Projet Industriel promotion 2011

PROJET INDUSTRIEL 27

PETIT François

SOYER Baptiste

NGOM Mor

BOUGHIDA Rafik

PAIS Vincent

DESGRANGE Yoann

Commanditaire – Inkscape France

Tuteur industriel - Steren Giannini

Tuteur ECL – René Chalon

Date du rapport – May 18, 2010

Final Report

Image properties dialog enhancements for Inkscape, a vector graphics editor.

Industrial Project

Inkscape

FINAL REPORT

Contents

Introduction

"Industrial Projects" are a part of every second-year students' degree courses at the École Centrale de Lyon, a French engineering college. Since 2008, Inkscape has been involved in several projects with the ECL in order to improve different parts of the software. In fact, our contact with Inkscape is a former student of the ECL: Mr. Steren Giannini who was the team leader of the first project aiming at improving Inkscape. Our team immediately took an interest in this year Inkscape project. Its goal was to improve how Inkscape handles images (to go from absolute to relative links), and enhance the image properties dialog.

In the school, our coordinator was Mr. René Chalon who is a teacher and researcher in the Computer Engineering and Mathematics department.

Throughout this report, we first describe the three main objectives of this project. Then we summarize the work done in order to fulfill each of them. Finally, we have gathered in the appendix complementary information about the project: organization and our general feelings about the project.

I. The project

I.1 What is Inkscape?

Inkscape is a vector graphic editor which is fully compatible with the SVG, XML and CSS standards. It is also a free open source software which means that its source code can be read by everyone and shared without any kind of restriction.

One of the predominant characteristics of Inkscape is that it is a vector graphic editor. That means that it deals with vector graphics instead of traditional images (also known as raster graphics) which are used by software such as Paint, Photoshop or Gimp to name but a few. To put it quite simply, a traditional image can be seen as a matrix of pixels whereas in vector graphics all the shapes are defined by points and curves or any other mathematical primitives. This difference entails two main consequences: files are usually smaller when encoded as vector graphics and even better there is no aliasing when you zoom in (as you can see below on Figure 1).

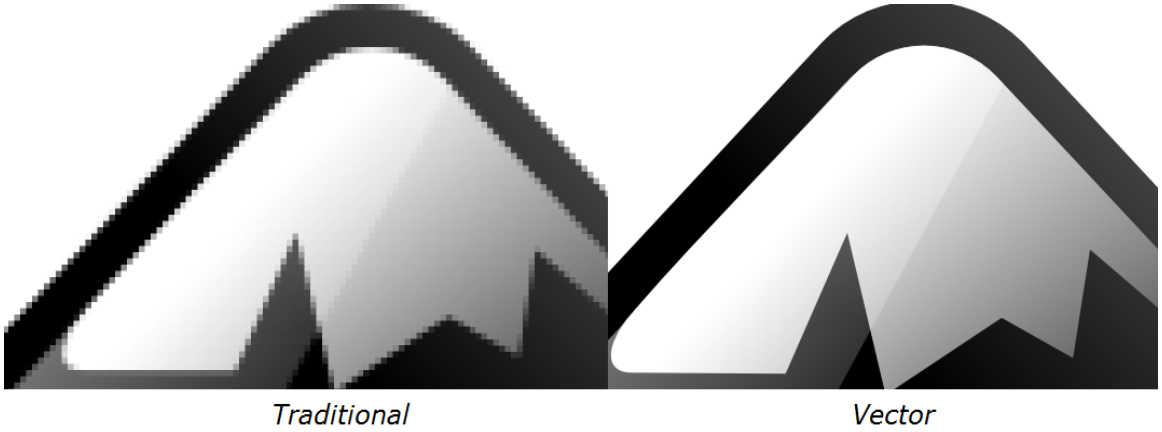

Figure 1: A traditional and a vector version of the Inkscape logo.

As Inkscape is open source, anyone can download the sources, modify them and have these modifications implemented to the main branch of the software if it is validated by the developer's community. Every features and

enhancements that are foreseen or necessary are listed and we have been tasked with one of them, which can be separated into three different sub-goals.

I.2 Absolute and relative links

These three objectives are closely related to the notion of relative and absolute links. So before going any further, we are going to explain what differentiate them and to sum up the pros and cons of each solution.

Absolute links are links which give us the whole address of a file. Indeed, they list every folder you have to go through from the root to the file. An example of absolute link in Inkscape is:

File//C:/Users/Yoann/Documents/Ecole/2009-2010/PI/Fichiers SVG/exemple.svg

Relative links are links which give us a shortened version of the address of a file. The way this address is interpreted by the computer depends on the position of the file where this address is used. For instance, if you are in exemple.svg and you use a PNG image which has for absolute link in Inkscape:

File//C:/Users/Yoann/Documents/Ecole/2009-2010/PI/Images/logo2009-ec-lyon.png

You have to use the following relative link where "../" is a command that means that you go to the parent folder of your file.

../Images/logo2009-ec-lyon.png

Now, only the absolute links are used in Inkscape when you import images into your SVG file. An absolute link is useful when you move your SVG file on another part of your hard drive, indeed Inkscape will still manage to find the images that are needed to display the file. But if you want to send the whole project (SVG file and the imported images) to a colleague (in our example, that

would mean sending the folder PI), the absolute links won't work anymore whereas if you had relative links there would be no problem as long as your colleague doesn't scatter the content of PI among different places of his hard drive. You can see how an image with a broken link is indicated on figure 2. From another point of view, we can say that relative links are likely to become very complicated to use and to read when you have to go up four or six parent folders.

Finally, an alternative way of using another image in a SVG file is to embed it. When you choose this option, the image is fully part of the file meaning that even if you delete the image from your computer, it will appear when you open the SVG file.

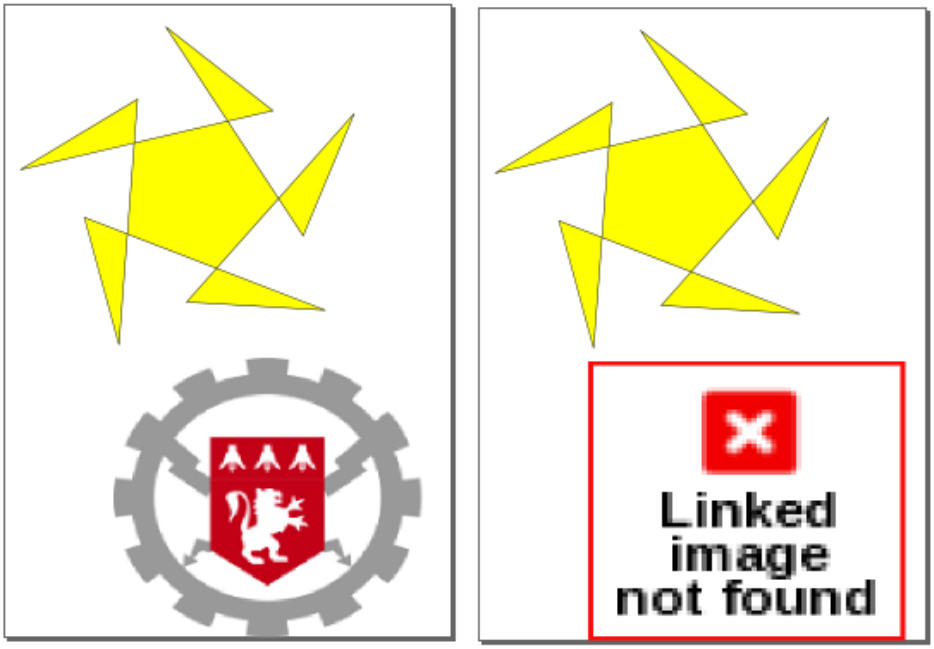

Figure 2: Example of working and broken links.

