Université Libre de Bruxelles
Faculté des Sciences Appliquées

Année académique
2002-2003

L'ingénierie collaborative : Application à la schématique électrique

Directeurs de mémoire :
Jean-Claude Maun
Esteban Zimanyi

Travail de Fin d'Etudes présenté par
Eric Delacroix en vue de l'obtention du
grade d'Ingénieur Civil Informaticien.

Remerciements

Je tiens à remercier chaleureusement toutes les personnes qui m'ont aidé dans mon travail :

Les Prof. Jean-Claude Maun et Esteban Zimanyi,
l'un pour la confiance qu'il m'a accordée en me confiant le travail, l'autre pour ses conseils et sa disponibilité tout au long du travail, ses idées et avis m'ont souvent éclairé le chemin

MM Jacques et Jean Goldstein,
pour leur suivi attentif et leurs conseils avisés sur le développement du projet, et pour le contact personnel par lequel ils m'ont mis en confiance pour ce travail

Mr Jean-Michel Dricot,
pour son écoute des moindres problèmes et ce à tout moment et pour son enthousiasme communicatif sur le projet

MM Olivier De Wilde et Alexandre Chau,
sans qui ce travail n'aurait jamais eu la même présentation

Mlle Gaëlle Lambin,
pour tout le temps qu'elle a accordé à la relecture de ce document, et le coeur avec lequel elle l'a fait.

Tous les gens qui ont contribués en petite main

Mais aussi sur un plan plus personnel :

Mes parents,
qui m'ont un peu poussé pour être là aujourd'hui, ou bien qui n'ont fait que m'éclairer...

Le 8 décembre 1978,
jour dans lequel j'ai trouvé le moteur pour décoller ainsi qu'une série de fondamentaux nécéssaires à mon équilibre

Max,
avec toi la traversée de ces études a su être animée et enrichissante, ton soutien et ta présence auront été indispensables.

Table des matières

Recherche de solutions

Chapitre 1

Analyse de la situation

Sommaire

1.1 Introduction

1.1.1 Contexte du travail

Dans le monde industriel d'aujourd'hui, les nouvelles technologies de communication incitent à développer des outils qui améliorent la productivité. Dans cette optique, un courant s'est répandu ces dernières années : l'ingénierie collaborative. Ce principe veut, au départ, que des projets puissent être menés en parallèle par différentes équipes. La nouveauté pour ce mode de travail est que ces équipes ne doivent plus se situer sur un même site.

Actuellement, beaucoup de société investissant dans des développements de projets, de quelque nature qu'ils soient, font travailler leurs équipes sur des logiciels leur permettant cette approche. Prenons un exemple : le développement de programmes informatiques. Dans ce cadre une application appellée CVS (Concurrent Versions System) est déjà largement utilisées. Il s'agit d'un programme dont la première fonction est de stocker toutes les classes qui composent un projet. Les différents participants au projet peuvent accéder aux différentes classes, et les rappatrier chez eux. Sur leur station, ils peuvent alors modifier à souhait le code, et tester ce qu'ils ont fait. Une fois satisfaits de leur travail, ils renvoient ce fichier à l'application qui va, avant d'officialiser les changements, tester la cohérence des classes que le développeur a mis à jour avec le reste du projet. En plus d'une application de stockage, c'est aussi une sorte de compilateur.

De cette manière, à partir du moment où les développeurs se sont donnés une ligne à suivre, l'emphase mise sur les incohérences possibles entre classes signalées directement suffit déjà à réduire considérablement les temps morts provoqués par d'éventuels oublis ou inconsistances.

Ce principe peut alors s'appliquer à tout développement de projet pouvant impliquer différentes équipes. Le développement de projets électrotechniques fait indiscutablement partie de cette catégorie. En effet, ceux-ci peuvent être composés de sous-projets, traités par des personnes ou des équipes différentes, éventuellement situés dans des lieux géographiques différents. Par exemple, la conception de la partie électrique d'un véhicule fait appel à divers bureaux d'études et sous-traitants. Un tel mode de développement pourrait donc tout à fait s'avérer utile. C'est ici le but de la démarche.

