3400 Hillview Avenue, Palo Alto, CA 94304. At the time of the work reported in this paper, Patrick Chiu was with Liveworks, Inc., A Xerox Company, working at Xerox PARC.

We then extended Tivoli by developing a set of simple and natural techniques for helping users spatially organize material on the board [10], most importantly providing ways to group materials into regions. Tivoli limits structured operations to the regions in which they are invoked.

Our next step was to build on these user interface and structuring features by adding facilities for handling a new kind of object, *domain objects*, in Tivoli. Before this, all of the materials on the Tivoli board have merely been graphical objects, whose interpretation or relationship to other data exists only in the minds of the users. For example, suppose a group's current action items were imported from a workflow system as a prioritized list and read (as simple text) onto the board; then users in a meeting altered, erased, added, and rearranged the action items on the board. The resulting board image would bear no relationship to the action item entities in the workflow system. However, by representing the action items as domain objects in Tivoli, they can be manipulated in a whiteboard style during a meeting and then the results (e.g., rearranged priorities) can be exported back to the workflow system. Thus, domain objects represent the subject matter (i.e., domain of knowledge) of meetings. We are able to tailor specialized domain objects as meeting tools that are tuned to particular contents and processes of meetings.

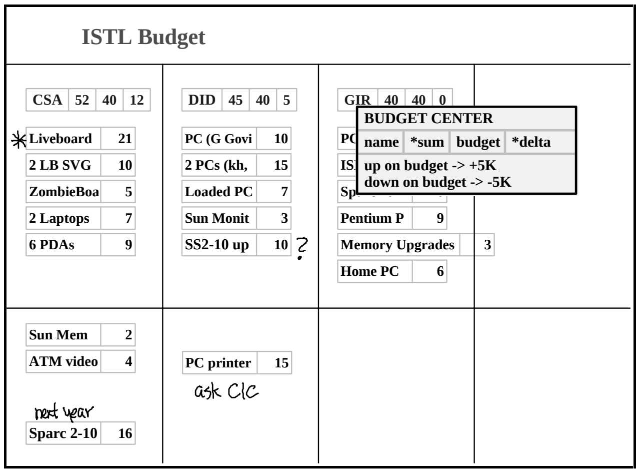
We introduced domain objects in a companion paper [14], where we motivated their utility for building meeting tools, described several applications, and related our experience in using the tools. In this paper, we focus on the user interface issues. We first give a concrete example (Section 2) and the general rationale for domain objects (Section 3). We describe the architecture of Tivoli (Section 4) and its facilities for domain objects (Section 5). We show various meeting tools that were built (Section 6). Finally, we discuss in detail various user interface issues encountered in this research: how Tivoli interprets spatial relations among objects (Section 7), the principles for designing gestures on objects (Section 8), and the techniques for smoothly blending the domain objects into the freeform whiteboard environment (Section 9).

2. AN EXAMPLE OF DOMAIN OBJECTS

Let us consider a scenario of a budget meeting (modeled after an actual meeting). This is perhaps the easiest example to understand, because the use of domain objects is like a "freeform spreadsheet." Figure 1 shows the Tivoli board for the budget meeting.

There are several kinds of graphic objects on the board: scribbles ("ink" strokes) in various places, text (the title at the top), and border lines (defining nine rectangular regions). But most of the graphic objects on the board are domain objects, which appear as boxed items. This is a freeform whiteboard: you can scribble anywhere with the pen in ink mode; you can erase with the pen in erase mode; you can select any graphic objects on the board by drawing

Figure 1. Tivoli Board Showing the Use of Domain Objects in a Budget Meeting.


ISTL Budget

CSA 52 40 12
Liveboard 21
2 LB SVG 10
ZombieBoa 5
2 Laptops 7
6 PDAs 9

DID 45 40 5
PC (G Govi 10
2 PCs (kh, 15
Loaded PC 7
Sun Monit 3
SS2-10 up 10 ?

GIR 40 40 0
PC
IS
Sp
Pentium P 9
Memory Upgrades 3
Home PC 6

BUDGET CENTER
name *sum budget *delta
up on budget -> +5K
down on budget -> -5K

Sun Mem 2
ATM video 4
next year
Sparc 2-10 16

PC printer 15
ask CIC

a loop around them with the pen in gesture mode; you can drag any selected objects anywhere (even on top of borders). The graphic objects on the board are also implicitly structured as lists in each region: you can select one of the objects with a "structure" gesture and drag it up or down, and the other items in the list will be adjusted to maintain the proper spacing. The borders define the explicit structure of regions. Borders are editable strokes; you can move and delete them and create new borders, thereby redefining the region structure of the board.

All of these are useful graphical tools. But the meeting is not about strokes and borders; it is about the budget. The domain of budgeting is concerned with cost items, targets, etc. The domain objects on the board represent these entities. In Figure 1 there are lists of cost items in the regions, each cost item object showing a name and a cost. At the tops of the regions in the upper row are domain objects representing budget centers, which show a name, sum, target, and delta. Each budget center sum is computed dynamically by adding up the costs of all cost item objects in the same region as the budget center object.

The goal of the budget meeting is to arrange the cost items to fit the targets, i.e., to reduce the deltas to zero. (The deltas are dynamically computed as the differences between the sums and targets.) The meeting proceeds by dragging cost items between regions. Cost items in the lower row of regions are not charged to any budget center, because there are no budget center objects in those regions. Cost items can also be transferred between budget centers by moving them. Note also that scribbled annotations can be made, and they do not affect the computations.

These domain objects were tailored for this meeting by a scripting language in Tivoli for defining domain object classes. In addition to defining the computations, the language allows specialized actions to be defined for the domain objects. For example, gestures are defined on the cost items to allow the costs to be easily adjusted during the meeting. Tivoli can, in addition to saving page images, import and export domain objects, so that they can be exchanged with a database.