I.3 What did we have to do?

As written before, the project has three different objectives and all of them have to do with the way of linking images in Inkscape.

I.3.1 Use of relative links

As explained in I.2, relative links are very useful when you send the folder of a whole project to a colleague as there will be no broken link as long as he does not scatter the content of this folder. But these links can also be quite confusing when you try to go up a few parents folder. So, we had to implement a system that would use relative links for images which are in the same directory or in children directories of the SVG.

I.3.2 Smart re-linking tool

In the current version of Inkscape if you have several broken links in a SVG file, you have to manually re-link each one of them. That's why our second objective is to provide the software with a tool that can simplify this task. Let us take the example of a SVG file where the following images were used:

/PI/Images/logo-ecl.png
/PI/Images/logo-inkscape.png
/PI/Images/example.png

Due to several handlings (of the SVG file or of the images), all the links regarding these images were broken (but the images still share the same folder). With our tool, as soon as the user has manually re-linked one of these images (with a newly implemented user interface), the software will offer to automatically re-link the other ones if:

- they have broken links,
- they have the same original path,
- they exist in the new location.

This dialog box should open itself automatically when you open a new svg file where there is an image with a broken link. Moreover it must be accessible in the newly redesigned image properties dialog.

I.3.3 Image Properties Dialog Re-Design

The current Image Properties dialog is quite annoying as another one pops up for each image. So part of the re-design is to make this window re-usable and dockable, meaning that when the user selects another image: its information should fill the same window and not a new one. Moreover this dialog should also be accessible from the Object menu whereas it is only accessible with a right-click on the image for now. A lot of work also has to be done on the contents of this window (Figure 3 is the current dialog and Figure 4 is what it should look like when we are done).

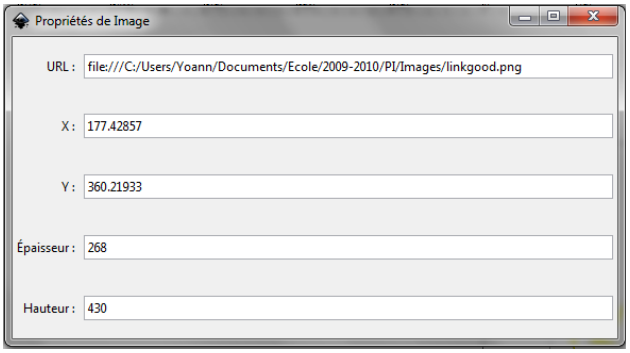

Figure 3 : Current dialog.

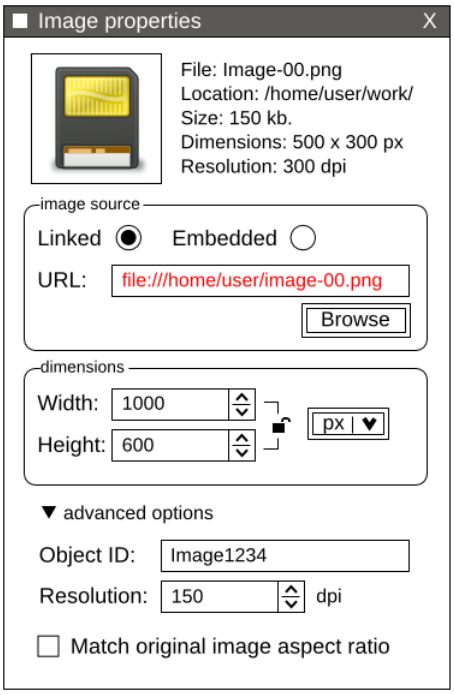

Figure 4: Re-designed dialog.

At the top, there should be an area that shows general information about the image, along with a thumbnail that will give a glimpse of the image we are working on.

The URL field should be equipped with a browse button for easy image re-linking. The URL text can also be rendered red if the link is broken making it easy for the user to spot that there is a problem. Moreover, we have buttons to select if we want the image to be embedded or linked (see I.2 for the difference between the two of them).

The fields for the dimensions of the image are to be kept but you should now be able to choose to make a proportional transformation and the unit of these numbers.

The position fields should be removed as they are more part of the object's attribute (ie which each and every Inkscape objects share) than of properties that are specific to images.

Finally, we should add an "advanced options" area in which the user will be able to enter the object ID, to change the resolution of the image and to use the original image's aspect ratio.

I.4 Methods of work

First of all, Mr. Giannini advised us to work on the source code of Inkscape in the Linux environment: Ubuntu. Indeed, this way it was easier for him to help us as during his project he has worked on Inkscape under Linux. So we all had to install Ubuntu and to configure it with the correct libraries, it also took some time to adapt ourselves to this new environment.

Then, Mr. Giannini created a branch for our project on launchpad.net. Each member of the team created an account on this website and we were added to the users of this branch. That allowed us to work on the same source code and to send our modifications when they were working for Mr. Giannini to examine them. These modifications were automatically merged with the source code of the branch and so it represented an easy way of gathering the work of everyone in the same source code. Sadly we did not manage to fully use this method, but as this project is not very complex in term of organization, it did not hinder our work.

II. Use of relative links

II.1 Approach

II.1.1 Initial solution

Our main problem in the beginning of the "relative path" part was to understand where to modify the source code. To locate this emplacement, we looked into a SVG file, to find what was written in it, and then to search in the files for a function which was writing the UML code. In the "src/xml/repr-io.cpp" file, we found the "sp_repr_save_rebased_file" whose function was obvious.

After having spent some time to understand what was the syntax of the paths (URI), and to find which argument of an "image" node contains the path of the actual image ("xlink:href"), we decided to modify the function which save the documents.

In Inkscape, a document has two kinds of representation: an XML tree (the objects the program works on) and the SVG file (raw text which represents the XML tree but which can be saved on the hard disk). This saving function converts the abstract XML tree used by the program into a concrete SVG file.

Our idea was to let the XML tree use absolute paths, but at the moment of saving the document, this function would write the corresponding relative path (if the image is in the same folder than the SVG file, or in children directories).

The first effective version of our code was this one (explanations included):

```
//Check if the object's attribute we are working on is a path
if(strcmp(g_quark_to_string(iter->key),"xlink:href")==0) {
    // Make a copy of the pass to be able to modify it before saving it.
    strcpy(relPathBuff, (const gchar*) iter->value);
    // If the path begins by "file://", remove it.
    if(strncmp(relPathBuff,"file://",7)==0) relPathBuff+=7;
```

```
        //If the path begins by “data://” , remove it.
        if(strncmp(relPathBuff,"data://",7)==0) relPathBuff+=7;
        // If the object’s path attribute beginning is the new path of the SVGfolder
        if(strncmp(new_href_base,relPathBuff,strlen(new_href_base))==0) {
              // Remove the corresponding beginning
              relPathBuff += strlen(new_href_base);
              // Modify the remaining path for it to be a correct relative path (beginning by “./”)
              if(strncmp(relPathBuff,"/",1)==0)  out.printf(".%s", relPathBuff);
              else out.printf("./%s",relPathBuff);
        }
        // If the paths of image and SVGfile do not match
        else {
        // Write the unmodified path
         repr_quote_write(out, iter->value);
        }
    }
  else
```

It was inserted in the “for” loop of the function “sp_repr_write_stream_element” (in “repr-io.cpp”), just before the line:

```
    repr_quote_write(out, iter->value);
```

The declaration of the variable relPathBuff had to be added at the beginning of this function:

```
    gchar * relPathBuff; relPathBuff = (gchar*) malloc(256);
```

Our code was doing what it was supposed to on our Linux computer, but the path format was still dependant on the operating system. To avoid that, we used a constant named “G_DIR_SEPARATOR” instead of the explicit “/”, specific to Unix operating systems (whereas Windows uses “\”). This constant is implemented according to the operating system, and is used to solve problems

like ours. A few modifications later, the code was adapted, and usable on different operating systems.

