Design of an Integrated Environment for the Automated Analysis of Architectural Drawings*

Philippe Dosch[1], Christian Ah-Soon[1], Gérald Masini[1], Gemma Sánchez[1,2], and Karl Tombre[1]

[1] Loria–Cnrs–Inpl–Inria–Uhp
B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
{dosch,ahsoon,masini,gsanchez,tombre}@loria.fr
http://www.loria.fr/isa/
[2] Computer Vision Center
Edifici C, Campus Universitat Autònoma de Barcelona
08193 Bellaterra (Barcelona), Catalunya-Spain
gemma@cvc.uab.es
http://www.cvc.uab.es/~gemma/

Abstract. This paper presents the principles which have guided the design of our graphics recognition software environment. A number of applicative modules have been constructed on top of the environment, for the purpose of analyzing architectural drawings. A flexible user interface drives these modules. Our choices are compared with those of similar systems.

1 Introduction

Our research group has been investigating various aspects of graphics recognition techniques for more than ten years: Map analysis [1], symbol recognition [2,3], dimension analysis [4], conversion of engineering drawings to Cad models [5], and lately interpretation of architectural drawings [6].

During the last two years, we have also conducted a "consolidation" activity, especially for low-level graphics recognition methods [7], in order to build up a set of stable software components, reusable from one application to the other. This work leads to a number of systems engineering issues, that we try to document and comment in this paper. Such design and integration problems are recurrent in the document analysis field, and more generally in the image processing field. Initiatives to solve them gave for instance rise to the *Image Understanding Environment* (Iue) [8,9].

Figure 1 gives an overview of our three-layered system. The first layer is the Isadora library (§ 2), which consists of basic graphics recognition methods. They are designed to be as general as possible, *i.e.* to be independent of application fields.

* This work is partially funded by France Telecom / Cnet.

S.-W. Lee and Y. Nakano (Eds.): DAS'98, LNCS 1655, pp. 295–309, 1999.

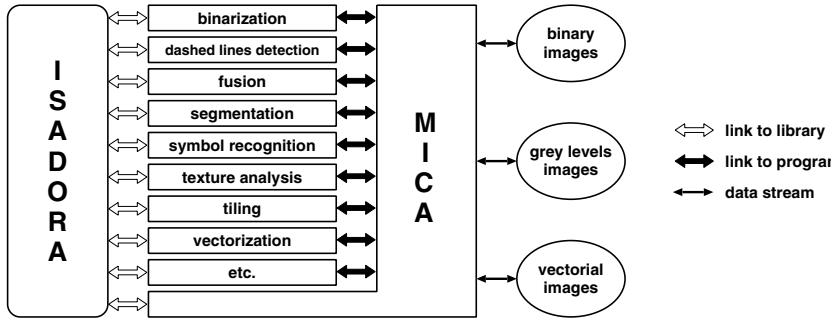


Fig. 1. Overview of our three-layered system.

The second layer includes useful so-called graphics recognition *applications* (§ 3). Low-level applications are more or less sequences of calls to the corresponding modules of the Isadora library, while higher-level ones are real programs including calls to Isadora functionalities. All applications are independent programs, which can either be run using a command line or be driven by the user interface.

The third layer is the user interface. In order for a document analysis system to work in practice, the user must be "in the loop": The interface must be tightly connected with the underlying layers, so that the user can easily guide the analysis process, and take quick corrective actions when necessary. The Mica user interface that we propose (§ 4) provides such functionalities. It is also an application, linked with the Isadora library as the others, but it is able to call applications of the second layer.

Some elements of comparison with other similar environments are given in § 5, before concluding the paper.

2 C++ Classes for Graphics Recognition

Our goal is the construction of a library of stable and reusable data structures, by using an object-oriented language. It rises two main categories of problems. If those related to the design of the components of the library do depend on the specific domain of the library, graphics recognition in our case, we first had to deal with those related to the design of the library itself.

2.1 Prerequisites and Tools

We have been confronted with three typical software engineering requirements. Firstly, the transition from existing code to new code had to be as easy as possible. As we had a lot of old code written in C, C++ appeared to be the ideal choice. It is not necessarily the best object-oriented language, but it is more or less a standard in industry, and it provides to a large extent compatibility with our previous C code.

Secondly, we wanted to use as often as possible available software, preferably public-domain, easy-to-find tools, and *de facto* standards. This explains, for example, why we use Jef Poskanzer's *Portable Bitmap* (PBM) format for image files, DXF for the representation of 2D graphics, and VRML for 3D graphics.

