TABS - A new software framework for document image processing, analysis, and understanding.

C. Cracknell, A. C. Downton, and L. Du
Department of Electronic Systems Engineering
University of Essex
Wivenhoe Park, Colchester CO4 3SQ, U.K.
e-mail: craccb@essex.ac.uk

Abstract

This paper provides an overview of a new software framework, TABS, which has been designed to support the rapid development of image processing and image analysis systems and components. Compared to other image manipulation software frameworks, TABS has a number of novel features which make it particularly suitable for use in applications where hypotheses rather than single "hard" results are generated by system components, and symbolic data are manipulated.

1 Introduction

Image analysis systems typically consist of a number of image processing and image analysis components which are connected together in a sequential manner. A *framework* dictates exactly how components may be connected, how data are to be transferred between components, and how components and systems may be developed.

Existing frameworks have a number of deficiencies which limit their suitability for use in certain domains (for example, support for components which generate a ranked list of result hypotheses is usually particularly poor). TABS [1] has been designed to attempt to overcome some of these deficiencies.

2 Existing frameworks and their limitations.

2.1 Khoros.

Originally developed for research in image processing, Khoros [JK94] is now being used in a variety of scientific applications. Khoros is a rather graphical framework, and includes a visual programming environment (VPE) called Cantata. Khoros contains a standard set of over 200 components for image processing, each of which exists as an independent executable program capable of being invoked either from the command line, or from within Cantata.

In several respects, Khoros is quite a good environment for developing image analysis components. It has serious weaknesses however, some of which are given in the list below.

1. Khoros components must be written in old style (Kernighan and Ritchie) C.
2. An interface description (written in a special "UIS" language) must be provided for each component.
3. It is far more difficult to implement and visualise potent control structures within Cantata than it would be using a language (for example, C).
4. The default Khoros data file format (VIFF) is overly complicated for simple 2-D image processing applications, and, since many standard Khoros components use VIFF exclusively, conversions to and from a more application-suitable format may be required with in a system.
5. Component development takes place outside of the Cantata visual programming environment.
6. Creating components which employ a graphical user interface is very difficult and involves knowledge of the xv library.
7. Khoros is slow, memory intensive, disk space intensive, and buggy.

[1] TABS is an acronym for "The Almost a Blackboard System system". This rather obfuscated label was chosen due to the author's initial desire to create a framework based on a blackboard architecture [Jac94].

0-8186-7898-4/97 $10.00

2.2 KBVision.

KBVision is a commercial product primarily designed for image understanding problems. Like Khoros, KBVision provides a visual programming environment and comes with a set of standard components.

Also supported by KBVision are a set of *intermediate symbolic representations* (ISRs). These allow manipulation of non-pixel data, for example, polygons, etc.). KBVision is currently used by a few researchers concerned with computer vision and image understanding problems. More details on KBVision can be found on Amerinex's world-wide web pages:

`http://www.aai.com:80/AAI/KBV/KBV.html`

Never having used KBVision, the author is unable to fairly criticise this product. However, from a brief study of the available information it appears that although KBVision's provision of ISRs make it slightly more favourable than Khoros for image understanding problems, it contains a VPE very similar to Cantata.

The use of a VPE is advantageous for demonstrating image processing techniques in the classroom. However, within the context of research, where maximisation of control, speed, and accuracy are important factors, the VPE can be much less favourable than, say, a good command language.

2.3 The IUE.

The image understanding environment (IUE) [CJC95] has been designed to support research in image understanding. Its primary purpose is to facilitate the exchange of research results between research groups, industry, and the government. The IUE is currently a fairly large C++ class library consisting of classes to aid the storage and manipulation of image data. The current version (1.1) is available freely, however, as the development of the IUE continues and the product becomes more stable it is expected that Amerinex AI will make a small charge for either the entire IUE, or, for individual parts of the IUE.

The IUE is still under development, and at this stage is not mature enough to be considered stable. Since the currently available IUE classes appear to be fairly plentiful and potent, actually learning the interface to the class library could take a considerable amount of time and effort. Also, it appears that rapid prototyping of systems is inhibited by the necessary re-compilation and current lack of supporting development tools.

In contrast to KBVision and Khoros, although the IUE is functionally very powerful, there is no construction framework provided to ease the process of creating new systems or re-using existing sub-systems.