II.1.2 Improvements

The problem with our modification in the saving function was that the XML tree always contained absolute paths, as we explained previously. Even when we loaded SVG files with relative ones, they were converted into absolute paths in the XML tree created. So, when they appeared into the "image properties" box, the modification could not be seen. To solve this, we decided to move the modification to the place in the source code where the image was added to the XML tree representing the working document. To find the right place, we identified every case where an image is created in the XML tree:

- importation using "File > Import"
- drag and drop a file from the desktop or a file explorer
- copy and paste an image (from another SVG file for example)
- open an image file (".PNG" for instance) directly from Inkscape

And we examined how these functionalities are implemented. During the process, we found an existing function ("sp_relative_path_from_path", defined in "dir-util.cpp") which converts absolute paths into relative ones so we decided to use it (according to the "Don't Repeat Yourself" principle of software engineering), even if it did not take into account the possibility of having a path containing "file://" before its actual beginning (and so doing nothing if it was the case).

After a while, we found the place where images are added to the XML tree (in the "sp-image.cpp" file). We noticed that the "sp_image_write" function was called in the four cases.

In order to create a relative path from an absolute one, we need the path of the folder the created path will be relative to. The real problem was to get the path of the SVG file opened in Inkscape (it was an argument of the saving function we modified before, so it was not a problem in our initial solution). First, we tried to use the function of the library Glib named "g_get_current_dir". This

function was supposed to return what we needed, but after some tries, we realized that it was not working properly. The result of the function was generally "/home/*user_name*/" instead of the actual path of the working file. The final solution we found to get the path needed was a global variable already present in Inkscape: SP_ACTIVE_DOCUMENT. The structure of this variable is complex (its type is SPDocument), but the only thing we needed to know was its "base" argument (containing the path), and we could then apply our modifications (the code is in part II.2.1).

After some tests, we realized that there was still a case that was not working well: the importation of images by "File > Import". For example, if we tried to import "/folder1/folder2/image1.bmp" into "/folder1/doc1.svg", the relative link obtained in doc1.svg was not "folder2/image1.bmp" as it should be but "folder2/folder2/image1.bmp", which does not exist (so that a "linked image not found" icon appeared).

The reason of this odd new problem is complicated and was not easy to understand. All the technical details are given in part II.2. Eventually, we identified the function responsible for this ("rebase_hrefs", in "rebase-hrefs.cpp") and found a solution which corrects the problem but which also involves a strange behavior in a very special case explained in part II.2.2.

Finally, we noticed another problem: the "copy and paste" function is not handled properly when it is applied to an image referenced to by a relative link. This time, it has nothing to do with our modifications: this problem already existed before we change the code. But as only absolute paths were used (if we did not change them manually into relative ones), almost nobody may have noticed it. Now that we have given the priority to relative paths, this issue has quite important consequences and that's why we have decided to try and solve it. Unfortunately, it is still a work in progress as we have not yet found a satisfactory solution. We will do our best to manage to solve this problem before the end of the project even if it was not explicitly part of our objectives at first.

II.2 Summarized final results

Two files have been modified: "sp-image.cpp" and "rebase-hrefs.cpp"

II.2.1 Modifications in "sp-image.cpp"

First, at the beginning of "sp-image.cpp", we had to add two "#include" instructions:

```
#include "inkscape-private.h" //for SP_ACTIVE_DOCUMENT
#include "dir-util.h" //for sp_relative_path_from_path
```

Then, the main part of our modifications is in the function "sp_image_write":

```
/* (François & Rafik)
    * The following lines replace : repr->setAttribute("xlink:href",image->href);
    * If image->href is an absolute path, we will try to transform it into a relative one.
    * The condition is that the image is in the same directory than the svg file, or in
    * children directories (we use 'sp_relative_path_from_path')
    */
    gchar * image_href = strdup(image->href);
    gchar * image_href2 = image_href;   //image_href2 points to the same string as image_href (a copy of image->href)
    if(!strncmp(image_href,"file://",7)) image_href2 += 7;
    //image_href2 is image_href without "file://". Indeed, with "file://", g_path_is_absolute
     //always return false
    if (g_path_is_absolute(image_href2) && SP_ACTIVE_DOCUMENT) {
  //we don't do anything if the path is already relative or if we are working in a new document
  //(not saved yet)
      strcpy(image_href2,sp_relative_path_from_path(image_href2,SP_ACTIVE_DOCUMENT->base));
    }
    g_warning("sp-image (image_href=%s ; image_href2=%s) :",image_href,image_href2);
    if(g_path_is_absolute(image_href2))    //sp_relative_path_from_path did not work
      repr->setAttribute("xlink:href", image_href);
    else
      repr->setAttribute("xlink:href", image_href2); //relative path must not begin with "file://"
    g_free(image_href);
```

All the explanations are in the comments. These lines were inserted just after:

```
if ((flags & SP_OBJECT_WRITE_BUILD) && !repr) {
    repr = xml_doc->createElement("svg:image");
}
```

II.2.2 Modifications in "rebase-hrefs.cpp"

Finally, there was the problem of the "rebase_hrefs" function, evoked in part II.1.2. The role of this function is to change all the relative paths of a document (the argument "doc" of the function), supposing that they were relative to the base of this document (for instance: the base of the file "/folder1/doc1.svg" is "/folder1") and that they now should be relative to the argument "new_base". For example, this function is called when the user changes the location of the SVGfile with the option "File > Save as…".

And we realized that it was also called during the importation of a file and that it was this function which added a second "folder2/" in the example we took in part II.1.2. Everything is detailed in the comments which precede the few lines of code we added:

```
/* (François & Rafik)
* The following 'continue' instruction solves a problem in image importations, which
* appeared when we modified the function 'sp_image_write' (in sp-image.cpp) in order to
* use relative paths instead of absolute ones for images when possible.
* In that function, the path is transformed to be relative to SP_ACTIVE_DOCUMENT->base.
* Unfortunately, in the process of file importation, a temporary document with only the
* imported images is created (their paths may be changed into relative ones by
* 'sp_image_write' in this temporary document), and then 'rebase_hrefs' is called to
* rebase the relative paths of the temporary document as if they were relative to the
* imported file (whereas the ones created by 'sp_image_write' are already relative to
* the active document).
*
* In order to solve this problem, rebase_href mustn't do anything if the relative path was
```

```
            * created from SP_ACTIVE_DOCUMENT->base. The only criterion we have found to detect
            * that case is testing if the path already points to an existing file without rebasing it.
            * This may cause some strange behavior if there are two different images with the same
            * name and the same relative path, one from the imported file, and the other from the
            * working '.svg' file. But we assumed this case to be really tricky and for it not to be
            * often seen. */
           if (Inkscape::IO::file_test(g_build_filename(new_abs_base, href, NULL),
G_FILE_TEST_EXISTS)) {
              continue;
           }
```

This code was inserted in the "for" loop (hence the instruction "continue"), just after the lines:

```
           if (!href || !href_needs_rebasing(href)) {
              continue;
           }
```

Let us come back to our example of part II.1.2: we were trying to import the image "/folder1/folder2/image1.bmp" into the Inkscape document "/folder1/doc1.svg". A temporary XML tree is built; it contains only one object, the image, with the absolute path "/folder1/folder2/image1.bmp" which is immediately transformed by our code in "sp-image.cpp" into "folder2/image1.bmp" (relatively to the base of the active document). Then, the "rebase_hrefs" function is applied, considering this path was relative to the base of the imported file ("/folder1/folder2") and that it should now be relative to the base of the SVG file ("/folder1"): the function concludes that it have to add "folder2/" at the beginning of all the relative links of the temporary file (hence the "folder2/folder2/image1.bmp" broken link). But with our modification of "rebase_hrefs", it will not change "folder2/image1.bmp" because it already points to an existing file without rebasing it: if we add the base of the SVG file, we obtain "/folder1/folder2/image1.bmp", which exists.

