Towards Flexible Graphical Communication Using Adaptive Diagrams

Kim Marriott[1], Bernd Meyer[1], and Peter J. Stuckey[2]

[1] School of Comp. Sci. & Soft. Eng., Monash University, Australia.
{marriott,berndm}@mail.csse.monash.edu.au
[2] NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
University of Melbourne, 3010 Australia.
pjs@cs.mu.oz.au

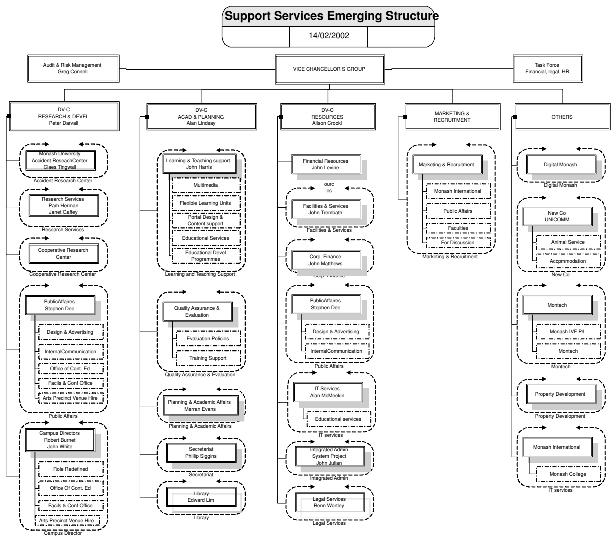
Abstract. Unlike today where the majority of diagrams are static, lifeless objects reflecting their origin in print media, the computer of the near future will provide more flexible visual computer interfaces in which diagrams adapt to their viewing context, support interactive exploration and provide semantics-based retrieval and adaptation. We provide an overview of the Adaptive Diagram Research Project whose aim is to provide a generic computational basis for this new type of diagrams.

1 Introduction

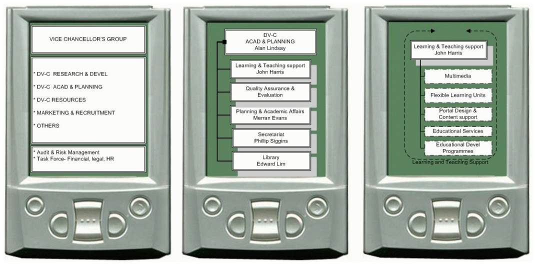
Our computing environment is rapidly changing to a ubiquitous information environment in which we access and interact with a variety of digital media using an incredible variety of devices: wearable computers, hand-held PDAs, tablets with styluses, TV like devices, large wall mounted displays, etc. We believe that diagrams and sketches will play an increasingly important role in this interaction, but that unlike today where the majority of diagrams are static, lifeless objects reflecting their origin in print media, the computer of the near future will provide more flexible visual computer interfaces that use diagrams—often in combination with text or speech—for natural communication.

There are five main differences between the old print media based view of diagrams and this new view based on interactive media and the web:

- *Adaption to viewing environment:* The first difference is the need for a diagram's layout and appearance to adapt to the viewing environment. The obvious reason for this is to make the best use of the viewing device by taking into account its capabilities and display characteristics such as size and aspect ratio. Secondly, adaptation can take into account user needs and desires, for example by adapting text in the diagram to a different language or by using larger fonts and more readily distinguishable colors for users with vision impairment. For example, if the user specifies a larger font size for the organisation chart shown in Figure 1(a) we would expect the layout to adapt by increasing the size of the boxes so that they continue to frame the text and then to appropriately modify the placement of the boxes, text and connectors.


Support Services Emerging Structure
14/02/2002
Audit & Risk Management Greg Connell
VICE CHANCELLOR S GROUP
Task Force Financial, legal, HR
DV-C RESEARCH & DEVEL Peter Darvall
DV-C ACAD & PLANNING Alan Lindsay
DV-C RESOURCES Allison Crook
MARKETING & RECRUITMENT
OTHERS

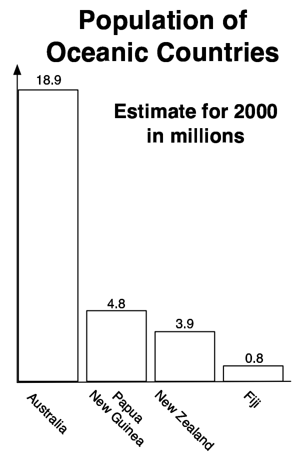
(a) Full layout suitable for large display


