Architectures to make Simple Visualisations using Simple Systems

Alan Dix
aQtive[3] and Lancaster University[1]
alan@aQtive.com

Russell Beale
aQtive[3] and University of Birmingham[2]
russell@aQtive.com

Andy Wood
aQtive[3]
andy@aQtive.com

aQtive limited, Birmingham Research Park, Vincent Drive, Birmingham, B15 2SQ, UK, +44 121 414 2626
http://www.hiraeth.com/alan/topics/vis/

ABSTRACT

In previous work, the first author argued for simple lightweight visualisations. These are surprisingly complex to produce due to the need for infrastructure to read files, etc. onCue, a desktop 'agent', aids the rapid production of such visualisations and their integration with desktop and Internet applications. Two examples are used dancing histograms for 2D tables and pieTrees for hierarchical numeric data. A major focus is the importance of architecture, both that of onCue itself and the underlying component infrastructure on which it is built – separation of concerns, mixed initiative computation and plug-and-play components lead to easily produced and easily used systems.

Keywords

Interactive visualisation, software architecture, hierarchical data, artificial intelligence, Internet–desktop integration

1. INTRODUCTION

Previously the first author has argued for simple lightweight visualisations based on interactive modifications of common paper representations of data. One problem is that although such visualisations can be produced rapidly, incorporating them into a usable system involves the ability to deal with appropriate file formats, table editors etc. The lightweight visualisations become heavyweight systems. This paper describes how onCue, a context sensitive desktop 'agent', has been used to integrate lightweight visualisations with desktop and Internet data sources. onCue is built upon a flexible component framework and the paper will demonstrate how appropriate underlying software architectures facilitate effective visualisation software. The paper will use as exemplars two visualisations, 'dancing histograms' and 'PIE-trees'. The former was presented at a previous AVI conference, the latter is a novel method for displaying hierarchical data where numerical information is associated with interior nodes as well as leaves (for example web site logs). Two features of the onCue architecture will be discussed in detail. First, the use of two types of components: 'recognisers' detecting different kinds of data from diverse sources, and 'services' invoking desktop or Internet applications. This separation makes adding new visualisations particularly easy. It also allows Internet-based services to be made accessible from the desktop in a near seamless manner and desktop services to easily use Internet data. The second feature is a mixed data-driven and demand-driven underlying component model. The key message is that architecture is important!

The next section recaps the arguments for simple visualisations including the problems found implementing dancing histograms and also the important role of architecture within user interface construction. Section 3 looks at onCue. It describes how onCue works, both its external behaviour and internal structure. We will also see how this has facilitated the integration of dancing histograms as a form of universal plug-in able to visualise tabular data from many sources including web pages, email messages, word processed documents and spreadsheets. In section 4, we will look at pieTrees, both the design concept and how it is implemented within onCue. Finally, in section 5, we will look deeper into the underlying component infrastructure of onCue, called aQtiveSpace and see how the model facilitated the construction of onCue and is suitable for other forms of visualisation project.

2. BACKGROUND

2.1. simple visualisation – promise and problems

In a previous paper at AVI'98, "starting simple", Geoff Ellis and the first author argued the importance of simple interactive visualisations [7]. Many exciting visualisation techniques have evolved over recent years, including the extensive use of 3D [1, 17] and dynamic query techniques [20, 18] involving many novel representation techniques. We argued that the most significant feature of these was their interactive nature and that this could be harnessed in simpler interactive visualisations. In particular, interactive variants of familiar paper-based representations are easy for users to understand and potentially easy to construct.

One of the driving examples from this paper was an interactive version of stacked histograms. Stacked histograms are often used where one wants to compare, for example, total sales between years, but where the sales in each year are also categorised by product line (figure 1). Other uses include agricultural production in different regions, where one wants a histogram giving total production within each region, but see the relative contributions of different types of production (arable, grassland, etc.).

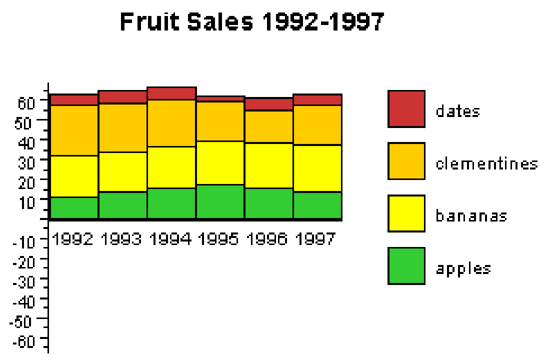


Figure 1. Stacked histogram of product sales by year

The problem with paper-based stacked histograms is that it is only possible to easily spot trends in the overall totals and the bottom category. The other categories have different baselines and are thus hard to compare. A simple interactive change was to make it possible to move the columns to align the bottom of different categories (figure 2).

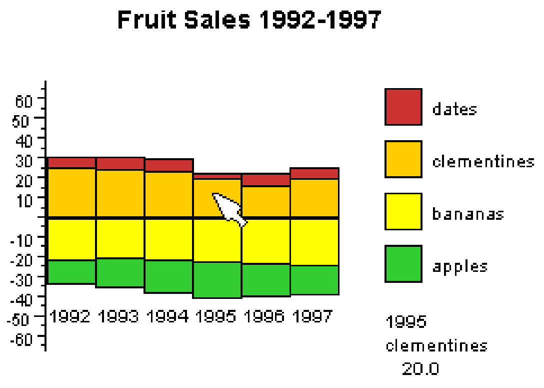


Figure 2. Dancing histograms change the baseline

To demonstrate the concept, an applet version of dancing histograms was produced very rapidly (less than two days effort). This was very successful as a presentation aid and as a demonstration on the web, but didn't give anything that it was easy for others to slot their own data into. Really it needed some form of import filters for formats such as CSV, tab delimited text, common spreadsheet formats, database data etc. and also some way to enter and edit data in a self contained program.

The effort to produce the demonstration of the visualisation concept was minimal, but the additional effort to turn this into a stand-alone program would be prohibitive.