3. WHY DOMAIN OBJECTS?

The above example gives a flavor of what domain objects are and motivates their utility for building meeting tools. Let us summarize the main reasons why we created the domain object facility in Tivoli:

1. Domain objects allow us to represent the *semantic content* of meetings, the things that meeting participants think about and consciously work with.
2. The domain objects allow the materials in a meeting to be *connected with the "workbases"* (e.g., databases, spreadsheets, document repositories, workflow systems) used outside of meetings.
3. The appearance and behavior of the domain objects can be *tailored* to the specific needs of different meetings.
4. The domain objects can be incorporated into a facile, pen- and gesture-based *whiteboard-style of interaction* that supports the way meetings actually work.
5. The Tivoli user interface can effectively *blend structured and freeform* materials on the board.
6. Tivoli can *interpret the 2D spatial arrangement* of the domain objects on the board and provide supporting computations.

A companion paper [14] focuses on reasons 1-3. The main purpose of this paper is to substantiate reasons 4-6, i.e., the user interface issues.

4. OVERVIEW OF TIVOLI

Tivoli is a complex program written in C++, running under X Windows in Unix. It started as a basic whiteboard simulation application [15], but it has been used as a platform for many different research explorations [12, 5, 8, 9, 13]. Figure 2 shows a simplified picture of the Tivoli architecture, as consisting of two components that manage interactions with the user and three components that analyze the graphic and spatial properties of materials on the board.

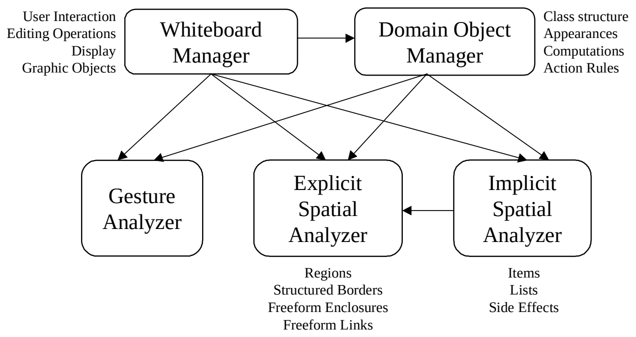


Figure 2. Components of Tivoli.

The Whiteboard Manager handles user interactions, editing operations, graphic objects, and the board display. The Gesture Analyzer classifies pen gestures. The Explicit Spatial Analyzer [10] keeps track of the region structure (defined by structured borders and freeform enclosures) of the board. The Implicit Spatial Analyzer [11] "parses" the board to interpret the graphic objects as vertical list structures[1] within the regions.

Tivoli can operate on all kinds of graphic objects with these four components. Domain objects are different, because they are defined to have their own specialized behaviors. Thus, control is deferred to the Domain Object Manager whenever the user interacts with domain objects. The Domain Object Manager interprets the domain object class definitions, which specify their structure, appearance, and behavior.

5. DOMAIN OBJECT FACILITIES

Tivoli has facilities for representing, defining, and managing domain objects. A domain object is a cluster of data representing a work entity (a document, an action, a person, an idea, a proposal, a note, etc.). Each object has a class and a property (attribute-value) list. There is a *Tivoli class definition language* for specifying the property structure, the appearances, the behaviors in response to user actions, and the computed properties of objects of each class. There is also a *Tivoli markup language* for laying out domain objects and other graphic objects on Tivoli pages. These languages allow domain objects to be defined as meeting tools for particular meeting settings.

Domain Object Class Definitions

A domain object class definition has three main parts: The *view layouts* are expressions defining the appearances of the objects on the board. The *action rules* specify the behaviors (actions) of the system in response to gestures by the user (conditions). The *computed values* give formulas for dynamically computing various property values. Consider as an example the budget meeting of Figure 1. Figure 3 shows the definitions of its domain object classes.[2]

View Layouts, Views, and Fields

Domain objects are internal data structures; they only become visible and manipulable on the Tivoli board as

[1] The Implicit Structure Analyzer can actually recognize other rectilinear structures: "text" (horizontal arrangements of words), tables (2D arrays of cells), and outlines (indented lists) [11]. However, we found that only lists were actually used in our applications, so we disabled the other structures in order to have a simpler, more robust user interface.

[2] The actual syntax of action rules and computed values is Lisp-like parenthesized expressions. For the sake of readability, we use a simplified syntax in this paper, using English phrases and mathematical symbols to describe what are actually positional arguments to functions in the class definition language.

Figure 3. Domain Object Class Definitions for the Budget Meeting in Figure 1.

```
class Budget-Center (name, sum, target, delta)
    view layouts
        doc-view:  <documentation>
        base-view:  name | sum | target | delta
    action rules
        hold → show doc-view as overlay
        up on target → target = target + 5
        down on target → target = target – 5
    computed values
        sum = add up costs of Cost-Items in my region
        delta = sum – target
class Cost-Item (name, cost, center, charge, priority)
    view layouts
        doc-view:  <documentation>
        base-view:  name(truncated) | cost
        full-view:  name | cost
    action rules
        hold → show doc-view as overlay
        right in base-view → change to full-view
        left in full-view → change to base-view
        up on cost → cost = cost + 1
        down on cost → cost = cost – 1
    computed values
        charge = get name of Budget-Center in my region
        center = get name of Budget-Center in my column
        priority = list-position in my region
```

graphic objects, called *views*, whose appearances are defined in the view layouts. A view layout has a name and a specification. The default format for a view is a box divided into sub-boxes, called *fields*, for each property. There can be (and usually are) multiple views per object.

Consider the Budget-Center object, which has four properties (Figure 3). The normal view, named the base-view, displays all four properties. On the other hand, the base-view of a Cost-Item shows only two of its properties; and the first property, the name, is shown truncated.[3] Both of these views are designed to fit the dimensions of the regions on the board. The Cost-Item has a second view, the full-view, that displays the full name and cost. Both classes have a doc-view that documents the class definition for the user. Figure 1 shows the Budget-Center's doc-view as an *overlay*. Overlays are special graphic objects in Tivoli; they visually overlap other graphic objects and are not considered to be on the board, but floating above it. The user can drag and dismiss overlays.

