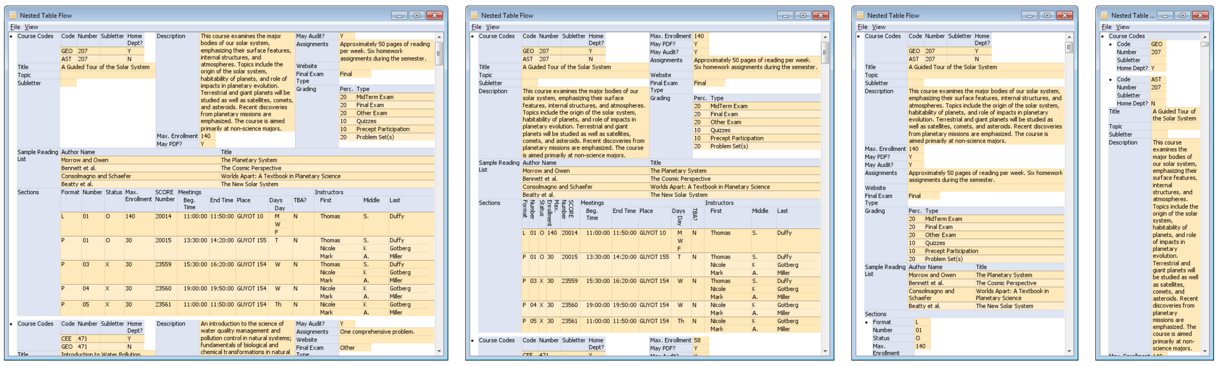
Automatic Layout of Structured Hierarchical Reports

Eirik Bakke, David R. Karger, and Robert C. Miller



Course Codes: Code | Number | Subletter | Home Dept?
GEO | 207 | | Y
AST | 207 | | N
Title: A Guided Tour of the Solar System
Topic:
Subletter:
Description: This course examines the major bodies of our solar system, emphasizing their surface features, internal structures, and atmospheres. Topics include the origin of the solar system, habitability of planets, and role of impacts in planetary evolution. Terrestrial and giant planets will be studied as well as satellites, comets, and asteroids. Recent discoveries from planetary missions are emphasized. The course is aimed primarily at non-science majors.
Max. Enrollment: 140
May PDF?: Y
May Audit?: Y
Assignments: Approximately 50 pages of reading per week. Six homework assignments during the semester.
Website:
Final Exam Type: Final
Grading: Perc. | Type
20 | MidTerm Exam
20 | Final Exam
20 | Other Exam
10 | Quizzes
10 | Precept Participation
20 | Problem Set(s)

Sample Reading List:

Author Name	Title
Morrow and Owen	The Planetary System
Bennett et al.	The Cosmic Perspective
Consolmagno and Schaefer	Worlds Apart: A Textbook in Planetary Science
Beatty et al.	The New Solar System

Sections:

Format	Number	Status	Max. Enrollment	SCORE Number	Meetings: Beg. Time	End Time	Place	Days: Day	TBA?	Instructors: First	Middle	Last
L	01	O	140	20014	11:00:00	11:50:00	GUYOT 10	M W F	N	Thomas	S.	Duffy
P	01	O	30	20015	13:30:00	14:20:00	GUYOT 155	T	N	Thomas	S.	Duffy
										Nicole	K	Gotberg
										Mark	A.	Miller
P	03	X	30	23559	15:30:00	16:20:00	GUYOT 154	W	N	Thomas	S.	Duffy
										Nicole	K	Gotberg
										Mark	A.	Miller
P	04	X	30	23560	19:00:00	19:50:00	GUYOT 154	W	N	Nicole	K	Gotberg
										Mark	A.	Miller
P	05	X	30	23561	11:00:00	11:50:00	GUYOT 154	Th	N	Nicole	K	Gotberg
										Mark	A.	Miller

Course Codes: Code | Number | Subletter | Home Dept?
CEE | 471 | | Y
GEO | 471 | | N
Title: Introduction to Water Pollution
Description: An introduction to the science of water quality management and pollution control in natural systems; fundamentals of biological and chemical transformations in natural
May Audit?: Y
Assignments: One comprehensive problem.
Website:
Final Exam Type: Other

Fig. 1. Interactive adaptation of the layout of the data to be displayed, based on the available horizontal width of an on-screen window.

Abstract—Domain-specific database applications tend to contain a sizable number of table-, form-, and report-style views that must each be designed and maintained by a software developer. A significant part of this job is the necessary tweaking of low-level presentation details such as label placements, text field dimensions, list or table styles, and so on. In this paper, we present a horizontally constrained layout management algorithm that automates the display of structured hierarchical data using the traditional visual idioms of hand-designed database UIs: tables, multi-column forms, and outline-style indented lists. We compare our system with pure outline and nested table layouts with respect to space efficiency and readability, the latter with an online user study on 27 subjects. Our layouts are 3.9 and 1.6 times more compact on average than outline layouts and horizontally unconstrained table layouts, respectively, and are as readable as table layouts even for large datasets.

Index Terms—Hierarchy data, tabular data, nested relations, layout management

1 Introduction

An important class of visualizations in everyday business use consists of the table-, form-, and report-style views found in most graphical relational database applications [2]. The data being displayed is typically *structured*, meaning that each value has an associated type and label in a schema. Furthermore, the data frequently needs to be presented in a *hierarchical* manner, because of the need to visualize one-to-many relationships between entities in the database. For instance, when users request “a list of employees, grouped by department” or “all information about a customer, including associated support tickets and a list of open orders,” what is being displayed is a structured hierarchical view of the underlying relational data.

Unlike other kinds of office productivity software, database applications tend to be tailored for specific target domains such as payroll, customer relationship management (CRM), or inventory tracking, or even more specific subdomains such as hotel reservation systems, inventory management for the offshore industry, or bug tracking systems for software developers [1]. The domain-specific nature of these systems makes them costly to develop and maintain, and there is seldom an opportunity to hire a graphic designer every time a new form needs to be created or an existing one updated. Yet, the software developer must invariably define the details of the view’s visual layout: the location of labels, the dimensions of text fields, the width of table columns, the organization of form fields into columns on a page, and so on. Since each database application can contain a sizable number of views—say, one table view and one form view per major (non-*weak*) entity type in the database schema, plus any special-purpose report views—maintaining the detailed visual layouts of each view is a significant development burden, and can lead to poor user interfaces.

The problem we approach in this paper is as follows: *How can the visualizations required of typical domain-specific database applications be generalized in such a way as to eliminate the need for manual tweaking of low-level visual details?* A general-purpose visualization system for nested relations data would be of direct utility to any one of many [15, 29, 27, 28, 16, 2] previously proposed database user interfaces based on structured hierarchical views. To scope our problem, we limit ourselves to the task of displaying simple textual data that has been pre-organized to conform to some hierarchical but non-recursive schema. This includes the documents of most data interchange formats based on XML and JSON, as well as the output of any object-relational or relational-to-XML mapping system that uses a finite number of standard SQL queries to produce its output, e.g. SilkRoute [8]. An equivalent data model is the *nested relational model*; see Levene [18] for an extensive bibliography.

We present a layout management algorithm that automates the display of structured hierarchical data using visual idioms seen in traditional hand-designed database UIs: tables, multi-column forms, and outline-style indented lists. The system gathers simple statistics about fields in the input schema, and uses these to make layout decisions which are then applied uniformly across tuples in each input subrelation. We compare our hybrid layout system with pure outline and nested table layouts with respect to space efficiency and readability, the latter with an online user study on 27 subjects. Our layouts are 3.9 and 1.6 times more compact on average than outline layouts and horizontally unconstrained table layouts, respectively, and are as readable as table layouts even for large datasets.

- *Eirik Bakke is at the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL). E-mail: ebakke@mit.edu.*
- *David R. Karger is at MIT CSAIL. E-mail: karger@mit.edu.*
- *Robert C. Miller is at MIT CSAIL. E-mail: rcm@mit.edu.*

Manuscript received 31 March 2013; accepted 1 August 2013; posted online 13 October 2013; mailed on 4 October 2013.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

2 Related Work

Tree Visualization. While our layout algorithm assumes the presence of a schema along with its hierarchical input data, it is always possible to derive a valid schema from the input data itself, and our system supports the visualization of any tree-structured input data in XML or JSON form, without further annotation. Our visual layouts make the most sense in cases where the schema can correctly be assumed to be non-recursive, which was the case for 9 out of 10 example datasets in the XML corpus we used for our user study. Thus, our system falls into the *tree visualization* category of information visualization research. Most differences arise from our assumption of a different variation of the tree-structured input data model than is typically seen in other tree visualization systems. Analogous data model specialization is seen for instance in Robertson's polyarchies [23]. While our system does not perform any kind of aggregation on input data, but merely displays every piece of data in full, the same is true of many standard kinds of tree visualizations.