2.2. architectures for user interfaces and interactive visualisation

Software architecture has long been regarded as an important issue in user interface construction. Early architectural models drew from the success of linguistic levels in compiler design, with emphasis on the distinction between presentation (lexical), dialogue (syntactic) and functionality (semantics). The early reference model for this was the Seeheim model [15] with three main components corresponding exactly to these levels: presentation or display management (which determines the appearance and low-level behaviour of the interface), dialogue control (which determines the order of interaction) and a link to the underlying data structures and semantics. This layered presentation/semantics distinction is evident in other abstract architectures, in particular the more recent Arch/Slinky model [19]. Similar distinctions can be found in more component or object-based implementation frameworks such as MVC [14] and PAC [4], although these focus on individual parts of an interface (e.g. the interaction with a particular item of data) rather than a global separation of the application into components for each linguistic level.

Effective architectural design has a number of benefits for user interface software. It can:

(a) make the software easier to construct and maintain (including reuse)

(b) divide the software into parts which can be run on different parts of a distributed system (especially important for web-based interfaces [16])

(c) improve the performance of the software

(d) encourage particular (hopefully good!) interface styles

These are of course inter-linked: for example, if interfaces are quicker and easier to construct, designers are encouraged to use more iterative design techniques, and also to try multiple design alternatives, which is likely to improve the eventual user interface. (Although rapid prototyping should not, of course, be used as an excuse for inadequate initial design effort.) Also (b) influences (c) in that it limits the kinds of interaction that are available locally for the user and also the overall rate of feedback. The same problems are also found in stand-alone systems and a poor separation between components can reduce the performance of the system and hence create unacceptable interface delays.

The crucial message for this paper is that architectural design is important in user interface design and, by extension, in interactive visualisation design. One of the reasons for the many exciting systems produced by Xerox Parc and Maryland are that both have constructed significant visualisation toolkits and frameworks, the former especially for 3D visualisation and the latter more for 2D dynamic visualisation. However, it is rare to find architectures dealt with in detail in visualisation papers (but there are some, for example, Jern [10]). Algorithms are often described in detail, but architectures rarely. Indeed, in Card, Mackinlay and Shneiderman's excellent collection [2], there are no papers on visualisation architecture.

3. ONCUE – LINKING USERS TO APPLICATIONS

onCue is a commercial product produced by aQtive that gives easy access to the Internet and to desktop applications. It has aspects of an active toolbar, an intelligent portal and a software agent. It watches everything that is copied to the clipboard, uses 'appropriate intelligence' to suggest suitable Internet services and desktop applications, and automates the use of the data in chosen services.

However, for this paper, the most important thing about onCue is that it is extensible. Developers can add their own components (called Qbits) into the onCue framework. Given onCue is able to work with data from any application, this effectively means one has a 'universal plug-in'.

The best way to understand onCue is by an illustrated scenario of the use of onCue from the user's perspective. We will then look at the architecture underlying onCue and how this has helped turn dancing histograms form a demonstration applet into an integrated product.

3.1. onCue at work

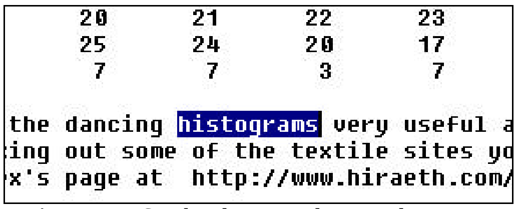
Imagine Sarah at work in the office …

Figure 3.

Sarah starts up onCue. Initially a small floating window appears (figure 3) with a few icons in it allowing her to get information and help about onCue and to set preferences etc.

Sarah then starts to look at her email, she finds a message from a colleague.. The message contains text and also a table (laid out with spaces) as well as the URL of a web page.

Sarah first selects the word "histograms" in the text (figure 4). When she does so, the onCue window changes. Several icons appear in it representing things she may want to do with the word "histograms".

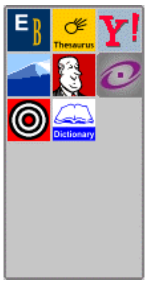


```
    20          21          22          23
    25          24          20          17
     7           7           3           7

the dancing histograms very useful a
ing out some of the textile sites yo
x's page at  http://www.hiraeth.com/
```

Figure 4. Sarah selects word in email message

onCue suggests looking up "histograms" in various online search engines: AskJeaves, Hotbot, AltaVista, and Yahoo; an online thesaurus and dictionary; and also suggests looking it up in the online Encyclopedia Britannica



She clicks the thesaurus icon and onCue launches a web browser and directs it to the thesaurus service which then returns a web page listing similar words such as chart, diagrams etc.

3.2. inside onCue

We'll go through the same scenario again, but this time look at what is happening inside onCue.

When onCue launches it loads a collection of Qbits.

Some of these are integral to the product:

- ♦ clipboard watcher – that watches for the users' cut/copy actions
- ♦ onCue window Qbit – for displaying onCue's suggestions
- ♦ browser Qbit – that is used to send the default web browser to a selected URL

Other Qbits are optional. A configuration file is used to record which need to be loaded and the user can modify this set via the onCue preferences (or edit the files directly if brave!). Some of these are coded in raw Java and some use the XML API, which allows some matching and invocation of web services.

The optional Qbits are of two kinds:

- ♦ recognisers – which use simple heuristics and AI to work out what kind of thing has been copied to the clipboard
- ♦ services – which encapsulate the things that can be suggested to the user

In addition to all these Qbits are the code for the aQtiveSpace, the underlying component infrastructure, and the onCue framework, the code built on top of aQtiveSpace which brings together the other onCue components.

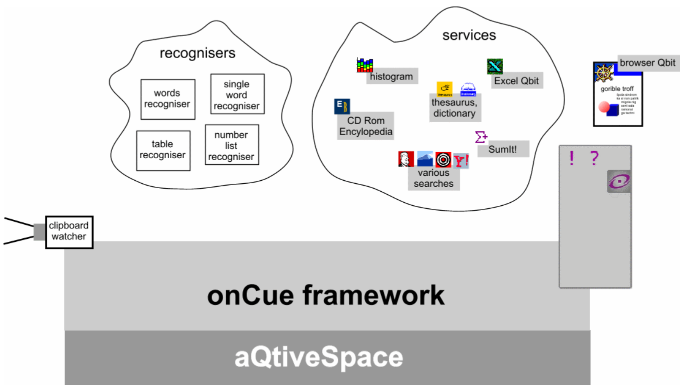


Figure 5. Components in the onCue architecture

In one sense the onCue framework (OCF) is simply another component, but it is special as it acts as the 'glue' between the other Qbits orchestrating their efforts. Furthermore, the other Qbits must be written to special patterns to enable the OCF to link them together.

