

Scan-to-XML for Vector Graphics: an Experimental Setup for Intelligent Browsable Document Generation*

B. Lamiroy[†]

LORIA – INPL / Projet ISA
Campus scientifique – B.P. 239
54506 Vandœuvre-lès-Nancy CEDEX – FRANCE

L. Najman

Océ Print Logic Technologies SA / AD Department
1, rue Jean Lemoine – BP 113
94015 Créteil CEDEX – FRANCE

R. Ehrhard C. Louis F. Quélain N. Rouyer N. Zeghache

École Nationale Supérieure des Mines de Nancy
Parc de Saurupt
54045 Nancy CEDEX – FRANCE

Abstract

This paper describes an experimental setup, conducted in collaboration with the ISA research group of the LORIA laboratory, OCÉ-PLT, and students from the École des Mines de Nancy.

The main objective is to experiment an approach to develop a high level document analysis platform by composing existing bricks from a comprehensive library of state-of-the art algorithms. The test-case of this methodology consists in the realization of a fully automated method of generating a browsable, hyper-linked document from a simple scanned image. We concentrated our work on cutaway diagrams, as shown in Figure 2. These documents present the advantage of containing simple “browsing semantics”, in the sense that they consist of a clearly identifiable legend containing index references, plus a drawing containing one or more occurrences of the same indices.

The setup described in this paper starts from a raw binary image of a cutaway diagram, and delivers an XML description matching the references of the legend with the indices in the image, and a browser for interpreting the XML generated map. The complete document treatment pipeline is conceived within a combined scripting and compiled library environment.

*This paper was presented at the Fourth IAPR International Workshop on Graphics Recognition, Kingston, Ontario, Canada, September 7-8, 2001

[†]Corresponding author. Send e-mail to Bart.Lamiroy@loria.fr

1 Introduction and Objectives

In this paper, we demonstrate an approach to develop high level document analysis tools by composing existing bricks from a comprehensive library of state-of-the art algorithms, by developing a fully automated method of generating a browsable, hyper-linked document from a simple scanned image. The work presented here was conducted in collaboration with the ISA research group of the LORIA laboratory, Océ Print Logic Technologies, and students from the École des Mines de Nancy.

The expressed need for composing existing algorithms naturally emerges from the observation that there exist a large number of papers describing ideas aimed at solving particular points of specific problems. Reusing this work for solving other applications seems to be quite a difficult task. In order to attain a functional level that can be used to tackle real-world problems, there is a pressing need to evaluate, experiment and combine existing approaches into an operational whole.

Moreover, the reuse issue is crucial, since the generic reasoning behind document analysis is to adopt a multi-level **syntax+context=semantics** paradigm. In other terms, most of the time, there is a loop where the semantics of a lower level algorithm, become the syntax of a higher level approach, and so on and so forth. The main problem is that the notion of the three terms : *syntax*, *context* and *semantics* tend to vary rapidly in function of a great number of parameters, one of the most important being the improvements of algorithms and advances in the state-of-the-art.

The syntax of a document tends to come from individual algorithms (vectorization, segmentation, skeletization, ...). These algorithms alone cannot decide on the “*truth*” (veracity, accuracy) of its output. It is only the whole treatment chain that can give information on that. Therefore, the question arises of how to use, to the best, imperfect results.

The context is given by the type of document and the problem that needs to be solved. It is generally expressed in terms of combination and conditions of syntactic expressions and expert knowledge. It is the correct formulation of this contextual information that gives the added value to a new approach that uses basic bricks to construct a higher level solution. Finding the correct contextual formulation is generally a process of trial and error, and requires a flexible framework for testing and reuse.