Finally, code reusability and efficiency had to be guaranteed. These two criteria are often antagonistic. Reusability is achieved through C++ thanks to data abstraction and encapsulation, but it sometimes leads to a lot of computation overhead. We therefore allowed ourselves to use well-known "programming tricks", based on low-level C constructions, for efficiency reasons. Since it is based on C, C++ makes it easier to implement such tricks. However, as classes designed in this way are often dependent on the implementation details, tricks have been hidden in so-called *private classes* (with private interfaces), which are not accessible to the common users of the library.

The programming tools had then to be chosen according to the previous criteria. This is out of the scope of this paper, but two points deserve a special attention. We had no intention to code a new set of common data structures like vectors, lists, sets, and so on. The C++ standard library provides most of them in what was previously known as the Standard Template Library (STL), on the notable exception of graphs, for which we decided to use an additional library, LEDA[1], from the *Max-Planck-Institut für Informatik* in Saarbrücken, Germany.

Moreover, a software library is useless without documentation giving detailed information about class interfaces. We chose Malte Zöckler and Roland Wunderling's documentation system, DOC++[2], which is similar to `javadoc` in the JAVA environment. It has low memory requirements and is very easy to use. It automatically generates online browsable HTML and high-quality hardcopy documentation directly from the C++ code by parsing the sources for special comments, that instruct how to create the documentation.

2.2 Class Design

In fact, the principal problem we had to solve is encountered when implementing image processing operations in an object-oriented language. The object-oriented paradigm is based on data encapsulation, *i.e.* describing abstract data types and their interfaces with the clients, whereas image processing generally concerns collections of operators, *i.e.* procedures, to be applied to images.

Let us consider an elementary example. The convolution of an image by a Gaussian yields a new image $J = I \otimes G$. The question is then: Should convolution be defined as a function member of the `Image` class that describes images, or should it be defined as a global operator? Our answer to the question is quite pragmatic, and is close to what is proposed by marketed libraries such as the *Image Vision Library*™ from Silicon Graphics.

The basic idea is very simple. `Image` is the base class of a hierarchy and its derived classes define image types: `BinaryImage`, `GreyLevelImage`, `FloatImage`...

[1] `http://www.mpi-sb.mpg.de/LEDA/`

[2] `http://www.zib.de/Visual/software/doc++/`

Each operation that processes an image of a particular type corresponds to a derived class of the class defining the image type, and is implemented by a constructor of this derived class. Different methods to perform the same conceptual operation can thus be easily implemented through derived classes of an abstract class.

Of course, this paradigm does not only apply to image processing, but also to all analysis and recognition tasks. It has the advantage of meeting understandability requirements [10]: Designers as well as clients of the library write compact and easy-to-read code. The idea may be illustrated with a very common image processing problem, edge detection. A straightforward algorithm can be written using our library in the following way:

```
PgmFile f("image.pgm");   // File containing the original image
GreyLevelImage myImg(f);  // Load it as a grey level image

// Perform Canny operator with sigma == 1.2
CannyGradientImage myDeriv(myImg, 1.2);
// Alternatively, if you choose the Deriche operator,
// comment previous line and uncomment next line
// DericheGradientImage myDeriv(myImg); // with default parameters

LocalGradientMaxima maxG(myDeriv); // Compute maximum of gradient
EdgeMap myEdge(maxG, 0, 5);        // Edge map with double thresholding
LinkedChainsList myChains(myEdge, 1, 0); // Link edges

// And so on... We can perform a polygonal approximation on the chains,
// and then save the segments to a DXF file, etc.
```

This obviously is an idyllic view as, in the general case, additional parameters are needed to accurately perform the different image processings. However, we firmly believe that the general philosophy holds for most low-level and intermediate-level operations in document image processing and graphics recognition applications.

2.3 The Class Hierarchy

ISADORA classes are hierarchically organized as a tree, with a single root named `IsaObject`, which defines information common to all objects, especially error handling facilities. This tree can be viewed as a collection of subtrees grouping together classes according to the different kinds of objects to be handled when designing a document analysis system. Each subtree is itself organized in the same way.