Each of the services has a data type that it is willing to accept. In this example:

Service	**type**
histogram	table
encyclopedia	words
thesaurus	single word
SumIt!	number list
Excel Qbit	table
Web searches	words

Each recogniser has a type it is willing to look at (in-type) and a type it recognises (out-type). The meaning of these will become clear as we discuss later stages:

Recogniser	**in-type**	**out-type**
words recogniser (Wr)	text	words
table recogniser (Tr)	text	table
single word recog (SWr).	words	single word
number list recogniser (NLr)	text	number list

onCue silently sits in the background, doing nothing except for the clipboard watcher, which simply waits for a copy or cut to happen.

When the user selects and copies the word "histograms", the clipboard watcher notices and passes the copied text to the onCue framework. aQtive Desk looks for recognisers or services that can use the text. The recognisers Wr, Tr and NLr are all activated.

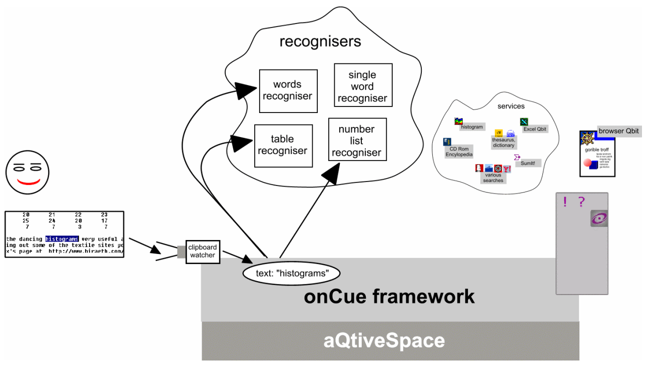


Figure 6. onCue activates recognisers that can accept text

Tr and NLr both fail to recognise the text (it is neither a table nor a list of numbers), but Wr does. The words recogniser simply looks at the text and decides whether it could be considered a sequence of 'words'. It clearly can and so it announces to the onCue that the text can be regarded as words. onCue records this.

Because it now knows the selected text is words it can activate the single word recogniser SWr (this is based on matching the in-type of SWr).

At the same time it activates the web search and encyclopedia services as all of these just expect words.

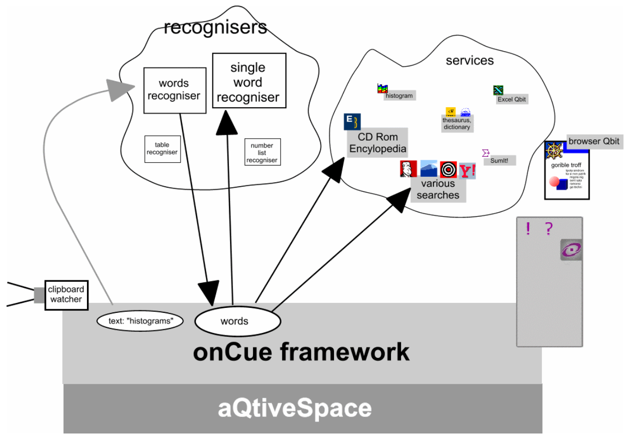


Figure 7. Text recognised as consisting of words

Note that the recognition that the text is a collection of words also involves posting back to onCue data structures that make it easy to view the text as a series of words.

SWr recognises that the words are in fact also a single word (there is but one of them!). It announces this back to the OCF. Now the OCF can activate the thesaurus and dictionary services, as they required a single word each.

Finally, the icons of all the active services are displayed in the onCue window.

Notice that:

- The selection of services offered depends dynamically on the kind of data selected by the user.
- The recognition of the type of the data may take several steps; e.g. text $\rightarrow$ words $\rightarrow$ single word

When the user selects an icon in the onCue window, the OCF goes back to the service and asks it to perform its action. In the scenario this was the thesaurus icon. This Qbit simply generates a URL, which the OCF passes to the browser Qbit, which in turn runs an external web browser to view the page.

The OCF treats services that generate a URL for the browser specially, as they are so common, it asks the service for the URL and then OCF passes this to the browser. Other kinds of Qbits have to do everything themselves!

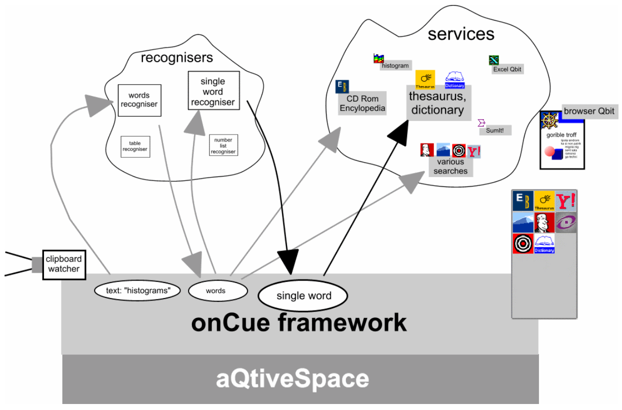


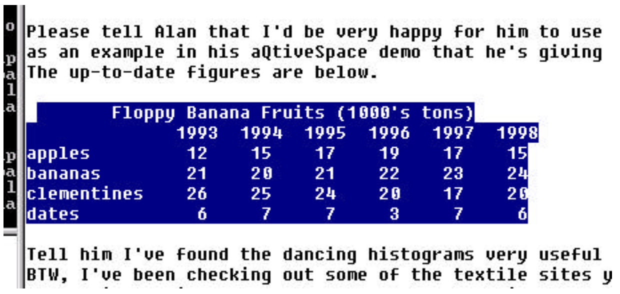
Figure 8. Text recognised as a single word

3.3. dancing histograms in onCue

The applet version of dancing histograms was modified slightly to launch in its own window and conform to the onCue 'service' Qbit API. Also a table recogniser was written that could look at some text, decide whether or not it corresponds to a table of data and, if it does, convert it into an appropriate internal format. This initial conversion took half a working day. Although some additional work was done later to add options to paste the histogram into user's own web pages and improve the table recogniser, the dancing histograms were demonstrable and usable after only the initial small amount of work.