1.1.2 Problématique

L'internationalisation croissante des sociétés implique une dispersion possible des différentes compétences nécessaires au développement d'un projet. La réunion de ces talents, précédemment, demandait soit beaucoup de communications par fax et téléphone, soit le déplacement d'équipes entières, avec les conséquences humaines que cela implique.

Dès lors, la perspective de fournir un outil qui permette la gestion dans une optique collaborative pour un programme de shématique électrique peut se présenter comme une réelle opportunité commerciale. La demande est présente, et l'utilité, même pour un outil plus basique que celui cité ci-dessus, est réelle. A partir de là, il s'agissait donc de cerner quelles étaient les premières fonctionnalités qui auraient une utilité pour l'allègement du procédé de développement d'un plan électrique.

Ces fonctionnalités doivent en effet s'inscrire dans les règles du processus industriel classique. Il y a donc plusieurs étapes intermédiaires à franchir avant de parvenir à une application aussi complexe que CVS, qui chacune déjà apporte ses différentes utilités. Dans un premier temps, le but sera de réduire le temps perdu dans les différentes communications qui doivent se faire entre des équipes pour l'avancement d'un projet.

1.1.3 Contributions et cheminement

Le sujet était proposé en collaboration avec Elsys([15]), une société belge commercialisant un logiciel de shématique électrique. C'est donc sur base de leur outil, Electre NT, que le développement d'une telle application a été envisagé. Il a fallu imaginer les techniques et technologies à utiliser. Dans cette perspective, différentes possibilités ont été explorées, dont les plus importantes sont expliquées dans ce travail.

La solution choisie devait alors s'inscrire dans une perspective de développement possible, pour permettre d'envisager, à long terme, l'implémentation d'une solution type CVS. Et c'est dans ce but que le choix du format graphique, certainement un des choix clé du travail, a été posé. Le choix du langage a alors suivi ce choix du format, lui aussi

s'inscrivant dans une vague qui a encore une bonne marge de progression.

C'est donc en partant de l'observation du marché actuel de la CAO électrique (au salon CAD CAM de Courtrai par exemple) que les différentes philosophies ont été examinées. Ceci, combiné à l'état des technologies actuelles, a mené à développer un outil fonctionnel que nous allons disséquer. Son but est de fournir une base de développement pour différentes fonctions ultérieures dont nous décrirons également les possibilités.

Dans le premier chapitre, nous exposerons le contexte de départ du projet. Ensuite nous chercherons à concilier la vue d'Elsys avec les avancées qui ont déjà eu lieu dans le domaine. Dans un troisième chapitre, nous tenterons de poser les premières spécifications dans lesquelles les choix seront justifiés, pour en déduire, dans le chapitre suivant, un design possible qui a été mis en oeuvre. L'avant dernier chapitre traitera des différentes techniques particulières qui ont été utilisées pour l'implémentation de la solution. Enfin, nous conclurons en mettant en évidence les bases posées, et en montrant qu'elles permettent une évolution vers une application plus complexe.

1.2 Le logiciel Electre NT

Electre NT est un logiciel de Conception Assistée par Ordinateur (CAO) destiné aux applications de schématique électrotechnique et de câblage.

Au départ, Jacques et Jean Goldstein, les créateurs d'Electre NT, ont développé et commercialisé un logiciel de CAO électrique en partenariat avec Hewlett Packard. Celui-ci tournait sous la plate forme UNIX. C'est en 1996, lorsque Hewlett Packard se retire du marché des logiciels de CAO que les frères Goldstein décident de poursuivre le développement du programme, mais cette fois utilisable sous Windows. Electre NT est sorti dans le courant de l'année 1998.