There are three main families of classes. The first one includes classes for image processing (mainly `Mask`, `Histogram` and `Image` subtrees), as graphics processing involves a lot of image processing, at least in the low levels. For instance, see figure 2 for a synthetical view of the `Image` subtree.

```
* Image
     o LabelImage
     o GreyLevelImage
          + ReconstructedImage
          + LinErodedImage
          + LinDilatedImage
          + LabeledSkeleton
          + ErodedImage
          + DistanceTransform34
          + DilatedImage
          + ContrastEnhancedImage
     o GradientImage
          + DericheGradientImage
          + CannyGradientImage
     o FloatImage
          + StandardDeviationImage
          + LocalGradientMaxima
          + LaplacianOfGaussianImage
          + GradientModuleImage
          + Convol2dImage
          + Convol1dImage
     o BinaryImage
          + TrierBinaryImage
          + NiblackBinaryImage
          + LinErodedBinaryImage
          + LinDilatedBinaryImage
          + IsotropicDilatedBinaryImage
          + GeodesicReconstruction
          + ExtractedComponentsImage
          + ErodedBinaryImage
          + EdgeMap
          + DilatedBinaryImage
          + CleanedBinaryImage
```

Fig. 2. The subtree of image classes.

All images are represented as arrays of pixels and can be either input data or results of some processing. We intentionally do not use a too general definition, to avoid the complexity found, in particular, in the IUE specifications [8]. Most of the common image processing tools are available: Histogram computing, basic processing using convolutions (Gradient, Laplacian, etc.), mathematical morphology on both binary and grey level images, edge detection (in particular Canny's and Deriche's methods), binary image processing, and so on.

The `Graphics` subtree provides classes for graphics processing, describing all the different kinds of graphical primitives that can be delivered by basic segmentation modules, and also groupings of such primitives, that are delivered by analysis modules: Points, segments, chains of segments, rectangles, arcs of circle, connected components, etc. as well as ordered collections of such objects (cf. § 3.3). They correspond to the semantics level in Koelma and Smeulders' hierarchy (cf. § 5).

Finally, classes for utilities, especially file processing (`IsaFile` subtree), allow graphics or image data to be stored in a selected format (cf. § 2.1).

The interfaces of the usual classes are not especially original, as we describe the fundamental operations in a very classical way. We first of all want to make the programming of complex algorithms easier, by providing a relatively sim-

ple design framework and by avoiding "reinventing the wheel" when reusable, efficient tools are available.

3 The Application Layer

Let us now present some of the application modules that we have designed, going from lower-level image processing tools to higher-level ones. The former are basically designed as sequences of operations applied to ISADORA objects. They can be easily reused for other graphics recognition tasks. The latter are more specifically designed for architectural drawing processing.

3.1 Binarization

When the binarization provided by commercial software or hardware is not satisfactory, we have decided to use the adaptive algorithm proposed by Trier and Taxt [11], with some minor adaptations [7]: Instead of using *ad-hoc* filters such as the Sobel gradient, as proposed by the authors, we use Gaussian filtering, which has become standard in edge detection, and which happens to be implemented in a robust way in our library.

Here are some lines from the implementation, to give once again a taste of the way existing classes can be easily reused and to emphasize the processing–through–constructor concept (cf. § 2.2):

```
TrierBinaryImage::TrierBinaryImage(const GreyLevelImage& anImage,
                                   const float postThresh,
                                   const float activityThresh,
                                   const double sigma)
{
  // Compute the Canny gradient of the original image:
  // Convolution by the 1st derivative of a Gaussian
  CannyGradientImage* gradient = new CannyGradientImage(anImage, sigma);
  GradientModuleImage modGrad(*gradient);
  delete gradient;
  // Compute the image activity
  Mask2d aMask2d(3, 1.0);
  double* myMask = aMask2d.mask();
  for (int i = 0; i < aMask2d.width(); i++)
    *myMask++ = (double) 1.0;
  // Compute the Laplacian of the smoothed image
  // by convolving with the Laplacian of a Gaussian
  Convol2dImage* activity = new Convol2dImage(modGrad, aMask2d);
  LaplacianOfGaussianImage* lapImg =
    new LaplacianOfGaussianImage(anImage, sigma);
  ...
 }
```

The last steps of the algorithm (not documented here) rely heavily on the `LabelImage` class, used to label the connected components of a binary image, as explained in next section.

3.2 Text/Graphics Separation

As most text/graphics separation methods are based on analyzing the connected components, we have designed the `LabelImage` class, whose constructor builds a tree of connected components from a `BinaryImage` [1]. The tree represents the inclusion relation between components and gives the oriented contours of all the components.

