Information Visualization (2003) 2, 258–269

www.palgrave-journals.com/ivs

A model and software system for coordinated and multiple views in exploratory visualization

Nadia Boukhelifa[1]
Peter J. Rodgers[1]

[1]*Computing Laboratory, University of Kent, U.K.*

Correspondence:
Nadia Boukhelifa and Peter J. Rodgers, Computing Laboratory, University of Kent, Canterbury CT2 7NF, U.K.
Tel: +44 1227 827590/+44 1227 827913;
Fax: +44 1227 762811/+44 1227 762811.
E-mail: n.boukhelifa@kent.ac.uk(NB), p.j.rodgers@kent.ac.uk(PJR)

Received: 1 August 2003
Revised: 19 September 2003
Accepted: 24 September 2003

Abstract
This paper describes a model for expressing coordination in multiple view visualization systems. We present the model and describe a prototype implementation that illustrates the features of the model. Current visualization systems tend to have an informal and inconsistent approach to coordination. Our model takes a formal approach to describing widely used coordination concepts. The model is based on views sharing abstract objects such as the visualization parameters of the dataflow model. Additionaly, this paper describes how current coordinations in exploratory visualization work and how novel coordinations can be constructed using our model.
Information Visualization (2003) **2,** 258–269. doi:10.1057/palgrave.ivs.9500057

Keywords: Multiple views; exploratory visualization; coordination; coordination objects; coupling; CViews

Introduction

Coordination is a powerful tool for interacting with multiple views when performing exploratory visualization tasks. For instance, selecting a group of data items in one view in coordination with the selection of the same items in another can reveal new relationships, such as distribution, grouping or subordination within these items, which might otherwise remain hidden. However, current visualization systems using coordination tend to have an *ad hoc* and informal approach. We present a model for coordination of multiple views that formalises coordination concepts in exploratory visualization, clarifies existing coordination systems and encourages the identification of new coordination combinations.

The model is designed to be unbiased towards any particular data, navigation or communication paradigm, and hence can be described as a generic model for coordination in exploratory visualization. This model differs from general visualization models in exclusively detailing coordination by describing a mechanism for linkage that relies on the sharing of visualization parameters. It defines coordination rudiments drawing on the findings of the interdisciplinary study of coordination and also the existing research in exploratory visualization. Furthermore, we describe the visualization process using the dataflow visualization model and our coordination model allows for the description of types of coordination that can occur in a system by detailing the sections of the dataflow model that the coordinations affect in each view.

We have developed a software system to demonstrate our model in use. The prototype, called CViews, implements coordinated views of the Surface and Underlying Structural Analysis of Natural English Corpus (SUSANNE). Currently, there are three types of views supported by our prototype. However, the implementation can easily scale to support additional views and coordination types, as the views do not need

knowledge of each other, instead all communication is through coordination objects, allowing 'plug and play' of views that support the required interaction features. Our experience with CViews shows that having a coordination model facilitates the implementation of a multiple view visualization system since the inter-relationships between views can be flexible.

The work presented in this paper is a substantially updated and revised version of the paper presented at CMV2003,[1] in particular the implementation is entirely new. This paper is divided into the following sections: related work; discussion of the salient features of coordination in exploratory visualization; the new model; discussion about the relationship of the model to current coordinated visualization systems; model implementation; and future work and conclusions.

Related work

There are many aspects of computing for which coordination is useful. Coordination increases user performance, allows discovery of unforeseen relationships and leads to unification of the desktop.[2] As a result, coordination has been the subject of study of many separate disciplines, but only recently have researchers such as Olson *et al*[3] appreciated the advantages of interdisciplinary viewpoints.

Interdisciplinary view of coordination

Coordination in exploratory visualization may benefit from observing how other disciplines address issues of inter-related information, multitasking and the interdependency and coupling of components. Each of these disciplines has its own architectures, models and protocols; some may be based on sharing memory and other resources, others on managing constraints or propagating data values or data parameters. Coordination in each of these fields is portrayed in a different context.

In the context of the multiple ontologies research, heterogeneous information services may be coordinated using an approximate ontology translation framework. For instance, the research done by Akahani *et al*[4] provides a representation formalism to define the mapping between different ontolgoies as well as the translation and approximation techniques. In this case the inter-ontology mapping of data attempts data integration by identifying semantically corresponding terms of different source ontologies.[5]

In context of middleware provision for cooperative work, Ciancarini *et al*[6] examine a suitable middleware for coordinating distributed active document-centric applications. This middleware is a software layer that abstracts from the heterogeneous characteristics of different architectures, operating systems, programming languages and networks in distributed systems. The main responsibility of the coordination middleware is one of data communication. Furthermore, middleware coordination often allows multiple views to occur in client-server architecture.

Another approach to coordination is the observation pattern as an ontology and a formal framework, as proposed by Viroli *et al*.[7] They write 'in general, *observation* occurs when a system *o [observer]* is interested in some information made available by a system *s [source]*. Typically, … *s* is modeling some portion of the world *o* is interested in, and providing *o* with some knowledge about it, as well as some mechanisms to access it'. The request–reply strategy of interaction in this paradigm works as a model-view-controller (MVC) pattern.

Although much of the interdisciplinary research on coordination is informal; for instance, the ontology translation uses informal mapping rules between the various ontologies.[4] The interdisciplinary study of coordination presents ideas that can be used for modeling multiple view exploratory visualization and entities, such as objects, events, processes, functions, agents and ontologies all may be coordinated. There is much communality between concepts from a variety of fields and a strong case for transference of ideas.

Coordination for exploratory visualization

One way of exploring data is through interactive visualization. This form of exploration is important because it enables the user to change the viewing parameters in one realization actively, which is the essence of visual information seeking. On that, Ahlberg and Shneiderman[8] state: the emphasis is on 'rapid filtering, … progressive refinement of search parameters, continuous reformulation of goals and visual scanning to identify results'. Within such environments, the user can subsequently perform one operation a number of times in various views. However, there is an obvious benefit in simultaneously coordinating the operation for the multiple views.