You may be wondering if there are some cases when the "rebase_href" original function is useful during a file importation and will not be prevented by

our code from doing its dangerous work. The answer is of course yes: it happens when the imported file is for example a SVG file itself and contains relative-linked images. In this case, these relative paths are not modified by “sp_image_write” (because they are already relative) so that this time, in the temporary document, they are really relative to the imported file and not already to the active document. Thus, they must be rebased, and they will be, except in a very particular case…

The example that follows is more complicated than the previous one, but it illustrates very well the functioning of the importation of files with our modifications.

We are now trying to import “/folder1/folder2/doc2.svg” into another Inkscape document “/folder1/doc1.svg”. “Doc2.svg” contains two images: image1.bmp and image2.bmp, which are both in the same directory (folder2) but the first one with an absolute link and the other with a relative one (“folder1/folder2/image1.bmp” and “image2.bmp”). When the temporary document is created, sp_image_write change the first one into “folder2/image1” (relative to the active document) and does not change “image2” as it is already relative. Then “rebase_href” tests if these files need rebasing: the first one does not because folder1/folder2/image1 exists, but the second one will be rebased as /folder1/image2.bmp does not exist. Thus, we eventually obtain two relative links: “folder2/image1.bmp” and “folder2/image2.bmp”, just as we wanted.

The “tricky case” evoked at the end of the comments of the code would be if there was another file named “image2” in “folder1”! In this case, the relative link “image2.bmp” is not rebased, so that it will eventually point to the image2 of folder1 instead of the one of folder2.

III. Smart re-linking tool

One of the purposes of the project was to provide Inkscape with a tool for the treatment of images with a broken link. A broken link occurs whenever the image has been moved or deleted from the directory it was previously saved in. Currently, the only solution that users have is to manually re-link these images, using the Image Proprieties dialog box to modify their path. Our goal is to develop a tool to make this re-linking easier.

First, it needs to warn the user as soon as he opens a SVG document containing at least one image whose link is broken. A window should pop-up and allows the user to indicate the new path thanks to a browser. Inkscape will then modify the corresponding attribute of the image proprieties and display the image on screen.

However, if a document contains an important number of broken links, re-linking every image one by one will still be long and annoying, particularly if they are all located in the same folder. Consequently we need to implement a smart tool: after the user has re-linked the first broken image, our tool will automatically check if the other images are present in the same folder (or any children folder) of the first image. Of course if the search is not successful, a new pop-up window will still appear to re-link the second image, but in a lot of cases it will be a time-saver for the user.

III.1 From XML Tree to display

Our first objective was to understand how Inkscape displays the images contained in a SVG document. A SVG document is represented by a XML tree, where every node is an element of the document (layer, image, geometric form…). Every type of element is represented by its own class, and all these classes inherit the same generic class *SPObject*.

A function, *_updateDocument* is called to realize the display of the document. It reads the XML tree recursively, and on every node calls the right update function - according to the node type - to lead to the full display of the document, as shown in Figure 5.

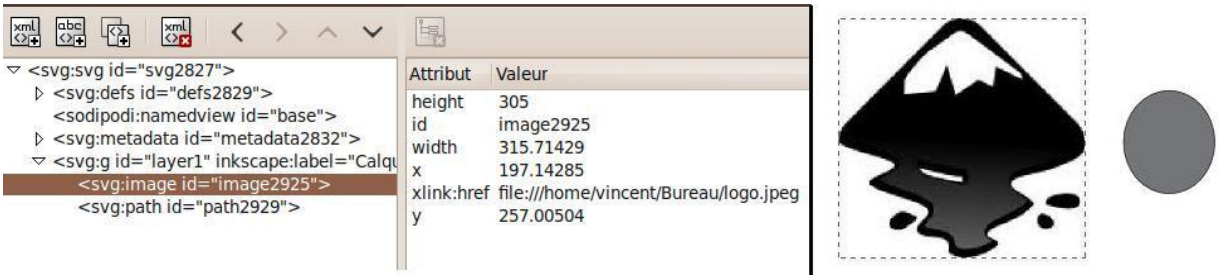

Figure 5: A XML Tree and the corresponding display

Let's now focus on how we update an image. The update function of the *SPImage* class tries to update the pixbuf (a data structure describing an image) according to the flag(s) it has as arguments. These flags indicate which kind of modification the image has suffered from since the last update. If the flag *SP_IMAGE_HREF_MODIFIED_FLAG* (i.e. the path of the image has been modified) is thrown, the function *sp_image_ repr_read_image* will try to create the display (i.e. updating the pixbuf) from the new path. In the current version of Inkscape, if the link is broken, it simply displays a default image shown in Figure 6.

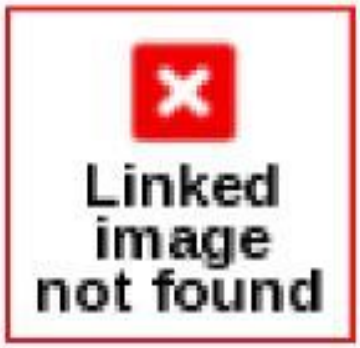

Figure 6: Default image for a broken link

III.2 General Scheme to retrieve the date

A simple idea would be: whenever a broken link is detected in the function *sp_image_repr_read_image,* we let the user choose the new path of the image and simply create the pixbuf with this path. But it won't be a practical solution, as it will display a window for every broken image. Moreover, as each window can't communicate with one another, it cannot know whether or not there are other broken images. Then a smart re-linking tool cannot be implemented.

Another solution would have been to modify all the function from _updateDocument to *sp_image_repr_read_image*, adding a new argument pointing the broken image. This image would have then been treated in _updateDocument. However this idea has quickly been rejected, because the SPImage update function is called not only by *_updateDocument* but also by some other functions, and this solution would imply to modify a lot of classes.

Still we decided to treat all the broken links in Document, as only one window would pop up in every document. The solution we choose to implement uses a new class named *Re-link*. Every time a new document is opened in Inkscape, a new instance of this class is created and is identified by a numeric ID. Then every document will have its own Re-link instance, which will treat the broken links, if needed. During the update, every broken link is stored in a matrix (i.e. an array of arrays). Each line of this matrix, indexed by the ID of its Re-link instance, stores all the broken links of a one document.

Once the update of all the documents is over, every Re-link instance will look if its line is empty or not. If it is not empty, the pop-up window appears, showing all the broken links and propose to re-link the first one. Furthermore, when we move the mouse over a link, we can see the full path of the image.

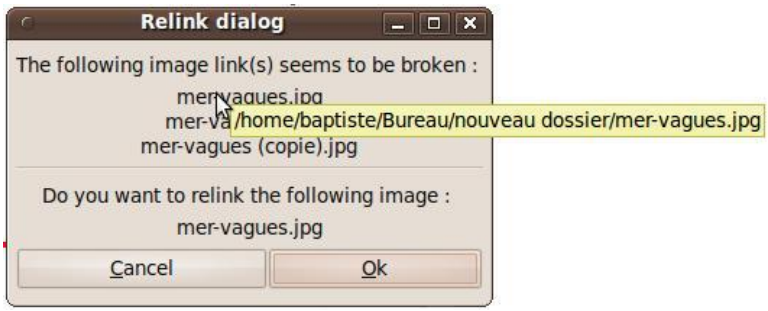

Figure 7: Pop-up window showing the broken links

This window has been made using GTK, a cross-platform toolkit for creating graphical user interfaces (GUI). It was first designed for the free graphics editor GIMP, but it has been involved in many projects including GNOME, a popular desktop environment, and Inkscape. It is an easy way to create all kind of windows (for further information, see 0.3).