Let's see what this means for Sarah …

After looking at the thesaurus for a while Sarah selects the table in the text (figure 9):



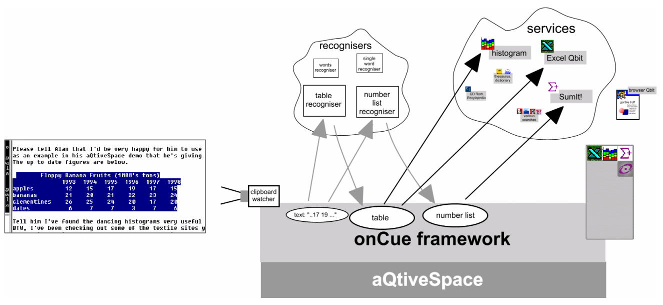
```
Please tell Alan that I'd be very happy for him to use
as an example in his aQtiveSpace demo that he's giving
The up-to-date figures are below.

         Floppy Banana Fruits (1000's tons)
               1993  1994  1995  1996  1997  1998
apples           12    15    17    19    17    15
bananas          21    20    21    22    23    24
clementines      26    25    24    20    17    20
dates             6     7     7     3     7     6

Tell him I've found the dancing histograms very useful
BTW, I've been checking out some of the textile sites y
```

Figure 9. Sarah selects table in email message

Inside, all the onCue Framework knows is that it has seen more copied text. It therefore passes this to the same three recognisers for processing: Wr, NLr and Tr. This time the words recogniser fails to recognise it (too long, split over several lines and too many numbers). However, the number list recogniser (NLr) does recognise it as it ignores the other words and looks for any numbers in the data. The table recogniser also recognises the data as a table.



```
Please tell Alan that I'd be very happy for him to use
as an example in his aQtiveSpace demo that he's giving
The up-to-date figures are below.

        Floppy Banana Fruits (1000's tons)
                1993  1994  1995  1996  1997  1998
apples            12    15    17    19    17    15
bananas           21    20    21    22    23    24
clementines       26    25    24    20    17    20
dates              6     7     7     3     7     6

Tell him I've found the dancing histograms very useful
BTW, I've been checking out some of the textile sites y
```

Figure 10. Text recognised as a table and as containing numbers

This time there are no repeat runs through the recognisers as none of the recognisers can deal with tables or number lists (just generate them). However, two services require tables (histograms and Excel) and one requires a number list. These three services, Dancing Histograms, SumIt!, and Microsoft Excel, are then activated and suggested to Sarah as icons in the onCue window.

She clicks the Dancing Histograms icon and onCue launches the Dancing Histogram window (figure 11)

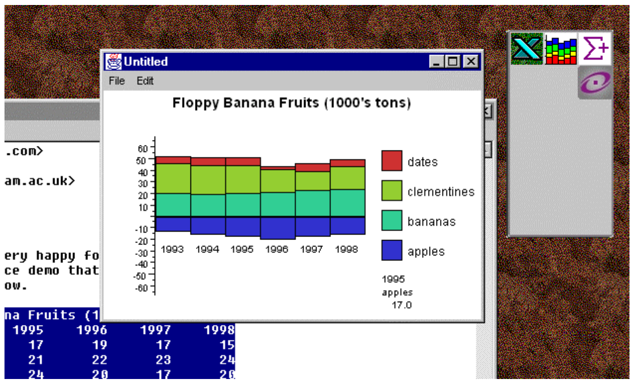


Figure 11. Sarah clicks histogram icon and dancing histogram is produced

In this case both the recognisers themselves and the data structures they post back are more complex.

For the table this includes:

- Title of table (where present)
- Number of columns
- Number of rows
- Column labels (where present)
- Row labels (where present)
- Numerical table data

However, the same principles hold as for the simple word recognisers. The text is recognised as having a certain form and the fact that it does together with transformed data are 'announced' to all Qbits that can use this type of data.

3.4. appropriate intelligence and the table recogniser

The table recogniser is in fact, one of the most complex individual recognisers within the current version of onCue. Many types of data can be recognised with fairly simple rules or regular expressions. For example, UK postcodes consist of one of a small number of patterns of letters and digits, names are small numbers of words or initials with initial capitals. None of these recognisers are perfect, they may occasionally miss an unexpected form of the type of data. For example, aQtive's business cards were misprinted with the postcode B15 25Q instead of B15 2SQ. This is still recognisable to a human eye as a UK post code, but does not conform the correct syntax. Although more semantics-rich artificial intelligence techniques could be employed to improve the recognition rate, none would be perfect. Furthermore, such algorithms would take far too long and consume too many resources for a desktop tool.

Where some form of artificial intelligence is being used without human intervention, for example, a neural network used to control a nuclear fusion chamber [9]. In such case the algorithms used must either be perfect (which of course won't be the case) or limited by simpler fail safe rules. However, user interfaces have a human at hand, indeed the purpose of the intelligence is to work with the user. Humans are used to dealing with uncertain information and fallible colleagues. This means that the raw algorithms do not need to be perfect, but instead they're weaknesses can be mitigated by well-designed interaction.

At aQtive, we use the term *appropriate intelligence* to refer to the use of simple rules and heuristics set in a fine tuned user interaction paradigm. The crucial things are that the interaction should

(a) be useful when it is right

(b) be right often enough to be useful

(c) *not* cause problems when it is wrong

To see examples of these principles, consider two uses of heuristic intelligence in popular Microsoft tools. The Microsoft Office paperclip is notorious. Although it often makes useful suggestions, it is modal: just at the moment that you enter full writing flow, it appears, with some suggestion and prevents further typing until it is explicitly dismissed. So although it satisfies (a) – it is useful – it fails (c) as it is costly when wrong. In contrast, consider the sum Σ button in Microsoft Excel. Around 80% of all spreadsheet use is the simple summing of columns of numbers. When the sum button is pressed it inserts "sum()" in the cell formula and then selects any contiguous cells with numbers in above the current cell (or to the left). It has some extra rules to deal with sub-totals, but is using relatively simple heuristics. When it gets it right it saves selecting the relevant cells – some benefit (a). Also because most use of summing is for simple columns it will get it right for a lot of people a lot of the time – (b). Finally, consider what happens when it selects the wrong cells, If the user simply ignores the selection and attempts to select a different range, the automatically selected range is overridden using the normal selection process. That is there is virtually no cost if the system guesses wrong – condition (c).

onCue attempts to use appropriate intelligence. The suggestion mechanism in the onCue window is deliberately low key, the window is non-modal, it does not flash or in any way force itself on the user's attention. The aim is to be available, but not intrusive. If, for example, the user selects a company name "Cooper Diamonds", this may be incorrectly recognised as a name and directory services offered by onCue. However, the user can simply ignore these icons. That is, we are attempting to carry a very low cost of failure – (c). In addition, the services offered are useful in various ways, either as short cuts to known Internet services and desktop applications, or in suggesting previously unknown ones – condition (a). The goal for the recogniser heuristics is therefore to be right often enough – condition (b).