Ce qui fait sa particularité est sans aucun doute son mode de développement. En effet, il est conçu pour rencontrer les besoins des concepteurs. Les clients d'Elsys ont continuellement à disposition une hot line pour les éventuels problèmes rencontrés, mais aussi la possibilité d'exprimer des besoins supplémentaires relatifs au logiciel. C'est principalement sur base de ces besoins que les fonctionnalités complémentaires sont développées. Ainsi, en plus des outils de dessin traditionnels, Electre offre des générateurs de sous produits que sont les informations structurées que l'on peut retirer d'un plan (bilan matériel, liste de connexions). Et pour répondre à des besoins de clients, un module de harnais (technique de câblage dans l'industrie du transport) a été développé et est maintenant diffusé avec un succès croissant dans les industries spatiale, aéronautique et automobile.

D'un point de vue graphique, Electre NT permettait de générer les folios en différents formats. Mais pour les formats exportables et visionnables par n'importe quelle application, toutes les informations importantes étaient perdues. A côté des différents formats graphiques, plusieurs autres fichiers peuvent être générés. Parmi ceux ci, une description mathématique des éléments du plan dans un fichier texte, avec une extension ".g". Nous reparlerons de son utilisation plus loin.

1.3 Définition de l'ingénerie collaborative

Tout projet industriel passe par plusieurs étapes de développement. Souvent, ces projets sont ralentis par les transitions entre ces étapes. Pour gagner du temps, plusieurs stades de développement peuvent être menés en parallèle, ou encore plusieurs personnes peuvent travailler au développement d'une même partie. Mais ces personnes ne travaillent pas forcément au même endroit. Afin d'avancer de manière cohérente dans le projet, chaque acteur se doit de savoir précisément tout ce qui a été réalisé dernièrement, mais aussi pouvoir observer toute l'évolution du travail, de manière à ne pas tourner en rond.

Le point auquel veut répondre le concept d'ingénierie collaborative est précisément la communication de ces informations, de la manière la plus rapide et le plus efficace possible. Le premier outil dans ce cadre est donc un interface accessible par tous les participants à un projet. Vu la disparité géographique potentielle de ceux ci, un moyen évident de réunir tout le monde est l'utilisation de réseaux, qu'ils soient internes ou externes. Précedemment, cette communication devait passer par des lignes téléphoniques et des fax, avec tous les inconvéniens matériels que cela engendre.

Pour une bonne utilisation de ces réseaux, les moyens nécessaires à mettre en œuvre sont une application à placer sur ce réseau. Celle-ci doit permette au minimum la consultation de l'évolution du travail, ainsi que la communication d'information pour garantir la cohérence des avancées. A partir de là, chaque participant au projet aura la possibilité de consulter tout ce qui a déjà été réalisé et, de lui même, déposer ses réalisations. Les systèmes les plus évolués permettront de synchroniser automatiquement les versions des fichiers concernés par le travail en cours, et même de soulever les incohérences éventuelles d'une version à l'autre. Mais pour cela, il faut le soutien d'une base de données qui permette de comparer les fichiers. Dans le même temps un programme peut servir à interpréter ces fichiers, établir les liens avec les folios concernés et vérifier que cela correspond.

1.4 Approche d'Elsys de l'ingénierie collaborative

Dans le cadre du programme Electre NT, les projets sont donc de l'ordre de plans de câblage. On imagine facilement les applications du collaboratif dans le cadre de tels projets. En effet, il est fréquent que la forme des câblages évolue en fonction des contraintes matérielles de l'endroit où ils sont placés. Si le concepteur change ces contraintes, cela peut modifier des longueurs, voire des types de câbles. C'est le genre d'information qui émerge à un endroit où l'on s'occupe de design matériel et qu'il faut pouvoir communiquer rapidement au concepteur du câblage. Si le designer a un accès direct aux fichiers sur lesquels travaille l'utilisateur d'Electre, il peut voir si le changement est réaliste, ou du moins signaler en temps réel le changement de contraintes et, de cette manière, éviter la perte de temps dans la communication de l'information.