VICE CHANCELLOR'S GROUP
* DV-C RESEARCH & DEVEL
* DV-C ACAD & PLANNING
* DV-C RESOURCES
* MARKETING & RECRUITMENT
* OTHERS
* Audit & Risk Management
* Task Force- Financial, legal, HR
DV-C ACAD & PLANNING Alan Lindsay
Learning & Teaching support John Harris
Quality Assurance & Evaluation
Planning & Academic Affairs Merran Evans
Secretariat Phillip Siggins
Library Edward Lim
Multimedia
Flexible Learning Units
Portal Design & Content support
Educational Services
Educational Devel Programmes
Learning and Teaching Support

(b) Interactive view suitable for PDA

Fig. 1. Adaptive viewing of organization chart (from [23])

- *Interactive exploration:* The second key difference is interaction. At the very least one would like the ability to interactively explore large hierarchical diagrams by collapsing or expanding components, thus allowing suppression of detail in areas of little interest so that the other areas can be displayed at a more detailed level. Such *semantic zooming* is particularly important on devices with small display areas, such as PDAs or mobile phones. Consider again the diagram shown in Figure 1(a). This layout is fine for an A4 sized page or laptop but not for a PDA. Figure 1(b) shows how the same diagram can be viewed on a PDA by using semantic zooming to expand and collapse nodes of interest. More ambitiously, we can imagine interactive on-line text-books that help students to understand and learn by letting them manipulate and experiment via manipulation of diagrams.
- *Access to semantics:* The ability to access the information contained in diagrams in a structured way is also very important. This is for at least three reasons. The first is semantics based retrieval. For instance, we might wish to query the web for information on the population of Papua New Guinea. This



Population of Oceanic Countries
Estimate fo 2000 in millions

Australia	18.9
Papua New Guinea	4.8
New Zealand	3.9
Fiji	0.8

Fig. 2. Diagrammatic and textual representation of population data

information may well be hidden in a diagram such as the bar chart shown in Figure 2. To enable search engines to find such diagrams, a semantic description must be attached to each diagram and the search engine must be able to use this determine the applicability of the diagram to the query. The second reason is that if the semantic representation is linked with the visual representation, we can use this to adapt the diagram so as to customize it to the user's needs. For example, when the bar chart shown in Figure 2 is retrieved it would be nice for the bar and value associated with Papua New Guinea to be highlighted. Or, as another example, we can use the contextual information that a user is from a particular department to determine which nodes should be collapsed or expanded in the initial view of a large organisation chart. The third reason is that associating a semantic representation with a diagram facilitates access by blind users and users with audio-only views. Given a semantic representation for a bar chart it is simple to generate an equivalent textual representation (Figure 2). Such extreme adaptivity is required if the web is to truly support universal access.

- *Dynamic generation of content:* Another important issue is the need to handle dynamic content such as that coming from a database. This requires automatic layout to be performed at display time since the diagram's layout cannot be determined beforehand. For example, the diagrams in Figure 1 and Figure 2 might well be generated automatically from information in a database.
- *Authoring:* The above requirements not only affect how the diagrams behave from the user's perspective but also raise significant new issues for the author. Traditional diagram authoring tools are designed for creating static fixed diagrams. For the full promise of this new kind of dynamic, flexible diagrams to be met, authoring tools which allow the construction of diagrams with the above capabilities by graphic designers, not programmers are required.

This paper provides an overview of the Adaptive Diagram Research Project whose aim is to provide a generic computational basis for this new kind of diagram. Reflecting the differences identified above, our research is built around four themes. The first is to extend web document standards for graphics to better support adaptive layout and viewer interaction. The second is to investigate formalisms and techniques for describing and reasoning with the semantics of diagrams. The third is to develop better techniques for automated static layout of a diagrams from abstract specifications and for dynamic re-layout of a diagram during interaction. The final theme is the development of authoring tools. Each theme will be examined from three complementary perspectives: mathematical theory (algorithm design), software engineering (generic software tools and methodologies) and human-computer interaction (HCI). In the remainder of the paper we discuss each of these in more detail. However as the project is only beginning much of the present paper consists of questions and possible approaches rather than solutions.

Geometric constraint solving provides a common foundation for our approach. Viewing diagram layout (and more generally any sort of document layout) as constrained optimization provides a general framework for understanding and supporting both static and dynamic layout in which geometric constraints capture the structure of the diagram or document and an objective function measures the aesthetic desirability of a particular layout.

This extends earlier suggestions for constraint-based specifications of web document layout, e.g. [34, 10, 6, 7]. The key idea is that geometric constraints allow the author to give a flexible, partial specification of document layout. The author can specify, for example, that a box should be large enough to contain its text or that certain objects should be aligned. The user's preferences and the viewing environment characteristics can be taken into account as additional constraints. Constraints can be preferences as well as strict requirements. The appearance of the document is the result of a *negotiation* between the author, the user and the viewing tool. Thus, although the author does not rigidly dictate the precise appearance of the document, the use of constraints allows the author to maintain a high degree of control over the document's appearance even in unanticipated viewing situations.