Another important aspect that needs to be considered in the light of reuse and combination of existing components is the need to describe the results of one brick in a way that can be easily translated (used) for further (more advanced) purposes. In other terms, *finding a flexible way to express the results*. There is some kind of intelligence already present in the way we express the result (there are two kinds of problems: the solved one, and the ill-expressed

ones). In today's world, the expression of choice is the XML format and its various derivatives (SVG, or in a way HTML). Most of the work in the XML context consists in using XSSL, XLT to transform the data, in a form easier to use for other purposes. We address this point in this paper.

As a summary we would like to emphasize the facts that

1. More than 20 years of advances in document analysis has given rise to an enormous pool of performant algorithms. However, they tend to address individual and specific problems, and (independently of any benchmarking or whatsoever) are more or less suited than competing approaches for particular tasks.
2. In order to solve higher level, real-world problems, there is a need for building upon those existing bricks in an open manner that integrates the demand for flexibility and interchangeability.

In that sense, we need :

- (a) a way to communicate results of one given treatment to another without being restricted to particular formats of their representation.
- (b) an experimentation and development framework that allows a flexible combination of different bricks, and facilitates their interchange. The principle objective is to form meta-bricks expressing an enhanced context and opening the way to a straightforward assembly of intelligent components, and finally achieve a full treatment pipeline architecture.

The outline of the paper is as follows. First, we more thoroughly analyze the needs in terms of operational requirements for a platform that would suit the needs of flexible inter-component communication, and allow for efficient experimental research. Secondly, we present a test-case for this environment, that consists in converting a cutaway diagram into a browsable document, and present the result of an automated treatment pipeline solving this problem.

2 The model

In this section, we describe our approach to achieve the objectives expressed in section 1. At first, we refine the requirements induced by the introduction, then we propose our solution.

2.1 Requirements

The model for testing ideas in graphic analysis is the pipeline: an image is run through a series of operations, resulting in other images and other kinds of information extracted from those images and in various formats.

Our model should comply to the following requirements:

1. We want this pipeline model to be evolutionary, in the sense that it should be modifiable as easily as possible. This is necessary for several reasons. First, there is a need for flexible experimentation of new algorithms, combining existing components, that are still in a non-finalized stage of development. Second, advances in the state-of-the-art may require fine-tuning of details or modifications and adjunctions of new concepts.
2. We want to be able to add our home-made components as easily as possible. If some (part) of those components play the same role, they need to be interchangeable for comparison purposes. Thus, our platform has to be modular.
3. Ideally, our platform should give us access to the greatest number of existing components, such components offering high-level services, like the ability to build a user-interface for demonstration purposes. Furthermore, the use of a console mode, *i.e.* a mode in which we can test step-by-step some idea, deciding only when seeing the result of the current step what will be the next step, is a must for debugging or fine-tuning of parameters.

2.2 Implementation Proposal : Scripting Languages

It is now well known that object-oriented technology alone is not enough to ensure that the system we want to develop is flexible and adaptable. Scripting languages go a step further than object oriented framework, as they integrate concept for component-based application development. As it is stated in [7], *Scripting languages assume that there already exists a collection of useful components written in other languages. Scripting languages aren't intended for writing applications from scratch; they are intended primarily for plugging together components.* They thus tend to make the architecture of applications much more explicit than in an object-oriented framework [9].

- Scripting languages are extensible [1]: this means that new abstractions can easily be added to the language, encouraging the integration of legacy code.
- They are embeddable, which means that they offer a flexible way to adapt and extend applications. As it is stated in [9], *the composition of components using a script leads to a reusable component. Therefore, components and scripts can be considered as a kind of component algebra.*
- Scripting languages support a specific architectural style of composing components, namely the *pipe and filter* style supported by Unix shells. As we have stated before, this is a good prototyping architecture, well adapted to image processing.

We more closely studied two classical scripting languages: Tcl [6] and Python [8]. Both languages do have numerous existing (and maintained) components covering a large part of classical development tasks. Moreover, they both enjoy the benefit of a “console” mode, in which one can test ideas step by step; this is a very important feature for experimentation. Currently, we use Tcl for its exceptional Tk component, dedicated to building user interfaces.