The principle of the algorithm consists in analyzing the input image one row after the other, comparing the current row with the previous one, and storing all necessary information while labeling the black and white runs of the current row. The contours of the connected components are chained "on the fly" and described by Freeman chain codes, instances of the `Freeman` class.

This class defines just one possible representation of a chain:

```
class Freeman : public GenChain<int> { ... };
```

and the template class `GenChain<T>` defines a common interface to all kinds of chains (cf. § 3.3). Hence, the `Freeman` class defines the specific encoding of Freeman chains, and implements the common interface given by `GenChain`.

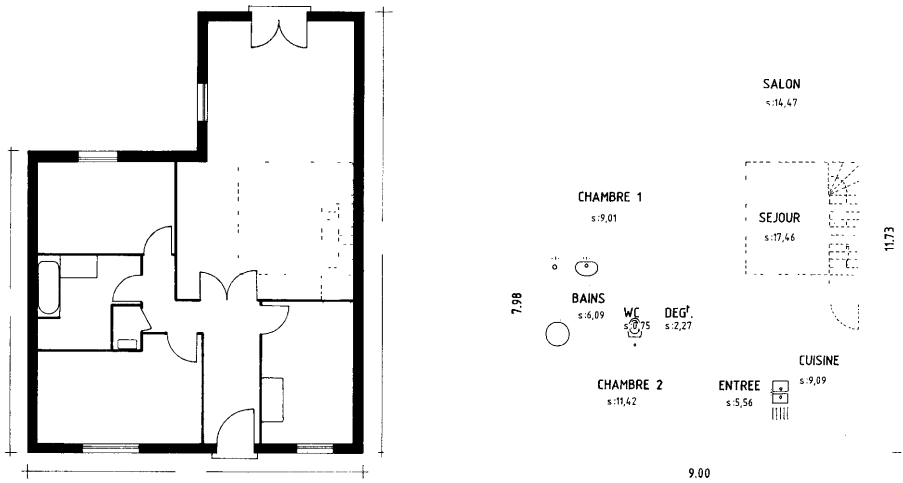
Fletcher and Kasturi have proposed one of the most robust text/graphics separation methods of the literature [12]. We therefore suggest that, instead of spending a lot of time on reinventing new methods, which most of the time do not give any real improvements on known methods, research groups do use this method, appropriately changing the parameters, if necessary, to fit the characteristics of the documents to be processed. Then, they will have more time left to concentrate on the really difficult problem of separating touching text and graphics, for which only partial solutions have been presented.

In our implementation, we added an absolute threshold for the size of a text component [7]. We thus end up having three thresholds, but their interpretation is straightforward, and they have proven to be very stable for a family of graphics documents: Once the best values are determined for a certain kind of application, they can be used for all related images.

Figure 3 shows some results obtained on an architectural drawing. As proposed by Fletcher and Kasturi, a string grouping is then performed, using the Hough transform. Further refinement of the graphics part can be obtained by separating thin and thick lines using morphological filtering, which is also available in the image processing part of the ISADORA library.

3.3 Vectorization

Among all the vectorization methods (*i.e.* raster-to-graphics conversion) currently available, our favorite is a skeletonization based on the 3–4 distance trans-



(a) Graphics. (b) Text and small symbols.

Fig. 3. An example of text/graphics separation using Fletcher and Kasturi's method on the architectural drawing of a simple private house.

form [13], followed by some polygonal approximation [7]. The distance skeleton is implemented by the `LabeledSkeleton` class.

Once a skeleton is computed, the skeleton pixels must be linked into chains. The `LinkedChainsList` class defines a list of linked chains of 2D points. We show here the context of this class, to illustrate how a specific implementation (such as using a list of points to represent a chain) can be encapsulated into a generic, abstract class.

First, the `GenChain` template class provides a generic interface for any chain, independently of the way it is internally represented and coded (note the use of Doc++ special comments).

```
template <class T>
  class GenChain : public Graphics {
public:
  /** Basic constructor. */
  GenChain() {}
  /** Get chain length (number of points). */
  virtual int length() const = 0;
  /** Get first point of chain. */
  virtual GenPoint<T> first() const = 0;
  /** Get last point of chain. */
  virtual GenPoint<T> last() const = 0;
  /** Set iterator to initial position (start of chain). */
```

```
  virtual void setInitialPosition() = 0;
  /** Access next point through iterator.
  The function returns 1 if it is possible to move forward in the chain,
  0 if not. When moving is possible, the new (x,y) translation is stored
  in 'transPoint' and the direction of the new code is stored in 'dir'.
  */
  virtual int next(GenPoint<T>& transPoint, Direction& dir) = 0;
  /** Reverse chain. */
  virtual void reverse() = 0;
  /** Is current chain empty?. */
  virtual int empty() const = 0;
};
```