The other main foundation for our project is a tripartite view of a diagram consisting of its visual appearance, its semantics, and its abstract syntactic structure which provides a bridge between the visual appearance and the semantics. For example, in the case of an organisation chart the visual appearance is the low-level graphic elements such as dashed lines and text elements making up the diagram, the abstract syntax representation is a labelled tree while the semantics might be a description of the logical relationships between the entities in the organisation chart such as "x reports to y." An important role of an authoring tool is to invisibly support this tripartite view of a diagram by allowing the designer to (largely) work at the level of the visual appearance and for the authoring tool to automatically update the associated abstract syntax tree and semantics.

2 Graphic standards that support interaction and adaptive layout

If flexible, interactive, adaptive diagrams are to become a reality, one of the most basic requirements is for web graphics standards to support and encourage such behaviour.

The desire to separate form from content, in part to allow adaptive layout, has been a major concern underlying the design of web document standards. But until recently the focus has been almost exclusively on textual documents. Older raster-based image formats such as JPEG and GIF do not even re-scale satisfactorily. The new resolution independent graphics web format Scalable Vector Graphics (SVG) [31] is a considerable improvement. It supports uniform rescaling of diagrams, specification of text attributes through style sheets, and allows alternate versions of document elements for different media and languages.

SVG provides the standard graphic primitives such as rectangles, circles, paths and standard graphical transformations. It also provides simple animation. The following SVG code specifies some text and a rectangle which frames the text.

```
<text id="t" x="300" y="50" text-anchor="middle">
  The rect should contain this text
</text>
<rect x="200" y="45" width="200" height="10"
  stroke="red" stroke-width="2" fill="none"/>
```

Unfortunately, the current SVG standard does not adequately support adaptation of the document's layout to the viewing context such as by adapting the layout to the size of the browser window or to the user's desire for larger fonts. In the above example, if the user changes the font size by, say, using a style sheet, the rectangle will not adapt to the change in text size.

Adding constraint-based layout mechanisms to SVG would provide such adaptive behaviour [2, 23] and we have explored this in an extension of SVG called Constraint SVG (CSVG) [21]. CSVG extends SVG by allowing attribute values to be specified with expressions that are evaluated at display time to determine the actual attribute value. In other words, CSVG extends SVG with one-way constraints.

One-way (or data-flow) constraints are the simplest, most widely used approach to constraint-based layout [33]. They form the basis for a variety of commercial products including spreadsheets, widget layout engines, and the customizable graphic editors Visio and ConceptDraw. A one-way constraint has form $x = f_x(y_1, ..., y_n)$ where the function f_x details how to compute the value of variable x from the variables $y_1, ..., y_n$. Whenever the value of any of the y_i's changes, the value of x is recomputed, ensuring that the constraint remains satisfied. One-way constraints can be used to ensure that, for instance, a box remains large enough to contain its component text or that lines or arrows connecting two objects follow the objects when they are moved.

Thus, using a CSVG example taken from [21], we can rewrite our previous SVG example so that the rectangle's attributes are specified in terms of the

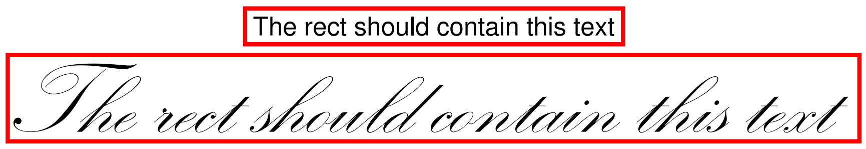

The rect should contain this text
The rect should contain this text

Fig. 3. How the layout of the CSVG example adapts to different fonts (from [21])
text's bounding box, so allowing the rectangle's location and size to adapt to different text sizes as illustrated in Figure 3.

```
<text id="t" x="300" y="50" text-anchor="middle">
  The rect should contain this text
</text>
<rect width="0" height="0" stroke="red" stroke-width="2" fill="none">
  <c:constraint attributeName="x"
                value="c:x(c:bbox(id('t'))) - 4"/>
  <c:constraint attributeName="y"
                value="c:y(c:bbox(id('t'))) - 4"/>
  <c:constraint attributeName="width"
                value="c:width(c:bbox(id('t'))) + 8"/>
  <c:constraint attributeName="height"
                value="c:height(c:bbox(id('t'))) + 8"/>
</rect>
```

Furthermore, if the location of the text is animated then the rectangle will automatically follow the text or if the text is changed by user interaction then the rectangle will automatically resize.