To obtain a fully operational experimentation platform, we build upon components from our own graphic analysis library QGAR [3, 4]. QGAR is a library written in C++, which implements basic methods for graphic recognition: among the methods implemented, it contains some low-level algorithms (binarization, segmentation, gradient methods, convolution masks, ...) and some vectorization methods. The use of SWIG [1] allows to automate the creation of the component: SWIG is a compiler that takes C/C++ declarations, turns them into the “glue” to access them from common scripting languages, including Python and Tcl.

As a side note, we remark that this approach proceeding has already been validated elsewhere a number of times. Where we are concerned, the idea was already implemented for the purpose of building a platform for OCR performance evaluation, combining various commercial OCR engines, and designing a common standard XML output for those OCRs [2]. We believe that this general procedure should be developed as a standard among the graphic recognition community.

One of the more difficult parts of succeeding this kind of generic architectures, as already mentioned before, is the realization of a component algebra. The basic idea being the flexible combination of existing components to form new components, there is a crucial need of representing data for interchange between components¹. We are conscient that the conception of such an algebra, and the representation of the data that is exchanged between components goes far beyond the scope of this paper. A thorough study of the formal requirements of such a construct will probably end at the fundamental difficulties of graphics analysis: what are the image semantics and how to represent them? In our paper, we restrict ourselves to *a priori* known and very simple semantics. It is our ambition to continue this study in depth and beyond.

3 An Example of Application: a Browsable Cutaway

The application we have chosen for illustrating our ideas is to render a cutaway diagram browsable. Cutaway diagrams present nicely identifiable image semantics, but extracting them requires the collaboration between a number of image treatments. They are composed of a graphical image, containing a number of *tags* (which we shall refer to as *indices*) denoting zones of interest

¹Hence the use of the term *algebra*: the combination of components forms a component

in their vicinity. A copy of all tags, together with an explicative text, is also present along the graphical image.

In this paper we take the following assumptions:

1. Tags are formed by alphanumeric characters we shall refer to as *strings*. There is no supplementary assumption concerning the tags, more particularly, we do not assume there are particular signs (such as surrounding circles, as in one of our examples – *cf.* Figure 2) that would allow easy detection of the tags.
2. The legend has an array structure, consisting of an undefined number of macro-columns, each of which contains two micro-columns: the leftmost containing the tag, the rightmost containing the explicative text.

Our goal is to detect the tags in the image, to detect the legend, and match the tags with the corresponding explicative text in the image by producing a mapping in the form of an XML document. A home-brew browser will allow navigation through the document: clicking on a tag in the image or in the legend highlights the corresponding tag(s).

3.1 General Algorithm Outline

In order to achieve this goal, we use the pipeline architected software model that takes a raw image as input, and sequentially executes the following treatment (Figure 1):

1. Separate text from graphics, and produce two new, raw images, one only containing text, the other only containing graphics.
2. Take the text image as input, analyze it, and locate the position of the legend containing the needed references. Separate the text image into another two raw images : one containing the legend area, another containing the rest.
3. Taking the legend area as input, analyze its structure and output a list of image zones, containing the references to be searched for in the rest of the image. These zones are fed to an OCR, and the final result of the treatment consists of a list of image zones and the corresponding references, as recognized by the OCR.
4. Taking the rest of the image as input, we also produce a list of the remaining text zones as well as the recognition results using the same OCR.
5. Both lists are analyzed in order to produce a coherent browsable structure, expressed in XML.

6. The XML file is then given as input for a customized browser-editor, written in Tcl/Tk, that allows navigation and editing of the final document.