3 Overview of TABS.

3.1 General features of the framework.

1. TABS does not have a restrictive visual programming environment, instead, application systems are implemented using a Tcl Tk [Ous94] script. This supports very rapid prototyping by maximising the ability to re-use sub-systems, and avoids costly recompilation.

2. TABS provides a layer of abstraction between the way data are stored and the way they are manipulated, hence allowing transparent access to databases.

3. TABS is small, simple, and compiles very quickly (a *total* re-build of all components and the framework takes under two minutes on a Sun SLC 20MHz Sparc station with 8Mb of RAM).

4. In common with other frameworks, TABS can save *workspaces* so that the current system and all of its associated data can be held on disk and restored at a later time.

5. TABS can perform simple performance monitoring of components and systems. This feature is suspiciously lacking from other frameworks.

6. TABS is capable of fully supporting components and systems which manipulate various types of intermediate symbolic representation.

7. The interface to TABS is totally separate to its engine, allowing one-to-many or many-to-one engine access via a network.

3.2 Features which support component development and use.

1. Components may be written in any combination of three languages: C, C++ and Tcl Tk.

2. Components written in alternative languages can be supported through the use of a small wrapper written in one of the native languages.

3. Components which generate or make use of either multiple hypotheses, ordered sets of data, or hierarchically structured data, can be easily utilised.

4. TABS makes use of a simple set of public domain image formats (namely pbm, pgm and ppm), rather than more general image formats, thus reducing the effort required for component development.

3.3 Features which support system development and use.

1. Systems can employ both top-down and bottom-up control methodologies, feedback, recursion and iteration [Nag82] by making use of the conditional, looping and control structures available in Tcl.

2. Systems do not have to halt when an error occurs (they can take another course of action).

3. Systems can be made smart (not requiring any user interaction), or dumb (requiring user interaction), as desired. Also, systems and components which require a GUI can be easily integrated.

The TABS framework is implemented by two executable files. The first file (the core / engine) is responsible for providing the functionality of the framework, whilst the second file (the interface) creates an interface which may be used to communicate with the framework. The executable core is produced by the compilation of several C++ and Tcl Tk source files. The executable interface is just a single Tcl Tk script. The interface (labelled "TABS command shell") is shown in figure 5.

Using two distinct executable files in this manner has several advantages. Firstly, the (potentially CPU intensive) framework can be run remotely, with only the interface being run locally. Secondly, changes to the interface do not require changes to the framework, and vice versa, thus minimising any required rebuilding during interface or framework enhancement. Thirdly, extending TABS to allow support for multiple users of the same framework, or extending it to allow a single user to access multiple frameworks becomes much easier.

Conceptually, the framework's core consists of a set of components, a *blackboard*, and a Tcl Tk interpreter which allows external access to the blackboard and the components. The blackboard is responsible for managing all of the data which are associated with a currently running system. Figure 1 shows the main lines of communication present within the framework.

3.4 The blackboard's architecture.

The blackboard which is present within the core of the TABS framework is of particular importance to the framework's operation and flexibility and is solely responsible for data management. Components and systems send requests to the blackboard and receive corresponding responses from the blackboard.

One of the main philosophies behind the TABS framework is that any data which are created during the normal operation of a system should remain available (for possible future use or inspection) unless explicitly removed. The reasoning behind this idea is that TABS has been developed primarily as an environment suitable for performing rapid configuration and testing of systems; whilst testing a new system, it is obviously useful if all of the data which have been generated are made available for inspection.

The blackboard fulfills two major requirements of the TABS framework. Firstly, it allows any structure associated with data to be represented and stored. Secondly, it allows easy integration of the framework with any desired database. These two capabilities are discussed in more detail below.

3.4.1 In-built support for structured data.

In an attempt to make the data presently available on the blackboard more comprehensible to system designers, a *tagging system* is enforced. Each datum is given a mandatory tag (a simple name) by which it may then be referred. When data are passed between system components, or sub-systems, only their tags, or parts of their tags, need to be communicated. Each tag is responsible not only for identifying a datum uniquely from all others, but also for recording the datum's position in any hierarchical or ordered structure which is associated with the available data.