We have found that adding expression-based evaluation of attributes to SVG adds little implementation effort but allows the document to adapt to different sized browsing windows, to adapt to changes in the size of text elements such as those arising from the use of a larger font or a different language, and to adapt to changes resulting from user interaction and animation [21]. Remarkably all three kinds of adaptation are supported by the same simple mechanism.

Currently the only way to obtain such adaptive layout with SVG is to use script to modify the SVG DOM at display-time. Constraint-based adaptation has two main advantages over script-based adaptation. First, it removes the need for the document author or authoring tool to worry about how to perform efficient incremental updates when document elements are modified since this is the responsibility of the SVG browser. Second, it removes most of the need for SVG authoring tools and document authors to understand script, thus facilitating editing and interchange of SVG documents.

3 Associating semantics with diagrams

The second theme of our research is to investigate how to associate an explicit representation of a diagram's semantics with a diagram and how to use this to support semantics based search and retrieval of on-line diagrams and to perform semantics driven adaptation. The issue is that an SVG document, or even worse a raster image, contains the semantics only implicitly.

This lack of an explicit, computer manipulable semantics is also true for textual documents and as a result currently most common retrieval mechanisms on the web use statistical methods that do not support more complex queries and retrieve many irrelevant documents. For this reason there is world-wide interest in the "semantic web" [5] and the use of logic-based languages for specifying document content and their associated ontologies which are designed to support semantics-based information processing and inference.

RDF (Resource Description Framework) forms the core of the semantic web [29]. It allows the specification of objects ("resources") and relations between them and provides a simple semantics for this datamodel. RDF Schema and OWL allow the specification of ontologies. Both provide a vocabulary for describing properties and classes of RDF resources organised into generalization-hierarchies. Of these OWL is the more powerful. It comes in three increasingly powerful variants: OWL Lite, OWL DL which is based on description logic and endeavours to provide maximum expressiveness while retaining decidability, and OWL Full which provides the power of first-order logic but is consequently undecidable.

RDF forms one of the two generic approaches to high-level data representation currently used for the web. XML [37] provides the other approach. It is a generic markup language primarily designed for data transfer between application programs. It allows the specification of customised languages for hierarchical description of data. RDF, RDF Schema, OWL and SVG all have an XML-compliant syntax.

We believe it makes sense to explore both of these approaches for diagram specification since the XML approach is suited to specifying the diagram's abstract syntax while the RDF approach is suited to detailing the diagram semantics. There has been some previous work on XML-based specification of particular kinds of diagram, e.g. XML notations for representing UML [36] and geographic data [12]. As a simple example, we might represent the bar-chart shown in Figure 2 by:

```
<ex:bar-chart orientation="vertical" style="value-annotated">
    <ex:title> Population of Oceanic Countries</ex:title>
    <ex:subtitle>Estimate for 2000 in millions</ex:subtitle>
    <ex:scale bottom="0" top="20" />
    <ex:bars>
      <ex:bar name="Australia" value="18.9"/>
      <ex:bar name="Papua New Guinea" value="4.8"/>
      <ex:bar name="New Zealand" value="3.9"/>
      <ex:bar name="Fiji" value="0.8" />
    </ex:bars>
</ex:bar-chart>
```

Using XML to provide a high-level description of a diagram's abstract syntax is useful but does not directly support reasoning about the diagram. We believe a generic approach to specification of the diagram semantics is also required in which the same generic logic engine can be used to reason about any diagram given some specific inference rules for the class of diagrams it belongs to.

The obvious approach is to use RDF and OWL since this allows us to leverage from the techniques and systems currently being developed to support the semantic web. Such support is important because more complex query answering and retrieval requires the fusion of information from different diagrams and from other contextual information such as surrounding text.

However, for this approach to succeed, three issues need to be addressed. The first is whether or not first-order logic based formalisms and, in particular, description logics are powerful enough to capture the semantics of most diagrammatic notations and does logical inference provide sufficient query answering power?

There is some history of using description logics for representing diagram semantics. For instance, the use of description logics to capture the semantics of sketched queries to a GIS system and visual programming languages [13] and for capturing the semantics of UML diagrams [4]. Thus we believe that description logic and so OWL DL is powerful enough to handle abstract notations, such as graphs and UML notations, which rely only on topological relationships and have a well-defined semantics and syntax. However, in the case of notations in which spatial location and distance between objects is important it is not clear that standard description logic is adequate. For instance, the boundary between the kitchen and living area may be ambiguous in a house plan and in a weather map it is not clear where the boundary between a high-pressure system and a low-pressure system occurs. Even though some approaches have integrated description logic with concrete continuous domains [13], vagueness in a notation can be a severe problem and appears to limit the usefulness of the logic-based approach for representing the semantics of vague notations such as sketches.