There are many ways through which we can describe coordination. Olson *et al*[3] define coordination as 'composing purposeful actions into larger purposeful wholes', where 'the additional information processing performed when multiple, and connected actors pursue goals that a single actor [or indeed the multiple actors working separately] pursuing the same goals would not perform'. We emphasize the point that the whole is greater than the sum of its components.

A coordination model improves understanding, and allows effective development and qualitative evaluation of systems that incorporate coordination. The field of visualization is full of overloaded terms and suffers from inconsistencies. Moreover, visualizations often are based on different models. Coordinating different visualizations requires a mechanism, which allows interoperability between these differing models. A coordination model should also provide guidelines for using multiple views such as those suggested by Baldonado and Woodruff.[9]

Current coordination models

Coordination is implemented in various visualization systems such as Xmdvtool,[10] Spotfire,[11] VIP, LinkWinds[12] and Visage.[13] Most common is the linked overview-detail views that is highly utilized in geovisualization, for example, see Andrienko and Andrienko.[14] This fits with North and Shneiderman[2] dual *selection* and *navigation* motivation for coordination.

There are only a few systems that design models for coordination, for example Data Exploration and Visualization (DEVise)[15] and GeoVISTA Studio.[16] However, like Pattison and Philips,[17] we believe in a wider view of coordination, which potentially may coordinate any aspect, such as data preparation, averaging, clustering and moving window positions. Recently, two models have been proposed: the Snap conceptual model[18] and the View Coordination Architecture.[17]

Snap The Snap conceptual model takes a data-centric approach to coordination. Relational database components are tightly coupled such that an interaction with one component results in changes to other components. Snap utilizes the concept of database design to promote better visual exploration. It provides a mechanism for constructing coordinations without the need for programming. In addition, new types of coordination are introduced, such as the compound join and the multiple alternative joins.[18]

In some respects Snap resembles our model; for instance, its architecture is event based and coordination is built from action associations. Snap also recognizes the need for a middleware party to ensure coordination operation and for a translation mechanism when dealing with heterogeneous information sources. However, our model handles coordination from a more general viewpoint and takes in consideration exploratory visualization needs for rich and varied user interactions. Furthermore, we are interested in modeling representation-oriented coordinations as well as data-centric coordinations.

View coordination architecture for information visualization Pattison *et al*[17] present an architecture for the implementation of generic view coordination in the MVC pattern. The proposed framework separates between the specification and implementation of mapping between data model to view model.

Coordination is managed by a new component (called *coordination*). Bidirectional coordinations can be achieved through directional coordination between presentation components, view model components or specification components. The more components there are and the more links exist between them, the more complex the implementation and debugging becomes, especially when linking different components.[17] Thus, to encourage reuse, presentation, content and the coordination itself should be – as far as possible – disparate and independent.

Rather than concentrating on the implementation architecture our work focuses on a layered approach based on the dataflow model. Like Pattison, we use an MVC fundamental design; however, we utilize multiple components and different facets of coordination.

Facets of coordination

From the related work and broadness of the interdisciplinary viewpoint, we see that coordination conjures some interesting challenges, such as relevance, design and visual depiction of each coordination as well as considering what and how to coordinate.

Coordination challenges and opportunities

First, due to the multiform nature of the multiple views, actions in one view cannot always be directly applied to other views. For example, it may be useful to rotate two three-dimensional views coincidentally, but if each uses a different mathematical projection then a translation needs to occur that converts user interactions in one view to a suitable format for the other. However, some coordinations that may be possible to achieve, may in fact not be useful; and yet others may be impossible to realize. However, at this abstract level it is possible to rely on the user to make such judgment of the usefulness of a particular coordination. Indeed, there is the whole question of how the system is implemented and whether the coordinations automatically occur or are created by user requests.[19]

Second, there are design and user-interface questions that a designer may wish to pose. For example, if the multiple views represent a visual history then is it feasible or relevant to coordinate between past variances of the exploration? Moreover, is everything coordinated or are aspects of a few windows coordinated (and if so who decides on what is coordinated – the user or the system?[20]) For instance, if may be beneficial to only coordinate views that are classified within the same group (the notion of Render Groups[21]).

Third, how does the system visually represent and notify to the user what is currently being coordinated (e.g. visual methods such as used by the spiral calendar,[22] or by the implicit laying out of modules in Waltz[21]). Many issues in visualization, such as synchronization, correlation of visual or non-visual information, occlusion, view explosion and multitasking could be more approachable through a coordination model.

Coordination in use – two examples

In order to develop some rudiments of coordination in EV, we investigate the use of coordination in a current tool (LinkWinds) and how coordination may be thought of as analogous to program variables.

LinkWinds uses a data-linking paradigm for coordination, which is comparable to the spreadsheet concept where cells are related to each other using a formula and changing the formula in one cell recalculates the value of the linked cell.[12] The basic entities that are coordinated

in LinkWinds are objects shown at the windowing level as either data, control or display objects. Objects of the same type sit in the same window making an object view. The general purpose of coordination is to detect possible relationships in data.

LinkWinds allows one-to-many links; for example, a slider broadcasts messages to all objects it is linked to when its value is changed. The user performs linking as well as unlinking interactively. In addition, there exist some constraints on coordination. For instance, data must be put into empty windows and messages are passed only between objects that are already linked. There is also a message-passing protocol that handles inter- and intraobject messages. The effect coordination causes on the user interface is the emanating flow between linked objects.

Program variables may be thought as analogous to coordination objects; for example, variables may be used in multiple places and accessed by reference, they must be instantiated, and they each have a type (if they are of a wrong type then they may be *cast* – either by default or explicitly). There are also notions of global and local scope.

Rudiments of coordination

Taking aspects from this analogy, the LinkWinds example, aforementioned challenges and other coordination tools we categorize the rudiments.

Coordination entities: This details what is actually being coordinated, such as aspects of the actual window, view, data, record, tuple, attribute, parameter, process, event, function, graphic or time.

Type: The type of the coordination determines the method by which the entities are linked. For example, simple coordination (such as rotation or transformations) may be implemented using primitive types (float, integer, etc.), while others may be more complex data structures. Translation (casting) may be required if the entities utilize different types. The types may also determine directionality of the links (unidirectional or bidirectional). For example, IRIS Explorer allows parameters to be coordinated but the events flow one way (as it disallows simultaneously connecting the reverse to inhibit circular event explosions taking place).