The table recogniser is designed with these constraints in mind. The initial (half-working day) version simply looked at the length of lines. Each line was split into 'words' by tabs or spaces and the length calculated. Two patterns are expected:

type I:	optional line of any length (title) line of length N (column headers) several line of length N+1 (row title plus data)
type II:	optional line of any length (title) several line of length N (possible row title plus data)

If the data did not conform with either of the above it was rejected. If it did then the words were checked to see which were numbers to verify in type II whether the table consisted of plain data, data with just column headers or data with just row headers.

Although this worked well with tabbed data it didn't deal with tables with multi-word row labels or with numeric column or row labels. However, tabbed data is already reasonably easy to use with existing tools. So this recogniser failed on condition (b) as it was least often right when it would have been most useful!

A second version of the recogniser (which did take longer than a half-day, but not too much more!), used more complex rules to grow a window of numbers from the bottom right corner of the potential table and used scoring to distinguish different potential ways of regarding the data as a numeric table. It has rules like:

- if the top line is very short it is probably a title, even if it is a number such as "1998"
- if the top line is one shorter than the rest it is probably column headings
- if most rows have N numbers and one has N+1, the extra number is probably part of the column label

The rules used are specific enough that it does not recognise non-tables, but general enough to be able to deal with explicit tables in spreadsheets and word processors, simple tabbed tables, spaced plain-text tables (as often use in email messages) and also tables laid out in web-browsers. The latter is especially important. Although web page tables are produced using explicit tags, they are typically reduced to spaced text when copy/pasted.

1.5. making the most if it

It is at this point that the value of the onCue recogniser/service structure is brought to bear. Although the table recogniser was built for the dancing histograms, it then becomes a part of the onCue world that other services can use. Although spreadsheets typically allow many graphing options, users are often uncertain quite what is available and how to use it. Also, if one copies data that is not in an appropriate format (for example a table of data copied form a web page) into a spreadsheet, the effect is often disastrous, often putting the entire line into the first column of each row!

To help users with this, another Qbit acts as a simple link to Microsoft Excel. It is activated by the same data produced by the table recogniser as dancing histograms. So, when a table is selected, the Excel icon is also offered to the user. If she selects this onCue converts the data into a form suitable for Excel and uses COM to tell Excel to create a new sheet and draw an appropriate graph.

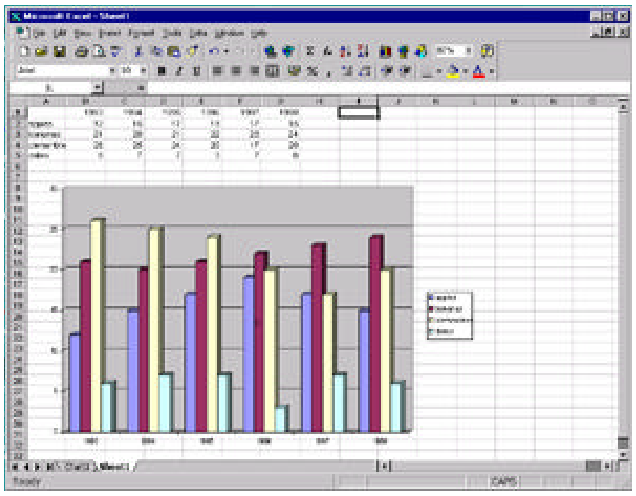


Figure 12. Sarah selects Excel icon and an Excel chart is produced

It is now possible to select a table in a web page, copy it and two clicks later have an Excel or dancing histogram chart. Moreover, because the same recogniser activates the different services, the work required is modest and the effort is reduced the more Qbits that are available already. Although re-use is often encouraged, it is comparatively rare in practice, except in very generic libraries. One of the reasons for this is that software reuse is often focused on methods or functions, whereas reuse in onCue is focused on shared data representations. Also, important is the form of the lower-level component framework upon which onCue is constructed. We will look at that in greater detail in section 5.

4. EXAMPLE 2 – PIETREES

We have seen how onCue was used to make an existing visualisation more accessible and more integrated with other applications. In addition, we are using onCue as a platform for the development of other visualisation techniques. In this section, we'll look at pieTrees a novel method for interactive visualisation of hierarchical numeric data sets.

In our AVI'98 paper Geoff Ellis and I suggested different ways in which simple interaction could enhance traditional representations. One of these was the pie chart. The paper suggested how segments of pie charts could be 'exploded' in concert with an outliner view of the data. This technique works for strict hierarchical data such as regions where each region is exactly decomposed into sub-regions etc. The number of people, factories or daffodils in any region is exactly equal to the sum of the number in each sub-region, hence the segment for the region can simply fan open.

However, there is a second form of hierarchical data where there are values associated with non-leaf nodes in addition to those at the leaves. One example of this is file system usage. The amount of space in the "documents" directory is the sum of the space used by the "documents/avi200", folder, the "documents/hci99" folder etc. *plus* the size of the files directly stored under "documents". Web usage follows a similar pattern: there are a number of visits to of the 'home page' for a portion of the site as well as visits to pages beneath that point.

pieTrees are a way of visualising such data extending the notion of exploding pie charts.

4.1. what are pieTrees

Suppose we have the following web usage statistics in visits per hour on a web site:

```
corporate/          4000
    info.html         3000
    press.html        2000
community/          3000
    user.html         2000
    research.html     1000
    developer.html    2000
product/            2000
    onCue/            2000
          online.html         1000
          download.html       1000
    vfridge           1000
    support           1000
```

Each number refers to the number of hits to the page, or to the index page if it is a directory. For example, there are 2000 hits per hour to corporate/press.html. The total number of hits to each region of the site can be simply calculated by summing everything below that level. So, the total hits to the community region is 3000+2000+1000+2000. Note including the community index page itself.