Academic research on visualizations for tree-structured data has been surveyed by Shneiderman [24] and, recently, Graham and Kennedy [11]. In our case, we are dealing with the subproblem of *single tree* visualization, with the additional constraint that we are working with structured data only, i.e. data that conforms naturally to some schema. In Graham's taxonomy, the visualizations produced by our system form a hybrid between the *nested* and *indented list* representations. While nested tables, an important base case in our layout system, organize data in a grid of rows and columns, they do not fall into the *matrix* representation category, but rather among the nested representations, since data leaf nodes are always contained within the visual boundaries of their parent nodes.

The study by Chimera and Shneiderman [6] compared three variations of the outline-style indented list view; two of these were interactive. Their results suggest that future versions of our system should include collapsible nodes in indented lists. Ziemkiewicz et al.[31] compare four different classes of tree visualizations, including three nested variants. None of the layouts tested include tables or hybrids between tables and indented list representations. Ghoniem et al. [10] provide a taxonomy of readability tasks on graphs and present a study comparing node-link and matrix representations.

Treemap [14] is maybe the most well-known nested or Venn Diagram-style tree representation. Treemaps fit an arbitrarily sized tree into a viewport of pre-defined dimensions, by making subsequent levels smaller and smaller. The relative size of siblings is determined by some semantically significant weight associated with each node in the tree, such as disk space consumed in a map of a file system hierarchy. In contrast, our own system allocates to leaf nodes whatever visual space is necessary to display the contained primitive text values at a constant font size. This is a requirement from the perspective of our target applications. Our system does, however, constrain layout dimensions in the horizontal direction, so that only vertical scrolling is needed if the layout can not fit in the desired viewport. In treemaps, siblings are arbitrarily stacked vertically or horizontally at every other step in the recursion. In our own system, tuples are always stacked vertically. Fields are stacked either horizontally, when contained in a nested table, or vertically, when part of an outline view. This ensures one kind of visual consistency while still allowing the widths of layouts to be constrained.

FISH [20] is a variation of the Treemap concept. In this system, styling attributes can be used to configure various node presentation details, including the choice to stack siblings either vertically or horizontally. Such styling must be applied to each individual data node, unlike in our own system, where styling is applied to schema nodes only. Strip Treemap [3] optimizes the presentation of a treemap by sizing rectangles such that they can be stacked in contiguous "strips" without broken horizontal lines. Like Treemap, and unlike our own system, both FISH and Strip Treemap assume that all data must fit into a viewport of predetermined size both in the horizontal and vertical direction.

Elastic Hierarchies [30] and EncCon [21] both combine node-link representations with some other representation in order to visualize tree-structured data. Node-link representations are typically less space-efficient than nested or indented representations, and seldom appear in traditional database GUIs, our target application. Thus, we did not use them ourselves. EncCon uses node-link representations in every level of the tree, but maintains an invisible Treemap-like layout to determine a favorable positioning for each node. In Elastic Hierarchies, subsequent levels may use either Treemaps or node-link representations, determined interactively. The Elastic Hierarchies paper also has a relevant discussion of the design space of hybrid tree layouts, but does not consider nested tabular layouts. Neither does Graham and Kennedy's survey. This is likely because tabular layouts apply only to structured data, where series of children in a tree all can be expected to have the same substructure. Systems that solve the more general problem of dealing with semi-structured (schema-free) data will not naturally be able to take advantage of tabular layouts.

XML Visualization. VisualXML and XMLAD [7], like Elastic Hierarchies and EncCon, use node-link representations as part of their interface, but are geared towards XML visualization. The XMLAD system does indeed use tabular representations as part of the output layout, but does not support nested tables. Thus, only the bottom logical relations in the data may be displayed as tables. Similarly, the VisualXML system uses outline or list-style representations for the bottom relation level of each subtree. Again, both XMLAD and VisualXML use a node-link representation for all higher levels. This is in contrast with our own system, which uses a column-enabled, indented list for a variable number of top levels in the tree, followed by nested tables for a variable number of bottom levels. Furthermore, our algorithm makes these layout decisions automatically, except where manually overridden.

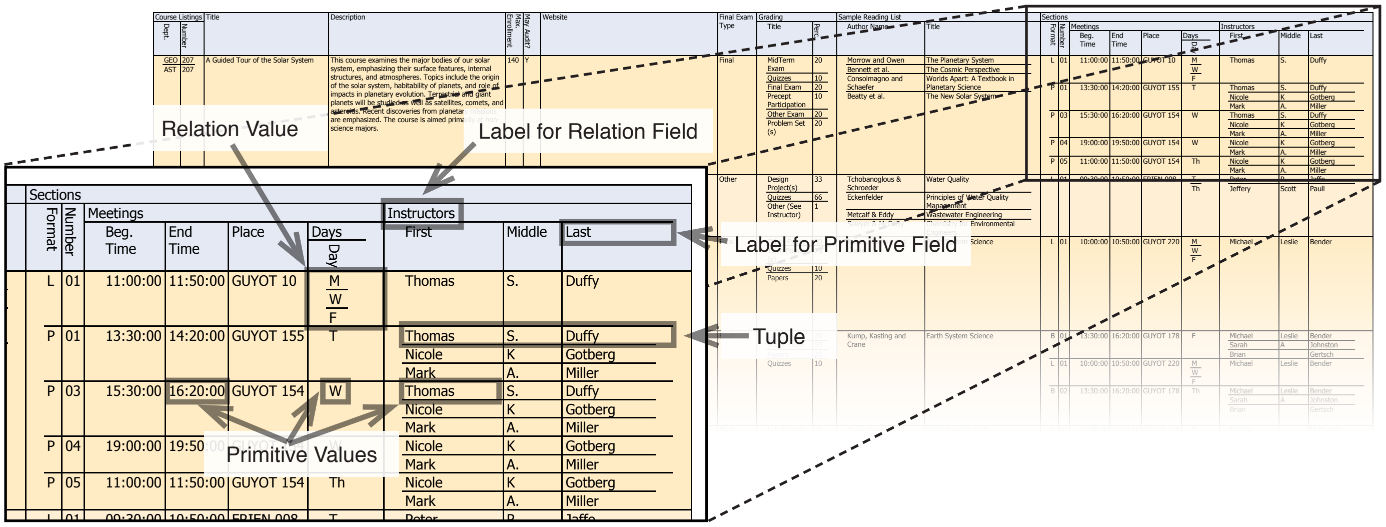
Tree Rewriting provides a visual language "very similar to the lambda calculus in simplicity and expressive power" [13]. While such systems are general enough to produce just about any layout from a set of input data, a user must manually specify how these layouts are to be constructed. Unlike our system, neither is able to produce a default layout subject to a constraint such as available page width. Furthermore, no mechanism is available to help ensure the careful alignment of column fields that is required in order to produce tabular or nested tabular sublayouts.

A relevant class of commercial systems consists of *report generator* tools such as Crystal Reports[1] and Altova XML-Spy/StyleVision[2]. An extensive survey is provided by Król [17]. These systems let the user build output layouts analogous to those produced by our own system, using a variety of input data sources. The layout building process is manual. Altova's "Grid View" is similar to a nested table view, but with single-level column headings repeated for every subrelation. Navigation requires heavy use of both vertical and horizontal scrollbars as well as manual collapsing and expanding of data nodes. Altova does combine vertical arrangements of tuple fields in an outline-style sub-view with the use of nested tables, but is unable to render subrelations as indented lists.

UI Generation. Supple [9] generates widget-based user interfaces for devices of various sizes and contexts using a cost optimization algorithm, but does not deal with table layouts or other layouts that take advantage of repeated structure in data that adheres to a schema. The Right/Bottom strategy [4] defines heuristics for widget placement in dialog boxes, but similarly does not deal with structured relational data. Database application builder tools such as FileMaker[3] and Microsoft Access[4] include wizards to help with the creation of new forms, but are only able to layout out a single level of fields automatically, and can not use statistics about the data in each field to make structural layout decisions.