III.3 Re-linking of a single image

Whenever a broken image is detected, the user should be able to simply and quickly re-link it. We decided to use a browser window, where the user can navigate in the computer directories to indicate the new path of the image (Figure 8). The same window will be used if the user wants to change the path by the Re-link button in the Image Proprieties dialog box.

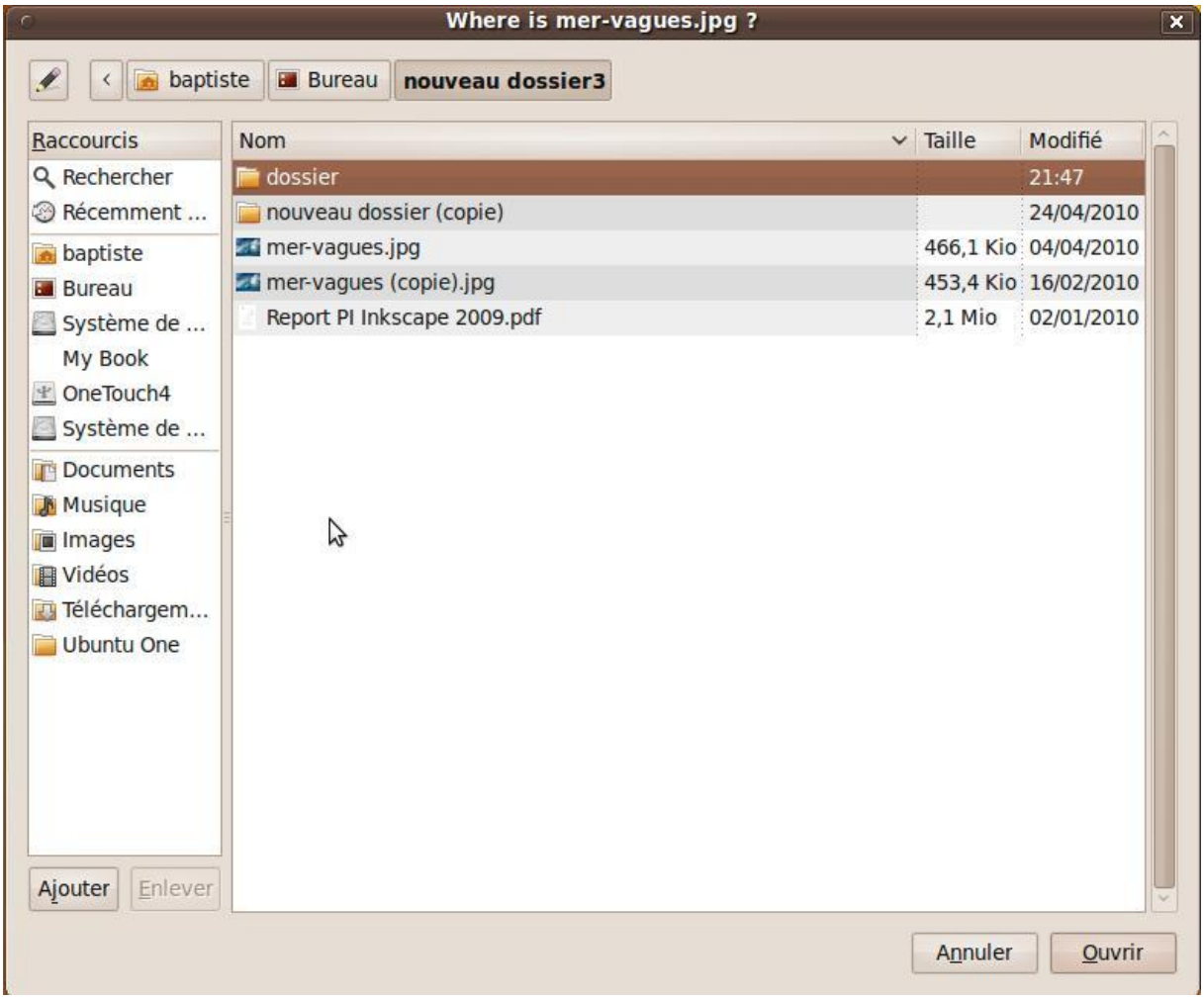

Figure 8: Browser window used to re-link an image

Once the user has selected the file, we simply set the path of the SPImage object to the new one. We then remove the image from the broken links matrix, and realize a display update. When this re-link is completed, we check whether there still are broken images. If so, the smart re-linking tool begins its work, as we will see in the next section.

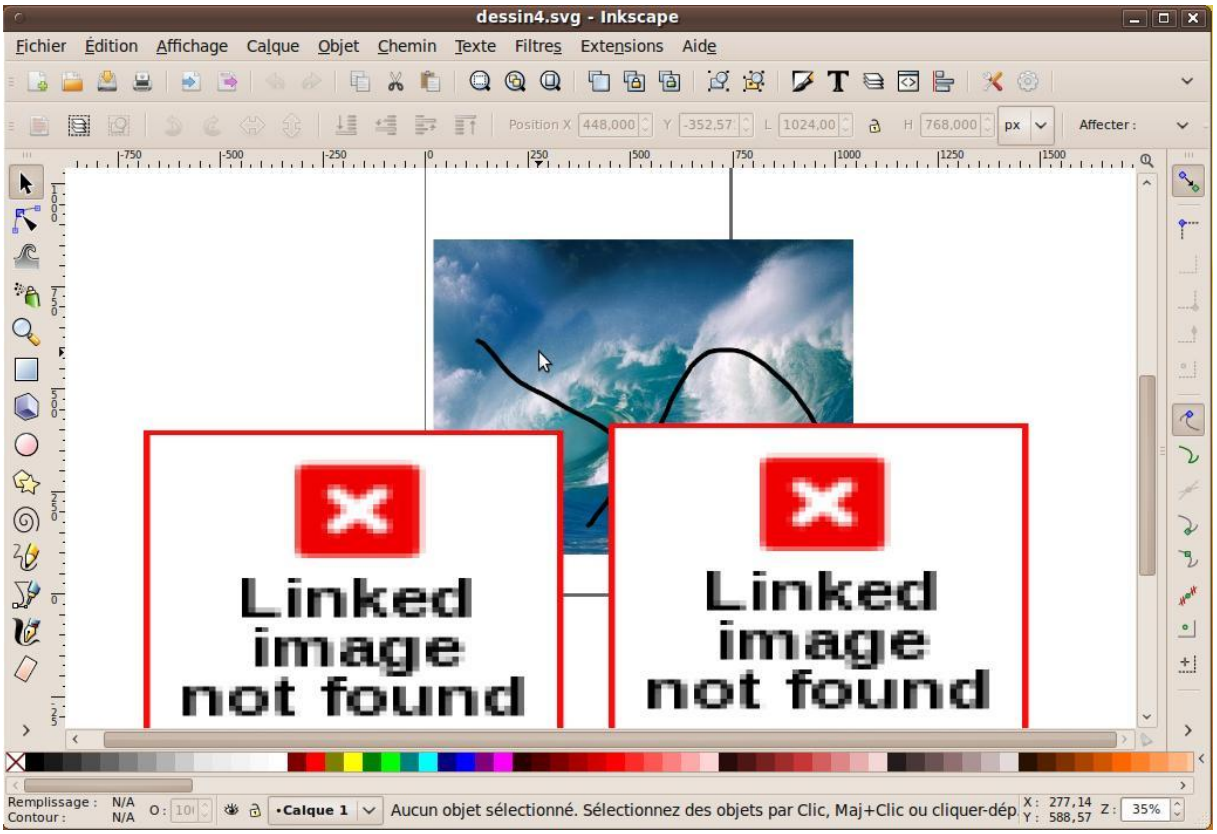

Figure 9: Document after the first image is re-linked

However, if for some reason the user decides to cancel the re-linking, he can still use the Image Proprieties dialog box to re-link his image(s), as we suggest in the exit window shown in Figure 10.