These region sums for the top-level regions can easily be represented as a normal pie chart (figure 13)

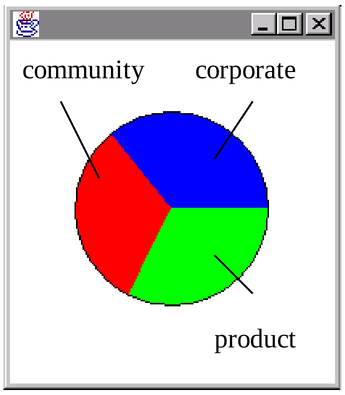


Figure 13. Pie chart of hits by region.

The parallel outliner view can be folded/unfolded and the pieTree segments will split/merge as the outlines are shown or hidden. However, the segments cannot simple split, as there is data associated with the index page of each region. Instead, when the products region is expanded, the pieTree retains a small pert-circle segment towards the center, representing the hits to the index page of "products" with smaller segment portions outside of this (figure 14). The sizes of these are chosen so as to preserve the proportionality between area and numbers of hits of a normal pie chart.

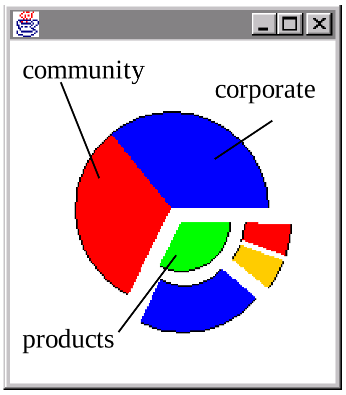


Figure 14. Opening products

As the user drills deeper into the data, the process continues (figure 15).

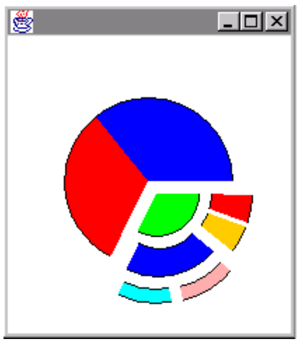


Figure 15. Drilling deeper

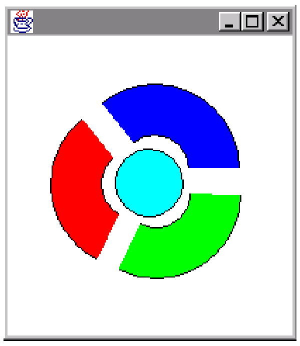


Figure 16. Root hits

Of course, web sites will usually have hits to the top-level home page and where this is so a small circle in the centre (figure 16) represents these.

The strengths of this representation are:

(a) preserves equal area/ equal value
(b) similar to existing pie charts
(c) small visual change as regions are opened/closed
(d) temporal fusion [6] induced by simultaneous change of outline/pieTree

Although it also has some problems:

(e) hard to judge areas (as with ordinary pie charts)
(f) poor when there is very little value at interior nodes
(g) becomes less useful towards the leaves (becomes lots of small rectangles)

In fact, web statistics are particularly well suited for this representation as index pages for regions of the site often have hit counts comparable with the whole of the region below. In particular, some sites enforce a strict menu-down/up style if interaction, which may mean more index page hits than all the pages below!

4.2. pieTrees and onCue

The same structure of recogniser/mini-application is used for pieTrees as for dancing histograms.

The hierarchical data recogniser needs to be able recognise indented data (as with the web usage data above), and also 'prefix' data as in the UNIX disk usage (du) listing in figure 17.

```
20      ./search
286     ./images
72      ./exercises/images
34      ./exercises/maths
54      ./exercises/projects
486     ./exercises
8       ./glossary
102     ./links
18      ./misc
102     ./overviews
2       ./prototyping
132     ./text
54      ./URLs
1582    .
```

Figure 17. UNIX 'du' (disk usage) listing

Heuristics used in the hierarchical data recogniser include:

- Is the first element in each line a number (like du)?
- Is the last number in each line a number (like web log)?
- Are some of the non-number items prefixes of others (e.g. "/exercises" and "/exercises/images")?
- Are some lines indented with spaces or tabs?
- Are the numbers associated with higher levels always greater than the sum of the lower levels?

The last of these rules is important in order to distinguish cases where the application has to produce higher level total (as with logs) or where the data has already been summed and the system needs to work out the differences (as in du). In the latter case, the disk used in files at the top level of "/exercises" must be 326 = 486-(72+34+54).

Although some care has to be taken in establishing such rules the result is (as with dancing histograms) that data from large varieties of sources: web pages, word-processor documents, spreadsheets, email messages, terminal sessions, can all be simply visualised with literally two mouse clicks.

4.3. related visualisations

pieTrees are closely related to three other visualisation/data exploration methods. First is Tree-Maps [11], which layout file-system size or similar data in a 2D, rectilinear structure where, like pieTrees, area corresponds to size. Tree-Maps use alternate vertical and horizontal slicing to represent levels of hierarchical structure. pieTrees have the advantage of leveraging off users' existing understanding of pie charts including exploded segments, however, the behaviour of pieTrees towards deep leaves is less good requiring interactive refocusing on subnodes to produce pieTrees of subtrees.

The second method is Disk Trees [3], which use a circular representation where the angular arc of each subtree is proportional to the number of items within the arc. This representation is used specifically to analyse web-usage data (as was a driving application for pieTrees) and uses line thickness to represent number of link traversals (ignoring cross-structure links). The difference between the Disk Trees and pieTrees is a simple trade-off. Disk Trees focus first on *number* of nodes and represent value as a secondary feature. This causes some problems of overlap towards the leaves of deep nodes and leaves have large values. pieTrees focus first on value with number represented implicitly by the number of segments. This causes some problems towards the leaves if some deep subtrees have very small total values and hence become much thinner than others. One major lesson from "starting simple" [7] was that interaction allows user selection of trade-offs, so perhaps it would be possible to dynamically choose representations that weight number and value with Disk Trees and pieTrees simply being ends of a continuum.

Finally, HIBROWSE [8] makes extensive use of multiple representations of fold/unfolded hierarchical and taxonomic structures with dynamic numerical data. This is not a graphical technique, but is highly effective in using interaction within a textual output medium and is a major inspiration behind pieTrees.