Hierarchical relationships are maintained by creating tags which look rather like widget names in Tk, or filenames specified from the root (/) directory in UNIX. For example, "Form.1.Field.1.Character.1" would refer to the first hand-printed character in the first field of the first scanned form image.

3.4.2 Data management.

If an external database is used, then it is the blackboard's responsibility to communicate with that database whilst ensuring that data integrity is maintained.

At any given moment during a system's operation, the blackboard will hold a dynamic list of all the tags which exist, and for each tag also hold some additional information (such as the age of the tag, the type of data it references, etc.).

The blackboard implements a file paradigm to allow components easy but controlled access to the actual data content associated with a tag. Like files in UNIX, a tag may be opened (in one of a number of possible modes), have its content analysed, or changed, and must then be closed. Each component is responsible for opening any tags it requires, performing any required functionality involving the data referenced by those tags, and then closing the tags again before returning control to the framework.

3.5 Component and system development in TABS.

Developing components for TABS is a two-stage process. Firstly, the component source is written in a supported language. Secondly, the component is registered with the Tcl Tk interpreter which is implemented within the core. Once these two processes are complete and the core has been re-built, the framework is ready for use (new component included).

Whilst developing new components requires the framework to be re-built, developing new systems using an existing framework requires no rebuilding whatsoever. Systems are written as Tcl Tk scripts (in this context called control scripts) which are passed to the core's interpreter via the framework's interface. Control scripts can call upon any of the standard Tcl Tk commands, as well as several other non-standard commands which have been registered with the core's interpreter. Non-standard commands are used for two purposes, firstly to allow control scripts to interrogate the blackboard, and secondly, to allow components to be called.

Developing sub-systems for use within TABS is particularly straightforward. Sub-systems are simply implemented as Tcl procedures within the control script. Once sub-systems have been created, not only can they be re-used within the same system, but they can also be re-used by different systems. An example control script which contains three sub-systems is shown in figure 3.

4 Implementation.

Whilst the framework's interface has been implemented entirely in Tcl Tk, the core has been implemented mostly in C++, but with some Tcl Tk embedded where required. The TABS framework incorporating the set of components currently used by the author consists of just over eighteen thousand lines of source code. Integrating Tcl Tk with C++ has been made possible thanks to the excellent ET *(embedded Tk)* package by Hipp [Hip96].

Having been developed by a single student rather than by a team of researchers, TABS must currently be considered to be immature and unsupported. Unlike other frameworks which typically contain over 200 fairly potent components, TABS only contains just over 30 at present, all of which are rather simplistic, performing rudimentary image handling tasks such as image resizing, binary connected components analysis, etc.

It has been stated that TABS makes the integration of external databases particularly straightforward; this will hopefully soon be demonstrated by integrating TABS with an object-oriented database such as Object-Store.

TABS currently runs happily under both X and Windows NT 4.0.

5 An example use of TABS.

Some figures are provided below to illustrate how TABS may be used to prototype a simple form processing system. The form image shown in figure 2 is placed on the blackboard and then processed by sourcing the control script shown in figure 3. Figure 4 shows the dialog generated by a simple TABS component (called "fn_DisplayBlackboard") which is capable of displaying various items of information about each datum currently on the blackboard. Finally, figure 5 shows TABS displaying some images and some classification results.

6 Conclusions.

A new software framework for image analysis, processing and understanding has been overviewed in this paper. It has been proposed that the new framework (TABS) is especially suited to applications where rapid system prototyping is required, where maximum use is required of control methodologies, and where probabilistic components are employed. The framework achieves considerable flexibility through a novel approach to the management of the data and components within the framework. TABS has been critically compared to several other frameworks and has been successfully used to implement a number of hand-printed form processing systems.

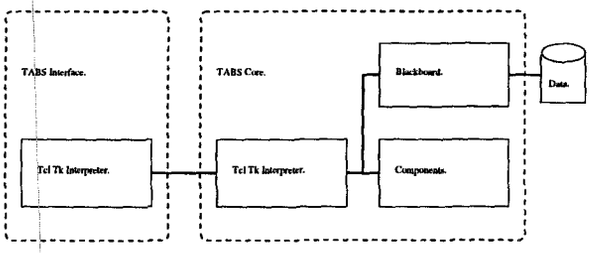

Figure 1: Communication within the TABS framework.