Tabular Systems. Show Me [19] is an autostyling system for Tableau, an interactive visualization system. Since Tableau operates on tabular input data and produces Pivot-table or *cross-tab*-based out-

[1]`http://crystalreports.com`
[2]`http://www.altova.com/xml-editor`
[3]`http://www.filemaker.com`
[4]`http://office.microsoft.com/access`



Sections									
Format	Number	Meetings				Instructors			
		Beg. Time	End Time	Place	Days / Day		First	Middle	Last
L	01	11:00:00	11:50:00	GUYOT 10	M W F		Thomas	S.	Duffy
P	01	13:30:00	14:20:00	GUYOT 155	T		Thomas	S.	Duffy
							Nicole	K	Gotberg
							Mark	A.	Miller
P	03	15:30:00	16:20:00	GUYOT 154	W		Thomas	S.	Duffy
							Nicole	K	Gotberg
							Mark	A.	Miller
P	04	19:00:00	19:50				Nicole	K	Gotberg
							Mark	A.	Miller
P	05	11:00:00	11:50:00	GUYOT 154	Th		Nicole	K	Gotberg
							Mark	A.	Miller
L	01	09:30:00	10:50:00	FRIEN 008	T		Peter	P.	Jaffe

Fig. 2. The nested table layout is the most common way to visualize a nested relation, and is supported as a special case in our algorithm. Here, we show nested relational data generated from an academic course catalog in nested table style, with one of the nested columns enlarged to show terminology. Nested table layouts stack tuples in the vertical direction and the fields of each tuple in the horizontal direction, with all field labels collected in a header on top.

puts, it does not fall into the tree-structured data visualization category.

Nested table layouts are used in Related Worksheets [2] and in applications produced with AppForge [28] or App2You [16]. These systems do not allow the nested table layouts to be re-styled as indented lists or deferred to lower layers of the tree-structure. They can not automatically constrain the width of the layout to the available page width.

FOCUS [25], TableLens [22], and the system by Tajima and Ohnishi [26] deal with the problem of displaying large, mostly flat tables. The latter system includes a "Record" viewing mode that resembles a single-level form view. While all of these systems support certain cases involving values spanning multiple cells, they do not operate on structured hierarchical data in general. Chi's visualization spreadsheets [5] combine a table layout at the outer level with cells containing plotted 3D graphics, but require the user to define each visualization using commands, and are primarily focused on numerical data.

Text Layout Systems. Text document layout systems, like the one described by Jacobs et al. [12], deal with the problem of rendering a given amount of text with a given font size on a set of dimensionally constrained pages. They are otherwise different from our own system, since they operate on a very different class of input data.

3 Layout Algorithm

We now describe our layout algorithm. For the purposes of exhibition, we start by discussing basic nested table and outline layouts, then discuss hybrid table/outline layouts, and finally show the complete steps to produce the layouts automatically.

The purpose of the algorithm is to produce a compact but readable view of tree-structured input data that conforms to a non-recursive schema, or, more precisely, nested relational data. Whenever possible, the regularity of the input data's schema should be used to maximize the readability of the data. For instance, tabular data should be rendered as a table, with proper column headings describing the name of fields in the schema. We may not require user input informing layout decisions beyond what can be derived from the data itself.

3.1 Nested Relations and Nested Table Layouts

Our algorithm operates on tree-structured data conforming to a non-recursive schema. We have chosen the nested relational model as the concrete data model for our implementation, since it tends to lead to simple tree traversal code. It is equally feasible to use a different tree-structured data model such as that of XML, and in connection with the user study, we did write a routine for importing arbitrary XML documents.

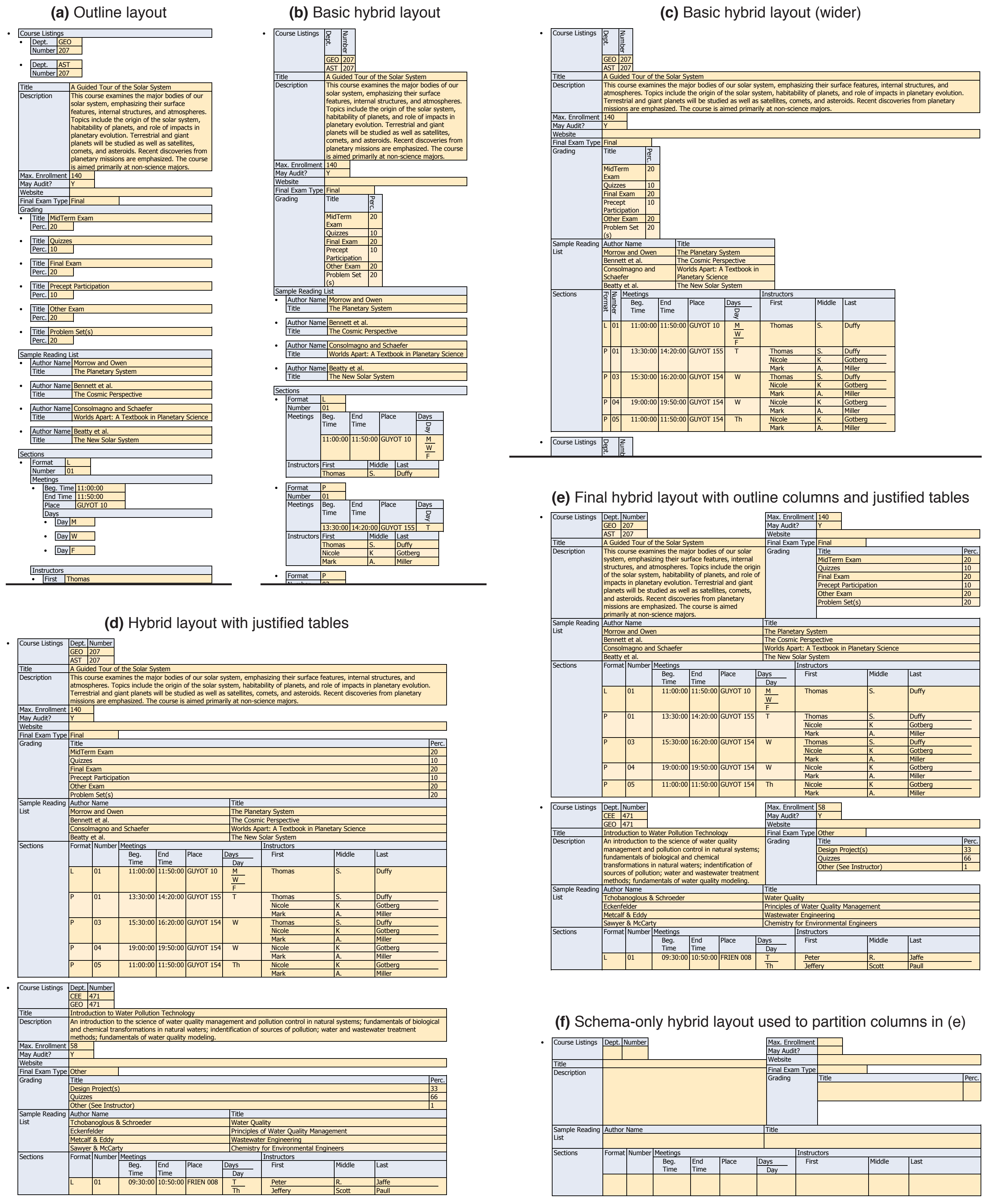
The terminology of the nested relational model is as follows: A *value* is either a *primitive* or a *relation*, where a relation is defined as a set of *tuple*s, each containing a set of *field*s identified by *label*s, each containing a value, recursively. The *schema* of a value either defines the value to be a primitive, or defines the value to be a relation, with schemas further specified for each of the latter's fields, recursively.

Nested relational data is most commonly illustrated in a *nested table* layout. An annotated example of such a layout is shown in Figure 2; it renders data about the first few courses in an academic course catalog. A nested table layout consists of a *header* area (shown in blue) and a *content* area (shown in beige). The header area reflects the schema structure of the displayed nested relation, with simple labels for columns containing primitive fields and recursive labels for columns containing relation fields. Columns containing relation fields may recursively contain columns for either primitive or relation subfields. In the content area, tuples in the rendered relation are stacked vertically with row separators between them. Each tuple renders its primitive values as simple strings and its relation values recursively as the content area of another nested table.

3.2 Outline Layouts

Nested table layouts, like the one previously seen, can quickly become extremely wide if many fields are to be displayed. For on-screen interfaces, the need for horizontal scrolling is undesirable, and for printing on paper, pagination in the horizontal direction is awkward. As another basic alternative to the nested table layout, we consider the *outline* layout, which is common among XML editing tools. Figure 3(a) shows the same data as before rendered using such a layout.