Figure 10: Exit window

III.4 Smart re-linking

The smart re-linking tool is the most interesting and important part of our work. It must be able to find the remaining broken images simply thanks to the path where the first re-linked image has been found. To understand how this tool works, let's consider a quick example, where there are four images to re-link.

C/File.jpg
C/File1.jpg
C/Directory1/File2.jpg
D/Directory1/File3.jpg

The image File.jpg has been re-linked to the path E/File.jpg. We then will start the search of the three other images in the directory E. Let's say this directory contains:

- a file File1.jpg,
- a folder Dir1, containing a file File2.jpg,
- a folder Directory1, containing two files, File2.jpg and File3.jpg.

The first image was initially at the path C and has been re-linked to E. So we take every broken image located in C or any child folder, we replace C by E in the full path of these images, and check if a file exists at this new path.

For example, File1.jpg was located in C, so we try to find it at E/File1.jpg. This file exists, so we re-link File1.jpg with it. We do the same thing for File2.jpg, which was located in a child folder of C. This file will then be re-linked to E/Directory1/File2.jpg.

However we cannot re-link File3.jpg because it was located in D/Directory1, despite the fact there is a file named File3.jpg in E/Directory1. In fact, there is no chance the smart re-linking tool could re-link this file from File.jpg as File3.jpg wasn't located in the same folder or a child folder.

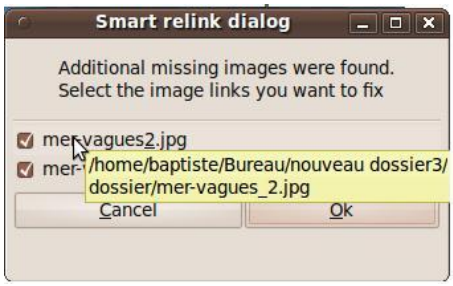

Figure 11: Smart re-link dialog

The smart re-linking is over. A window, like the one shown in Figure 11, appears, summarizing all the images we were able to found, and asks the user for a confirmation before proceeding to the re-linking.

After that, there is still one image which is not re-linked, so a new window will pop-up. If there were still more than one link broken, the pop-up window allows re-linking the first one in the list, and then a new smart re-linking occurs

for the remaining images and so on until every image is re-linked, or the user choose to cancel the re-linking.

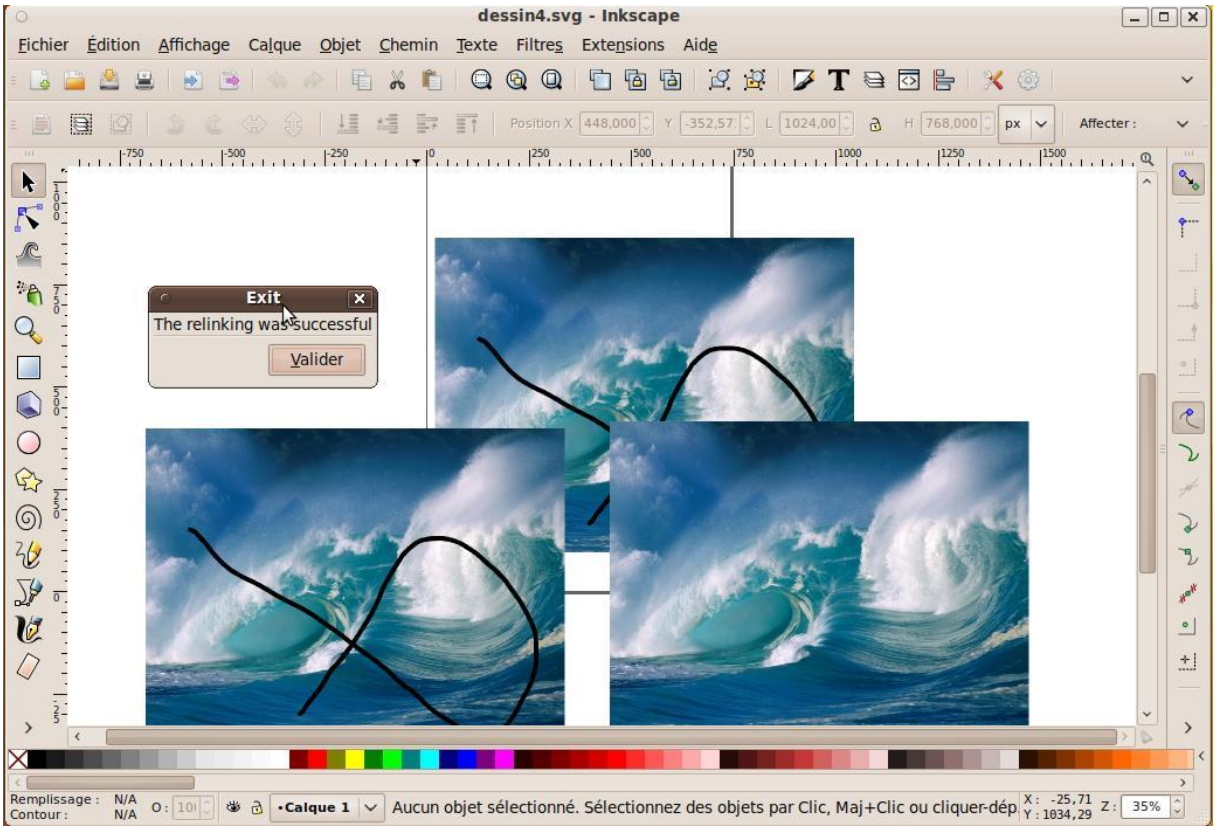

Figure 12: Successful re-linking

This program is very efficient in some case. For example, if all the images were in the same folder, and this folder has been renamed, the smart re-linking tool can easily re-link them. However if every image has been moved in a different folder, the smart re-linking tool won't give any result, and the user would have to re-link all the images one by one. A solution could be to make a complete depth search from the folder where were found the first image, but it would be a long process if for example we re-link from the root of the hard drive (every file in the computer should be examined).

IV. Image Properties Dialog Re-Design

This part of our project aimed at completely re-designing the Image Properties dialog. All the changes that were to be implemented are listed in I.3.3. In order to successfully re-design the dialog we extensively used the GTK language which is described in 0.3.

IV.1 Using Glade

At the beginning, we wanted to use a software named Glade in order to program the general structure of the new dialog more easily. It would have allowed us to put more time into the more difficult parts of this objective.

Glade is a tool which is very useful when it comes to creating GTK graphic interfaces. Indeed, it generates the code regarding the different windows and buttons automatically and thus allows the programmer to design new windows by dragging and clicking rather than typing lines of code. It saves the graphic interface in XML files and these files can then be used by several languages (such as C, C++, Java and PHP to name but a few) thanks to a library named "libglade".