Chronology (lifetime and scheduling): How long entities are coordinated is governed by its lifetime (which is also known as the persistence of the coordination). It may be coordinated permanently, for a given action, or determined by some scope. Moreover, the coordination may be synchronous, asynchronous, reactive or proactive. For example, it may be useful to rotate the view of a fast and a slow renderer coincidentally; one solution is that the time-consuming one could update at a slower rate by taking every n events from a coordination event queue, or merely notified at every n seconds.

Scope: Scope determines both the global/local connection and the lifetime of the links; a global scope would mean that any entity (wherever and whatever it is) could be connected; whereas some links may be restricted to being only used in a local area. For example, the user may set up a group where simply adding a new member to the group automatically coordinates it to each of the others in the group (which is commonly known as a Render Group). Moreover, the scope also may restrict the lifetime.

Granularity of links: Many entities may be connected together via various links. Granularity determines: the number of entities in one coordination $\{2 \ldots n\}$, number of views in one coordination $\{1 \ldots n\}$, and number of links an entity contributes to coordination $\{0 \ldots n\}$.

Initialization: Initialization determines how the coordination is created. For example, it may be automatic (such as using a Render Group) or user specific, scheduled in some fashion, etc. Moreover, the user may need to connect entity A to B explicitly, for every type by (say) connecting ports from modules A to B; alternatively, the user may only need to drag and drop the whole module into a Render Group to instantiate and link everything in modules A–B.

Updating: Coordinated views require that the information is dynamically up-to-date. However, there may be conflicting uses especially if, for example, the multiple views represent a visual history. Commonly, in a dataflow paradigm, the downstream modules always reflect the information upstream. However, it may be prudent that sometimes some views become out of synchronization: such that they reflect a previous time in history. The displays may be updated by various means, such as eager or greedy update, lazy update or user initiated.

Realization (link realization, user control): How is the expression of coordination conveyed to the user? It may be that explicit lines are used (such as the Spiral Calendar[22]) or via some formal layout mechanism.[21] Moreover, how does the user control the information to be linked, do they use direct manipulation or indirect means via (say) dynamic sliders?

The model

The challenge is to develop a model that addresses the coordination design issues mentioned above without bias towards a particular data, navigation or communication paradigm. Effectively, it should be flexible, adoptable, extensible and foster better visual exploration.

The model should allow visualization designers to specify existing and novel coordinations formally in multiple view exploratory visualization and so facilitate early testing of the proposed coordination designs before they are implemented by programmers or constructed visually by the user.

Abstract model for coordination

The model we define includes 'coordination objects' that manage combinations of *entities* (e.g. parameters) that control aspects of the linked views. A single coordination object is associated with each separate coordination in the system. A view is said to be coordinated if it shares a

common coordination object. All the coordination objects for a visualization system are held in a 'coordination space' as shown in Figure 1. This is similar to the middleware layer component we mentioned in the Interdisciplinary view on coordination.

The views that are being coordinated need to define a translation function (for instance, $f_{1,1}$ and $f_{2,1}$ for Coordination Object 1 as shown in Figure 1) from the shared coordination object to the linked view parameters.

The views must also register to be informed of a notify event when a coordination object is changed. If an event occurs, which might typically be a user initiated action in one of the linked views, it alters the coordination object, which sends a notify to all the linked views registered. Registration may depend on a given *scope*. Then, those views that were notified of a change consequently use the information provided by the coordination object via their translation function to generate the new view.

A single coordination object is considered to be present for each *type* of coordination in the system. So that if multiple views define coordination for both corotation and brushing, this will be represented by two separate coordination objects: a rotation coordination object for corotation and a selection coordination object for brushing.

In the previous example, brushing uses the same visualization parameter for both views, selecting in one view results in selecting in another view and the same applies to corotation. However, a coordination object may hold more than one visualization parameter. For instance, one action in view V_1 may be linked to two actions in view V_2. In this case, we have three parameters in the coordination object.

An advantage of this model is its dynamic nature, as views may be added and removed without other views that also access the same coordination object necessarily having knowledge of this activity. Importantly, views do not need to know about other views in the coordination.

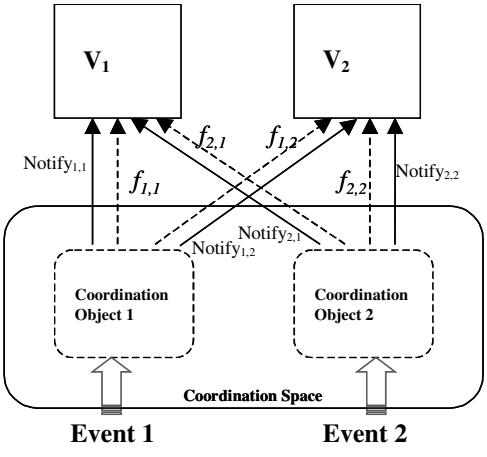

Figure 1 Abstract model for coordination in exploratory visualization. This diagram shows two different coordinations between two views.

A layered approach to coordination

The views themselves are a result of parameter changes to the visualization process; an interaction or exploration would generate a new view, likewise viewing the same data by a different display technique (multiform) would provide a new view. These different instances may be displayed in different windows, overlaid into the same window or in fact replace the current window (replication, overlay, replacement, respectively[20]).

Often in exploratory visualization, visual correlation is seen as the focus of coordination, which tends to limit the techniques to brushing and navigational slaving. However, coordination may be understood in a wider context and occur on any variable or data at any level within the whole visualization process.