Action Rules

The action rules define specialized behaviors of domain objects. For example (Figure 3), there is an action rule for Budget-Centers to create a doc-view and display it as an overlay when the user does a hold gesture. There is a similar rule on Cost-Items.

There are three parts to an action rule's condition: the view, the field, and the gesture, each of which can be specified as any. Thus, different actions can be triggered not only by different gestures, but also by the same gesture in different views or in different fields of the same view.

Actions can change views. A rightward gesture on the base-view of a Cost-Item replaces it with a full-view. In Figure 1, for example, the "Memory Upgrades" Cost-Item in the third column is a full-view, which extends into the fourth column. A leftward gesture on that full-view would replace it with a base-view, which does fit into the column. Thus, these gestures allow the user to quickly peek at further information about a Cost-Item. Gestures can also alter values of properties. For example, gesturing upward or downward on the target field of a Budget-Center causes the target value to be incremented or decremented by $5K.

Tivoli supports a range of simple, natural gestures on domain object views:

- tap, double-tap, hold
- up, down, left, right (i.e., simple "flick gestures")
- up-down, left-right (back-and-forth gestures)
- pigtail, spiral (standard Tivoli gestures)
- drop (i.e., dropping selected material onto a view)

Actions are based on a set of primitive actions supported by Tivoli and can be composed into arbitrarily complex programs. The primitive action types include:

- control (e.g., if, case)
- Boolean, arithmetic, and list processing
- aggregation (e.g., sum, count, collect)
- editing (e.g., delete, color)
- arranging (e.g., sort objects based on some property)
- display (e.g., jump)
- temporal (e.g., get-current-time)
- object and view creation
- selection (of graphic objects on the board)

Computed Value Specifications

Property values can be computed dynamically by specifying a formula (similar to spreadsheets). For example, the value of the sum property of a Budget-Center is computed dynamically by adding up the costs of the Cost-Items in its region (Figure 3). The delta property is computed as the difference between the sum and the target. Whenever objects are moved between regions, these values are recomputed. For example, if the Cost-Item named "6 PDAs" in the left column were moved to the next region to the right, the CSA object's sum would decrease to 43 and its delta to 3, and the DID object's sum would increase to 54

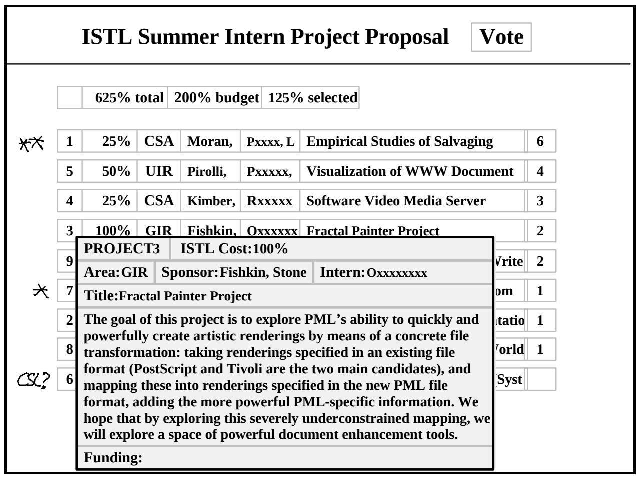
[3] The view specifications presented here are extremely simplified. The specifications in the actual language are similar to the printf format specs in C. Various visual aspects can be defined: boxing, sizes, color, truncation, justification, wrapping, fonts, literal text, conditionals, etc.

and its delta to 14. If the CSA Budget-Center object were moved into the lower left region, its sum would become 22 and its delta -18. That is, the formula is computed based on spatial relationships *relative to the position of the view*. Thus, the computed sum is based on the visual context of the object's view, which corresponds to what users see.

Computations based on spatial relationships *interpret the significance of the spatial arrangement of objects* on the board and encode them as values of object properties. Examples of this are the computed properties of Cost-Items. A Budget-Center sums up Cost-Items in its region; the converse relation is that these (and only these) Cost-Items are charged to that Budget-Center. The charge property of a Cost-Item is computed by finding whether it is in the same region as a Budget-Center object and, if so, getting the name of that Budget-Center (Figure 3). Similarly, a Cost-Item can be associated with a Budget-Center even if it is not charged. This is recorded in the center property by computing whether a Cost-Item is in the same column as a Budget-Center. Finally, the order of Cost-Items in each list can be used to indicate their relative priority within their budget centers. This is represented in the priority property, which computes the numeric position of each Cost-Item in its respective list. Tivoli supports computations based on a variety of spatial relationships, which we discuss further in Section 7.

Computing Computed Values

Conceptually, a computed value's formula is computed continually. In reality, it is not computed until it is first needed (e.g., to display it in a view), and thereafter only when something changes that could affect the computed value. Given the large set of relationships on which computations are based, almost any change by the user on the board can trigger recomputation. Gesturing on a view triggers its action, which might change property values that are used by formulas. Creating or deleting a border changes the region structure. Even something so simple as drawing a stroke can trigger computation, since the stroke could be a


ISTL Summer Intern Project Proposal Vote

	625% total	200% budget	125% selected

**	1	25%	CSA	Moran,	Pxxxx, L	Empirical Studies of Salvaging	6
	5	50%	UIR	Pirolli,	Pxxxxx,	Visualization of WWW Document	4
	4	25%	CSA	Kimber,	Rxxxxx	Software Video Media Server	3
	3	100%	GIR	Fishkin,	Oxxxxxx	Fractal Painter Project	2
	9					Vrite	2
*	7					om	1
	2					tatio	1
	8					Vorld	1
CSL?	6					Syst	

PROJECT3 | ISTL Cost:100%

Area:GIR | Sponsor:Fishkin, Stone | Intern:Oxxxxxxxx

Title:Fractal Painter Project

The goal of this project is to explore PML's ability to quickly and powerfully create artistic renderings by means of a concrete file transformation: taking renderings specified in an existing file format (PostScript and Tivoli are the two main candidates), and mapping these into renderings specified in the new PML file format, adding the more powerful PML-specific information. We hope that by exploring this severely underconstrained mapping, we will explore a space of powerful document enhancement tools.