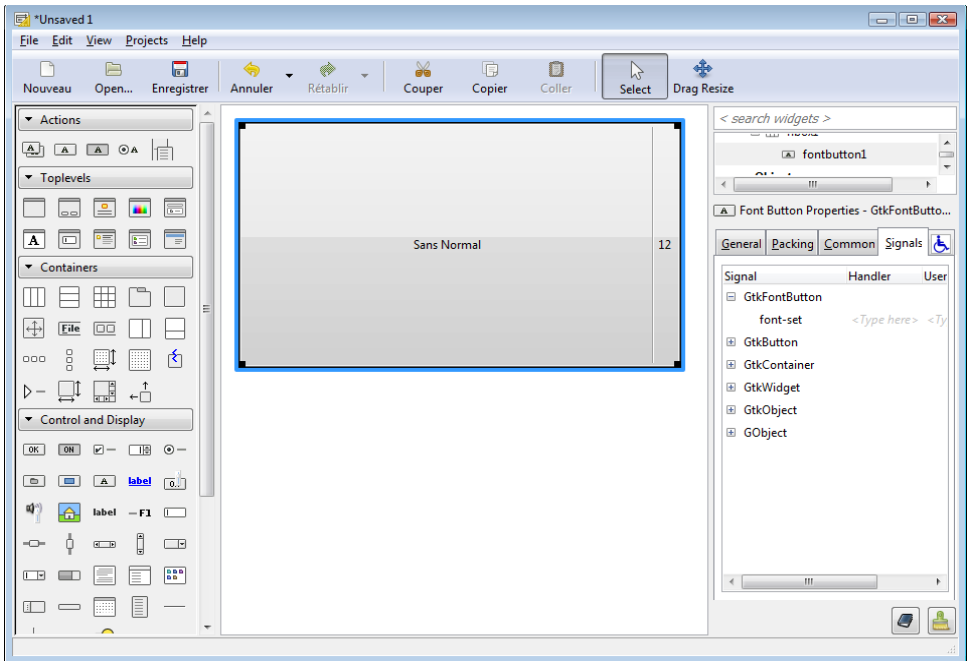

Figure 13: Glade Interface

Unfortunately, we realized after a while that programming the whole dialog using Glade would have two major consequences:

- each and every makefile would need to be modified in order to add instructions regarding "libglade",
- any person who would like to compile Inkscape on his computer would need to install the "libglade" library just because of this dialog that we added.

This led us to change our plans and to start programming in GTK without using any other library. This means that compiling Inkscape with our dialog implemented in it will not be any more complicated than compiling the current version of Inkscape.

IV.2 Programming the dialog

In order to realize the Image Properties dialog, we mostly modify sp-attribute-widget.cpp. This document can be found in the directory /src/widget and include some other files of interest such as sp-attribute-widget.h. All the functions and structures linked to sp-attribute-widget.cpp are defined in the .h. Among these structures, the most interesting is SPAttributeTable which is defined as written below:

```
/* SPAttributeTable */
struct SPAttributeTable{
   GtkVBox vbox;
   guint blocked : 1;
   guint hasobj : 1;
   GtkWidget *table;
   union {
      SPObject *object;
      Inkscape::XML::Node *repr;
          } src;
```

```
    gint num_attr;
    gchar **attributes;
    GtkWidget **entries;
    sigc::connection modified_connection;
    sigc::connection release_connection;
};
```

As we can see in this piece of code, SPAttributeTable is a structure (keyword "struct") containing a vertical GTK box (GtkVBox) and storing information and variables such as num_attr or attributes (which is here a 2D array of characters). But the variable that interests us the more is GtkWidget *table, as it is an array that stores all the attributes of an image. Also, we need to modify sp-attribute-widget.cpp in order to display the dialog shown on Figure 4. Modifications in this document start to appear at line 568 and some of them are listed below.

```
/*Create a table that will store the properties of the image (spat being a SPAttributeTable)*/
spat->table = gtk_table_new (num_attr, 2, FALSE);
/*[…]*/
for (i = 0; i < num_attr; i++) {
        GtkWidget *w;
        const gchar *val;
        spat->attributes[i] = g_strdup (attributes[i]);
        /* we create a label matching the name of the attribute (URL, Width, …)*/
        w = gtk_label_new (_(labels[i]));
        gtk_widget_show (w);
        gtk_misc_set_alignment (GTK_MISC (w), 1.0, 0.5);
        /*we store the label in the table*/
        gtk_table_attach ( GTK_TABLE (spat->table), w, 0, 1, i, i + 1)
                GTK_FILL,
                (GtkAttachOptions)(GTK_EXPAND | GTK_FILL),
                XPAD, YPAD );
/*[…]*/
/* we display the different frames containing the image properties*/
 gtk_widget_show (spat->table);
```

This loop allows us to run through the whole structure and to display each piece of information in the corresponding frame. Comparing the current dialog (Figure 3) and the dialog we are aiming at (Figure 4), we can see that we have to create four frames. In GTK, in order to create a frame, you use the function Gtk_frame_new(name of the frame). The first frame has no label and contains the following properties: name of the image file, location of the image on the hard drive, size of the file, dimensions of the image and resolution. The second frame is named "image source" and contains two buttons "Linked" and "Embedded" (one of which must always be selected), a "browse" button to run through the directories and an URL field to display the path to the image. The third frame is named "Dimensions" and should contain the current dimensions of the image (the first frame contains the natural dimensions of the image while with this one we can modify of the image appears in Inkscape). Finally, the last frame is labeled "Advanced options" should be a concealable list of every advanced options of the image. At the moment this report is written, the dialog is the following state:

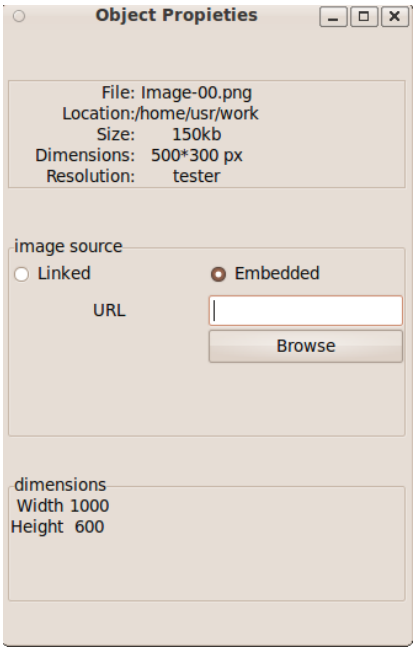

Figure 14: Our Image Properties dialog

It is still clearly a work in progress. Some things are yet to be corrected or added such as the title of the dialog, the layout of the elements, the display of the resolution or the tool to modify the dimensions. Nevertheless, we have good hopes that all these modifications will be made before the end of this project and that we will be able to present them during the oral examination.

Conclusion

The objectives of this project were:

- Generalize the use of relative links,
- Implement a smart re-linking tool,
- Re-design the Image Properties dialog.

As this report is written, we can say that the first and the second goals are reached. Indeed the problem we still encounter with relative links is due to a pre-existing bug in Inkscape that we are trying to solve even if it was not part of our explicit objectives. Regarding the third goal, we still have some work to do but have good hopes that it will be reached by the end of this Industrial Project (meaning May the 25th). Although our work may not be bug-proof, we have already corrected many of them.

As soon as the third goal is reached, we will be able to submit our work to the developers' team. If they found it worth of implementation into the next Inkscape version, we will have to fill in the Inkscape wiki that acts as a documentation reference.

The hope of seeing our work implemented into a future version of Inkscape and the feeling of helping users is what kept us motivated all along this project. But whatever the outcome, we are really proud of having tried to add our contribution to Inkscape.