In an outline layout, tuples are stacked vertically in an indented bullet list fashion; we show one bullet per tuple. Unlike the nested table layout, which puts the fields of each tuple next to each other on a row, the outline layout stacks the tuple fields vertically as well. Thus, having more fields in a schema makes the layout taller, not wider. In a tuple in an outline layout, primitive fields are rendered as strings with their respective schema labels to the left, whereas relation fields are rendered recursively with their schema labels above, extended to the full width of the layout area. The color scheme is the same (blue and beige) for labels and values in an outline layout as for the header and


(a) Outline layout
(b) Basic hybrid layout
(c) Basic hybrid layout (wider)
(d) Hybrid layout with justified tables
(e) Final hybrid layout with outline columns and justified tables
(f) Schema-only hybrid layout used to partition columns in (e)

Course Listings: Dept.	Number
GEO	207
AST	207

Title	A Guided Tour of the Solar System
Description	This course examines the major bodies of our solar system, emphasizing their surface features, internal structures, and atmospheres. Topics include the origin of the solar system, habitability of planets, and role of impacts in planetary evolution. Terrestrial and giant planets will be studied as well as satellites, comets, and asteroids. Recent discoveries from planetary missions are emphasized. The course is aimed primarily at non-science majors.
Max. Enrollment	140
May Audit?	Y
Website	
Final Exam Type	Final

Grading: Title	Perc.
MidTerm Exam	20
Quizzes	10
Final Exam	20
Precept Participation	10
Other Exam	20
Problem Set(s)	20

Sample Reading List: Author Name	Title
Morrow and Owen	The Planetary System
Bennett et al.	The Cosmic Perspective
Consolmagno and Schaefer	Worlds Apart: A Textbook in Planetary Science
Beatty et al.	The New Solar System

Format	Number	Beg. Time	End Time	Place	Days	First	Middle	Last
L	01	11:00:00	11:50:00	GUYOT 10	M, W, F	Thomas	S.	Duffy
P	01	13:30:00	14:20:00	GUYOT 155	T	Thomas / Nicole / Mark	S. / K / A.	Duffy / Gotberg / Miller
P	03	15:30:00	16:20:00	GUYOT 154	W	Thomas / Nicole / Mark	S. / K / A.	Duffy / Gotberg / Miller
P	04	19:00:00	19:50:00	GUYOT 154	W	Nicole / Mark	K / A.	Gotberg / Miller
P	05	11:00:00	11:50:00	GUYOT 154	Th	Nicole / Mark	K / A.	Gotberg / Miller

Course Listings: Dept.	Number
CEE	471
GEO	471

Title	Introduction to Water Pollution Technology
Description	An introduction to the science of water quality management and pollution control in natural systems; fundamentals of biological and chemical transformations in natural waters; indentification of sources of pollution; water and wastewater treatment methods; fundamentals of water quality modeling.
Max. Enrollment	58
May Audit?	Y
Website	
Final Exam Type	Other

Grading: Title	Perc.
Design Project(s)	33
Quizzes	66
Other (See Instructor)	1

Sample Reading List: Author Name	Title
Tchobanoglous & Schroeder	Water Quality
Eckenfelder	Principles of Water Quality Management
Metcalf & Eddy	Wastewater Engineering
Sawyer & McCarty	Chemistry for Environmental Engineers

Format	Number	Beg. Time	End Time	Place	Days	First	Middle	Last
L	01	09:30:00	10:50:00	FRIEN 008	T, Th	Peter / Jeffery	R. / Scott	Jaffe / Paull

Fig. 3. We illustrate our algorithm by enabling its features one by one and producing successive layouts of the data from Figure 2. All layouts are at the same scale. (a) is a basic outline layout; this layout renders tuples in relation values as indented bullets, stacks the fields of each tuple vertically, and puts labels to the left of primitive values and above relation values. (b) and (c) show basic hybrid layouts, at two different widths, that use the outline layout at the first level but switch to nested table sublayouts wherever a table can fit within the available horizontal space. (d) justifies the columns of the table sublayouts to fill the remaining available horizontal space. (e) adds columns to the outline sublayout to use horizontal space more efficiently. (f) is a special schema-only layout generated by the algorithm to calculate ideal break points for the columns in (e).

content area of nested tables, respectively.

The outline layout, unlike the nested table layout, supports the concept of a horizontally constrained width. Like the layouts produced by our final algorithm, an outline layout can be produced for any available width (with some minimum constraints), breaking text in primitive value areas as necessary. Our system distinguishes between fixed- and variable-length primitive fields; value areas for fixed-length primitive field are always rendered at their predicted width, whereas areas for variable-length primitive fields are rendered to the full available width. This ensures that the visual width of a particular primitive field remains the same between successive tuples, even when the actual value differs. The rationale is two-fold. First, if layouts are to be used for data entry or data editing in a database application, the width of an input field should be commensurate with the expected size of its values. Second, keeping the perceived overall shape (*gestalt*) of successive tuples of the same relation schema similar should make it easier for a user to visually scan for specific fields in those tuples.

3.3 Hybrid Layout

Whereas nested table layouts quickly grow wide, outline layouts tend to be tall and narrow. Outline layouts use space inefficiently by only starting values on the left-hand side of the page, and by repeating schema labels once for every value. They are also harder to read than table layouts, since they tend to put values from the same schema field but from different tuples far apart.

In Figure 3(b), we introduce a hybrid layout that embeds nested tables inside an outline layout. This saves vertical space compared to the pure outline layout in Figure 3(a). Like for the outline layout, we assume that we are given a constrained horizontal width to work with, such as screen size or page width for printing. We then create a layout that is guaranteed to fit within this horizontal width. This avoids horizontal scrolling or pagination, and ensures that the layout will only grow in the vertical direction as more tuples or fields are added.

When building a hybrid outline/table layout, the algorithm must start at the root level of the relation schema and decide, for each relation field, whether to render that relation field using a nested table or another level of an outline layout. If the decision is made to render a relation field using an outline layout, the decision process is repeated recursively for each of its fields. If the decision is made to render a relation field using a nested table layout, all child fields are rendered as a nested table layout as well. While our recursive layout generation algorithm technically supports embedding outline sublayouts into nested table layouts, this makes little typographical sense, and we do not make use of this case.

The decision to use an outline layout vs. a nested table layout for a given relation schema could reasonably be made using a cost optimization strategy, for instance based on the total area consumed by the layout in each case. However, because nested table layouts almost invariably consume less space than corresponding outline layouts, a simpler heuristic is possible: always use a nested table layout if there is enough horizontal space available for it. This is the rule used in our algorithm; it is illustrated by contrasting the narrow layout in Figure 3(b) with the wider layout of Figure 3(c). In Figure 3(b), the layout is constrained to a small width, and small nested tables have been chosen by the heuristic for relation fields Course Listings, Grading, Sections/Meetings, and Sections/Instructors. All higher-level relations are rendered using outline sublayouts. In Figure 3(c), the same data is rendered at a larger constrained width, allowing both the Sample Reading List relation field and the entire Sections relation field to be rendered as a nested table. Note that a hybrid layout that is given enough horizontal space to work with will always degenerate to a pure nested table layout.

Since the outline vs. nested table decision heuristic depends on whether or not there is enough horizontal space for a table, the determination of table column widths is important at this stage. For relation fields, table columns are as wide as the sum of their child fields' columns, plus separator lines and side margins, as well as any extra space needed to accommodate the relation column's own header label. For fixed-width primitive fields, the width of the table column is simply the width of the field. For variable-width primitive fields, we experimented with various heuristics, and found the average width of values in the field to be a sensible minimum, limited upwards to a constant value (on the order of 50 characters, a standard book column size). Primitive columns also need to be wide enough to accommodate their schema labels, which may often be wider than the actual values in the fields. For the latter case, we automatically use vertical column labels in tables if this makes the column narrower for the purposes of the outline vs. nested table decision. Examples are the Format and Number column header labels in Figure 3(c).

After the minimum widths of table columns have been determined and the decisions to use outline vs. nested table sublayouts at each relation level have been made, additional horizontal space may be available to the right of nested table sublayouts. A separate *table justification* step uses the remaining space to first, for readability, make any previously vertical column headers' labels horizontal. This is done in a greedy order to minimize the number of remaining vertical column labels. Then, any remaining horizontal space is distributed among columns holding variable-length primitive fields, in proportion to their fields' average lengths. The table justification step is illustrated in the transition from Figure 3(c) to Figure 3(d).

Note again that layout styling decisions, such as table column widths or whether to use an outline or a table sublayout for a given subrelation, are made once for each field in the schema of the input data rather than once for each value in the input data. This means, for instance, that in a given layout like Figure 3(c), every instance of the Sections relation will be rendered in the same way (either as a table or an outline), regardless of its actual content in each instance. The rationale is similar to that for making primitive fields in outline mode always the same width.

3.4 Columns in Outline Layouts

While the hybrid layouts layout shown in Figures 3(b), (c), and (d) save a substantial amount of space compared to corresponding outline layouts, they still use horizontal space inefficiently in cases where small primitive fields can not be made part of a table and where only tables with narrow content can be used. In form-style database user interfaces, the traditional solution is to make use of multiple columns of fields. Note that these are a different kind of columns than the columns in nested tables; they allow different fields in an outline layout to be organized in multiple adjacent stacks. We now show how our system can automatically incorporate columns in outline sublayouts with no manual styling required.