Une autre situation a amené Elsys à conclure à la nécessité d'une application permettant une approche collaborative : la simple consultation de plans. En effet, il a été remarqué que la plupart des concessionnaires d'une marque automobile n'avaient parfois les plans mis a jour des produits dont ils s'occupent que 18 mois après les changements. Soit, souvent assez longtemps après que de nouvelles modifications aient été enregistrées. La possibilité de consulter à distance les plans appropriés à chaque type de produit et à chaque période de validité serait un gain à plusieurs points de vue en plus du fait de pouvoir être à jour. Cela épargne la production des livres de mise à jour ainsi que leur envoi.

Ce dont Elsys a donc besoin, c'est un module de consultation de plans, pour permettre à leurs clients une publication sur un interface accessible sur internet. De plus ce module devrait pouvoir permettre de fournir certaines informations matérielles sur les plans consultés, sans pour autant fournir les outils d'Electre. Enfin, un utilisateur de ce module devrait pouvoir signaler des zones de problèmes ou de modifications nécessaires sans pour autant modifier le fichier lui-même, afin juste d'envoyer ce fichier au concepteur de schéma.

Dans cette optique, on peut se demander pourquoi ne pas permettre à l'utilisateur de carrément suggérer une modification dans le dessin même, et soumettre le changement au designer. Il faut rappeler à ce moment que, dans le processus industriel, il est impensable qu'une modification d'un plan vienne de l'extérieur sans être passée par plusieurs étapes de confirmation et ce afin que la modification éventuelle soit prise en compte à tous les niveaux. Ce genre d'application serait alors plus de l'ordre d'une application pour une équipe de développement, strictement interne à la société productrice des plans.

Pour Elsys, tout ceci doit être développé en restant un maximum indépendant des technologies utilisées. En effet, il faudrait pouvoir diffuser le module de collaboration sans imposer de condition supplémentaire au client.

Chapitre 2

Analyse des possibilités

Sommaire

2.1 Etat de l'art

Nous nous situons ici dans le domaine de la CAO électrotechnique. Le marché est occupé par plusieurs grandes sociétés. Parmi elles, les plus connues dans le milieu, Auto-Desk, Eplan, IGE-XAO ou Cim-Team. En regardant de plus près ce que propose chacune, on se rend compte que le point clef est sans doute la publication des productions sur internet.

Cependant, dans le domaine de l'ingéniérie collaborative, d'un point de vue software, on observe deux tendances. Tout d'abord, on trouve sur le marché des programmes dont le but est de gérer, avec une orientation collaborative, les différents fichiers générés à lors de conception d'un projet. Ces programmes sont en quelque sorte des boîtes creuses que l'on place par dessus des applications. Bien sûr, pour chaque programme de ce type, il existe une liste exhaustive des applications prises en charge par celui-ci. L'approche de l'ingénierie collaborative par ce biais est donc assez rigide, mais est une solution efficace pour des bureaux utilisant des outils moins spécialisés. Il est évident que ce type de gestionnaire de projet n'est développé que pour des applications très répandues. Cela représente donc ce que j'appellerai "la couche supérieure des applications à dessein collaboratif".

Dans un deuxième temps, on trouve les applications plus spécialisées, qui ont aussi un besoin d'offrir des solutions vers le collaboratif. C'est là qu'on retrouve les solutions proposées par les compagnies précitées. Chez chacune, on retrouvait (en octobre 2002) l'intention d'offrir un outil qui ouvre la porte des réseaux. Chez l'un on peut trouver un viewer, une application web qui permet de consulter à distance les productions dans un interface très sobre. Chez l'autre, il ne s'agissait encore que d'exportation de fichiers

sous plusieurs formats différents, ce qui les rend lisibles par plusieurs applications. En fin de compte, la solution la plus évoluée se trouvait chez Eplan, qui proposait une application web permettant non seulement de consulter des plans, mais aussi de signaler une éventuelle erreur par le placement d'une croix distinctive sur le plan; la communication de la remarque était effectuée par mail.