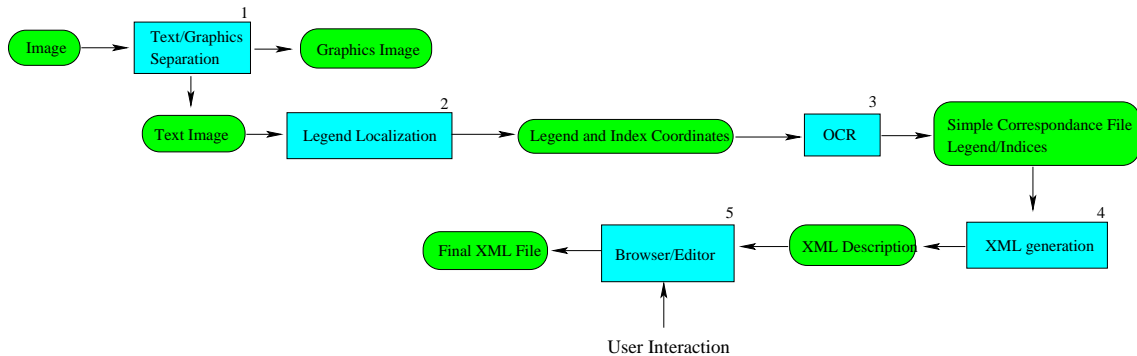


Figure 1: General Software Architecture : A modular pipeline with easily interchangeable bricks.

In what follows, we shall briefly describe the conception and functioning of each module, and show some results and screenshots (Figures 2 and 5) of our prototype.

3.2 Text – Graphics Separation

The text–graphics separation is an enhanced implementation of the FLETCHER–KASTURI algorithm [5]. It consists in an analysis of the connected components of the initial raw binary image. The algorithm is based on a statistical analysis of the distribution of the bounding boxes of all connected components. Given the fact that alphanumerical characters roughly have small squarish bounding boxes, a simple thresholding with respect to the bounding boxes’ surface and ratio allows it to quite nicely separate text from graphics, as shown in figure 2.

It is noteworthy to mention that the quality of the Text/Graphic segmentation does not need to be absolutely perfect for our algorithm to work properly. As we explain in section 3.4, we can easily cope with noise in the Text layer. The Graphics layer, however is (currently) completely discarded, and undetected text elements (most likely being tags) cannot be recovered.

An interesting side-effect of the algorithm is that we have identified individual bounding boxes of text elements. Therefore, every identifiable tag is necessarily one of the found text elements. The inverse is false, however. A found text element is not necessarily a tag.