We considered various approaches to the problem of introducing columns into outline layouts. Design questions include how to pick the right number of columns to use, how to pick the width of each column, whether to allow certain fields to span multiple columns, and after what fields a new column should be started. We decided to make two simplifying assumptions which seem to work reasonably in practice: (1) the number of columns to use is based solely on the available horizontal width, and (2) every adjacent column has the same width. So for a layout of a width corresponding to a typical letter-size page, for instance, the algorithm would use a two-column layout in the root level outline. However, one complicating issue must still be dealt with: relation fields that could be rendered in a table layout need to be allowed to span multiple columns if necessary. We settled on the following rule: any relation field in an outline is excluded from participating in a set of multiple columns if that would cause it to be rendered as an outline. There is no requirement that the field would actually have to be rendered as a nested table if excluded, but if the excluded field is still rendered as an outline, that outline layout is still subject to the usual heuristics about whether to use columns or not at that next level.

For implementation purposes, the algorithm divides the fields of a relation displayed in an outline layout into multiple *column sets*, each which contains again a list of *columns*, each which contains the fields in the column. To populate column sets, the algorithm iterates over outline fields in the order they appear in the schema, assigning each to the current column set. If an excluded field is encountered, it is assigned to a new column set of its own, and a new current column

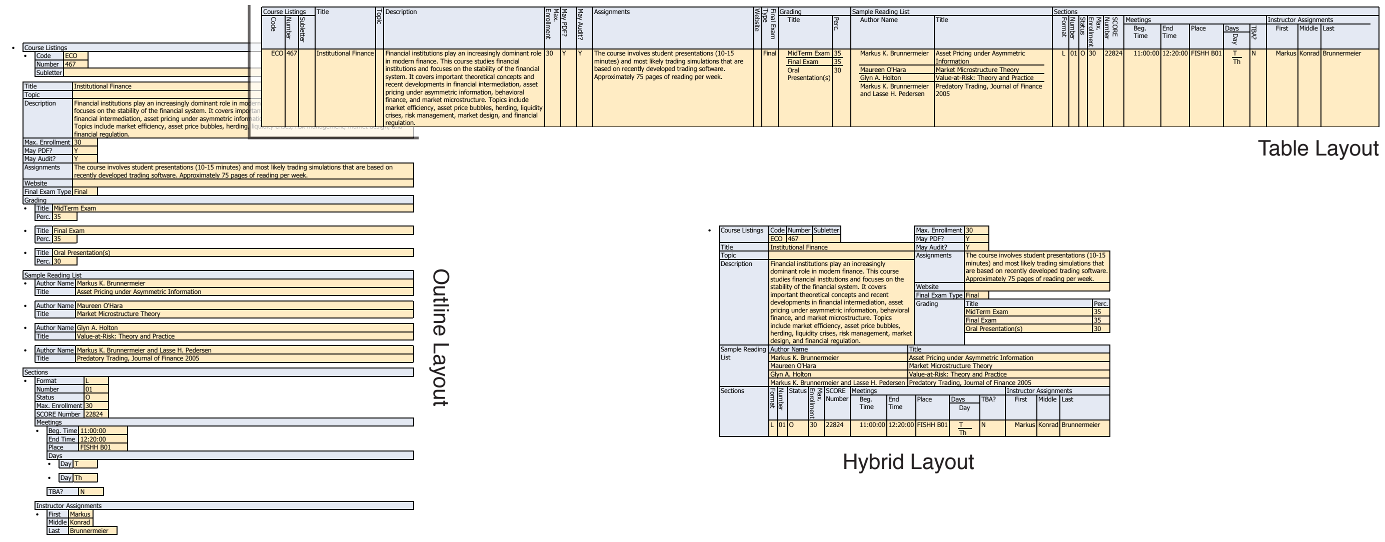


Fig. 4. A comparison between our hybrid outline/table layout and a pure nested table layout and a pure outline layout, for the case of displaying a single tuple with many fields, including relational fields containing other nested tuples. Each layout is showing the same data in its entirety, at the same scale and font size. Outline layouts waste space by concentrating data to the left of the screen and by repeating labels for each value. Table layouts waste space when different fields in the same subrelation require different amounts of vertical space. Table layouts also tend to become very wide, requiring horizontal scrolling if viewed on a screen.

set started. No reordering of fields is done at any point, as the order of fields in the schema is considered significant for presentation purposes. After all fields have been assigned to column sets, each column set partitions its assigned fields into columns. Figure 3(e) shows the final hybrid layout with column support. The example layout has a single outline sublayout with 3 column sets; the first column set has two columns and contains the fields Course Listings through Description in the first column and the fields Max. Enrollment through Grading in the second column. The second and third column sets each have only one column with a single field in each, containing the fields Sample Reading List and Sections, respectively.

The partitioning of columns in a column set, that is, after what fields to start each column, requires a heuristic. Simply giving each column the same number of fields does not work well, since some fields frequently take up more vertical space than others. A better approach is to split the columns in such a way as to minimize the total vertical space consumed; this can be done easily with a dynamic programming routine. However, if this is done independently for each tuple being rendered, two tuples might end up having differently partitioned columns, with different fields starting the columns in each case. This is not ideal for readability. Instead, as before, we make the decision of where to begin new columns only once for the entire layout.

To allow calculation of optimal column partitioning positions on the basis of only aggregate information about the input, we allow our layout generation algorithm to generate *schema-only* versions of sublayouts. For instance, the schema-only layout for the layout in Figure 3(e) is shown in Figure 3(f). In a schema-only layout, variable-length primitive fields are sized according to the average width of the field in the entire dataset. Relation fields are rendered, in outline or table form, with a single placeholder tuple only, but subsequently padded such that the size of the subrelation layout is proportional to the average cardinality of that relation throughout the entire dataset. Thus in Figure 3(f), the Description field is taller than the Title field, and there are about two rows' worth of space allocated in the Sample Reading List table. Optimal column partitioning decisions are then made using these estimated schema-only layouts.

The final class of hybrid layouts produced is considerably more compact than both the outline and the nested table layouts, and can be produced automatically with no manual input. Figure 4 shows a scale comparison between the three layout styles, each showing a single nested tuple from the course catalog example. In this case, the hybrid layout would permit significantly more data to be fit on a single screen without scrolling in either the vertical or the horizontal direction. We can also see that large nested table layouts often waste space whenever two values in the same tuple take up different amounts of vertical space.

3.5 Implementation

To produce the hybrid layouts described, our system makes two passes over the input data while maintaining a *stylesheet* as the only other common data structure. The stylesheet maps schema paths and property types to property values, that is, each node in the schema has one value for each property. See Table 1 for the list of stylesheet properties. Before the algorithm starts, a subset of properties will already have been set as constants, such as the choice of fonts and separator styles. Our algorithm's first pass over the input data is during the Measure phase, which finds the average rendered width of each primitive value and the average cardinality of each relation value. The application of heuristics to set remaining stylesheet properties is then done in a subsequent pass over the schema only, the Auto-Style phase. Finally, the output layout is constructed in the Layout phase, which is the second pass over the input value. The Auto-Style and the Layout phases execute the same code, but with the Auto-Style phase traversing a schema-only version of the data structure used to maintain context, and using the aforementioned heuristics to set undefined stylesheet properties whenever they are encountered. The heuristics for setting stylesheet properties during the Layout phase are described in Table 1.

4 Interactive Features

While our evaluation focuses on the static aspects of our layout management algorithm, our implemented system includes multiple features oriented towards interactive use.[5]

A requirement for many kinds of interactions is to be able to make selections among displayed elements in the layout. Our system supports a spreadsheet-like cursor which can be used to select any *cell* in the layout, where a cell is defined as either a label for a primitive field, a label for a relation field, or a primitive value. Selections can be made either by clicking the mouse or by moving the cursor with keyboard navigation keys (arrow keys, Home/End, and Page Up/Page Down).

[5]To see the interactive features described in this section in action, we recommend watching the demo video accompanying this article.

Table 1. Properties defined, for each schema node, by a stylesheet. P, R, and P R indicate properties applicable to primitive nodes, relation nodes, or both, respectively. We have omitted color- and border-related properties.