Consider the dataflow model[23] (which is used in many systems such AVS, IRIS Explorer, Amira, Data Explorer DX to describe the whole visualization process); the data is enhanced or enriched in some form, then mapped into an abstract visualization object (AVO) that can be rendered into an image (Figure 2). Multiple views are generated by splitting the dataflow at any stage of the pipeline (generating a fan-out). Aspects of the replicated modules may be readily associated with engender coordination. In addition to the traditional dataflow model, aspects may also be coordinated at the viewport transform (see Figure 3). Such coordinated transform operations might include simultaneously rotating viewpoints or altering projections. Moreover, coordination may occur at Window level, for example, moving or closing windows concurrently. Incidentally, the same visualization process can also be described using the data state model as it is equivalent to the dataflow model.[24]

One criticism directed at the dataflow metaphor concerns the granularity of its processes. The dataflow paradigm describes a coarse model for the visualization process; for instance, the entire graphics field is encapsulated in one process 'rendering'.[25] In addition, a single process represents the mapping stage, which is the essence of information visualization whereby information objects are mapped to visual objects.

Moreover, the map and render stages of the dataflow model are tightly associated in many application areas (such as 'information visualization') and thus often treated as one process.

Therefore, theoretically any process could be coordinated with anything (relevant translations applied); however, this may not be feasible or relevant. Our model

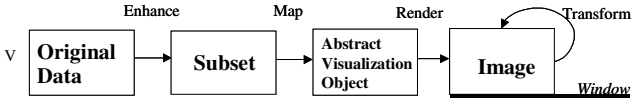

Figure 2 The data flow paradigm.

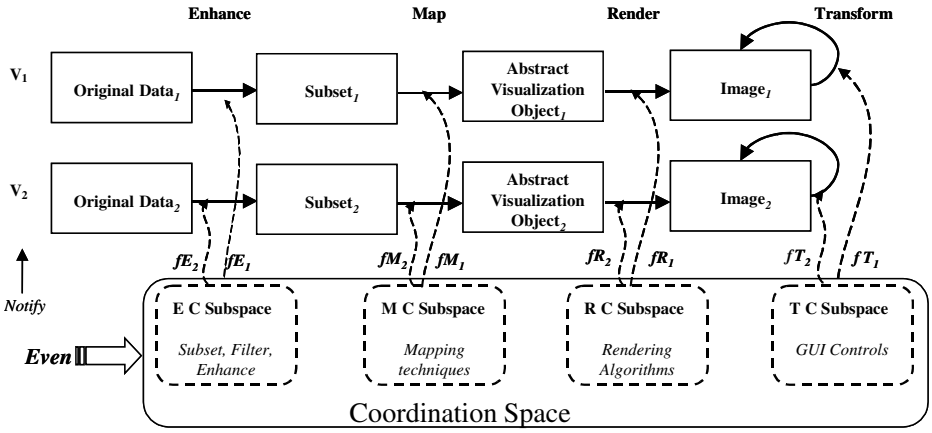

Figure 3 Coordination model for exploratory visualization showing subspaces.

allows a coordination to apply to a group of different process layers within the dataflow paradigm. However, we note that typically coordination tends to occur within a particular layer.

Components of the model

Our model is divided into four components. They are the basic visualization processes and states, the coordination space, the events and the translation and notification functions.

Basic visualization processes and states These are enhance, map, render and transform processes of the dataflow pipeline as explained in section on A layered approach to coordination. Data are transformed in the dataflow paradigm from the raw data to an image, firstly by enhancement to produce a derived data set. This is then mapped onto some geometry described by the AVO. The final data state is the rendered image. These provide the coordination *entities* (see section on [Rudiments of coordination]).

Coordination space

Each visualization system has one coordination space that holds various coordination objects. Coordination objects connect events and a number of linked coordinated views. In interactive exploratory visualization the event is often user initiated in one of the views.

One view may be associated with many coordination objects in a coordination space since a view may play various roles in different coordinations: it might be part of a focus-and-overview coordination, as well as a rotation coordination (see *granularity* in Rudiments of coordination).

The abstract parameters in a coordination object might be simple thresholds, coordinates of mouse clicks, or more complex notions, such as modifications required to color maps or rendering algorithms (the concept of *type* in Rudiments of coordination). In the simple cases, the abstract parameters could easily be the actual parameters used by a view, such as a bounding box when filtering data in the Enhance section of the visualization pipeline. Even such simple parameters are not suitable in a raw form for all views, as some views may measure screen distances in different units, or from different origins, so translation functions may still need to be defined, for example, to convert measurements between inches and centimeters.

This demonstrates the need for a more neutral format for storing these abstract parameters in some cases. Naturally, we might want to store the shared parameters in the same format as the event-generating view parameter format. Hence, there will be no translation needed between the coordination space and that particular view. However, there might be a more suitable format for storing this parameter that suits more than one view, which is not the current format of any view parameter, hence, in this case, all views would need to define a non-trivial translation function on the abstract parameter.

The abstract parameters stored in the coordination space can be grouped into four varieties of coordination subspaces (see Figure 3), since we describe the visualization process in terms of four processes: the enhance coordination subspace, the mapping coordination subspace, the rendering coordination subspace and the transform coordination subspace. However, in many situations the division between these subspaces is less rigid and coordination objects may include various parameters from each of the coordination subspaces.

Events In this model, abstract parameters held in coordination objects are changed by events. Events can be generated by explicit user actions or automatically, by for example continuous analysis of the input data. Some user actions, such as selection with a mouse, are

connected to a particular view, whereas others, such as keystrokes for altering parameters, are not.

Events modify abstract parameters, so they must be aware of the nature of the format and extent of the abstract parameters in the coordination objects. Where events are generated in particular views, the event-generating view will only be updated in the course of the event-notify cycle along with the other views.

As events associated with views are usually indirect in any case, and typically the cycle defined in this model is fairly immediate, this system seems an elegant manner of allowing multiple events, generated by any view linked to the coordination object, to modify the abstract parameters.

While in this model we do not consider the time lag to be significant between the occurrence of the event and the notification event, this model should be adaptable to more critical real time visualization applications where large numbers of multiple events are occurring.

There may be many events associated with a particular coordination object, so allowing the modeling of various types of coordination. A simple master–slave relationship, where all the actions are in one view and which simply reflected in other linked views might have only one event inputting into the coordination object, from the master view. On the other hand, where the coordination allows input from any or all of a group of similar views, the number of events may be at least the number of linked views.