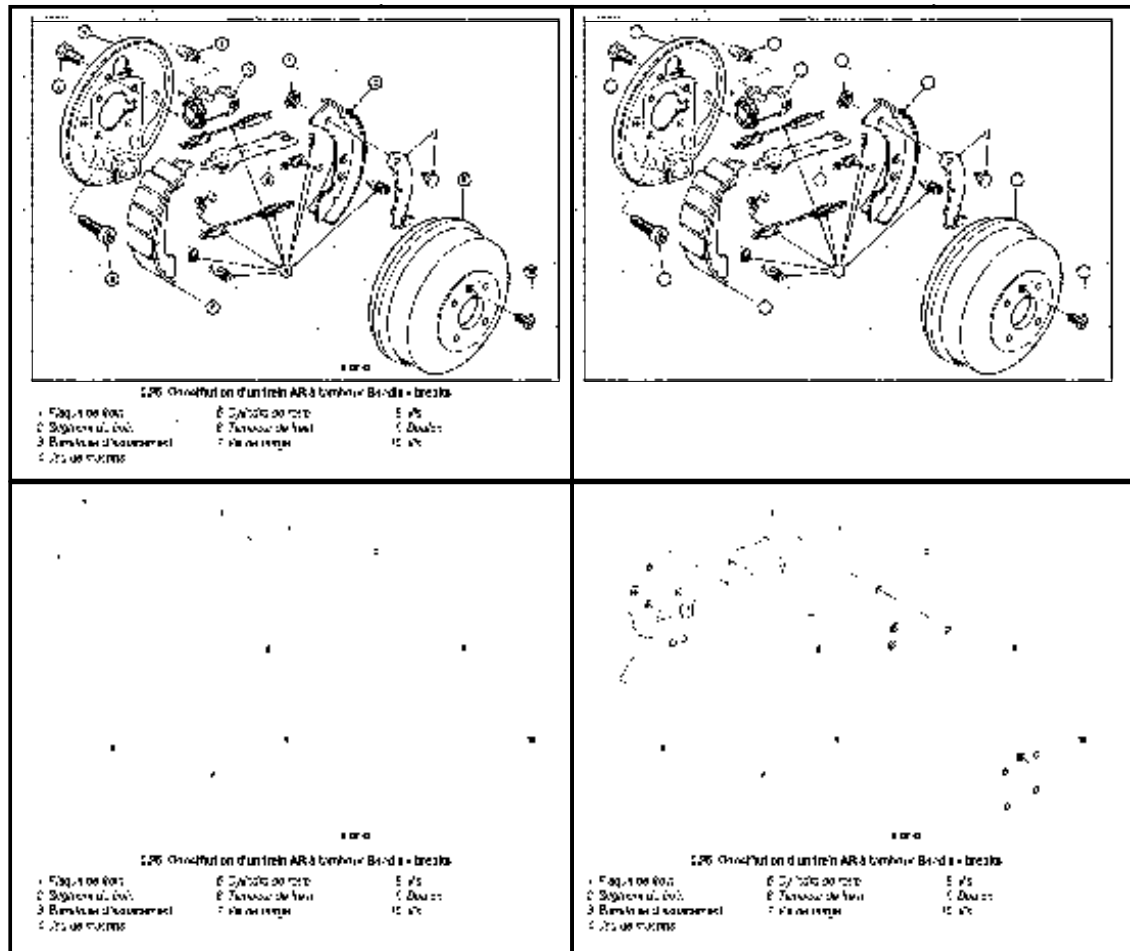


Figure 2: Cutaway diagram and the aimed results after a run of the text/graphics separation. *From upper left to lower right* : original cutaway with legend at bottom, aimed graphics image after separation, aimed text image after separation, currently obtained text image after separation without particular filtering.

3.3 Identifying the Legend

Once the text elements have been separated from the rest of the drawing, the analysis of the semantics, as we defined them in the beginning of this document, can be extracted. The main element describing these semantics consists of the legend containing the index references.

Since we know that the legend is generally a concentrated “blob” of texts (independent of its content), and that the referenced indices in the drawing are most often sparsely scattered over the image, the following algorithm is general and robust enough to extract the location of the legend (although it might need some tuning in the case of awkwardly formed, *i.e.* non-rectangular, legends) :



Figure 3: Closure on the (cluttered) Text Layer of our Cutaway Diagram.

1. Apply a Run-Length Smoothing Algorithm closure [10] on the image, with its parameter roughly the size of a text element. This will result in the legend form a homogeneous zone of filled black pixels. Figure 3 illustrates the results of this step for a combined horizontal and a vertical closure.
2. A simple search of very high density pixel clusters rapidly locates the legend with respect to the rest of the image or other textual elements. Our current implementation can be easily enhanced, but for the moment we simply take the legend to be the intersection of the 30% highest values of the horizontally and vertically projected pixel value histograms (which implies we consider the legend zone to be a set of disconnected rectangles).
3. We currently have not implemented a detailed analysis of the found legend zone. For the same reason that we already mentioned concerning

the clutter in the Text/Graphic separation, the following phases of our pipeline already manage relatively well without this detailed analysis. Further versions will need to incorporate this finer level of detail, however.

At this stage, we have identified, on the one hand, the tags within the drawing, and, on the other hand, the legend zone containing, among other data, occurrences of the same tags. The remaining step consists in identifying which tags appear in the legend, and where.