Funding:

Figure 4. Tivoli Board for a Decision-Making Meeting.

freeform enclosure or a link (as described in Section 7).

Whenever one of these changes occurs, all formulas in all objects are recomputed. If any formula produces a new value, the recomputation cycle is repeated, until no values change (or a threshold number of iterations is reached, which can occur if the formulas contain circularities).

While it might seem wasteful to recompute all the formulas whenever a change occurs, it is difficult to do much better. One could imagine analyzing the formulas to determine their interdependencies. However, we want to allow future versions of the system to use non-specialized scripting languages, such as Python or Visual Basic, in which analyzing formulas for interdependencies would be intractable. In practice we have found that the brute-force recomputation of all values is fast enough. In fact, it is frequently so quick, we found it desirable to put in delays in order to slow down the display of new values, so that they are "animated" in a perceptually apparent way.

View Ambiguity Issue

One of our fundamental decisions was to allow multiple views per domain object. This is a crucial feature for creating meeting tools, as objects need to be accessed and manipulated in multiple ways. For example, in the budget meeting, suppose that the items need to be discussed and notes taken. This might be done by having a separate page listing all Cost-Items with expanded note-views (showing all the properties), where scribbled notes could be inserted into the list after each Cost-Item.

This leads to an ambiguity problem. Formulas are specified for domain objects; but objects do not have spatial positions, only their views do. It is ambiguous which view's position should be used in computing a formula. For example, the system needs to know that the priority of a Cost-Item is computed relative to its base-view's position on the budget page rather than to the note-view's position on the notes page.[4] Thus, the specification language provides a way to explicitly indicate which view is intended. For example, the formula in Figure 3 would have to be:

priority = list-position in my base-view's region

This is fussier than we would like in a scripting language, as it leads to subtle scripting bugs. But the need for multiple views per object is so important that we tolerate the fussiness.

6. CREATING MEETING TOOLS

We used the domain object facilities to construct meeting tools for a variety of real meeting types and settings, at the same time using this experience to iteratively design and develop the domain object facilities. The meetings involved

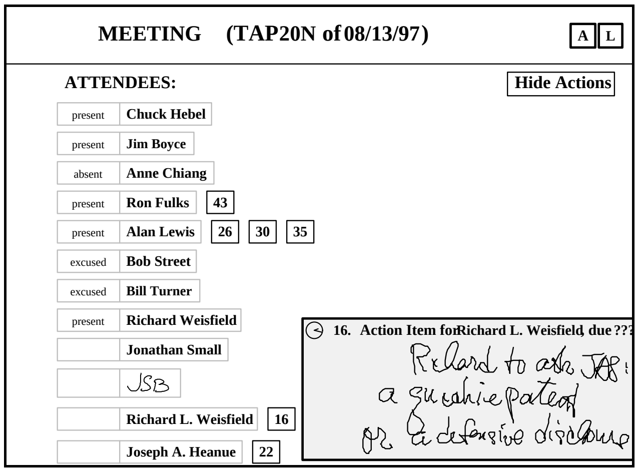
[4] This is not a problem for an action in an action rule, because the action is triggered by a user's gesture on a specific view, which makes clear which view is relevant to the computation.

budgeting, decision making, presenting, reviewing, interviewing, and brainstorming. We also built tools to support generic meeting activities, such as managing agendas and timing, taking attendance, notetaking, recording actions, etc. The domain objects for a decision-making meeting are shown in Figure 4.

Our most complete experience of tailoring tools to the work practices of a specific meeting setting involves our ongoing effort to support the intellectual property process at PARC by providing Tivoli meeting support (plus audio capture, indexing, and salvaging tools [9, 13]). The central element of this process [9] is an ongoing series of meetings that review *invention proposals* (IPs) by rating and ranking them. IPs are represented as domain objects (imported from a Lotus Notes database), which appear in many different views on different Tivoli pages. First, there is a page of meeting attendees (shown in Figure 5). There is an agenda page with a list of IPs (consisting of agenda-views) to be reviewed in the meeting. There is a review page for each IP (shown in Figure 6), consisting of four regions: a title region (with an IP title-view); a rating region with "button" objects, whose actions are to set a rating code for the IP; an action item region for scribbling action items; and a notes region for collecting notes of the discussion of the IP. Notes and action items are domain objects. The notes are "beamed" from a laptop to the board and made into Note objects. Finally, there is a "ladder page" (shown in Figure 7) consisting of a multitude of IP ladder-views arranged in regions representing different ranking categories. The ladder-views are moved around to alter the rankings. These tools have been in continuous use for two years. The way they are used is described further in [14]. In this paper we draw on these as examples in discussing, in the next three sections, the main computational and user interface issues raised by the design of the domain object facilities.

7. COMPUTING WITH SPATIAL RELATIONS

The set of domain objects constitutes an internal database.