Translation and notification functions A translation function takes the abstract parameters in the coordination object and converts them to view parameters, which are used in the visualization process to produce the final image. Each view registered with the coordination object has one such function. The result of the translation function at the view level is the replacement of current view parameters affecting the operations in the dataflow pipeline. Often a translation function might affect only one operation, but it is perfectly valid for the function to coincidentally affect all of enhance, map, render and transform. The view integrator might define the translation function. It is also possible for a set of default registration methods to link a coordination object inside the coordination space to a certain class of views, which would then define a render group. The coordination object designer would define these defaults.

As well as defining a translation function, a linked view must be registered to receive notify events when the coordination object has changed, signifying that the view's image needs to be updated. As there may be concurrent coordination objects defined over one view parameter, the notify indicates which coordination object has changed, and so which translation function must be accessed. The view must define a notify handler that is triggered by the notify event. At its simplest the notify handler merely forces a regeneration of the image by resending the current data through the dataflow pipeline. Where temporal issues exist, for example with time consuming visualizations, so that perhaps several notify events occur during the production of one view, then the notify handler in the view must include a scheduling algorithm to deal with the queue that develops in the best way for that particular view, deciding whether to restart the visualization process or discard some notify events.

Other elements in the coordination object As mentioned above, it may be desirable to include some notion of default registration for a render group of views so that a plug and play approach to linking views to the coordination object is possible. There are other possible components in sophisticated coordination objects. It may be possible to place constraints on both the type and number of views that can connect to an object. For instance, if one view wants to register to coordinate with another specific view over a particular coordination object, a constraint can be added to that coordination object as to only give those views permission for access and change. A further constraint on linking views might be related to its lifetime, so restricting linking to limited periods, or timing out views after a certain amount of time.

Discussion

Our model fits well with current visualization systems that implement coordination. For instance, Amira is a modular- and object-oriented software system for scientific visualization. It allows the simultaneous display of multiple data sets in different views or in a common view. Amira's components are modules and data objects, each of which has a set of parameters, which can be modified using a parameter editor in an interaction area of the application. Views are coordinated if they share some parameters displayed to the user in the object pool view. The user specifies which views to coordinate.

Similarly, IRIS Explorer users interactively create their application by linking modules; each module has some associated set of parameters, which describe its behavior. The control panel editor creates, modifies and links module control panels, and a parameter function editor creates relationships between parameters in linked modules.[26] The parameter value in the downstream module is then expressed as a function (*P-Func*) of the upstream parameter values (the translation function in our model).

Many visualization systems, such as IRIS Explorer, AVS and Amira provide capabilities for user-oriented design; users could choose modules, edit parameters and link components. This facilitates coordination design. We note that these systems use the dataflow model to build applications for scientific visualization.

Abstract example using the model

Using the abstract example in Figure 4, we illustrate our model using visualizations of geographical map data. The

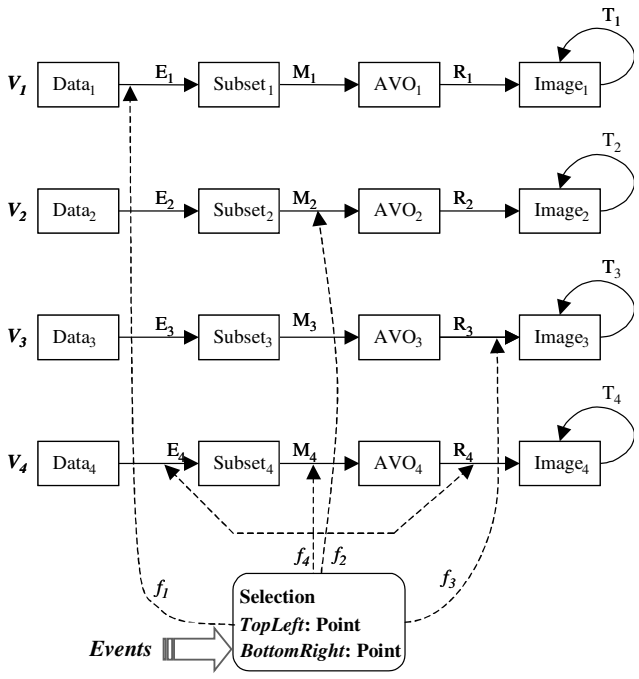

Figure 4 Schematic showing selection coordination.

event handled by the Selection coordination object is that of the user selecting a rectangular area of a map, an event that could be generated by any of the views. The effect in V_1 is to filter the data to show only the selected rectangle whereas V_2 changes the color map to highlight particular objects, such as road junctions inside the rectangle. The result in V_3 is to modify the rendering inside the rectangle, increasing the level of detail to show smaller roads. Finally, the consequence for V_4 is to perform a combination of all these actions, cropping the data, highlighting certain objects and increasing the level of detail by altering the rendering.

Furthermore, f_1 is a simple identity function, as the Selection object holds parameters indicating which area of the map to crop, which is the information required by the Enhance flow of V_1. f_2 is slightly more complex, as it is alters part of the color mapping, so this function indicates that a subsection of the color map will be replaced by the mapping indicating selection. f_3 will pass the selection area to the renderer of V_3, with a flag indicating the desired level of detail. f_4 combines all the effects of f_1, f_2 and f_3 in V_4. We can describe this form of complex coordination using the dataflow paradigm, where coordinations are described by the parts of EMRT (Enhance, Map, Render and Transform) composition affected. The type of coordination in Figure 4 is then E,M,R,EMR as E_1 is affected by f_1, M_2 is affected by f_2, R_3 is affected by f_3 an all of $E_4M_4R_4$ are affected by f_4.

Model implementation

We have developed a concrete prototype that implements our model of coordinated views. The result is a system called *CViews* (see Figure 5). Here, CViews is applied to visualizing the SUSANNE. SUSANNE was created as a taxonomy and annotation scheme of the English grammar.[27]

A corpus is a body of linguistic data, typically some hundred million words. New data are continuously acquired. Analysts often want to extract qualitative as well as quantitative information from this pure text data. Some typical tasks would be to generate frequency lists, concordances, collocations, keyword search and grammatical, gender and social variations. While some visualization systems exist for corpus data,[28] information extraction is dominated by query-based retrieval techniques. There is much space for further exploratory visualization research in this area.