	Basic Styling Constants
R	**OutlineBulletStyle.** Bullet type for bulleted tuples in outline sublayouts.
R	**OutlineIndentWidth.** Indentation amount for bulleted tuples in outline sublayouts.
R	**OutlineTupleSpaceHeight.** Vertical space between tuples in outline sublayouts.
R	**TableNestSpaceSideWidth.** Horizontal margin amount for nested tables.
P R	**LabelTextStyle.** Text style for outline labels or table headers.
P	**ValueTextStyle.** Text style for values in outline or table sublayouts.
	Constant Heuristic Parameters
R	**OutlineMaxLabelWidth.** Maximum width of labels in outline sublayouts.
R	**OutlineColumnMinWidth.** Minimum width of each outline column.
P	**OutlineMinValueWidth.** Minimum width of a primitive value in an outline.
P	**OutlineSnapValueWidth.** Multiple to round up to when setting the width of a non-variable primitive value in an outline sublayout.
P	**TableMaxPrimitiveWidth.** Maximum width that can be allocated to a table sublayout column with variable-length primitive values, before table justification.
	Properties Set During Measure Phase
R	**OutlineLabelWidth.** Width of labels in outline sublayouts. Sibling fields all use the same width, which is defined at the parent relation level.
R	**AverageCardinality.** The average number of tuples in each subrelation.
P	**IsVariableLength.** Whether a primitive field holds long strings of variable length.
P	**ValueDefaultWidth.** Average width of primitive values in this field when rendered with ValueTextStyle, or maximum width for non-variable length fields.
	Properties Set During Auto-Style Phase
P R	**StartNewOutlineColumn.** True for the first field in each column of an outline column set. The heuristic partitions columns based on schema-only layouts that use ValueDefaultWidth and AverageCardinality to estimate field sizes.
P R	**StartNewOutlineColumnSet.** True for the first field in each outline column set. Column sets allow some sibling fields to be organized in multiple columns and others not. The heuristic puts a field in its own single-column column set if it would otherwise contain an outline sublayout.
P R	**UseVerticalTableHeader.** Whether to display a label in a table header vertically. The heuristic initially assumes vertical labels for primitive fields if this makes the column narrower, but restores as many horizontal labels as possible when the table is justified.
P R	**TableColumnWidth.** The width of each table column. Primitive columns are ValueDefaultWidth wide before justification; relation columns are the sum of their children plus twice TableNestSpaceSideWidth. Columns are also extended to accomodate their labels, as necessary.
R	**UseTable.** Whether to use an outline or a table sublayout for this relation. A table sublayout is used iff it its width before justification is less than or equal to the available horizontal space.

Because the aforementioned definition of a cell serves to ensure that no two cells can ever overlap, determining the cell to be selected in the case of a mouse click is a simple matter of determining what cell occupies, or is closest to, the point at which the mouse was clicked. For keyboard-based cursor movement, we found the cursor behavior to feel the most natural when the relative motion of the cursor followed the physical location of the cells in the visual layout rather than the logical location of the cells in the schema. To make keyboard cursor movement work well when traversing cells stacked in various nontrivial configurations, we store the cursor state as an actual point on the layout rather than simply as a pointer to the selected cell. This generalizes the behavior seen in existing spreadsheets for instance when moving the cursor across merged cells.

An additional feature is the ability to interactively override the stylesheet settings made by the automatic layout manager. This has the potential to significantly improve readability of output layouts, since the user can use their domain knowledge to decide where labels are superfluous and can be omitted, what fields should serve as titles and thus be emphasized with larger fonts, and such.

Finally, our system supports "frozen" table headers which stay put at the top of the screen for as long as a table is partially visible in the scrolling viewport. This works both for pure table layouts and where table layouts are contained within outline layouts.

Potential future features include multiple selection, collapsible relation fields, editing of data values, spreadsheet-style filtering, and more general-purpose query-building.

5 Evaluation

We evaluated three aspects of our system: runtime performance, the amount of space consumed by generated layouts, and the readability of large layouts as measured by the time taken for human subjects to solve question tasks about the rendered data. As sample datasets, we picked one XML file from each of the 10 categories in the XML Data Repository at the University of Washington[6], except for the Treebank dataset, which is the only one with a recursive schema. We used the "preview" version of each dataset to make sure visualizations would be of a realistic size for human perusal. We also included one dataset from a relational database containing the complete course catalog for a semester at Princeton University. See Table 2.

Table 2. Quantitative statistics related to the size and complexity of datasets referred to in this paper. The *depth* of a primitive value is the number of enclosing relation values that must be traversed to reach the primitive value from the root. The *plural depth* only counts non-singleton enclosing relations.

Dataset	*# Characters in Primitives*	*# Primitives*	*# Tuples*	*Mean Depth*	*Max. Depth*	*Mean Plural Depth*	*Max. Plural Depth*
RelationalPrincetonHuge	828 463	59 887	18 751	2.25	4	2.25	4
RelationalPrinceton	21 811	1 359	419	2.20	4	2.20	4
Auction321Gone	13 679	250	61	2.96	4	1.00	1
SigmodRecord	12 360	953	560	5.01	6	2.51	3
SwissProt	13 559	1 566	809	3.15	5	1.94	3
DBLP	22 012	1 493	310	2.10	3	1.09	2
Mondial	15 046	3 032	1 020	3.63	5	2.48	4
NASA	18 218	854	731	6.16	8	3.43	5
TPCHPart	10 607	901	101	2.00	2	1.00	1
ProteinSequence	11 236	953	494	3.85	6	1.94	3
CoursesReed	4 858	800	200	2.33	3	1.00	1

Table 3. Runtime measurements for each phase of the layout algorithm. Standard error is within 3% in each case.

Dataset	*Pages*	*Algorithm Phase Runtime (s)*			
		Measure	*Auto-Style*	*Layout*	*Paginate*
RelationalPrinceton	12	1.03	0.29	0.83	0.0014
RelationalPrincetonHuge	455	40.16	0.31	42.43	0.0451

5.1 Runtime Performance

For runtime measurements, we ran all phases of the layout algorithm in sequence, and repeated the entire sequence multiple times. The runtimes for individual phases were averaged, less initial dry runs. Resulting runtime statistics for two datasets are shown in Table 3, for 30 runs plus 3 dry runs. The machine used had an Intel Core 2 Duo CPU and 4GB of RAM.

Our two sets of runtime measurements suggest, as expected, that both the Measure and Layout phases run in time roughly proportional to the size of the input data, as measured by the size of the output layouts. Also as expected, the time consumed by the Auto-Style phase does not depend on the size of the input data, as it depends only on the schema and input stylesheet. For the larger RelationalPrincetonHuge dataset, the fact that the Layout phase does not take significantly more time to run than than the Measure phase suggests that the main bottleneck of the Layout phrase is the line breaking code that determines the size of rectangles assigned to display primitive values. Profiling has confirmed this to be the case. For the smaller RelationalPrinceton dataset, the time to perform Auto-Style and Layout for a new width is interactive.

[6]G. Miklau, `http://www.cs.washington.edu/research/xmldatasets/www/repository.html`

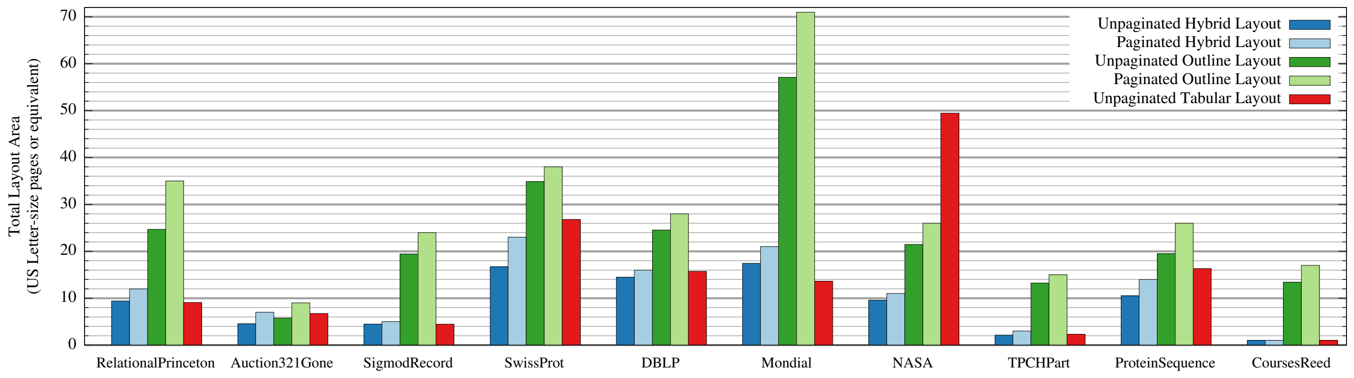


Fig. 5. Total area consumed by layouts of each of the three types. Outline and Hybrid layout widths are constrained to 8 inches, and are shown both with and without pagination.

5.2 Layout Space Efficiency

To evaluate the space efficiency of our layouts, we compared, for each dataset, the area consumed by our own hybrid layout vs. the area consumed by a pure outline layout and a pure nested tabular layout. Each layout was produced by our layout manager, with the latter two using hard-coded values for the stylesheet property that selects whether a table or an outline style is used at any given level in the schema.