3.4 Cutaway–Legend Matching

In order to obtain a matching between the tags and the legend we apply a very simple algorithm: we run an OCR over all extracted text elements, and link identical results together.

This approach has the advantage to be rather robust. On the one hand, clutter in the Text layer will give at best a “not recognized” result from the OCR, or a random text label, at worst. The probability to encounter an identical text label (provided it contains a sufficient number of individual characters) is very low. Clutter therefore has very few chances to be matched with anything. A similar reasoning holds for the micro-analysis of the legend we mentioned before. Since the legend normally is composed of a tag, followed by a longer text string, there are few chances that the text string will be mismatched by a tag.

3.5 XML Generation

At this stage of our treatment chain we express the result of all different modules in XML [11]b. The reason for using this description language, rather than any other format, comes from two reasons.

First of all, the work presented in this paper is a first step in developing a framework for generating browsable graphics documents, with a non dissimulated aim to extend the notion of both “*browsable*” and “*document*” as our work advances. We therefore need an expression language in which to represent the underlying semantics of such documents that offers the possibility to easily extend and enrich their description. XML was designed for this purpose.

Secondly, since this is work in progress, and one of the objectives is to obtain qualitative study of the suitability of individual graphics recognition algorithms, we need a coherent framework for data interchange. Here again, especially in the light of the previous point, XML is the way to go.

3.5.1 XML for Browsing Semantics

Our main goal remains to express the content of a browsable document. We therefore decided on a DTD that would allow the interpretation of a cutaway

tags, as detected by our algorithm. Figure 4 contains this DTD.

```
<!-- DTD proposal of 23/01/2001 -->

<!-- Tag value as recognized by OCR -->
<!ELEMENT OCR_RESULT (#PCDATA)>
<!ATTLIST OCR_RESULT confidence (sure|doubtful) #IMPLIED>

<!-- Point Elements, containing two integers: x,y -->
<!ELEMENT UPPER_LEFT (#PCDATA)>
<!ELEMENT BOTTOM_RIGHT (#PCDATA)>

<!ENTITY % text_zone "(OCR_RESULT?,UPPER_LEFT,BOTTOM_RIGHT)">

<!ELEMENT NAVIGABLE_IMAGE (IMG,LEGEND,DRAWING)>

    <!ELEMENT IMG (SRC)>

        <!-- path to the source image -->
        <!ELEMENT SRC (#PCDATA)>

    <!ELEMENT LEGEND (EXTRACTED_ZONE*)>

        <!-- Text zone -->
        <!ELEMENT EXTRACTED_ZONE (%text_zone;)>
        <!ATTLIST EXTRACTED_ZONE id ID #REQUIRED>
        <!-- For evolution towards word pointers -->
        <!ATTLIST EXTRACTED_ZONE type (number|text) #IMPLIED>

    <!ELEMENT DRAWING (PARTS*)>

        <!-- Tag in the drawing -->
        <!ELEMENT PARTS (%text_zone;)>
        <!ATTLIST PARTS id ID #REQUIRED>
```

Figure 4: DTD of a Browsable Cutaway

Its fairly straightforward and simple. A browsable document (`NAVIGABLE_IMAGE`) contains three sub-parts:

- The path to a source image (`IMG`), mainly for displaying reasons.
- A list of rectangles (`LEGEND`), that contains a unique identifier as an attribute, and is supposed to be composed of two corners (`UPPER_LEFT` and `BOTTOM_RIGHT`) and an OCR recognition tag. This list represents the recognized items of the legend zone of the document.

- A similar list of rectangles (`DRAWING`, that also contains a unique identifier as an attribute, two corners (`UPPER_LEFT` and `BOTTOM_RIGHT`) and an OCR recognition tag. This lists enumerates the tags scattered over the drawing part of the document.