This section examines CViews prototype from five different areas; the program structure, coordination initialization, view registration, event notification and program scalability and extendibility.

Program structure

CViews is developed using Java 1.4 and has three main classes for implementing coordination. They are the *View* abstract class, the *cObject* abstract class and the *Link* class. The remaining classes are data or domain specific. In this case, they are closely related to SUSANNE data files (see Figure 6).

First, the *View* class implements the common features of views. Currently, there are three types of views that extend this class: *tabularView*, *textView* and *concordanceView*. The tabularView displays SUSANNE's raw data in a tabular format. The textView class constructs the original text from SUSANNE's tags and annotation scheme. Finally, the concordanceView displays occurrences of a user selected word as it appears in multiple contexts.

Second, the *cObject* class implements the coordination objects described in our model (see section on Coordination space). For each coordination such as brushing there exists one cObject instance. This class is responsible for setting the parameters of each coordination object and for notifying system views of any change to these objects. As it stands now, CViews supports only one type of coordination object: *brush*.

Third, the *Link* class is responsible for initialization of coordinations and registration of views.

Coordination initialization

We create a coordination initialization table which defines default coordinations between View instances and coordination objects. For example, a coordination which links a number of views has this general entry form

$$C : ((\text{View}_1, \text{Event}_1), (\text{View}_2, \text{Event}_2), \ldots) \rightarrow \text{cObject}$$

For brushing coordination, the default entry is

$$b : ((V_1, \text{select}), (V_2, \text{select}), (V_3, \text{select})) \rightarrow \text{brush}$$

where V_1, V_2 and V_3 correspond to instances of tabularView, textView and concordanceView classes.

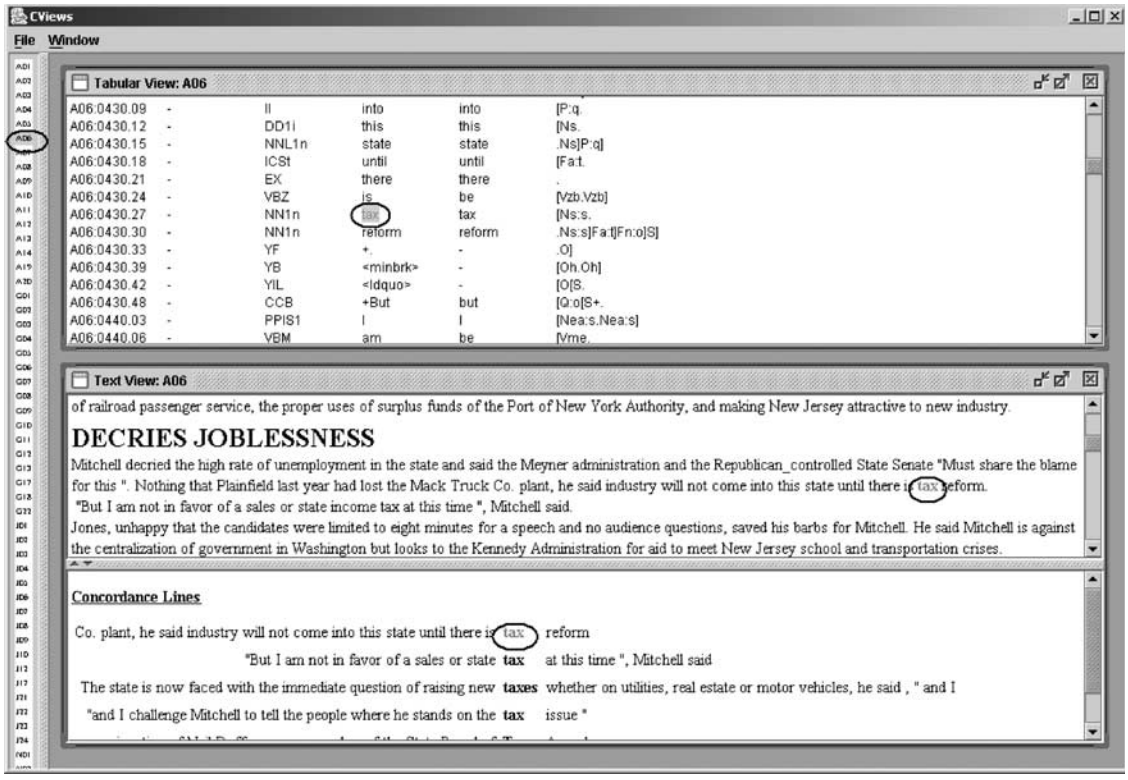

Figure 5 CViews system showing three coordinated views.

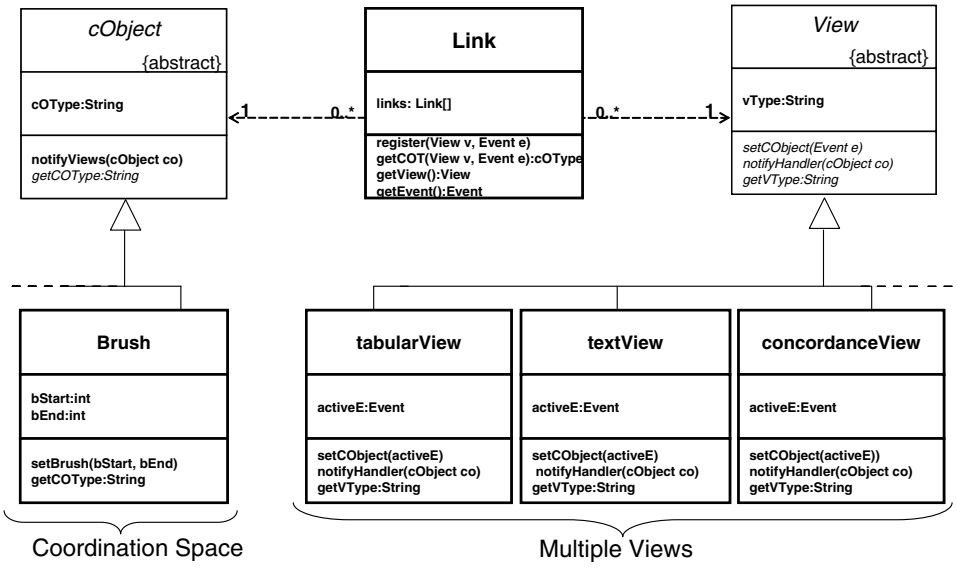

Figure 6 CViews UML diagram.