The second issue is the need to develop new (good!) user interaction metaphors for search and retrieval that utilise semantic information to highlight the relevant parts of the diagram. More ambitiously, we can imagine a system that merges information from a variety of media and generates a mixture of diagrams and text to provide a summary.

User interaction is even more of an issue when presenting a diagram to a blind user: If one is only interested in the population of Papua New Guinea one does not want to have to listen to an audio description of the entire bar-chart in Figure 2 and the population of all countries in it. On the other hand one might be interested in summary information such as trends which is implicit in the diagram. There has been relatively little work in this field, although see [27].

The final issue is how to associate a diagram with an RDF description of its semantics in terms of a notation specific ontology. The simplest method is for the RDF description to be generated automatically by the authoring tool from the diagram's abstract syntactic structure. The harder (but arguably more useful method) is to search for relevant diagrams and then generate the RDF description from the visual representation of the diagram on the fly.

Computational approaches to understanding of diagrams have been studied for the last 40 years and syntax-based diagram interpretation is now comparatively well understood [24], but it relies on a precise and narrow definition of the notation to be interpreted. In the context of the web this is inadequate and more flexible interpretation techniques are required. To understand the difficulties consider the bar-chart in Figure 2. This has many semantically equivalent representations: the axes could be rotated, values could be associated with elements indirectly by using a scale on the axis rather than by directly annotating the bar with a value, or bars might not be rectangles but instead vertical stacks of stick figure icons. On the other hand, interpretation of diagrams specified in SVG is considerably simpler than for raster-based graphics since SVG's vector-based graphic specifications details immediately the graphic primitives such as lines, text, polygons etc that the diagram is made up of.

4 Automated layout

Automated layout is at the core of diagram adaptation. It is useful to distinguish between static layout in which a new layout is computed from scratch, and dynamic layout in which the current layout is modified.

Static layout takes an abstract specification of a diagram, for instance an organisation hierarchy, and determines a good layout for the diagram. This is most relevant to generation of diagrams from on-line data but is also required in authoring tools when data is imported from other application programs such as spreadsheets or databases.

Constrained optimization provides a general framework for modelling static layout. Geometric constraints capture the structure of the diagram and an objective function measures the aesthetic desirability of a particular layout. For some diagrammatic notations, such as trees, we can solve the resulting constrained optimisation problem sufficiently rapidly using standard operations research techniques such as linear or quadratic programming. Typically, however, the resulting constrained optimisation problems are impractical to solve using standard generic techniques since they involve complex non-linear and non-convex arithmetic constraints and objectives, such as minimising the number of edge crossings. For this reason, we must usually devise algorithms that are specialised for a particular kind of diagram and drawing convention.

Most research into specialised algorithms for diagram layout has been for graphs and this has largely focussed on the static layout of idealized mathematical graphs (including the important sub-case of trees) composed of nodes and edges [3]. Surprisingly little work has gone into layout of other types of diagrammatic notations.

We plan to explore layout of blob charts. These are an important and common kind of diagram used in business, science and engineering for illustrating processes and sub-processes and relationships between them. Concept maps and state charts are both examples of blob chart like notations. Syntactically, blob charts are a labelled graph notation enriched with hierarchical grouping of nodes indicated by inclusion. Often they contain significant amounts of text.

These seemingly minor extensions make their layout significantly harder than that of normal graphs and to date no general layout procedure for such diagrams is known, despite their practical importance [14]. One possible approach to such more complex layout problems is the use of optimization meta-heuristics. Unfortunately meta-heuristic approaches usually suffer from being slow which makes them unsuitable for interactive applications, so that specialized algorithms are required.

Dynamic layout is required to support interactive exploration of graphs and other notations such as organisation charts and blob-charts in which the diagram is hierarchically organized and sub-parts can be collapsed or expanded to show internal structure. This is naturally modeled as a constrained optimization problem in which the new layout is required to be as “similar” as possible to the old layout so as to preserve the user’s mental map [8]. Unfortunately, most research has focused on static layout and there has been surprisingly little work on dynamic layout even in the context of graphs.

We are planning to investigate how to best model dynamic layout as constrained optimization by conducting user studies to better understand which layout properties (e.g. ordering of nodes) it is most important to preserve during re-layout. Then we will develop techniques for solving constrained optimization problems of the form identified. These need to be fast enough to support user interaction. Inspired by the approach taken in [16], we plan to extend dynamic linear approximation to handle the non-linear objective functions arising in force directed graph layout [3].

