A DESCRIPTION OF THE ADAGE PROJECT

J. ALEX STARK

OUTLINE

This is a short description of the Adage project. It describes: (1) the motivation behind the project; (2) application areas for which it might be used; (3) scenarios in which it might be invoked; (4) the general design strategy; and (5) its main features. This introduction is quite terse, and there are no pictures: the project website provides example drawings, executive summaries, and much more information besides.

CONTENTS

For further information email `info AT mdag DOT org`.

1. Concept

The vision behind the Adage project is to provide two complementary systems. The first is a stand-alone interactive drawing package. The second is a library-style package that can be used by programs to draw graphical output. The project is organised around a central engine with a custom-made data manipulation language. This language is designed for modularity and extensibility, and the majority of project, at least in term of code quantity, consists of extension modules and scripts.

Note that this document is written in the present and past tenses, even when referring to things not yet created: it is documentation in advance.

1.1. **Unity and diversity.**

There are many open-source and research programs being distributed today that produce graphical output. Such programs typically use similar drawing elements. For instance, a program that plots an audio signal and a program for drawing flow diagrams both use mainly lines, points and text. The needs of a program that analyses DNA and plots trees of descent are much like those of a program that plots the exchange rates between currencies.

In view of this, it is surprising that a truly flexible and reusable drawing engine has yet to be created. Presently, programmers primarily concerned with other tasks have to write their own output generators. This is wasted effort: a solution off the shelf would be much better. An important aim of the Adage project is to meet this need.

1.2. **An integrated system.**

The interactive program can be used not only to create and edit drawings, but also to design macros, scripts and drawing templates. These can in turn be used to build extension libraries, or to serve as templates for master programs to use. Consider the following scenario:

> A master data analysis program includes commands that are used to plot a graph. The Adage engine, which is linked as a slave library, provides this ability. A "smart export" function is used to save the data and graph details with tagging information. The user launches the interactive Adage program separately and adds annotations, creating an enhanced plot. Then a batch file is written that invokes the (first) data analysis program on multiple data sets, such as monthly results. New data is exported for each. The stand-alone Adage program is invoked on each results file, generating enhanced graphs that *include* the additional annotations.

This is not an exceptional example. However, no current drawing package supports it. Adage only requires that the master program can export data with consistent tagging information, and the rest is made straightforward by the basic design of the Adage language.

1.3. **Extensibility and flexibility.** Many of the most successful software projects have been based around a community of developers. Examples are Gimp, Perl and Latex. A characteristic of these is that scripts, modules and plug-ins build on a well-designed core. The Adage package makes even greater use of extensibility. It has a small core that is not specific to the task of drawing. The vast majority of function is derived from this by extension. This is done not only to facilitate a community effort, but also so that the engine can be used in varying scenarios.

The flexibility of the Adage system is supported by a characteristic of its drawing files, non-GUI commands, the data structure for drawings, and the code that describes drawing elements. All four have the same format, differently expressed. For instance, when Adage saves a drawing file it basically writes a set of text commands that can be invoked to recreate the drawing. A new drawing element can be created within the interactive program for a library using a similar process. The first step is to draw an example using a small set of parameters such as points and lengths. The second step is to group the drawing elements. Adage automatically identifies the parameters on which the group depends. The third step is to export the group of elements as a function.

2. Applications

The potential range of drawing applications was an initial motivation for the Adage project, and surveying applications became a key part of early work on it. It has proved particularly important to keep in mind the diversity and similarity between applications when making basic design decisions. Here the applications have been organised loosely in three groups: plots and charts, drawings with nodes and connections, and more intricate drawings that represent real objects.

2.1. **Plotting and charting.** Drawings are often made to illustrate data from two kinds of source, in one case from measurements, in the other from a mathematical function or computer program. In some ways this distinction is not important to us, especially since artificial measured data is often generated by computer simulation. In both cases a sequence or set of data values is to be represented graphically. However, there are a number of different genres of drawing, and we can crudely categorise them as either plots or charts.

In a *plot*, values are drawn against others, the most common being the simple 2-dimensional graph. Styles of plot include connected lines, points, impulses and filled areas. More elaborate features include statistical error bars. Graphs also comprise axes, grid lines, tic marks, legends and labels.

In a *chart*, values are displayed in a more disconnected fashion are used more for collections than sequences of data. The types of display are more varied than with plots. They include pie charts and histograms, and are often seen in newspapers and reports.