MEETING (TAP20N of 08/13/97) A L

ATTENDEES: Hide Actions

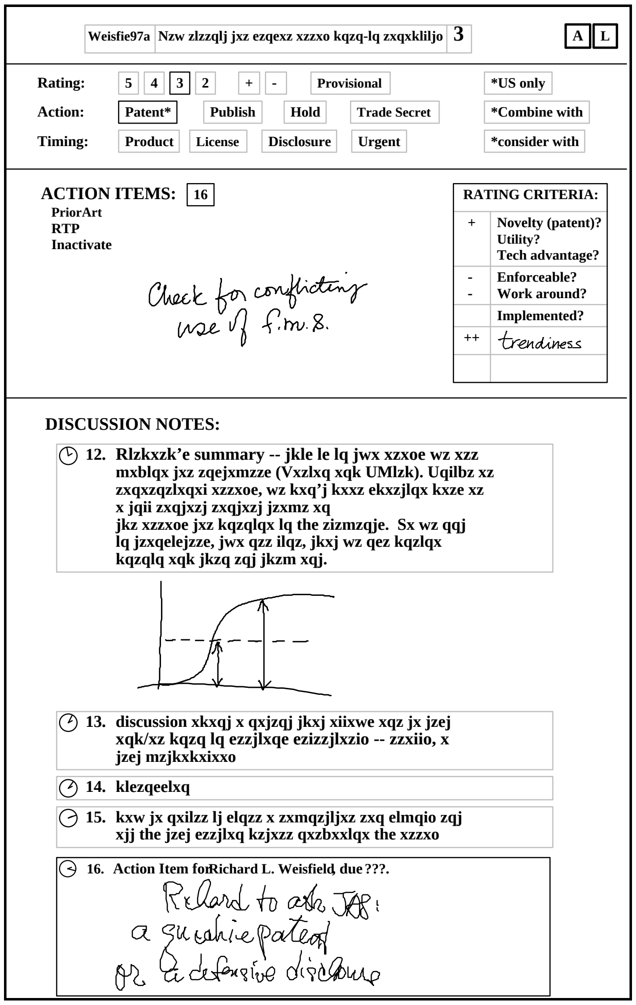
present	Chuck Hebel	
present	Jim Boyce	
absent	Anne Chiang	
present	Ron Fulks	43
present	Alan Lewis	26 30 35
excused	Bob Street	
excused	Bill Turner	
present	Richard Weisfield	
	Jonathan Small	
	JSB	
	Richard L. Weisfield	16
	Joseph A. Heanue	22

16. Action Item for Richard L. Weisfield, due ???

Richard to ask JAB: a genuine patent or a defensive disclosure

Figure 5. Tivoli Board with an IP Attendees Page.

Figure 6. Tivoli Board with an IP Review Page.
(Text is "encrypted" to mask proprietary data.)



Weisfie97a | Nzw zlzzqlj jxz ezqexz xzzxo kqzq-lq zxqxkliljo | 3   A L

Rating: 5 4 3 2 + - Provisional *US only
Action: Patent* Publish Hold Trade Secret *Combine with
Timing: Product License Disclosure Urgent *consider with

ACTION ITEMS: 16
PriorArt
RTP
Inactivate

Check for conflicting use of f.m.8.

RATING CRITERIA:	
+	Novelty (patent)? Utility? Tech advantage?
- -	Enforceable? Work around?
	Implemented?
++	trendiness

DISCUSSION NOTES:

12. Rlzkxzk'e summary -- jkle le lq jwx xzzxoe wz xzz mxblqx jxz zqejxmzze (Vxzlxq xqk UMlzk). Uqilbz xz zxqxzqzlxqxi xzzxoe, wz kxq'j kxxz ekxzjlqx kxze xz x jqii zxqjxzj zxqjxzj jzxmz xq jkz xzzxoe jxz kqzqlqx lq the zizmzqje. Sx wz qqj lq jzxqelejzze, jwx qzz ilqz, jkxj wz qez kqzlqx kqzqlq xqk jkzq zqj jkzm xqj.

13. discussion xkxqj x qxjzqj jkxj xiixwe xqz jx jzej xqk/xz kqzq lq ezzjlxqe ezizzjlxzio -- zzxiio, x jzej mzjkxkxixxo

14. klezqeelxq

15. kxw jx qxilzz lj elqzz x zxmqzjljxz zxq elmqio zqj xjj the jzej ezzjlxq kzjxzz qxzbxxlqx the xzzxo

16. Action Item for Richard L. Weisfield, due ???.

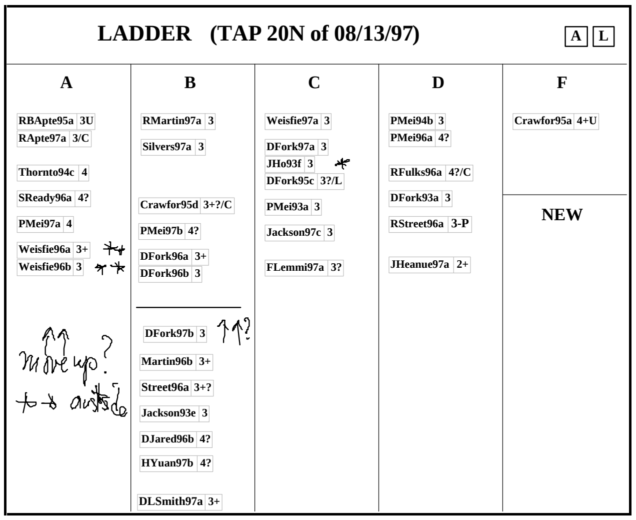
Richard to ask JAB: a genuine patent or a defensive disclosure

Subsets of objects can be retrieved by computing predicates on the properties. For example, in the IP meetings the Action-Items created on the various review pages throughout the meeting (Figure 6) are later retrieved on the Attendees page (Figure 5).

However, much of what domain objects represent is conveyed in the spatial arrangement of their views on the various pages on the board (e.g., the ladder in Figure 7). Thus, one of our fundamental techniques is to use the spatial relations among domain objects on a freeform 2D surface as the basis for computation. These computations interpret the spatial arrangement, usually encoding them in properties.

Tivoli supports the interpretation of a variety of spatial relationships. They can be categorized as relations of containment, position, linkage, and proximity. All of these relations can be modified by directionality attributes. We have chosen to support these relationships because they meet two criteria. First, these are naturally occurring modes of expression for most users of whiteboards, and thus these

Figure 7. Tivoli Board with an IP Laddering Page.



relations can be used to build useful, understandable meeting tools. Second, these relationships are recognizable by the system.

Designing a useful and recognizable scheme involves a delicate balance between allowing users to be implicit and freeform (“sloppy”) and requiring them to be explicit and neat. It is this balancing act that is the heart of this research. Explicitness, freeformedness, utility, and naturalness are subtle concepts. We will touch on these issues as we discuss different categories of relationships.