5 Authoring tools

The final research theme is the design and development of authoring tools. We believe that the best approach is to use a generic constraint-based graphics editor which can be customized for particular diagrammatic notations by providing notation specific graphic elements and layout templates. Tools such as Microsoft Visio and ConceptDraw are a first step into this direction. They are based on one-way constraints and provide extensible templates for common notations, but we believe that they fall short when it comes to usability, flexible customisability and more complex geometric constraints. Furthermore, they are not intended to generate adaptive diagrams or an associated semantic description.

As we discussed in the introduction we believe that the authoring tool should have tripartite view of a diagram consisting of its visual representation, its abstract syntax and its semantics. As the author changes the visual representation it is the role of the editor to appropriately update the abstract syntax representation and to generate the adaptive visual representation, say in CSVG, and associated semantic description, say in RDF, from this tree.

Creating an editor for a new notation requires a (possibly implicit) specification of the abstract syntax and ontology for that language as well as mappings between abstract syntax and semantics and abstract syntax and visual representation. One possible starting point are grammar-based techniques for specifying abstract syntax for diagrams and automatic generation of customised diagram editors [19].

Requiring the authoring tool to keep a high-level representation of the diagram allows more than generation of RDF. It allows the tool to provide high-level notation specific transformations, such as for instance transforming a bar chart to a pie chart. Furthermore, it can provide default adaptive layout behaviour for a particular notation.

Of course the default adaptive layout behaviour will not be appropriate for all diagrams. A key question is how the author can specify diagram specific adaptive layout behaviour. Clearly they should not have to write textual attribute expressions when, say, specifying one-way constraints in CSVG. Previously, we have suggested two possible approaches that have been used for specifying widget layout in user interfaces: learning from examples and spring-based specification [23]. However, it is unclear whether either of these will be practical.

We would like the authoring tool to behave as if it understands the abstract syntax and semantics of a diagram during editing. Constraint solving plays a crucial role in supporting such natural behaviour since it allows the abstract syntax (and so the semantics) of the diagram to be maintained during manipulation by preserving the geometric relationships between the diagram components. For example, constraints can be used to ensure that if a component in a network diagram is moved, its internal components and external connections move with it. Thus, constraint-solving is the heart of our approach to diagram editing.

Starting with SketchPad [30], constraint solving for graphics editors and other interactive graphical applications has been investigated for four decades [17]. The requirement for very fast re-computation of solutions during direct manipulation has led to the development of specialized constraint solving algorithms.

As previously discussed, one-way constraints are the most widely used approach. They are simple to implement and can be solved extremely quickly. Their main limitation is that constraint solving is directional and cyclic dependencies between variables are not allowed. This means that only fixed, pre-determined modes of interaction can be supported. For instance, a text box will change size if the text is changed, but if the size of the box is changed, the text remains the same size.

The customizable grahics editor Microsoft Visio provides one-way constraint-based tools, such as alignment and distribution for layout, but a preliminary user study [35] suggests that many users find it difficult to understand what will happen if an object is involved in more than one constraint since only one constraint will be enforced and the others ignored. Our study suggests that if so-called multi-way constraints are used this difficulty vanishes. Thus, we believe that multi-way constraints are required in graphic editing.

Unfortunately, there is no single best approach to solving multi-way constraints. One popular approach is to use propagation based solvers, e.g. [28, 32], while linear arithmetic constraint solvers are a more recent approach, e.g. [11, 1, 22]. Iterative numeric approaches have also been tried, e.g. [26], as well as degrees of freedom analysis for CAD [20].

However, it is fair to say that none of the existing techniques for solving multi-way constraints is powerful enough. For example, consider a constraint-based authoring tool for blob chart diagrams or a document authoring tool that allows constraint-based placement of floating figures in a page. In both cases, we would like to support direct manipulation of arbitrary diagram components in the context of non-overlap, containment, alignment and distribution constraints. Existing approaches to constraint solving for interactive graphical applications cannot handle the kind of (possibly cyclic) multi-directional non-linear constraints arising in these scenarios: we need to develop more powerful constraint solving techniques.

Currently we are exploring a very simple idea: we model more complex geometric constraints, such as non-overlap and containment in non-convex objects, by a dynamically changing conjunction of linear constraints. Consider a complex non-linear "non-overlapping" constraint on two rectangles. At each stage during an interaction, this constraint can be locally approximated by a conjunction of linear constraints (e.g. "left-of" and "above") which can be efficiently solved using linear arithmetic constraint solvers. The solver automatically switches between local approximations at appropriate moments during the interaction. Our initial prototype implementation [25, 18] is very promising, but we need to develop a methodology and theoretical justification for designing such linear approximations.

However, the lack of powerful constraint-solving algorithms is not the only reason limiting the adoption of constraint solving techniques in graphics editors. Probably the most pressing issue is the need for good user interaction techniques and metaphors for constraint-based authoring. A key question is how to provide adequate visualization of constraints without cluttering large diagrams with representations of the constraints such as guidelines etc.