2.2. **Getting connected.** A surprisingly wide range of drawings are in essence composed of connected nodes. In an organisation chart, a node is a person, job function or organisation subgroup. The arrangement and connections show the hierarchy, reporting structure, or other interactions.

Organisations and their activities are often represented using connected nodes. Project plans, time-lines and dependencies, business processes, reporting and communications structures all take this form.

Many diagrams are composed of functional blocks with inputs and outputs. Examples are flowcharts, system diagrams and process illustrations. Schematics for electrical, electronic, plumbing, pneumatic and hydraulic systems all have broadly this form.

Software design makes increasing use of connected drawings. And tree structures also come under this category, as do network diagrams and topologies.

2.3. **Representing reality.** Many of the kinds of drawing described above represent reality in some way. However, a separate category can be usefully drawn for the more elaborate task of drafting. It includes engineering drawing and architectural drawing. The distinguishing characteristic is that, although diagrammatic, there is a direct correspondence between elements on the page and physical reality. Distance on the page corresponds to distance in space.

Any attempt to develop a package that can handle drawings of this complexity must be considered a long-term project. Nevertheless, the Adage system is designed with this future potential in mind. In the shorter term, the project does aim to provide libraries for simple drafting.

3. Means

Some mention was made previously of design and implementation. We now focus attention on some details.

3.1. **Building blocks.** The basic building blocks of Adage drawings are lines, points, areas, and text. Lines have a variety of patterns and widths; areas can be filled with colour and patterns; and there is a set of points, some of which are small versions of drawing elements such as circles and squares. Some output devices support slightly more complex elements such as circles and rectangles.

The basic elements are built into more complex elements in a hierarchical fashion. A rectangle can be defined as a closed polygon which in turn can be broken up into a set of line segments. An architectural element for a washing machine might comprise polygons and circle arcs. A drawing is composed of elements realized on a composition frame which is rendered on a page. Modification and transformation operations are provided. These include rotation, scaling and translation.

Function specific to interaction is provided. Examples include snapping points to a grid and restricting vector angles.

3.2. **Language and data.** The Adage system is built upon the POStiche type system and MDAG data manipulation system. POStiche is also known as the *Protean Object System*. Within it data objects have both a major type and a minor substype. An example might be a rectangle expressed as the centre point, width and height. The major type is the rectangle, and the specific set of data fields is one subtype. Another subtype might be the bottom-left and top-right corners. The object system is protean in that the same information can be represented in many different ways. POStiche has rules that determine how objects are transformed, combined and broken up.

The *MDAG* (M directed acyclic graph) system is in large part an implementation of the POStiche system. Drawing elements are defined in a hierarchy. Suppose, for instance that a rectangle is defined using the polar coordinates of its centre point and the lengths of its width and height. Four raw values are combined with length and angle units to create one angle and three length objects. The angle and one length object are combined into the polar subtype of the coordinate major type. This and the other two lengths combine to make the rectangle. Within the MDAG system, at the level of drawing definition, a node is used for each object, unit choice, or raw data value. Directed edges (node links) indicate the dependencies and combinations. This hierarchical framework of POStiche objects is the basis for the MDAG system.

The MDAG system has some sophisticated capabilities for handling transformations, and for optimising them. For instance, all lengths are converted to universal units. We might start with some lengths specified in centimeters and others in millimeters, and these might be scaled then converted to inches. The optimisation engine is reliable in collecting transformations together.

3.3. **Engine.** The base layer of the Adage implementation is the MDAG system in combination with the Adage language. This language is used across the whole MDAG system for a variety of purposes. First, POStiche types and their transformation operations are defined using it. Second, drawings are specified as MDAG nodes and connections. Third, macros, drawing elements and transformations are defined as MDAG nodes and connections. Hence, a drawing is realized by recursive breaking down until a network of irreducible nodes is reached. Fourth, interactive operation is implemented as a special set of transformations and irreducible nodes. Fifth, file formats are ordinary commands that rebuild the network of nodes at the uppermost level. Sixth, batch scripts and master programs using Adage as a slave library utilize special invocation commands that are realized as nodes.

Thus the main engine is an implementation of the Adage language that manipulates MDAG networks and POStiche objects. All drawing elements are provided in libraries that build on this foundation. Libraries at the lower level define basic drawing elements. At the top level are elements that are specific to applications. Specialized nodes provide external interaction. Through these a library of Adage functions provide for GUI operation and use as a slave library.