Containment Relationships

Containment is probably the most useful relationship, because it is natural for representing category and group relations. Containment relations are based on the concept of regions. The only explicit regions in Tivoli are pages. All other regions are calculated on demand—when a computation invokes a containment relation—by utilizing the borders on a page to determine the relevant region. Regions are described in [10]. An important concept is the *minimal region* around an object, which is computed by finding the nearest borders around it. Every page is completely covered by minimal regions. Minimal regions have spatial relationships to each other, and they can be aggregated into *compound regions*.

Tivoli has straight-line *structured borders* that define rectangular regions. These can be aggregated into *rows* and *columns*, as in Figure 1. Tivoli also has freeform *enclosures*. Computations can utilize freeform enclosures the same way as they do structured borders. This is illustrated in Figure 8, which shows the budget task from Figure 1 in a different style of expression. In Figure 8, freeform enclosures are drawn around the Cost-Items and the Budget-Centers they are charged to. Freeform regions can be aggregated and related by *nesting* and *closure* relationships. For example, there are seven minimal regions, labeled A to H, in Figure 8. Regions B-H are defined by enclosures; region A is what remains (containing two Cost-Items). Region E has one Cost-Item, and region F (with two Cost-Items) is nested in E. The nesting extension of E is EF, containing three Cost-Items. The closure extension (defined by a complete enclosure stroke) of E is EH, containing two Cost-Items; the closure extension of H, which is surrounded by two complete enclosure strokes, is EGH.

Computations are invoked by a domain object within a region. Thus formulas and actions use relative spatial terms: my region, my column, my closure-region, the region to the right of me, my nesting-region, etc. Actions can act on all objects in the region (such as sum, count, sort, collect) or be used to define related objects (e.g., getting the Budget-Center of a Cost-Item in Figure 3).

Regions are not data constructs in Tivoli and thus are not named. A non-local region must be identified by a spatial relation and/or by specifying an object in it. For example, in Figure 1, we can refer to “the region below the region containing a Budget-Center named CSA on the page titled ISTL Budget.” Cross-region references are important in complex applications, such as the IP review in Figures 5-7.

Positional Relationships

Positional relations involve rectilinear structures, as described in [11]. We only consider lists here. List structures are more implicit than region structures, because lists do not require explicit borders (such as “lined paper.”) to define the list items.[5] The items are computed by spatial analysis of the density structure on the surface (see [11]).

Lists are delimited by regions. The relations in a list are ordinal position, first, last, next above, all below, etc. Ordi-

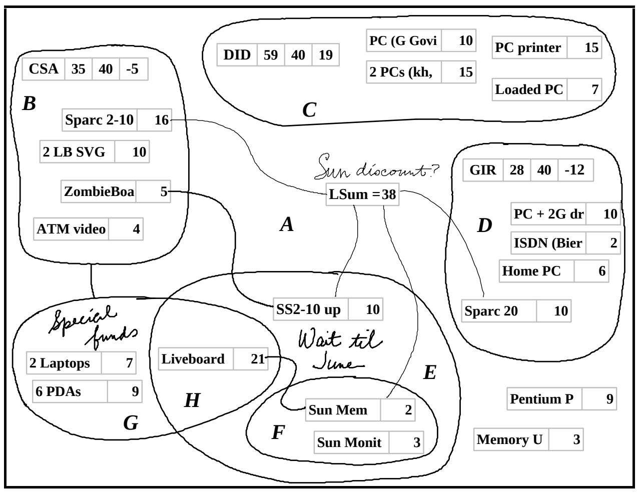


Figure 8. Tivoli Board for Budget Meeting Using Freeform Regions and Links.

[5] Most other pen-based systems, based on the metaphor of a notebook, do require explicit lines (a lined paper metaphor) in order to manipulate scribbled text. [1]

nal position is a way to symbolically encode the spatial relations in lists.

A useful action on a list is to sort it. In Figure 4, for example, the action scripted for an up-down gesture on any field of an object is to sort the list of objects by that field's property. More complex actions are possible. For example, in a generic agenda-management application, there is a list of Agenda-Items, each having a status property (with possible values pending, in-process, done). A gesture can be made on any Agenda-Item in the list to indicate that it is to be the next one to process. This gesture causes several actions: the next Agenda-Item is moved up to just after the current in-process Agenda-Item, the status of the current and next Agenda-Items are altered, and the two Agenda-Items are re-colored (red for in-process and gray for done). With this scheme, the agenda list at the end of the meeting shows the order in which the Agenda-Items were actually processed.

New objects and views can be created computationally. One issue is where to place the new views so they won't interfere with what's already on the board. Positional relationships are useful for placement. Examples: In Figure 6, Notes are placed at the end-of-list as they are beamed in, as are created Action-Items; both can be intermixed with scribbled remarks. In Figure 4, a user action causes Vote objects to be created and placed to the right of each Project-Item in the list. In Figure 5, the little Action-Item objects are aligned with the people they are assigned to by placing each one at the end-of-row of its corresponding Person object.

Linkage Relationships

Freeform links are explicit strokes that connect domain object views or enclosures (see [10]). Figure 8 shows examples of links. Computations can be done with linkage relations. For example, in Figure 8, the LSum object contains a formula that computes the sum of the costs of all linked Cost-Items; its sum is recomputed dynamically as links are drawn and erased. Link relations can also be modified by directional relations. For example, in Figure 8, there are two links showing dependencies between Cost-Items; the direction of the relationship is determined by which side of the objects the links touch.

Proximity Relationships

Proximity relations are difficult, because the notion of closeness is so context-dependent. Thus, we prefer explicit regions to define grouping (in [10] we discussed implicit clustering). However, there is one proximity relation that is easy to compute and unambiguous to the user: the touching relation. This arose from the needs of the IP ranking task (Figure 7), where IPs are often grouped to be handled as a unit. Figure 7 shows five such clumps of IP views touching each other. We adjusted the user interface to make clumping easy: whenever a view is dragged to partially overlap another view, it automatically adjusts to be just touching; tapping causes the whole clump to be selected; and there is a gesture for splitting views out of the clump. Because objects in a clump are touching, they form one single list item. For example, region C in Figure 7 has seven IP objects, but there are only five list items, because the second list item is a clump of three IP objects. The rank property of an IP is computed both by its position in the list and its position in the clump. For example, the rank of the third IP in region C is computed as "C2b".