Our browser (*cf.* Figure 5) simply parses both lists and activates links between items of the `DRAWING`-list with those of the `LEGEND`-list, when they present the same OCR recognition label. Unlinked zones can be highlighted as warnings for possible further fine-tuning and error correction methods. Moreover, the formal description of the document and the availability of a browser/editor, allows for a straightforward inclusion of human intervention: the browser/editor can allow for modification of the tag dimensions (merge/split/rezise) and tag labels (and indirectly the links) or even allow for creation and deletion of tags. Having a formal description of the document as well as a browser/editor nicely turns our experiment into a complete, exploitable platform for semi-automated hyperlinked document generation.

3.5.2 XML for Data Exchange

If we take some height with respect to the work that has been conducted in the context of this paper, one notices that the expressive power of XML, can be used far beyond expressing the semantics of the final result (*i.e.* a browsable drawing). As a matter of fact, XML can be the exact missing link for interoperable modules. Indeed, in the light of what we intend to obtain by using scripting languages: *i.e.* a component algebra, any image treatment module can be either a final implementation, a brick within a more complex treatment pipeline, or both. A sound, and universal knowledge representation scheme is absolutely necessary in this context, and XML and its derivatives seem to be the perfect key to this framework.

4 Conclusion

In this paper we have presented a first approach towards the creation of intelligent browsable documents, applied to cutaway diagrams. We identify the tags in the drawing and correlate them with the occurrences of the same tags in the legend adjoining the graphics.

Moreover, we identified the need for a real component algebra for document analysis applications and research and defined the major requirements for such an environment:

- flexibility and interchangeability, which we obtain through scripting basic image treatment bricks
- and interoperability, which we want to obtain by defining a sound XML-based description format for image and document analysis results.

The very first implementation we made of this component algebra, based on the Tcl/Tk scripting language and our own graphic analysis library QGAR, has demonstrated the power of such a design for rapidly obtaining a complete, exploitable platform for semi-automated hyperlinked document generation.

References

- [1] D. M. Beazley. SWIG and automated C/C++ scripting extensions. *Dr. Dobbs Journal*, (282):30–36, February 1998.
- [2] A. Belaid, L. Pierron, L. Najman, and D. Reyren. *Bibliothèques Numériques*, chapter La numérisation de documents : Principe et évaluation des performances. ADBS, 2000.
- [3] Ph. Dosch, C. Ah-Soon, G. Masini, G. Sánchez, and K. Tombre. Design of an Integrated Environment for the Automated Analysis of Architectural Drawings. In S.-W. Lee and Y. Nakano, editors, *Document Analysis Systems: Theory and Practice. Selected papers from Third IAPR Workshop, DAS'98, Nagano, Japan, November 4–6, 1998, in revised version*, Lecture Notes in Computer Science 1655, pages 295–309. Springer-Verlag, Berlin, 1999.
- [4] Ph. Dosch, K. Tombre, C. Ah-Soon, and G. Masini. A complete system for analysis of architectural drawings. *International Journal on Document Analysis and Recognition*, 3(2):102–116, December 2000.
- [5] 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.
- [6] J.K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.
- [7] John K. Ousterhout. Scripting: Higher-level programming for the 21st century. *Computer*, 31(3):23–30, March 1998.
- [8] G.V. Rossum. *Python Tutorial*. iUniverse.com, 2000.
- [9] J.-G. Schneider and O. Nierstrasz. Components, scripts and glue. In J. Hall L. Barroca and P. Hall, editors, *Software Architectures - Advances and Applications*, pages 13–25. Springer, 1999.
- [10] K.Y. Wong, R.G. Casey, and F.M. Wahl. Document analysis system. *IBM J. Res. Develop.*, 26(2):647–656, 1982.
- [11] Extensible markup language (XML) 1.0 (second edition). Technical report, w3C, 2000. <http://www.w3.org/TR/2000/REC-xml-20001006>.