It is natural to imagine that the authoring tool and the browser provide the same constraint solving capabilities. However, we believe that this may not be the best model. Instead in our approach the authoring tool provides a powerful multi-way constraint solver for constructing the diagram and then compiles the resulting constraints into much less powerful constraints such as one-way constraints or even scripting commands that encode a plan for satisfying the author's constraints. The advantage is that the browser and web standards such as CSVG do not need to support more powerful constraints. This is possible as long the browser only allows the user to interact in limited ways such as by changing browser window size or the default font size.

We therefore plan to study "compilation" of linear arithmetic constraints and more complex geometric constraints into one-way constraints to support this model and also to improve the efficiency of constraint solving in general. We have previously investigated the use of a projection-based algorithm for compiling linear arithmetic constraints [15], but there is considerable work to be done.

6 Conclusion

Diagrammatic notations are ubiquitous, but current web and computer-mediated communication technologies have primarily focussed on text, video and static image data. The research program we have sketched is intended to remedy this bias by developing the basis for computational handling of diagrams which supports sophisticated, intelligent user interaction, visualization and retrieval. It will support universal access from different devices and better access to diagrammatic information by people with vision impairment and blindness.

However, it is important to realise that diagrams usually occur as part of a larger (usually textual) document. Thus an important question in the longer term is how to provide better support for adaptation and user interaction for compound documents containing text, diagrams and variety of other media. Adaptation of multimedia documents so as to take into account device capabilities, bandwidth, temporal and simple layout constraints has been studied for some time, see e.g. [9], but sophisticated layout adaptation, semantics based retrieval and generic support for user interaction has not been explored. In particular, adaptive layout is a hard problem since we may need to trade-off the areas assigned to different document components in order to find the best overall layout. We are hopeful that the techniques we are developing for diagrams will provide a suitable basis for more complex, compound documents.

Acknowledgements

It is a pleasure to acknowledge the many useful (sometimes heated but always interesting) discussions with the other members of the Adaptive Diagrams Project: Stan Gunawan, Nathan Hurst, Cameron McCormack, Linda McIver, Peter Moulder, Michael Wybrow and those colleagues who have worked on precursors to the project: Sitt Sen Chok, Tony Jansen, Weiqing He and Laurent Tardif.

References

1. G. Badros, A. Borning, and P.J. Stuckey. The Cassowary linear arithmetic constraint solving algorithm. *ACM Transactions on Computer Human Interaction*, 8(4):267–306, 2001.
2. G. Badros, J. Tirtowidjojo, K. Marriott, B. Meyer, W. Portnoy, and A. Borning. A constraint extension to Scalable Vector Graphics. *ACM Conference on the World Wide Web (WWW10)*, pages 489–498, Hong Kong, May 2001.
3. G. Di Battista, P. Eades, R. Tomassia, and I. Tollis. *Graph Drawing.* Prentice Hall, 1999.
4. D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams using description logic based systems. *Proc. of the KI'2001 Workshop on Applications of Description Logics*, CEUR Electronic Workshop Proceedings `http://ceur-ws.org/Vol-44/`, 2001.
5. T. Berners-Lee, J. Hendler and O. Lassila. The Semantic Web. *Scientific American*, May 2001.
6. A. Borning, R. Lin and K. Marriott. Constraints for the Web. In *Fifth ACM International Multi-Media Conference*, pp. 173–182. Seattle, November 1997.
7. A. Borning, R. Lin and K. Marriott. Constraint-based document layout for the Web. *ACM/Springer Verlag Multimedia Systems Journal* **8**(3): 177-189, 2000.
8. P. Eades, W. Lai, K. Misue and K. Sugiyama. Preserving the mental map of a diagram. In *CompuGraphics'91*, Vol. 1, pp. 34–41, 1991.
9. A semantic framework for multimedia document adaptation. J. Euzenat, N. Layada and V. Dias. *International Joint Conference on Artificial Intelligence (IJCAI'2003)*, 2003.