A possible implementation of such a chain is then obtained by using a list of points provided with an iterator. Of course, the functions of the generic interface must be defined, although these implementations are not given here for the sake of brevity.

```
template <class T>
  class GenLinkedChain : public GenChain<T> {
protected:
  /** The list itself. */
  list< GenPoint<T> > theList;
  /** An iterator on the list. */
  list< GenPoint<T> >::iterator theIter;
public:
  // Implementation of the interface defined by GenChain<T>
  ...
};
```

Finally, the complete result of a chaining can be defined as a list of such chains, with integer coordinates:

```
  class LinkedChainsList : public list< GenLinkedChain<int> >
    { ... };
```

This class is provided with both a general constructor from a `BinaryImage` object, that performs the chaining of any kind of binary image, and a specific constructor from a `LabeledSkeleton` object, that implements a linking algorithm taking the topological properties of the skeleton distance into account.

The skeletonization itself is followed by a polygonal approximation. The way we have designed our library allows us to have several algorithms in store for that. They can be easily tested on any list of objects implementing the generic interface for chains: A contour of a connected component represented by a Freeman code, a contour computed from a grey level image by some edge detector, or a skeleton.

All the polygonal approximation methods use the generic interface provided by class `GenChain<int>`. We have implemented Wall and Danielsson's

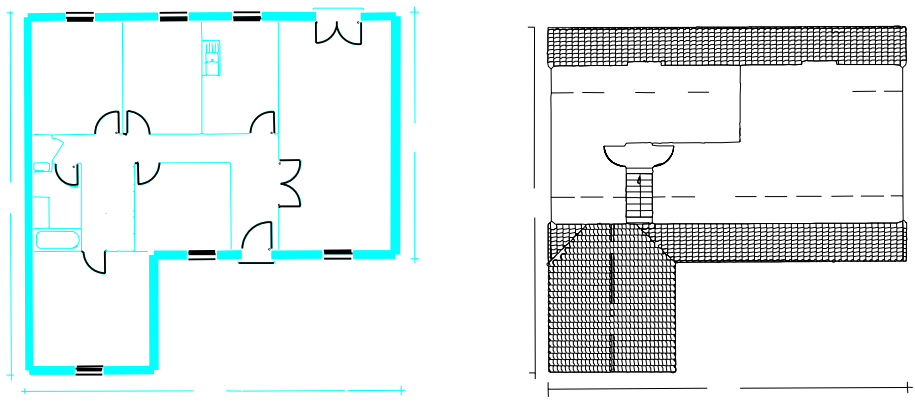
method [14], as well as the recursive method proposed by Rosin and West [15], which actually is an evolution of an algorithm first proposed by David Lowe. Both corresponding classes, `WallDanielssonSegList` and `RosinWestSegList`, have a constructor from a `GenChain<int>` object: They can work on any kind of linked chain.

If necessary, vectorization is followed by arc and dashed-line detections, inspired by Dov Dori's ideas [16,17]. The corresponding algorithms are implemented by constructors of classes, in the same way as our other tools.

3.4 Symbols and Textures

A flexible recognition system performs the identification of symbols representing building elements, such as doors or windows [2]. Each symbol is described by a set of constraints on the graphical features of the symbol, using a language we have specifically designed for that purpose.

All the description files are parsed to dynamically create a network representing a compact description of all the symbols. A node of the network represents a constraint. A one-pass symbol detection can subsequently be performed by propagating graphical features (*i.e.* segments and arcs of circle) through the network: When a feature conforms to the constraint of a node, it moves to the adjacent node. The recognized symbols are retrieved from the terminal nodes. They are represented as instances of ISADORA classes and thus can be easily handled by higher-level applications. This method allows the recognition of most of the symbols with a very low computation time (Fig. 4.a).



(a) Symbols. (b) Textured regions.

Fig. 4. Symbols and textured regions extracted from the architectural drawings of the first and second levels of the same private house.

Texture detection is also useful, mainly to recognize and locate specific areas representing staircases and pieces of roofs. A texture usually is a structure composed of similar lines presenting a regular repetition. It is made from an element, called a *texel*, placed according to a regular pattern. In our case, texels are straight lines or polygonal shapes.