In addition, there are many visualisation techniques used for plain hierarchies, that is, for visualising the hierarchies themselves, not numeric data associated with them. These include cone–trees [17], the PDG tree-browser [12] and the hyperbolic browser [13]. All are currently especially important given the interest in web site management and analysis.

5. LOW-LEVEL ARCHITECTURE

onCue is built upon a lower-level component architecture called aQtive space. aQtiveSpace is a software framework for producing context-sensitive applications from small components, which we call **Qbits**. It is particularly suited for systems that dynamically reconfigure themselves as new Qbits are added. aQtiveSpace is used as the underlying framework for various aQtive products, in particular onCue. aQtiveSpace is itself built using Java.

aQtiveSpace has been developed from a strong theoretical standpoint. It builds on earlier work on Cameo an architecture for context-sensitive applications [21,22]. In addition, both aQtiveSpace and Cameo before it were heavily influenced by status-event analysis [5]. The discrete nature of computation means that at an implementation level everything reduces to events and the system's response to them. This is reflected in the majority of specification notations and implementation platforms. However, many aspects of the physical world are of a different kind, status phenomena, which always have a value that can be sampled. In context-aware applications, such as onCue many of the contextual elements are better viewed as status rather than prematurely decomposed into events. The primitives in aQtiveSpace, although by their nature discrete, are designed to enable an effective and natural encoding of status phenomena. This makes it easier to build context-aware applications, such as onCue, on top of aQtiveSpace. Also the direct mapping between intended user interface behaviour and underlying architecture means that the resulting systems behave correctly and consistently.

5.1. Qbits

The components in aQtiveSpace are called Qbits. In quantum mechanics, qubit refers to a property that is effectively half a bit. A single qubit carries no information in itself, but, when combined with suitable other qubits, does yield useful information. In a similar fashion the Qbits in aQtiveSpace are usually impotent individually, but when combined yield substantial power to the user.

Each Qbit in aQtiveSpace has a series of named **Nodes**. The nodes act somewhat like the named attributes and methods of an object, but with some differences and additional semantics. The nodes are like plugs and sockets by which the Qbit can be connected to its environment and to each other.

5.2. nodes

Each node performs one or more of 6 kinds of interaction:

set	– a value can be given to the node (e.g. setting an attribute)
get	– a value can be requested from the node (e.g. getting the value of an attribute)
call	– the node can be called as in a normal object method call
listen	– the node can give a value to a 'settable' node
give	– the node can request a value form a 'givable' node
supply	– the node can invoke a 'callable' node

Table 1. Node interactions

These interactions can be classified in two ways:

by data Flow – In the case of set and give, data flows into the node. In the case of get and listen data flows out from the node. In the case of call and supply the flow is bidirectional.

by initiative – In the case of set, get and call, the control comes from the outside (external initiative), another Qbit (or arbitrary Java code) has invoked the relevant set, get or call method on the node. In the case of listen, give and supply, the control comes form within (internal initiative) as the node invokes the appropriate intercation when it is ready.

The internal initiative interactions correspond to 'callbacks' found in many systems. They each have a means (in the reference implementation, listen, give and supply methods) of establishing a connection to one or more other nodes and they then invoke those nodes when ready.

The interactions can be matched in pairs as each internal initiative interaction has a corresponding external interaction. For example, a listenable node is given a settable node in its listen method. It invokes the set method on the node everytime it is ready to donate a value.

Table 2. Interaction characteristics

interaction	data flow	initiative	pair
set	in	external	
get	out	external	
call	bidirectional	external	
listen	out	internal	set
give	in	internal	get
supply	bidirectional	internal	call

The input and output data of each node (where relevant) are also typed (e.g. number, text, image).

5.3. plug and play

The nodes of one Qbit can be connected to another where they are compatable (i.e. they can function as complementary pairs and have compatable types).

We can represent the node interactions s a 'Lego' block where control flow runs from left to right, input is represented by a hole (wanting to be filled) and output asa peg (figure 18).

In this representation, two nodes can be connected if they have a corresponding hole/peg combination (and to take the analogy further, if we regard tpyes as the shape of peg, then the shapes must also correspond).

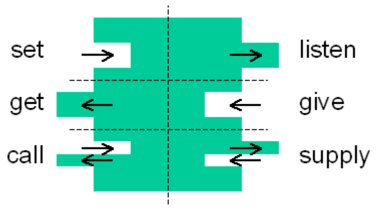


Figure 18. Node as a Lego block

The method by which nodes are described means that Qbits can be dynamically connected together by other Qbits or program code. This is different from, for example, object oriented programming languages, when objects tend to 'know about' a lot of other objects. Of course, Qbits may have internal structure, including other Qbits, but their external behaviour is very like a Lego brick that can be freely connected to others.

Although, connections can be established statically when an application is configured, the plug-and-play nature of Qbits mean it is particularly easy to connect them and disconnect them form one another while a program is running.

5.4. asynchronous interactions

All the node interactions have synchronous and asynchronous versions. For example, one can invoke the 'set' method and wait for the method to return indicating that the set was successful (synchronous). Alternatively, one can use a variant that establishes the request to set the node, but allows the 'setting' Qbit/code to continue to execute (asynchronous). This asynchronous version of 'set' can be thought of as a sort of 'fire and forget' mode. For 'call' and 'get', where a return value is required, a 'callback' can be registered for the value when it is ready (this may be a 'settable' node, or a Java callback object).

These asynchronous interactions are particularly useful in accessing Internet information resources where there may be considerable delays if Qbits access data on different parts of the local network or Internet.

5.5. onCue in aQtive space

As mentioned above, the aQtiveSpace component model was driven by the theoretical concepts of status–event analysis [5]. This emphasises that there are status relationships that continually hold in interfaces, such as the relationship between the outline view and pieTree, as well as things that happen at particular times (events). Although the low-level implementation of any computer interfaces is ultimately event based, the conceptual model should involve status phenomena and the implementation framework should allow an easy mapping between the two. The alternative initiative interactions within the aQtiveSpace component model are designed specifically to allow flexibility in the mapping between status relationships and the underlying event structure.