Figure 5 compares the total area of each kind of layout for every dataset. The unpaginated area of a layout is that of the smallest rectangle enclosing it. The paginated area, for hybrid and outline layouts, is the number of pages consumed by the layout times the imageable (non-margin) area available on each page. Thus, the latter includes space wasted at the end of each page whenever the pagination algorithm has opted to break the page at an earlier but less awkward place. Since pure tabular views of an entire dataset are generally too wide to fit on a regular letter-size page, the tabular layout is rendered as a single, very large page. In cases where the tabular layout is actually narrow enough to fit on a page, notably the the SigmodRecord, TPCHPart, and CoursesReed datasets, the hybrid layout is nearly identical to the tabular layout, except that the pagination algorithm may be used to break up the hybrid layouts in the vertical direction. The ratio of the area consumed by an unpaginated outline layout to that consumed by an unpaginated hybrid layout is 3.9:1 on average. Similarly for tabular to hybrid layouts, it is 1.6:1 on average.

Looking at the data from Figure 5, we see that the hybrid layout consumes less space than the corresponding outline layout for every dataset, with or without pagination enabled. The difference is greatest, between 4 and 13 times, in the cases where the hybrid layout corresponds to a pure nested tabular layout, namely SigmodRecord, TPCHPart, and CoursesReed. In these cases the schema of the data was flat or almost flat, and so a standard table layout would make very efficient use of the space. In the other cases, the hybrid layouts are about half the size of outline layouts on average. The smallest difference was for the Auction321Gone dataset, where the outline layout was 1.3 times the size of the hybrid layout. In this case the schema was nested in several levels, but contained only singular relations (relations only ever holding a single tuple) beyond the top level, so there were no opportunities for the hybrid layout algorithm to introduce tables into the lower levels of the layout. The modest saving over the outline layout came from the hybrid layout's ability to display data in an outline tuple over two columns. A more significant difference was for the Mondial dataset, where the outline layout was 3.3 times the size of the hybrid layout. Here the hybrid layout made good use of both tuple columns and the ability to render subrelations as tables, and only used one level of outline bullets.

While pure nested tabular layouts cannot be constrained to a page width like the outline and hybrid layouts, they tend to consume less total area than the outline layouts. See again Figure 5. For Auction321Gone, NASA, ProteinSequence, SwissProt, the hybrid layout still consumes between 30% and 80% less space than the tabular layout. This is because nested tables waste large amounts of space whenever a row contains cells of variable heights, such as when one empty and one well-populated subtable are placed horizontally adjacent to each other on the same row of a parent table.

5.3 Readability

To evaluate the readability of our layouts, we conducted a between-subjects online user study using Amazon Mechanical Turk[7] and StudyCaster, a Java-based tool we developed to allow test subjects to stream timestamped recordings of their computer screens to our server with a minimum of effort. In an initial public recruiting stage of the study, workers were offered $0.25 to launch the StudyCaster and solve a chart-making task that required the workers to have Microsoft Excel installed on their machines. In the second and main stage of our experiment, we gradually invited qualified Mechanical Turk workers from the first stage directly to do a second task, worth $3.00. This task contained, for each subject, 9 different two-part questions, each two-part question being based on a separate PDF file with a layout generated from one of our 9 XML sources from the UW XML repository. The questions were a mix requiring the subjects to do both scanning across multiple similar entities (e.g. "What is the Brand number of the product sold in a Jumbo Bag container at a Retail price of less than $950?" or "How many articles were published in Volume 12, Number 3?") and lookup between the attributes and related entities of a single entity (e.g. "What is the name of the person who was responsible for digitizing the earlier work by authors X?"). To reduce potential noise from subjects' varying familiarity with their PDF readers' search feature, the PDF files were rasterized, effectively disabling the feature for everyone. All subjects were given the same questions and datasets in the same order. However, the type of layout provided for each dataset was randomized, with the constraint that each subject would see 3 datasets rendered with each of the 3 kinds of layout types. The order of the layout types was round-robined such that datasets number 1, 4, and 7 would use the same layout types, as would 2, 5, and 8, and as would 3, 6, and 9. Each of the 18 total questions (from 9 two-part questions) would be shown in the StudyCaster pop-up window, which allowed us to measure the exact amount of time the subjects spent viewing, and hence presumably spent working on, each question. The StudyCaster software also allowed us to further limit timings to when workers had the correct PDF file in focus in their PDF reader to answer the currently shown question (sampled from the Win32 API at 5Hz), and to exclude time idle more than 5 seconds from keyboard or mouse activity (same). The idle time rule was used to decrease noise from workers taking a break from the computer while having a question open on the screen.

Our user study yielded data from 27 subjects. An additional 6 subjects completed the study, but were not included in the dataset due to

[7]`http://www.mturk.com`

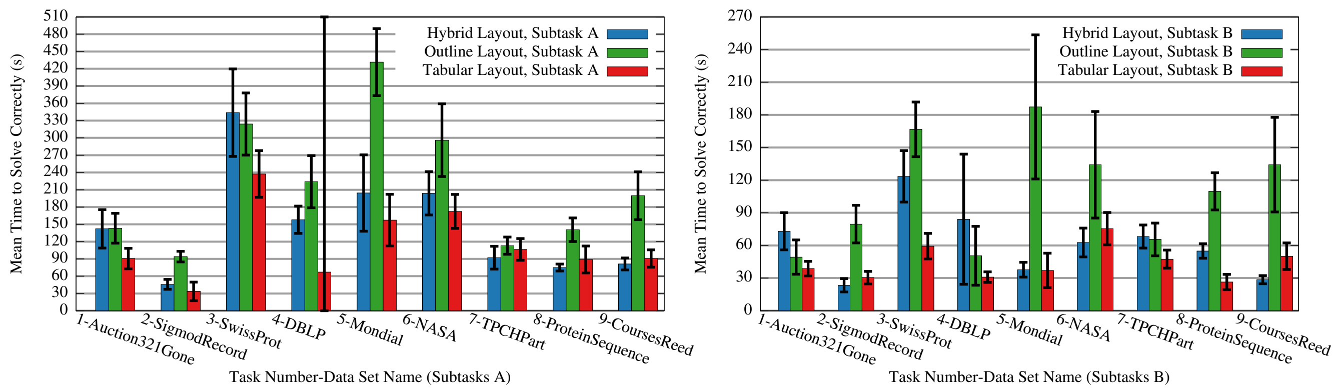


Fig. 6. The mean time to solve each task in the user study, grouped by the kind of layouts that were used to generate the PDFs subjects used to solve the task. The error bars show the Standard Error of the Mean. Subtasks B were given to subjects directly after Subtasks A, and were in each case identical to Subtask A except for a small emphasized change in the question text.

technical problems uploading their screen recordings and timing data. The 18 task questions were answered correctly by 88% of subjects on average, with too limited variation to draw conclusions about possible impact of layout type on correctness. Figure 6 shows the average time taken to complete each subtask for each layout type. Each subject's timing is included in the average for each subtask only if the subject answered that question correctly.

To test our user study for statistical significance, we considered the timing data from each of the 18 subtasks separately. In each case, we thus had 3 sets of measurements of the time taken to solve the task correctly, one for each kind of layout presented to the user. We first ran Levene's test to confirm that our experiment design conformed to ANOVA's assumption of homogeneous variances between the 3 measurement populations in each case. In only 1 of the 18 cases (Task 5B) was Levene's test significant (indicating non-homogeneous variances) with $p < 0.05$, suggesting that this is a reasonable assumption. We thus proceeded to use ANOVA to analyze the results, with a Tukey Honest Significance Test (HSD) as a follow-up test in cases where the ANOVA was significant. Since we are doing 18 tests, we should require $p < 0.05/18$ for strictly significant ANOVAs, as per the Bonferroni correction. There are two significant results to this confidence level, for tasks 2A and 8B. For the purposes of discussing results, however, we have done the Tukey HSD follow-up test for all tasks with ANOVAs up to $p < 0.05$. This allows us to list all the most significantly different pairs of timings between different layout types, as shown together with the relevant p-values in Table 4. Note that we can expect about one $(18 * 0.05)$ of the borderline-significant ANOVAs in this table to be due to chance.

Looking at the follow-up tests from Table 4, we can see no significant differences in the task completion times between the tabular vs. the hybrid layouts. We do however see consistent differences both between the outline and the hybrid layouts as well as between the outline and the tabular layouts. These differences are present in both subtasks of several questions, suggesting that they are relevant both when users are first learning to do a task and when they immediately after do a second similarly structured task. In terms of relative task completion times, it is clear that both the hybrid and the tabular layouts outperform the outline layout, in both cases being completed 2.9 times faster on average for the tasks listed in Table 4. This is consistent with the hypothesis that larger layouts are harder to use because more scrolling is needed to look through the same data.

The relatively high performance of the nested tabular layouts suggests that for the kinds of large datasets we had our users work with, the very regular structure of the tabular layout can outweigh its disadvantages of taking up more space and requiring both horizontal and vertical scrolling. However, the tabular layout would be a poor choice for smaller datasets, such as the common database application requirement of showing the details of a single entity with all its attributes and

Table 4. Summary of statistical tests run on the dataset from Figure 6. Only tasks for which the ANOVA yielded $p<0.05$ are shown. For Tukey HSD pairs with $p<0.05$, we also show the relative differences in average task completion times.