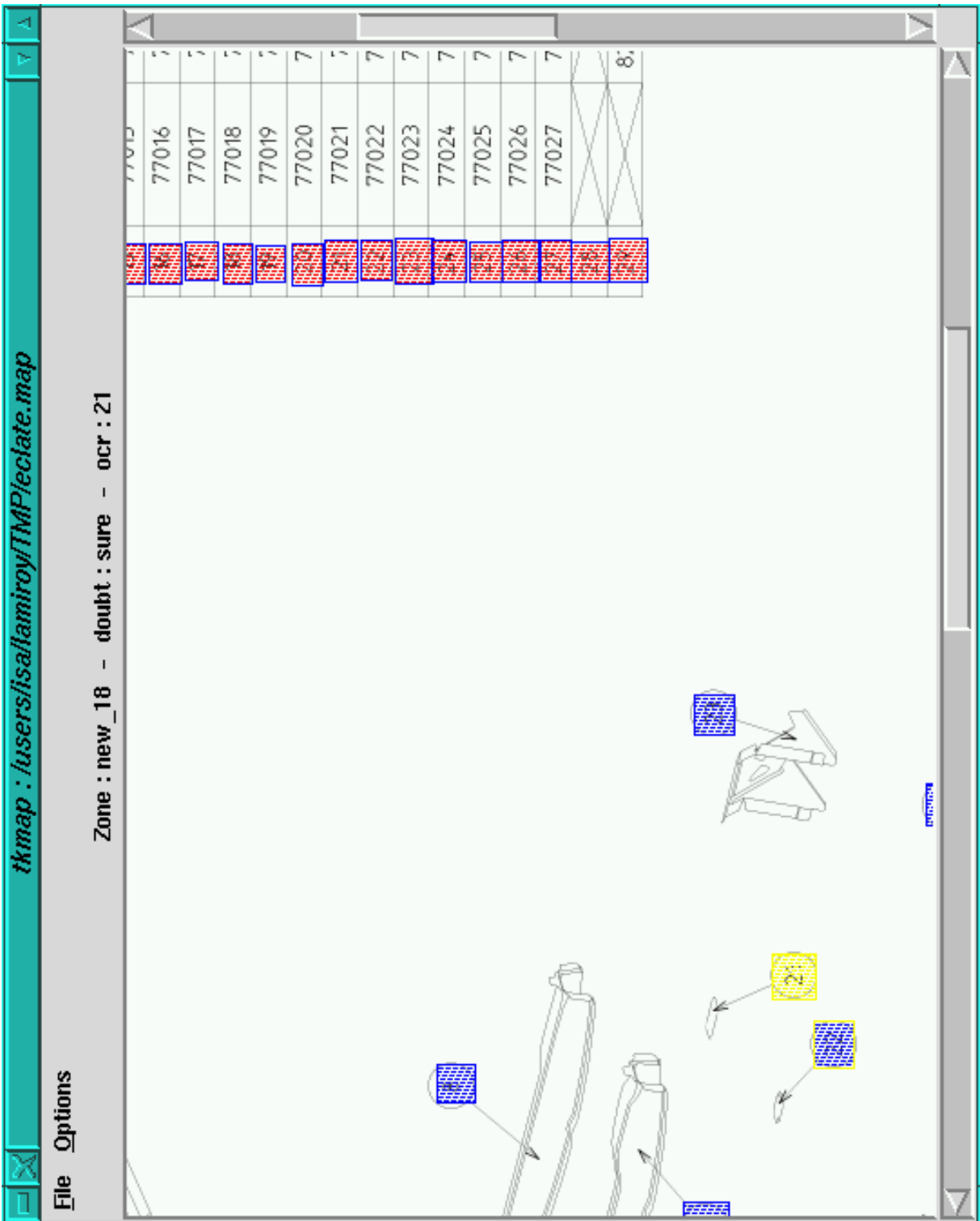


Figure 5: Screenshot of our Browser/Editor, showing a zoom of a fully browsable cutaway diagram.