Ces applications sont donc spécifiques au programme auquel elles sont rattachées. Leur mode de fonctionnement, pour l'affichage des graphiques, semblait reposer sur le noyau même du programme de base. En effet chaque application génère son propre format graphique, lisible par son moteur d'affichage. Il n'y a donc a priori pas tranformation du fichier avant sa publication.

2.2 Problématique dans le cadre d'Elsys

Comme dit plus haut, Elsys se veut offrir des solutions aux problèmes rencontrés. Ces solutions sont bien évidemment vendues, ce qui impose quelques contraintes pour le développement d'une application allant dans ce sens. Tout d'abord, il est nécessaire que l'application soit utilisable dans un maximum de configurations possibles, et donc un maximum libérée des contraintes éventuellement imposées par les technologies utilisées. Ce qui signifie en fait que l'application ne doit pas nécessiter d'achat particulier pour le client. Ensuite, les technologies utilisées ne doivent pas non plus se trouver sous licence. Enfin, le tout doit offrir une possibilité d'utilisation intuitive, dans la mesure où cette application ne se destine plus uniquement aux spécialistes de la CAO mais aussi à tout utilisateur d'internet qui voudrait juste pouvoir lire un de ces plans.

La spécialisation d'Electre ainsi que ces dernières contraintes l'inscrivent clairement dans la deuxième tendance, mentionnée ci-dessus. Cependant, au début du travail, Electre ne possédait que des fonctions d'export de plans sous d'autres formats, dont celui utilisé chez AutoDesk. La publication sur un réseau ne pouvait donc se faire, a priori, que soit via le noyau d'Electre lui-même, soit sur bases de formats non propriétaires déjà exportés, c'est à dire de bitmaps.

Une autre solution, plus flexible mais plus importante en quantité de travail, était de suivre une tendance progressiste et exporter un nouveau format graphique. Pour que ce changement ait du sens, il était nécessaire d'envisager un format qui possède des propriétés différentes que celles des bitmaps. Or, dans le domaine graphique, un phénomène se répand : le dessin vectoriel. Il existe quelques standards de dessins vectoriels, le plus connu actuellement pour son utilisation sur Internet étant Flash. Un autre format,basé sur le langage XML (eXtensible Markup Language), gagne en popularité([5]) ces derniers temps : le SVG (Scalable Vector Graphics) qui offre des possibilités sérieusement étendues par rapport aux images bitmaps. Nous détaillerons cela un peu plus loin. Ce format commence à être envisagé, et parfois même implémenté, également chez les concurrents.

Toutefois, le Jpeg offre un format assez léger pour une image, mais entraîne une perte de précision quand il s'agit de pratiquer des zooms ceci étant dû à l'effet de pixélisation.

D'un autre côté, dans les formats connus, on en trouve un qui, lui, ne pose pas ce problème de pixélisation : le format PDF. Cependant s'il permet des zooms sans perte de qualité, il ne permet aucun affichage intelligent ni d'identification par élément. Ces deux formats auraient donc pu être utilisés, par facilité, mais ne se seraient pas inscris dans une perspective de progrès.

Pour savoir dans quelle voie aller, il a fallu analyser les différents fonctionnements d'Electre. En effet, le programme génère plusieurs fichiers qui sont exploités chacun différemment. Dans le cas présent, ce qui nous préoccupait était la description graphique. Plusieurs fichiers sont générés dans ce cadre : un fichier binaire que le noyau graphique d'Electre utilise pour produire un rendu des plans dans l'espace d'utilisation du programme, mais aussi un fichier en format texte, donnant une description mathématique de l'image. Cette description mathématique est justement un des points clés du format SVG. Il était donc envisageable que cette description soit utilisée également dans le cadre du SVG, ne fût-ce que pour produire le fichier.