Ultimately, the whole of the onCue interface can be seen as embodying a form of status-status relationship – the icons that show in the onCue window should always be the relevant ones for the data currently in the system clipboard. Typically clipboard data is accessed within applications in a demand-driven manner. The user selects 'paste' and the application 'asks' the operating system for the data from the clipboard. However, in onCue we require the opposite form of initiative, when the data in the clipboard changes, we need to react in a *data-driven* manner to update onCue and maintain the status–status relationship.

onCue also incorporates Internet awareness Qbits which can tell other parts of the system whether the user is currently connected to the Internet or not (important for mobile systems or home-computers with modems). There is another status–status mapping here: the true/false value of the Internet awareness Qbit should *always* reflect the current Internet connectedness. However, this is not continually visible to the user, is expensive to test and is only required at specific times when other Qbits are about to do actions that require the Internet. The Internet awareness Qbit is therefore implemented using *demand-driven* features in aQtiveSpace, only checking the connectedness status when asked.

6. CONCLUSIONS

There are three principle results in this paper.

First is the pieTree, a simple interactive visualisation technique for hierarchical numeric data, especially useful for web log and file system data.

The second is the suitability of onCue and the underlying aQtiveSpace infrastructure as a platform for implementing such visualisations. The separation of recognisers and services allows easy integration between applications within onCue and other desktop and Internet applications. This has allowed dancing histograms to effectively become a universal plug-in. It is possible to copy a table from a web page, view it as a dancing histogram and then paste the live histogram back into a new web page or as a static image in a document.

However, the third result is the most important, not the success of the particular onCue architecture, but the general importance of software architecture in effective visualisation construction and design.

7. REFERENCES

[1] Brown C., Benford S. and Snowdon D. (1996). Collaborative Visualization of Large Scale Hypermedia Databases.*ERCIM workshop on CSCW and the Web*, (Sankt Augustin, Germany), Arbeitspapiere der GMD 984, GMD/FIT. pp. 115–123

[2] Card S.K., Mackinlay, J.D., and Shneiderman, B. (1999). *Readings in Information Visualization – using vision to think*. Morgan Kaufmann.

[3] Chi, E.H., Pitkow, J., Mackinlay, J., Pirolli, P., Gossweiler, R.and Card, S.K. (1998). Visualizing the evolution of web ecologies. *Proceedings of CHI98*, ACM Press, pp. 400–407

[4] Coutaz, J. (1987). PAC, an object oriented model for dialogue design. *Human–Computer Interaction – INTERACT'87*, Eds. H.-J. Bullinger and B. Shackel. Elsevier (North-Holland), pp. 431-436

[5] Dix, A. and Abowd, G. (1996a). Modelling status and event behaviour of interactive systems. *Software Engineering Journal*, **11**(6) pp. 334-346.

[6] Dix, A.. (1996b) Time, space and interaction *Proc. of FADIVA 3*, Gubbio, Italy, University of Rome. pp 99–103, `http://www.comp.lancs.ac.uk/computing/users/dixa/papers/FADIVA/`

[7] Dix, A., and Ellis, G. (1998). Starting Simple - adding value to static visualisation through simple interaction. *Proceedings of Advanced Visual Interfaces – AVI98*, Eds. T. Catarci, M. F. Costabile, G. Santucci and L. Tarantino. L'Aquila, Italy, ACM Press. pp. 124–134.

[8] Ellis G.P., Finlay J.E. and Pollitt A.S. (1994) HIBROWSE for Hotels: bridging the gap between user and system views of a database *IDS'94 2nd International Workshop on User Interfaces to Databases*, (Ambleside, UK, April 1994) Springer Verlag. Workshops in Computer science, pp. 45–58

[9] Greake, E. (1991) Neural network keeps fusion plasma in shape. *New Scientist*, page 27, 12th October 1991.

[10] Jern, M. (1996). "Thin" vs. "fat" visualization clients. *Proceedings of Advanced Visual Interfaces – AVI98*, Eds. T. Catarci, M. F. Costabile, G. Santucci and L. Tarantino. L'Aquila, Italy, ACM Press. pp. 270–273

[11] Johnson, B. and Shneidermann, B. (1991). Tree-maps: a space filling approach to the visualisation of hierarchical information structures, *Proc. of IEEE Visualization'91 Conference*, San Deigo. pp. 284–291

[12] Kumar, H.P, Plaisant, C., and Shneiderman, B. (1997). Browsing hierarchical data and multi-level dynamic queries and pruning. *International Journal of Human–Computer Studies*, **46**(1), pp.103–124.

[13] Lamping J. and Rao R. (1995) The Hyperbolic Browser. A Focus+Context Technique for Visualizing Large Hierachies *Journal of Visual Languages and Computing*, **6**(4)

[14] Lewis, S. (1995). *The Art and Science of Smalltalk*. Prentice Hall

[15] Pfaff, G., and Hagen. P., (Eds.) (1985). *Seeheim Workshop on User Interface Management Systems*, Springer-Verlag, Berlin.

[16] Ramduny, D., and Dix, A. (1997) "Why, What, Where, When: Architectures for Cooperative Work on the World Wide Web", Proceedings of HCI'97, Springer-Verlag, pp. 283–301

[17] Robertson G.G., MacKinlay J.D., and Card S.K. (1991) Cone Trees: Animated 3D Visualizations of Hierarchical Information. *Proceedings of CHI'91* (New Orleans, April 1991) ACM Press, pp. 189-194

[18] Tweedie, L., Spence, R., Dawkes, H., and Su, H.. (1996). Externalising abstract mathematical models. In *Proceedings of CHI'96*, ACM Press, pp. 406–412.

[19] UIMS (1992). The UIMS tool developers workshop: A metamodel for the runtime architecture of an interactive system, *SIGCHI Bulletin*, **24**(1), pp 32-37,

[20] Williamson, C.. and Shneiderman, B.. (1991) The Dynamic HomeFinder: Evaluating Dynamic Queries in a Real-estate Information Exploration System SIGIR'92 *Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, ACM Press, pp. 338–346

[21] Wood, A., Dey, A. K., and Abowd, G. D. (1997). CyberDesk: Automated Integration of Desktop and Network Services, *Proceedings of CHI'97*, ACM Press, pp. 552-553.

[22] Wood, A. (1998) CAMEO: Supporting Agent-Application Interaction, PhD Thesis (University of Birmingham, UK).

8. GETTING ONCUE

onCue is available from:

`www.aqtive.com`

more information about the onCue architecture can be found at:

`www.aqtive.com/community/research`