Note that the order of the tuple components is not significant. Moreover, the initialization table does not take in consideration the detail of how events are implemented at each view level. For instance, the select event is implemented by highlighting the chosen text in the tabular view, and in the other two views by underlying the text.

View registration

All views of our prototype system need to register if they are to play a role in any form of coordination. Registration is dynamic and involves associating views to coordination objects. In particular, a user event occurring in a view is linked to a coordination object cObject. A link can be defined as follows:

$$\text{Link} : (\text{View}, \text{Event}) \rightarrow \text{cObject}$$

The *register* method of the Link class creates a cObject instance using the definition above. We also ensure that view registrations always meet the specification of the initialization entry of this coordination object. For example, in order for a view to register for brushing, the relevant tuple needs to be in the domain of the brush coordination entry in the initialization table. If this is the case, then during exploration the view can change the brush coordination object using a select event. For a coordination to be constructed there should be at least two views registered for the same coordination object. Brushing data items between the views V_1, V_2 and V_3 is only possible if we allow the following three definitions to coexist:

Link_1: (V_1, select) → brush

Link_2: (V_2, select) → brush

Link_3: (V_3, select) → brush

This in effect describes the directionality of our coordination. In this example, brushing is bidirectional among views V_1, V_2 and V_3. More generally, this method of defining link directions allows bidirectional linkage across more than two views.

During exploration, parameters of one or more coordination objects may be changed. The abstract class View has abstract method *setCObject* that sets the coordination object parameters upon change and update the view parameters in consequence. Hence, setCObject contains the translation function (see section on Translation and notification functions). Whenever a coordination object is changed, the *notifyViews* method of the cObject class is called. This method finds the views registered for this coordination object and calls the *notifyHandler* method of each of the affected views.

Program scalability and extendibility

Currently, there are three types of views supported by our prototype. However, the implementation can easily scale to support more views and the coordination space can be extended to include other coordination types. This is possible because our coordination space is separated from view implementations. In addition, new coordination definitions can be handled by our registration mechanism.

The scope of our brush coordination object is global (i.e. it is linked to all registered views for brushing). However, as we include more objects in the coordination space, we may want to restrict the scope of some of its members to certain types of views. Moreover, if a coordination object is changed all registered views for this object are updated. This tight coupling could be made more relaxed later on when we introduce constraints on the scope of our view update.

There is a trade off between the formatting of the parameters of the coordination objects and the translation function to the corresponding view parameters. This may become problematic if the translation functions are expensive. If the number of multiform views in the system increases this will cause more view updates and hence bigger computational overhead.

In the current CViews version, the neutral parameter format of the coordination object 'brush' is the same as the format of the tabularView parameters and so the translation function for that View is the identity function. For the textView and concordanceView, their translation functions take the data in the neutral format and generate text indexes in their own view format.

Moreover, our layered approach to coordination is based on the dataflow model. Within this pipeline, the user can change the visualization parameters that belong to the E, M, R or T processes. However, in our prototype, there is no indication of what dataflow process is affected as a result of which event. Nevertheless, the user can interactively change the Enhance visualization parameters and some of the Transform parameters. The mapping and rendering parameters cannot be changed at the current version of CViews. Hence, coordination is still limited to visual correlation.

Furthermore, the mapping between the (event, view) space into the coordination type space is performed statically during the initialization stage (see Coordination initialization). Later on, when constraints are introduced to our system, users will be allowed to change those mappings interactively. Hence, a coordination manager will be added to CViews. It will serve as a visual representation of user exploration and coordination. Conclusions may be drawn from this subsystem to establish new types of coordination based on frequency of associated tasks performed in the multiple views.

Future work and conclusions

Coordination as described by this paper is the mechanism through which views interact together to achieve purposeful goals that could not otherwise be achieved efficiently by these individual views working uncooperatively. In this paper, we have detailed a model for describing how coordination objects in multiple view exploratory visualization are built from simple user interactions. Our model borrows ideas from recent research in visualization and other disciplines. It handles any combination of data sets and any number of linked views as it is based on sharing objects (parameters) that

control the rendered view and not the sharing of the data that is being visualized. Indeed, a view parameter can be part of more than one coordination object.

The issues involved in coordinating different data sets and visualization methods are dealt with at the translation function stage, where abstract parameters are converted to meaningful parameters in each view. Interactions are variations on the basic visualization functions: Enhance, Map, Render and Transform. If a view interaction is to play a role in coordination, it changes the coordination space, which is then reflected upon the linked views that use that coordination object. We notify all linked views upon change. Hence, we use the eager notify mechanism for our model and implementation. However, we do not allow channels for storage and we are not concerned with establishing a protocol for notification.

Our future work includes extending the CViews prototype system to include more views, more coordinations and constraints in the coordination space. For example, semantic brushing is an obvious extension to our coordination space because it provides the corpus analyst with not only the exact occurrences of a selected word in multiple contexts but also the occurrences of a similar word in various contexts, where similar means concepts such as synonym and antonym.

Moreover, more sophisticated features could be introduced. We could have default coordinations based on system or user settings. Furthermore, we can have recommended types of coordination if the system learns about user interactions, goals and existing coordinations.

More research is required in the area of coordination design to provide rules and guidelines. In addition, comparative studies regarding users' ability to work with independent views compared to working with cooperative multiple views are still underinvestigated. For example, how many coordinated events can one user keep track of during visual exploration? (some work has been done, such as by North and Shneiderman[29]). Moreover, further research is required in the area of multitasking for multiple view exploratory visualization.

The nature of coordinated multiple views is a necessarily vague concept and open to interpretation, hence we cannot claim that our model encapsulates all forms of coordination in this application area. However, we feel that the model covers more concepts than previous attempts, and we have not yet found a counter example of coordination in multiple views that our model cannot capture.