Task	*Levene* (hetero-scedasticity)	*ANOVA*	*Tukey HSD* (follow-up to find differences between pairs)			*Mean-Time-to-Solve Ratio*	
			Outline v. Hybrid	Outline v. Tabular	Tabular v. Hybrid	Outline: Hybrid	Outline: Tabular
#	*p*	*p*	*p*	*p*	*p*		
2A	0.5312	0.0012	**0.0019**	**0.0169**	0.8006	2.05	2.80
2B	0.0627	0.0034	**0.0036**	**0.0448**	0.9279	3.40	2.62
3B	0.2889	0.0032	0.3047	**0.0023**	0.0962		2.81
5A	0.7976	0.0072	**0.0200**	**0.0142**	0.8521	2.11	2.74
5B	0.0389	0.0174	**0.0251**	0.0532	0.9999	4.97	
8A	0.1099	0.0101	**0.0092**	0.0996	0.8201	1.88	
8B	0.3409	0.0004	**0.0048**	**0.0008**	0.3129	2.00	4.16
9A	0.4104	0.0034	**0.0050**	**0.0119**	0.9553	2.46	2.20
9B	0.1812	0.0107	**0.0108**	0.0530	0.7919	4.69	

related subentities. Future evaluation could focus on layout performance on smaller, form-size datasets.

6 Conclusion

We have presented a layout management algorithm that automates the display of structured hierarchical data using the traditional visual idioms of hand-designed database UIs: tables, multi-column forms, and outline-style indented lists. By default, the widths of generated layouts are constrained so that only vertical scrolling may be necessary to view the data in its entirety. Our stylesheet system is further designed such that two input values mapping to the same schema node will always be styled in a similar way. Our hybrid layouts are 3.9 and 1.6 times more compact on average than outline layouts and horizontally unconstrained table layouts, respectively, and are as readable as table layouts even for large datasets. We believe that our system can function as a single output system for most of the data views commonly required in domain-specific database applications, whether they be large tables required to display information about many entities at once or form views that must display many details of a single entity compactly without scrolling.

Acknowledgments

We would like to thank Michael S. Bernstein, Christian Bernt Haakonsen, Jeffrey Heer, and anonymous reviewers past and present for their helpful input during various stages of this project.

REFERENCES

[1] E. Bakke and E. Benson. The schema-independent database UI: A proposed holy grail and some suggestions. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, 2011.

[2] E. Bakke, D. R. Karger, and R. C. Miller. A spreadsheet-based user interface for managing plural relationships in structured data. In *Proceedings of the 29th International Conference on Human Factors in Computing Systems (CHI '11)*, New York, NY, USA, 2011. ACM.

[3] B. B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Transactions on Graphics (TOG)*, 21(4):833–854, October 2002.

[4] F. Bodart, A.-M. Hennebert, J.-M. Leheureux, and J. Vanderdonckt. Towards a dynamic strategy for computer-aided visual placement. In *Proceedings of the workshop on Advanced Visual Interfaces (AVI '94)*, pages 78–87, New York, NY, USA, 1994. ACM.

[5] E. H.-H. Chi, P. Barry, J. Riedl, and J. Konstan. A spreadsheet approach to information visualization. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '97)*, pages 17–24, 1997.

[6] R. Chimera and B. Shneiderman. An exploratory evaluation of three interfaces for browsing large hierarchical tables of contents. *ACM Transactions on Information Systems (TOIS)*, 12:383–406, October 1994.

[7] P. Chmelar, R. Hernych, and D. Kubicek. Interactive visualization of data-oriented XML documents. In T. Sobh, editor, *Advances in Computer and Information Sciences and Engineering*, pages 390–393. Springer Netherlands, 2008.

[8] M. Fernández, Y. Kadiyska, D. Suciu, A. Morishima, and W.-C. Tan. Silkroute: A framework for publishing relational data in XML. *ACM Transactions on Database Systems (TODS)*, 27(4):438–493, 2002.

[9] K. Gajos and D. S. Weld. SUPPLE: Automatically generating user interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI '04)*, pages 93–100, New York, NY, USA, 2004. ACM.

[10] M. Ghoniem, J. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '04)*, pages 17–24, 2004.

[11] M. Graham and J. Kennedy. A survey of multiple tree visualisation. *Information Visualization*, 9(4):235–252, 2010.

[12] C. Jacobs, W. Li, E. Schrier, D. Bargeron, and D. Salesin. Adaptive grid-based document layout. *ACM Transactions on Graphics (TOG)*, 22(3):838–847, July 2003.

[13] J. Jelinek and P. Slavik. XML visualization using tree rewriting. In *Proceedings of the 20th Spring Conference on Computer Graphics (SCCG '04)*, pages 65–72, New York, NY, USA, 2004. ACM.

[14] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd Conference on Visualization (VIS '91)*, pages 284–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[15] H. Kitagawa, M. Gotoh, S. Misaki, and M. Azuma. Form document management system SPECDOQ—its architecture and implementation. In *Proceedings of the second ACM-SIGOA conference on Office Information Systems (COCS '84)*, pages 132–142, New York, NY, USA, 1984. ACM.

[16] K. Kowalzcykowski, A. Deutsch, K. W. Ong, Y. Papakonstantinou, K. K. Zhao, and M. Petropoulos. Do-It-Yourself database-driven web applications. In *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research (CIDR '09)*, 2009. A newer version of this paper may be found at `http://db.ucsd.edu/pubsFileFolder/319.pdf`.

[17] D. Król, J. Oleksy, M. Podyma, and B. Trawiński. The analysis of reporting tools for a cadastre information system. In *Proceedings of the 9th International Conference on Business Information Systems (BIS '06)*, 2006.

[18] M. Levene. *The Nested Universal Relation Database Model*, volume 595 of *Lecture Notes in Computer Science*. Springer Berlin/Heidelberg, 1992.

[19] J. D. Mackinlay, P. Hanrahan, and C. Stolte. Show Me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 13(6):1137–1144, November/December 2007.

[20] R. Mitchell, D. Day, and L. Hirschman. Case study: Fishing for information on the Internet. In *Proceedings of the First Information Visualization Symposium (InfoVis '95)*, pages 105–111, Los Alamitos, CA, USA, October 1995. IEEE Computer Press.

[21] Q. V. Nguyen and M. L. Huang. EncCon: An approach to constructing interactive visualization of large hierarchical data. *Information Visualization*, 4(1):1–21, 2005.

[22] R. Rao and S. K. Card. The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '94)*, pages 318–322, New York, NY, USA, 1994. ACM.

[23] G. Robertson, K. Cameron, M. Czerwinski, and D. Robbins. Polyarchy visualization: visualizing multiple intersecting hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '02)*, pages 423–430, New York, NY, USA, 2002. ACM.

[24] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, page 336, Los Alamitos, CA, USA, 1996. IEEE Computer Society.

[25] M. Spenke, C. Beilken, and T. Berlage. Focus: The interactive table for product comparison and selection. In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology (UIST '96)*, pages 41–50, New York, NY, USA, 1996. ACM.

[26] K. Tajima and K. Ohnishi. Browsing large HTML tables on small screens. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology (UIST '08)*, pages 259–268, New York, NY, USA, 2008. ACM.

[27] L. Wegner, S. Thelemann, J. Thamm, D. Wilke, and S. Wilke. Navigational exploration and declarative queries in a prototype for visual information systems. In C. Leung, editor, *Visual Information Systems*, volume 1306 of *Lecture Notes in Computer Science*, pages 199–218. Springer Berlin/Heidelberg, 1997.

[28] F. Yang, N. Gupta, C. Botev, E. F. Churchill, G. Levchenko, and J. Shanmugasundaram. WYSIWYG development of data driven web applications. *Proceedings of the VLDB Endowment*, 1(1):163–175, 2008.

[29] S. B. Yao, A. R. Hevner, Z. Shi, and D. Luo. FORMANAGER: An office forms management system. *ACM Transactions on Information Systems (TOIS)*, 2(3):235–262, August 1984.

[30] S. Zhao, M. J. McGuffin, and M. H. Chignell. Elastic hierarchies: Combining treemaps and node-link diagrams. In *IEEE Symposium on Information Visualization (InfoVis '05)*, pages 57–64, October 2005.

[31] C. Ziemkiewicz, R. J. Crouser, A. R. Yauilla, S. L. Su, W. Ribarsky, and R. Chang. How locus of control influences compatibility with visualization style. In *2011 IEEE Conference on Visual Analytics Science and Technology (VAST '11)*, pages 81–90, October 2011.