List of figures

Appendix

0.1 Internal organization

At the beginning of this project and for a month or so, we have worked all together. During this period, we installed Ubuntu, downloaded the source code of Inkscape and tried to get a good grasp of what we were to do. It allowed each and every one of us to have a global vision of our project before focusing onto a specific part of it.

Then, as the project was pretty well divided into three parts, we decided to split the team into three groups. François and Rafik took care of the objective concerning the use of relative links. Baptiste and Vincent tried to implement the smart re-linking tool. Finally, Mor and Yoann were to address the re-designing of the Image Properties dialog.

0.2 Working on an open source software

During our project we have come to realize that Inkscape shows all the characteristics either good or bad that we were expecting in an open source software.

The main drawback of such a software is to be found in the source code. Indeed unlike company issued ones, open source software are often written by enthusiast programmers who find more interesting to add new features than to provide further programmers with understandable code. Also, it was often quite difficult to find the pieces of code we needed and we had to ask Mr. Giannini for help a few times. Having experienced the problems entailed by a lack of information about the code, we tried and commented the parts we added as much as possible.

But working on an open source software also has positive sides. For instance, we had the feeling of belonging to a community as a few articles on the Internet spoke about what we were trying to do. This feeling of doing something useful,

something users were interested in, helped us to cope with the discovery of the huge source code which was quite frightening at first glance. Another benefit of working on an open source software is the fact that we were not hindered with restrictions or confidentiality contracts as we would have been in an actual company. It allowed us to focus solely onto the improvement of Inkscape and thus to make the most of the time we had.

0.3 Using GTK

GTK is a language that was initially developed in order to program GIMP (GNU Image Manipulation Program) which is an open source images editor. Still, it is now used in several other projects and has been given birth to GTK+. GTK+ is a library which allows people to create graphic interfaces compatible with several environments (Windows, UNIX …). It can be used and modified freely and is available in a lot of languages such as C, C++, Ada, Perl, Php …

GTK is composed of two main kinds of elements: the widgets and the signals. Widgets (also known as window gadgets) are the objects at the base of graphic development in GTK+ (GtkWidget). Any graphic object inherits the attributes and methods of a widget. You can see below how it is possible to create a dialog with GTK+.

```
Gtkwidget* window; //creation of a widget
window=gtk_window_new(GTK_WINDOW_TOPLEVEL); //creation of a window
GtkWidget* bouton=gtk_button_new(); //creation of a button
GtkWidget* label)gtk_label_new("button"); //creation of a label
gtk_container_add(GTK_CONTAINER(button), label); //adding the label to the button
gtk_container_add(GTK_CONTAINER(window), button); //putting the button in the window
```

Any action performed by the user onto the graphic interface (clicking for instance) is gathered by a loop. The widget that is concerned by this action emits a signal (it can for example be "clicked" for a button) which will be connected to a function called callback that the programmer has to complete.

```
GtkWidget *button = gtk_button_new_with_label("button"); //Creation of a button
g_signal_connect(G_OBJECT(button),"clicked",G_CALLBACK(callback),NULL);
//connection of the signal “clicked” with the function “callback”
void * callback(void * arg){}  //definition of the function “callback”
```

0.4 Personal comments

Yoann Desgrange

I find that this project proved to be both very interesting and difficult. I had no real experience at programming beforehand and getting used to Ubuntu also took me some time. But now that this project comes to an end, I realize that I am now capable of finding most of the pieces of code I am looking for quite easily and even if I am still not confident when trying to change big chunks of code, I do not feel powerless anymore. I also realized how thinking things through is very important in programming. Indeed, the fact that Mor and I started a little too fast finally slowed us down a lot as we had to start again from scratch.

Moreover, the opportunity of writing a full report in English proved to be very interesting too. Indeed, I intend to work in an English-speaking country sooner or later and getting a first idea of what it is to sum up your work in another language was quite fulfilling.

Mor Ngom

This project allowed me to improve myself on certain points particularly programming under GTK. It is a project which also allowed me to see my limits on certain aspects of programming. Through this project, I namely realized that understanding a program from another person and modifying it, is not an easy thing. In addition to that, it also permitted me to see that the absence of methodology in the way of programming can lead to unexpected results. Indeed, Yoann and I started to develop the Image Properties window by using a software

(Glade) to go faster, but in the end it is the thing which slowed us down because we didn't enquire too much about this software.

All in all, this project proved to be rich in learning for me even if I am a little disappointed by my results.

Baptiste Soyer

What motivated me first in participating in this project was to work on a widely used tool like Inkscape. Our work could benefit a lot of people, unlike many others Industrial Projects.

I mainly worked on the re-linking tool. With Vincent we spent a lot of time trying to understand how Inkscape update its display. We went in many wrong directions as one of the main flaws in Inkscape is its lack of documentation. However we manage to understand this process. The second step was to find a way for the re-link windows to know which images where broken, and finally find a way to re-link them. It was quite difficult as one problem could mean that I had to rethink the whole architecture of the program I was trying to implement (and it happened many time). However after such an arduous process being finally able to see a window popping up when there is a broken link and enabling the user to re-link them is very rewarding.

This project was very interesting. I have learnt a lot concerning the structure of a real software, and the complexity behind what seems at first very simple.

Vincent Pais

I was really enthusiastic to participate in the development of a widely used software. With this project, I realize there is an immense gap between programming, as it is taught in Centrale Lyon, and what is really done in a big open-source project like Inkscape. It explains some of the difficulties we had in the first months when we tried to get familiar with a large, and often uncommented, code.

From a personal point of view my contribution hasn't been as important as I expected, because of the problems we had using Bazaar, but still I think I learnt a lot about software engineering from this project.

I am really hoping that our work will be integrated in a future version of Inkscape, as it would be a great accomplishment to see our work recognized and approved by the developer community. Consequently I wouldn't mind keeping working on the re-linking tool, as many enhancements can probably still be made, so that one day every user of Inkscape can benefit from it.

Rafik Boughida

I was interested in a computer-engineering Industrial Project because it corresponds to what I want to do after my studies in the Ecole Centrale de Lyon. As for this particular project, it was a very good experience for me because it was the first time I had been confronted to the source code of a professional-like software application with hundreds of files. It made me realize the importance of the documentation for such a collaborating project: it took us much more time understanding the existing code than writing our own lines.

I appreciated too the repartition in three groups of two persons: I realized that working with a partner on a code could be very efficient, much more than working alone or in a bigger group.

Moreover, it made me more familiar to the C++ programming language. I had already programmed in many different languages, and I had always thought it was a shame that I was not at ease with this one, which is probably the most used professionally.

François Petit

Personally, I took this project as an introduction into the world of software engineering as I want to work in that domain later on. My experience in this domain was very rewarding. As project leader, I could realize the importance of

making regular meetings with the rest of the team, in order to be able to synthesize the overall advancement of the project. I also learned that a project in software engineering is very slow to begin, and it was sometime worrying to think about the time we took in order to understand the code that was already written. But in the end, applying the modifications was really fast, so the largest amount of work was the understanding, not the writing (at least for the relative links handling part). The most rewarding aspect would be for our updates to be published in a “released” version of Inkscape, although it might take some time (on the basis of the previous Inkscape projects).

Summary

The Industrial Project n°27 aimed at improving the open source vector graphic editor Inkscape. A team of six students was given the responsibility of improving the way Inkscape dealt with images. This general objective was divided in three roughly independent parts: to make the software use more relative links, to implement a tool for smart re-linking and tore-design the existing Image Properties dialog.

Throughout this report, you will be able to grasp what are the differences between a vector graphic editor and a traditional graphic editor and why the changes made by the team will prove useful. Moreover, you will be able to see the work that has been done during the last five months. Finally, each member of the team will explain what this project brought to him and what he preferred in it.