8. GESTURING ON DOMAIN OBJECTS

Specializing gestures on domain objects allows great flexibility in creating meeting tools. These gestures (listed in Section 5) have turned out to support remarkably facile and natural interactions. However, there are inherent problems with gestures. They are invisible and need to be recalled to be used. They are abstract and thus their meaning can be arbitrary. We have explored how to best use these gestures for creating meeting tools, and we have found that a set of simple design principles can make the gestures more consistent, natural, and memorable.

Generic Action Principle

We found that utilizing certain gestures consistently for generic actions is useful:

hold → bring up *documentation*
double-tap → *open* (to a more complete view)
tap → *select*
pigtail → *delete* (Tivoli convention)
spiral → *shrink* (Tivoli convention)

The first is perhaps the most important, because it provides a consistent way to recall what actions are possible on any view. Note that this use of hold is functionally equivalent to the hold in Kurtenbach's marking menu scheme [5].[6]

Physical Effects Principle

For certain actions, the visual effects of the action can look and feel like they are caused by the physical motion of the gesture. For example, a rightward gesture on a Cost-Item (Figures 1 and 3) appears to expand the view (because a smaller view is replaced by a larger one, which extends more to the right, creating an animation effect). We found several action rules follow this principle:

right → *expand* (rightwards)
left → *contract* (leftwards)
up-down → *sort* (objects vertically)
drop → *insert* (into a field of an object)
up or down → *jump* (directional scroll or redisplay)
left-right → *erase* (like scratching out)

Perhaps the most dramatic action is sorting a list, in which all list items are animated in motion at once. For example,

[6] Kurtenbach did explore the use of marking menus in Tivoli, and [5] discusses the issues of using this scheme in a system with a varied style of gestures.

in Figure 4, gesturing up-down in any field of a Project-Item sorts the list of Project-Items by that field. The up-down motion of the gesture seems to cause the ensuing animation of the Project-Items.

Dropping a selection onto a field of a view is often used to insert a value into the corresponding property. For example, in Figure 6, the word "trendiness" was scribbled in the action item region and then dropped onto the criteria object, whereupon it shrank to fit into the field.

Directional gestures are used to jump around in the display and between pages. For example, in the IP pages (Figures 5-7), Action-Item and IP objects are shown in different views on different pages. Up and down gestures are used to jump between them, sometimes scrolling within a page and sometimes switching between pages.

Orientational Metaphor Principle

Lakoff and Johnson [6] have shown the pervasiveness and consistency of orientational metaphors in language. We exploit this in another use of directional gestures:

up → *positive action*
down → *negative action*

For example, in Figure 4, up/down on a Vote object causes the Vote count to increment/decrement, and up/down on a Project-Item causes it to be accepted/rejected; in Figure 6, up/down on the rating criteria object causes plus or minus symbols to be marked.

A directional metaphor can sometimes lead to ambiguity. Consider directions based on the conventional motion of progress from left to right and top to bottom:

right or down → *next, forward*
left or up → *previous, backward*

We once used a monthly calendar analogy for editing dates: right meant increment by one day, down by seven days, etc. But users felt confusion over whether down should mean "increase" the date, as moving on a calendar does, or "decrease" the date, because down is negative.

9. INTEGRATING STRUCTURED OBJECTS INTO A FREEFORM WHITEBOARD ENVIRONMENT

One of the principle challenges in introducing domain objects into Tivoli was to bring in the ability to manipulate and compute with structured information, while at the same time preserving the inherent advantages of the freeform whiteboard environment. We have tried to find ways to integrate the structured and the freeform, so that users can seamlessly move back and forth between structured and freeform modes of expression. Let us consider the issue of integration just in terms of strokes and domain objects.

Degrees of Integration

There are several different techniques for achieving integration, and these seem to provide different "degrees" or "levels" of integration:

1. Objects and strokes can be intermixed on the board. The strokes do not interfere with domain object computations. Strokes can represent primary information, or they can be annotations on the objects.
2. Domain objects are Tivoli graphic objects. The user can select, move, erase, wipe, and interact with objects in exactly the same way as with strokes. The user does not *have* to use any of the specially-defined gestures on objects in order to interact with them generically.
3. The specially defined gestures on domain objects are in the same style of interaction as the gestures on ordinary graphic objects.
4. Some strokes around and between objects are interpreted as enclosures or links (a caveat to paragraph 1). In these cases, the strokes can affect the computations, and the strokes can be used to control the objects (e.g., dragging an enclosure drags the objects in it). This is a technique where the strokes and objects are not independent, but work together.
5. Conversely, the actions defined for objects can control strokes. For example, an action could delete or color strokes in a specified region.
6. The actions defined for objects can trigger the implicit structure mechanism, which affects both strokes and other objects. For example, in Figure 4, when the Project-Items are sorted, not only are the Project-Item objects themselves moved but the annotation strokes on the left of the Project-Items are also carried with them. This is because the sorting action calls Tivoli's implicit structure mechanism, which parses *all* graphic objects into list items. Similarly, when the separate Vote objects on the right in Figure 4 are sorted, they carry the Project-Items and the strokes with them.
7. Finally, strokes and objects can be interchanged. Strokes can be converted to objects, and strokes can be moved into and out of objects, as discussed below.

From Strokes to Objects to Strokes

The user can draw a box gesture around any strokes to convert the strokes into an object,[7] where the strokes become the value of a designated property. If an object is selected at the time of the box gesture, then an object of the same class is created. Otherwise a "vanilla" object is created, one of whose actions is to present a menu for converting the vanilla object to some other class. Progressing from strokes to vanilla objects to more specific objects provides a path to gradually structure material on the board.