As textures vary a lot from one drawing to another, we do not use predefined models of regularly structural textures. In fact, we try to find all structures composed as a regular arrangement of a same pattern, whatever this pattern may be.

The method we have integrated in our system results from previous work by one of the authors, at *Universitat Autònoma de Barcelona* [18]. Similar neighboring texels are grouped together using a hierarchical clustering method [19]. The similarity between texels is estimated from their area difference and their shape similarity. The latter is itself computed using the cyclic string edit distance between the boundary strings, proposed by Maes [20], in association with the merge operation of Tsai and Yu [21]. The costs are defined as a weighted sum of an angle cost and a length cost, in a similar way as [22].

The result is a set of regions containing similar neighboring shapes with the same orientation. These regions represent the different textured zones, as illustrated in figure 4.b, where they are displayed as areas surrounded by thick lines.

4 The User Interface

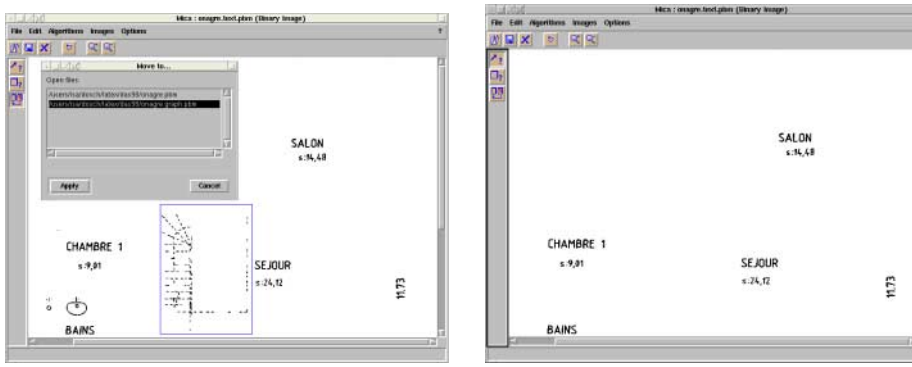
All the processes presented in the previous sections are incorporated into a user-friendly software system, provided with a sophisticated user interface giving a visual feedback about the working of the system.

In fact, it would not be reasonable to run the whole interpretation chain without giving the user any possibility to interact. Indeed, we have to deal with many problems like the noise introduced by artefacts (especially folds) in drawings, by the fact that a same architectural component (door, window, etc.) can be drawn in many different ways, and, of course, by the very limitations of our methods.

A human assistance is anyway required to determine when and how a specific process is to be performed, and to set values to parameters and thresholds of the different applications of the analysis. The user may also want to experiment several applications related to a given problem, in order to be able to select the one giving the best results.

All the applications of the second layer are therefore connected through simple links with what constitutes the third layer of our system, named Mica. In this way, an application can be easily substituted for another or can be upgraded when necessary. Each link is used to transmit the values of the parameters to the corresponding application: Names of images, options, thresholds, etc. Such parameters are set with default values which can be customized by the user.

The interface supplies the following basic functionalities:



(a) Interactive correction of segmentation results.

(b) Final text layer.

Fig. 5. Correction of text/graphics segmentation: Some dashes are misinterpreted as hyphens in the text layer (they actually represent stairs) and are moved to the graphics layer by the user.

- Display of the content of all kinds of files, with common editing functionalities like multi-file editing, zooming, and so on.
- Parameter and threshold tuning, after examination of the results of the current application using the display facility.
- Direct manipulation of resulting data, *i.e.* add missing results, delete or alter erroneous results. In particular, special editing operations are supported for both bitmap and vectorial images: Cut and copy of bitmap images, creation and modification of vectorization components. (Fig. 5), etc.

5 Comparison with other Work

We are aware that we are not the only research group working on this topic. For instance, the duality between object-oriented programming and operator-based image processing (cf. § 2.2) has been studied by several teams, most notably in our area. Dov Dori's group, for instance, proposes a similar environment, Mdus, the *Machine Drawing Understanding System* [23], based on the object-process methodology. Our approach is probably more pragmatic than theirs, but the resulting environments are quite comparable.

The Tabs system [24] has also been designed according to choices similar to ours, whereas its applications deal with form processing and handwriting recognition. The system is implemented in C++, but the user interface is written in Tcl/Tk. It also includes a supervision level driven by a blackboard.