10. W.H. Graf. The constraint-based layout framework LayLab and its applications. In *ACM Workshop on Effective Abstractions in Multimedia* in conjunction with ACM Multimedia, 1995.
11. M. Gleicher and A. Witkin. Drawing with constraints. *The Visual Computer* 11(1), 1994.
12. GML - the Geography Markup Language. `http://opengis.net/gml/`
13. V. Haarslev, R. Möller, M. Wessel. Visual spatial query languages: A semantics using description logic. In *Diagrammatic Representation and Reasoning*, M. Anderson, B. Meyer, P. Olivier (eds), 387–402.
14. D. Harel and G. Yashchin. An algorithm for blob hierarchy layout. *The Visual Computer*, 18:164–185, 2002.
15. W. Harvey, P.J. Stuckey, and A. Borning. Fourier elimination for compiling constraint hierarchies. *Constraints*, 7:199–219, 2002.
16. W. He and K. Marriott. Constrained graph layout. *Constraints* **3**(4): 289-314. 1998.
17. W. Hower, W.H. Graf. A bibliographical survey of constraint-based approaches to CAD, graphics, layout, visualization, and related topics. *Knowledge-Based Systems*, 9:449–464, 1996.
18. N. Hurst, K. Marriott and P. Moulder. Dynamic approximation of complex graphical constraints by linear constraints. *ACM Symposium on User Interface Software and Technology*, 191–200, Paris, Nov. 2002.
19. A. Jansen, K. Marriott, and B. Meyer. CIDER: A component-based toolkit for creating smart diagram environments. *Proc. of the 2003 International Conference on Visual Languages and Computing (VLC 2003)*, Miami, Sep. 2003, pp 353-359.
20. G. Kramer. A geometric constraint engine. *Artificial Intelligence*, 58:327–360, 1992.
21. C. McCormack, K. Marriott and B. Meyer. Adaptive layout using one-way constraints in SVG. *3rd Annual Conference on Scalable Vector Graphics (SVG Open).* Japan, Sep. 2004.
22. K. Marriott and S.S. Chok. QOCA: A constraint solving toolkit for interactive graphical applications. *Constraints* **7**(3/4): 229–254, 2002.
23. K. Marriott, B. Meyer, and L. Tardif. Fast and efficient client-side adaptivity for SVG. *ACM Conference on the World Wide Web (WWW 2002)*, 496–507, Honolulu, May 2002.
24. K. Marriott, B. Meyer and K. Wittenburg. A survey of visual language specification and recognition. In K. Marriott and B. Meyer, editors, *Theory of Visual Languages.* Pages 5–85. Springer-Verlag. 1998.
25. K. Marriott, P. Moulder, P.J. Stuckey, and A. Borning. Solving disjunctive constraints for interactive graphical applications. In T. Walsh, editor, *Proceedings of the Sixth International Conference on Principles and Practices of Constraint Programming*, LNCS, 361–374. Springer-Verlag, 2001.
26. G. Nelson. Juno: A constraint-based graphics system. In *SIGGRAPH '85 Conference Proceedings*, 235–243, ACM Press, 1985.
27. H. Petrie, C. Schlieder, P. Blenkhorn, D. Evans, A. King, A-M. O'Neill, G. Ioannidis, B. Gallagher, D. Crombie, R. Mager, M. Alafaci. TeDUB: A System for Presenting and Exploring Technical Drawings for Blind People. *Computers Helping People with Special Needs, 8th International Conference (ICCHP).* LNCS Springer-Verlag, pages 537-539, 2002.
28. M. Sannella, J. Maloney, B. Freeman-Benson, B., and A. Borning Multi-way versus one-way constraints in user interfaces: Experience with the DeltaBlue algorithm. *Software—Practice and Experience* 23(5), 529–566, 1993.
29. Semantic Web. `http://www.w3.org/2001/sw/`

30. I.E. Sutherland, Sketchpad: a man-machine graphical communication system. In *Proceedings of the Spring Joint Computer Conference*, pp. 329–346, IFIPS
31. Scalable Vector Graphics (SVG) `http://www.w3.org/Graphics/SVG/`
32. B. Vander Zanden An incremental algorithm for satisfying hierarchies of multi-way dataflow constraints. *ACM Transactions on Programming Languages and Systems* 18(1), 30–72, 1996.
33. B. Vander Zanden, R. Halterman, B. Myers, R. McDaniel, R. Miller, P. Szekely, D. Giuse, and D. Kosbie. Lessons learned about one-way, dataflow constraints in the Garnet and Amulet graphical toolkits. In *ACM Transactions on Programming Languages and Systems* 23(6), 776–796, 2001.
34. L. Weitzman and K. Wittenburg. Automatic presentation of multimedia documents using relational grammars. In *Proceedings of 2nd ACM Conference on Multimedia*, 443–451, 1994.
35. M. Wybrow, K. Marriott, L. McIver and P. Stuckey. The usefulness of constraints for diagram editing. *Proc. of the 2003 Australasian Computer Human Interaction Conference (OzCHI 2003)*, Brisbane, Nov. 2003, pp 192-201.
36. XML Metadata Interchange (XMI). `http://www.omg.org/technology/documents/formal/xmi.htm`
37. Extensible Markup Language (XML). `http://www.w3.org/XML/`