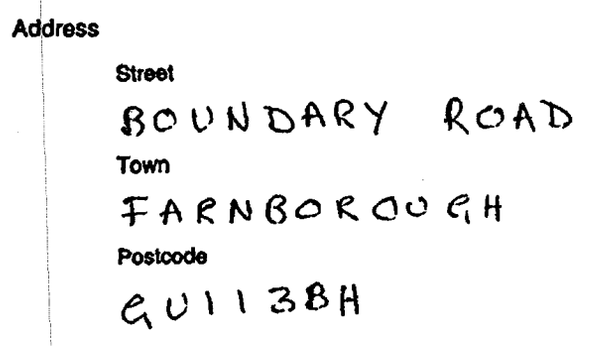

Figure 2: A form image to be processed.

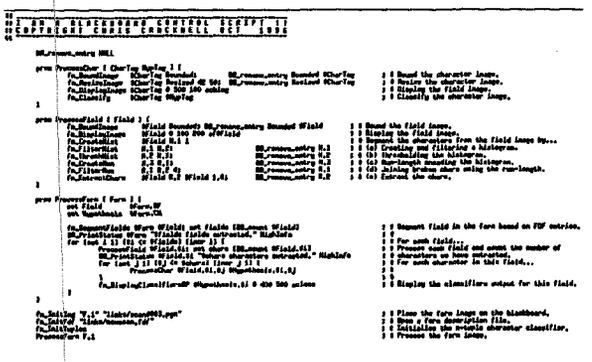

Figure 3: Complete TABS control script which implements a form processing system.

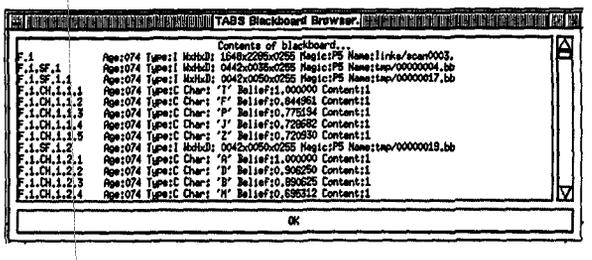

Figure 4: The dialog generated by a simple TABS blackboard browser component.

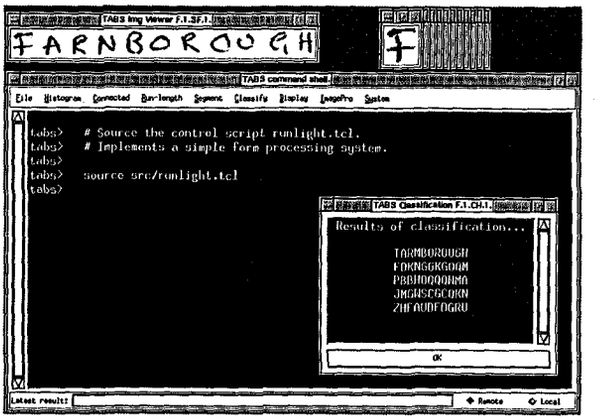

Figure 5: TABS displaying some results from a form processing system.

It is hoped that in the near future TABS will be released into the public domain.

7 References.

References

[CJC95] Charles Kohl, J. Jeffrey Hunter, and Cynthia L. Loiselle. Towards a unified iu environment: Coordination of existing iu tools with the iue. In *Proceedings of the IEEE International conference on computer vision*, pages 2–7. IEEE, jun 1995.

[Hip96] D. Richard Hipp. Embedded tk. `http://www.vnet.net/users/drh/ET.html`, 1996.

[Jac94] Peter Jackson. *Introduction to expert systems.* Addison-Wesley publishing company, 1994.

[JK94] John R. Rasure and Konstantinos Konstantinides. The khoros software development environment for image and signal processing. In *IEEE Transactions on image processing*, volume 3, pages 243–252. IEEE, may 1994.

[Nag82] Makoto Nagao. Control strategies in pattern analysis. In *Proceedings of the IEEE International conference on pattern recognition*, volume 2, pages 996–1006. IEEE, 1982.

[Ous94] John Ousterhout. *Tcl and the Tk toolkit.* Addison-Wesley publishing company, 1994.