A number of people have proposed different solutions, for example Koelma and Smeulders, who designed a library for image processing, with a hierarchy

based on the number of dimensions of the image, the form of the pixel representation and the semantics of the image content [25]. Other systems include PhotoPix [26], and Piper and Ritovitz's work on C and C++ classes [27,28].

In our opinion, a system like Khoros belongs to a different category. Its main strength comes more from the flexibility of its user interface (based on the visual programming paradigm) and from the completeness of its image processing library. Although it seems to be appropriate to teaching or to prototyping, it remains quite slow when used for real applications.

6 Conclusion

We have described a framework for designing reusable graphics recognition software components. It comprises a library of basic image and graphics processing operations, ISADORA, and a set of higher-level graphics recognition applications. A user interface has been added on top of these two layers, to drive the applications.

The design of this software platform first constrained us to deal with software engineering problems: We had to choose tools to be integrated as our programming environment, and to specify detailed programming recommendations to which each member of the research group had to conform. The latter point is particularly important when coding with a programming language like C++, which encourages the use of tricks increasing efficiency.

Some could think that we are not concerned by such problems. On the contrary, it appears that they fully condition the future of a project like ours: Without the long time passed in the careful study of the prerequisites, we could not be able to work together and to achieve a project involving the coding and the integration of a great number of complex methods. This experiment taught us that rigor and simplicity must be privileged, even at the cost of a certain loss of efficiency. The most important fact is that software components are easy to upgrade, to complete and to reuse.

The principles that we have initially chosen to design each separate method and tool allow our group to subsequently develop this environment all together, thus going from single-user programming to group development. Some software engineering problems remain open, but our platform can be considered as operational and we are now experienced enough to progressively solve them. Of course, our research work about graphics recognition is still simultaneously carried on. When methods become mature enough, they are integrated into the common environment, and become thus available to the whole group. We hope this paper has demonstrated the pertinence of our choices.

Ultimately, our work, as well as the others, might end up becoming contributions to the *Image Understanding Environment* (IUE) [8], which aims at providing a sophisticated environment for all kinds of image understanding activities. However, before joining this large effort, we prefer mastering a lower level of complexity by concentrating on our own know-how. We want to prove that we can "serve our own group" before being able to "serve the community".

References

1. D. Antoine, S. Collin, and K. Tombre. Analysis of Technical Documents: The REDRAW System. In H. S. Baird, H. Bunke, and K. Yamamoto, editors, *Structured Document Image Analysis*, pages 385–402. Springer-Verlag, Berlin, 1992. 295, 301
2. C. Ah-Soon and K. Tombre. Network-Based Recognition of Architectural Symbols. In A. Amin, D. Dori, P. Pudil, and H. Freeman, editors, *Advances in Pattern Recognition (Proceedings of Joint IAPR Workshops SSPR'98 and SPR'98, Sydney, Australia)*, *Lecture Notes in Computer Science 1451*, pages 252–261. Springer-Verlag, Berlin, 1998. 295, 304
3. A. H. Habacha. Structural Recognition of Disturbed Symbols Using Discrete Relaxation. In *Proceedings of 1st International Conference on Document Analysis, Saint-Malo (France)*, volume 1, pages 170–178, 1991. 295
4. S. Collin and D. Colnet. Syntactic Analysis of Technical Drawing Dimensions. *International Journal of Pattern Recognition and Artificial Intelligence*, 8(5):1131–1148, 1994. 295
5. P. Vaxivière and K. Tombre. CELESSTIN: CAD Conversion of Mechanical Drawings. *IEEE COMPUTER Magazine*, 25(7):46–54, July 1992. 295
6. C. Ah-Soon and K. Tombre. Variations on the Analysis of Architectural Drawings. In *Proceedings of 4th International Conference on Document Analysis and Recognition, Ulm (Germany)*, pages 347–351, 1997. 295
7. K. Tombre, C. Ah-Soon, Ph. Dosch, A. Habed, and G. Masini. Stable, Robust and Off-the-Shelf Methods for Graphics Recognition. In *Proceedings of the 14th International Conference on Pattern Recognition, Brisbane (Australia)*, pages 406–408, 1998. 295, 300, 301, 302
8. C. Kohl and J. Mundy. The Development of the Image Understanding Environment. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Seattle (USA)*, 1994. 295, 299, 307
9. J. Mundy, T. Binford, T. Boult, A. Hanson, R. Beveridge, R. Haralick, V. Ramesh, C. Kohl, D. Lawton, D. Morgan, K. Price, and T. Strat. The Image Understanding Environment Program. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Urbana Champaign (USA)*, pages 406–416, 1992. 295
10. B. Meyer. *Object-Oriented Software Construction, Second Edition*. The Object-Oriented Series. Prentice-Hall, Englewood Cliffs (NJ), USA, 1997. 298
11. Ø. Due Trier and T. Taxt. Improvement of "Integrated Function Algorithm" for Binarization of Document Images. *Pattern Recognition Letters*, 16(3):277–283, 1995. 300
12. L. A. Fletcher and R. Kasturi. A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images. *IEEE Transactions on PAMI*, 10(6):910–918, 1988. 301
13. G. Sanniti di Baja. Well-Shaped, Stable, and Reversible Skeletons from the (3,4)-Distance Transform. *Journal of Visual Communication and Image Representation*, 5(1):107–115, March 1994. 302
14. K. Wall and P. Danielsson. A Fast Sequential Method for Polygonal Approximation of Digitized Curves. *Computer Vision, Graphics and Image Processing*, 28(2):220–227, 1984. 304
15. P. L. Rosin and G. A. West. Segmentation of Edges into Lines and Arcs. *Image and Vision Computing*, 7(2):109–114, May 1989. 304
16. D. Dori. Vector-Based Arc Segmentation in the Machine Drawing Understanding System Environment. In A. L. Spitz and A. Dengel, editors, *Document Analysis Systems*, pages 338–362. World Scientific, 1995. 304