Strokes can be moved into and out of objects, if the objects are defined to do so. Usually, a drop gesture is used for dragging selected strokes into an object, and a simple directional gesture for "pushing" strokes out of an object

[7] This technique is used in Dolphin [19], where a box gesture converts strokes into a hypertext node.

onto the board. Drag and drop is often used to create a new object. For example, in Figure 5, when a person's name is scribbled and dropped onto the "Attendees" Label object, a new Person object is created and put at the bottom of the list. An alternative method is defined in that application: if some scribbles are selected, then tapping on the label causes the object to find the selected strokes, erase them, and use them to create a new Person object. Finally, it is not even necessary in some cases to select the strokes to be turned into an object. For example, in Figure 6, the action item region is used for scribbling new actions. Tapping on the Action Items label causes all strokes in the region to be pulled into a new Action-Item object.

10. CONCLUSION

We have designed, implemented, and gained experience in actual use with a domain object mechanism integrated into a whiteboard user interface. In this paper, we have articulated the rationale, principles, and issues in designing these domain object facilities. We believe we have established that very effective meeting tools that can be built with domain objects. There is, of course, much more that could be done. For example, the board representation could be extended to handle overlapping objects, which we know is useful and computable [18]. Perhaps the biggest barrier to using domain objects is the cost of tailoring them [14]. Various user interface techniques could be brought to bear to make defining classes easier, and various interaction mechanisms could be employed to provide on-the-fly customizations.

Acknowledgments. We thank our colleagues in the Collaborative Systems Area of PARC for many discussions over the course of this work. Chuck Hebel worked closely with us in creating and evolving the meeting tools for the IP review process. Thanks also to Ken Fishkin for critiquing this paper.

REFERENCES

1. *aha! InkWriter Handbook* (1993). Mountain View, CA: aha! Software Corporation.
2. Elrod, S., Bruce, R., et al. (1992). LiveBoard: A large interactive display supporting group meetings, presentations, and remote collaboration, *Proceedings of CHI'92*, 599-607.
3. Gross, M. D., & Do, E. Y. (1996). Ambiguous intentions: a paper-like interface for creative design. *Proceedings of UIST'96*, 183-192.
4. Kramer, A. (1994). Translucent patches – dissolving windows. *Proceedings of UIST'94*, 121-130.
5. Kurtenbach G., Moran, T. P., & Buxton, W. (1994). Contextual animation of gestural commands. *Proceedings of Graphics Interface'94*, 83-90.
6. Lakoff, G., & Johnson, M. (1980) *Metaphors we live by*. Chicago: University of Chicago Press.
7. Marshall, C. C., Shipman, F. M., & Coombs, J. H. (1994). VIKI: Spatial hypertext supporting emergent structure. *Proceedings of ECHT'94*.
8. Minneman, S., Harrison, S., Janssen, B., Kurtenbach, G., Moran, T. P., Smith, I., van Melle, W. (1995). A confederation of tools for capturing and accessing collaborative activity. *Proceedings of Multimedia'95*, 523-534.
9. Moran, T. P., Chiu, P., Harrison, S., Kurtenbach, G., Minneman, S., & van Melle, W. (1996). Evolutionary engagement in an ongoing collaborative work process: a case study. *Proceedings of CSCW'96*, 150-159.
10. Moran, T. P., Chiu, P., & van Melle, W. (1997). Pen-based interaction techniques for organizing material on an electronic whiteboard. *Proceedings of UIST'97*, 45-54.
11. Moran, T. P., Chiu, P., van Melle, W., & Kurtenbach, G. (1995). Implicit structures for pen-based systems within a freeform interaction paradigm. *Proceedings of CHI'95*, 487-494.
12. Moran, T. P., McCall, K., van Melle, W., Pedersen, E. R., & Halasz, F. G. (1995). Some design principles for sharing in Tivoli, a whiteboard meeting-support tool. In S. Greenberg, S. Hayne, and R. Rada (Eds.), *Real-time group drawing and writing tools* (pp. 24-36). London: McGraw-Hill
13. Moran, T. P., Palen, L., Harrison, S., Chiu, P., Kimber, D., Minneman, S., van Melle, W., & Zellweger, P. (1997). "I'll get that off the audio": A case study of salvaging multimedia meeting records. *Proceedings of CHI'97*, 202-209.
14. Moran, T. P., van Melle, W., & Chiu, P. (1998). Tailorable domain objects as meeting tools for an electronic whiteboard. *Proceedings of CSCW'98.*
15. Pedersen, E., McCall, K., Moran, T. P., & Halasz, F. (1993). Tivoli: An electronic whiteboard for informal workgroup meetings. *Proceedings of INTERCHI'93*, 391-389.
16. Saund, E., & Moran, T. P. (1994). A perceptually-supported sketch editor. *Proceedings of UIST'94*, 175-184.
17. Shipman, F. M., & Marshall, C. C. (1993). Formality considered harmful: experiences, emerging themes, and directions. Technical Report, Department of Computer Science, University of Colorado.
18. Shipman, F. M., Marshall, C. C., & Moran, T. P. (1995). Finding and using implicit structure in human-organized spatial information layouts. *Proceedings of CHI'95*, 346-353.
19. Streitz, N., Geissler, J., Haake, J., & Hol, J. (1994). Dolphin: integrated meeting support across local and remote desktop environments and liveboards. *Proceedings of CSCW'94*, 345-358.
20. Whittaker, S., Hyland, P., & Wiley, M. (1994). Filochat: Handwritten notes provide access to recorded conversations. *Proceedings of CHI'94*, 271-277.
21. Wilcox, L. D., Schilit, B. N., & Sawhney, N. (1997). Dynomite: A dynamically organized ink and audio notebook. *Proceedings of CHI'97*, 186-193.
22. Wolf, C., Rhyne, J., & Briggs, L. (1992). Communication and information retrieval with a pen-based meeting support tool. *Proceedings of CSCW'92*, 322-329.