Acknowledgments

We thank Dr. Jonathan Roberts for his contribution to the research described in this paper. This work has been supported by EPSRC (grant reference: GR/R59502/01). CViews uses SUSANNE, a project led by Professor Geoffrey Sampson at Sussex University supported by the Economic and Social Research Council (UK).

References

1 Boukhelifa N, Roberts JC, Rodgers PJ. *A coordination model for exploratory multi-view visualization*. IEEE Proceedings of the International Conference on Coordinated and Multiple Views in Exploratory Visualization CMV2003 (London, UK, 2003), IEEE Computer Society Press: Chicago; 76–85.
2 North C, Shneiderman B. *A taxonomy of multiple window coordinations*. Technical Report #CS-TR-3854 (College Park, U.S.A., 1997) University of Maryland Computer Science Dept.
3 Olson GM, Malone TW, Smith JB. *Coordination Theory and Collaboration Technology*. Lawrence Erlbaum Assoc: London, 2001.
4 Akahani JI, Hiramatsu K, Kogure K. *Coordinating heterogeneous information services based on approximate ontology translation*. Challenges in Open Agent Systems, at the International Conference on Autonomous Agents and Multiagent Systems, AAMAS (2002), ACM Press: New York.
5 Wache H, Vögele T, Visser U, Stuckenschmidt H, Schuster G, Neumann H, Hübner S. *Ontology-based integration of information – a survey of existing approaches*. Proceedings of the Workshop Ontologies and Information Sharing (Seattle, U.S.A., 2002), IJCAI; 108–117.
6 Ciancarini P, Tolksdorf R, Zambonelli F. *Coordination middleware for XML-centric applications*. Proceedings of the 16th ACM Symposium on Applied Computing (Madrid, Spain, 2002).
7 Viroli M, Moro G, Omicini A. *On observation as a coordination paradigm: an ontology and a formal framework*. ACM Symposium on Applied Computing, Proceedings of 16th International Conference SAC01 (Las Vegas, NV, U.S.A., 2001); 166–175.
8 Ahlberg C, Shneiderman B. *Visual information seeking: tight coupling of dynamic filters with starfield display*. CHI'94 (1994), ACM Press: New York; 313–317.
9 Baldonado M, Woodruff A, Kuchinsky A. *Guidelines for using multiple views in information visualization*. Proceedings of the Advanced Visual Interfaces (Palermo, Italy, 2000), World Scientific Publishing Company, Incorporated: 110–119.
10 Ward M. *XmdvTool: integrating multiple methods for visualizing multivariate data*. Proceedings Visualization '94 (Washington DC, 1994), IEEE Computer Society Press: Chicago; 326–333.
11 Ahlberg C. *Spotfire: an information exploration environment*. SIGMOD Record 1996; **24**: 25–29.
12 Jacobson A, Berkin A, Orton M. *LinkWinds: Interactive scientific data analysis and visualization*. Communications of the ACM 1994; **37**: 43–52.
13 Roth SF, Lucas P, Senn JA, Gomberg CC, Burks MB, Stroffolino PJ, Kolojejchick JA, Dunmire C. *Visage: a user interface environment for exploring information*. IEEE Proceedings Information Visualization (San Francisco, USA, 1996); 3–12.
14 Andrienko GL, Andrienko NV. *Interactive maps for visual data exploration*. International Journal of Geographical Information Science 1999; **13**: 355–374.
15 Weaver CE, Livny M. *Improving visualization interactivity in Java*. Proceedings of the SPIE Conference on Visual Data Exploration and Analysis (2000) 62–72.
16 Takatsuka M, Gahegan M. *GeoVISTA studio: a codeless visual programming environment for geoscientific data analysis and visualization*. The Journal of Computers and Geosciences 2002; **28**: 1131–1144.
17 Pattison T, Philips M. *View coordination architecture for information visualization*. Australian Symposium on Information Visualisation (invis.au) (Sydney, Australia, 2001), ACS volume 9; 165–171.

18 North C, Conklin N, Indukuri K, Saini V. *Visualization schemas and a web-based architecture for custom multiple-view visualization of multiple-table databases*. Information Visualization 2002; 211–228.
19 Roberts JC. *On encouraging coupled views for visualization exploration*. Visual Data Exploration and Analysis VI, Proceedings of SPIE (San Jose, 1999), Vol. 3643; 14–24.
20 Roberts JC. *Issues of dataflow and view presentation in multiple view visualization*. CISST Annual Conference (Las Vegas, NV, 2001) CSREA Press; 177–183.
21 Roberts JC. *Waltz—an exploratory visualization tool for volume data, using multiform abstract displays*. Visual Data Exploration and Analysis Proceedings of SPIE (San Jose, 1998); **3298**: 112–122.
22 Mackinlay J, Robertson G, DeLine R. *Developing calendar visualizers for the information visualizer*. ACM Symposium on User Interface Software and Technology, Proceedings of UIST'94 (California, U.S.A., 1994); 109–118.
23 Haber R, McNabb D. *Visualization idioms: a conceptual model for scientific visualization systems*. In: Nielson GM, B Shriver, Rosenblum LJ (Eds). Visualization in Scientific Computing, IEEE Computer Society Press: Chicago, 1990; 74–93.
24 Chi EH. *A taxonomy of visualization techniques using the data state reference model*. Proceedings of InfoVis (2000), IEEE Computer Society: Chicago; 69–76.
25 Duclos AM, Grave M. *Reference models and formal specifications for scientific visualization*. Scientific Visualization Advanced Software Techniques (1993), Ellis Horwood Workshops: Chichester, UK; 3–14.
26 Walton J. *Data visualization with IRIS Explorer—what's new?* Technical Report TR10/96 (NP3070), Numerical Algorithms Group Ltd (1996).
27 Sampson G. *English for the Computer: the SUSANNE Corpus and Analytic Scheme*, Clarendon Press: Oxford, 1995.
28 Gaizauskas R, Rodgers PJ, Humphreys K. *Visual tools for natural language processing*. Journal of Visual Languages and Computing 2001; **12**: 375–412.
29 North C, Shneiderman B. *Snap-Together Visualization: can users construct and operate coordinated visualizations?* International Journal of Human–Computer Studies 2000; **53**: 715–739.