17. D. Dori, L. Wenyin, and M. Peleg. How to Win a Dashed Line Detection Contest. In R. Kasturi and K. Tombre, editors, *Graphics Recognition—Methods and Applications*, *Lecture Notes in Computer Science 1072*, pages 286–300. Springer-Verlag, Berlin, 1996. 304
18. G. Sánchez, J. Lladós, and E. Martí. Segmentation and Analysis of Linial Texture in Planes. In *Proceedings of 7th Spanish National Symposium on Pattern Recognition and Image Analysis, Barcelona, Spain*, volume 1, pages 401–406, 1997. 305
19. S. W. C. Lam and H. H. S. Ip. Structural Texture Segmentation Using Irregular Pyramid. *Pattern Recognition Letters*, 15(7):691–698, 1994. 305
20. M. Maes. Polygonal Shape Recognition Using String-Matching Techniques. *Pattern Recognition*, 24(5):433–440, 1991. 305
21. W. H. Tsai and S. S. Yu. Attributed String Matching with Merging for Shape Recognition. In *Proceedings of 7th International Conference on Pattern Recognition, Montreal (Canada)*, pages 1162–1164, 1984. 305
22. Y. T. Tsay and W. H. Tsai. Model-Guided Attributed String Matching by Split-and-Merge for Shape Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 3(2):159–179, 1989. 305
23. D. Dori. Representing Pattern Recognition-Embedded Systems Through Object-Process Diagrams—The Case of the Machine Drawing Understanding System. *Pattern Recognition Letters*, 16(4):377–384, 1995. 306
24. C. Cracknell and A. C. Downton. TABS: Script-Based Software Framework for Research in Image Processing, Analysis and Understanding. *IEE Proceedings – Vision, Image and Signal Processing*, 145(3):194–202, 1998. 306
25. D. Koelma and A. Smeulders. An Image Processing Library Based on Abstract Image Data-Types in C++. In C. Braccini, L. De Floriani, and G. Vernazza, editors, *Proceedings of 8th International Conference on Image Analysis and Processing, San Remo (Italy)*, *Lecture Notes in Computer Science 974*, pages 97–102. Springer-Verlag, Berlin, 1995. 307
26. A. A. S. Sol and A. de Albuquerque Araújo. PhotoPix: An Object-Oriented Framework for Digital Image Processing Systems. In C. Braccini, L. De Floriani, and G. Vernazza, editors, *Proceedings of 8th International Conference on Image Analysis and Processing, San Remo (Italy)*, *Lecture Notes in Computer Science 974*, pages 109–114. Springer-Verlag, Berlin, 1995. 307
27. J. Piper and D. Rutovitz. Data Structures for Image Processing in a C Language and UNIX Environment. *Pattern Recognition Letters*, 3(2):119–130, 1985. 307
28. J. Piper and D. Rutovitz. An Investigation of Object-Oriented Programming as the Basis for an Image Processing and Analysis System. In *Proceedings of 9th International Conference on Pattern Recognition, Rome (Italy)*, pages 1015–1019, 1988. 307