Il existe également des fichiers qui permettent de passer les plans vers des imprimantes utilisant aussi bien les protocoles (formats) Postscript ou hpgl. Voici un tableau des formats pris en charge par Electre NT.

interface	in	out	standard	remarque
DXF	√	√	√	Attributs, symboles
DWF	√	√		Attributs, symboles
AMF		√	√	Lien Excel, Word
PDF		√	√	
TIF		√	√	
JPG		√	√	
BMP	√	√	√	

interface	in	out	remarque
HPGL	√		
IGES	√		
ME10	√	√	
MIF		√	Framemaker
Euclide schématique	√		Irisbus
Saviplan	√		Irisbus, Renault Trucks
DSI	√		Faisceaux

TAB. 2.1 – *Formats pris en charge par Electre NT*

2.3 Comparaison SVG-Bitmap

La différence fondamentale entre ces deux formats se situe au niveau du contenant de l'information. Dans le cas d'un SVG, il s'agit d'une description de l'image à l'aide de fonctions mathématiques suffisamment complexes pour permettre la description d'images détaillées. Ces images sont décomposées en leurs différentes parties, chacune descriptible par une fonction particulière, et ces fonctions sont reprises dans un fichier, type xml, dont l'agencement des balises donnera la configuration du dessin.

Les fonctions, ou plutôt les balises, permettant de tracer ces dessins sont répertoriées dans la recommandation du W3C([3]). On y retrouve les formes de base (polygones, cercles) mais aussi des possibilités de tracer des courbes plus complexes en les décrivant, dans leur balise, par les points par lesquels on veut que passe une courbe (polyline)([6]). Il est même possible d'utiliser des courbes de Bézier, en les décrivant par leur code spécifique. Illustration simple:

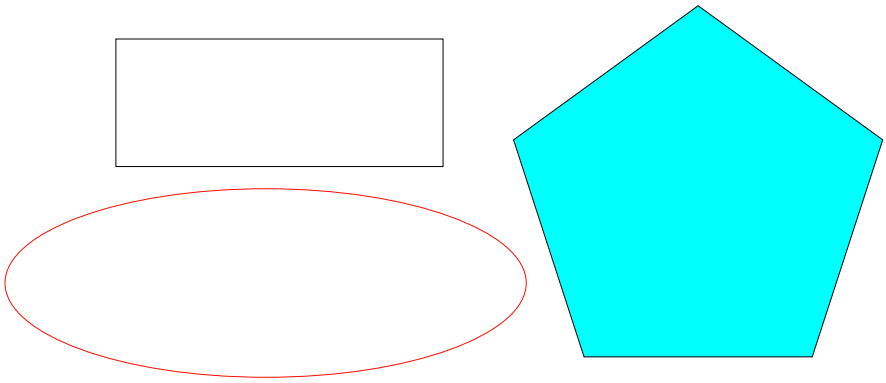


Fig. 2.1 – *Quelques formes simples*

Ces formes sont rendues sur base du code suivant :

<svg xmlns“http://www.w3.org/2000/svg” xmlns:odm=“http://product.corel.com/CGS/11/cddns/”
xml:space=“preserve” >
<g id=“Layer 1”>
<ellipse class=“fil0 str0” cx=“2042” cy=“1627” rx=“833” ry=“301”/>
<path class=“fil1 str1” d=“M3425 740l295 215 295 214 -113 347 -112 347 -365 0 -365 0
-113 -347 -112 -347 295 -214 295 -215z”/>
<rect class=“fil2 str1” x=“1563” y=“847” width=“1046” height=“408”/>
</g>
</svg>

Intuitivement, on remarque déjà la possibilité de structurer en couches les différentes formes du dessin, de manière à jouer avec les avant et arrière plans. Cette propriété sera un avantage dans le cadre d'un développement plus poussé, nous en reparlerons plus loin.

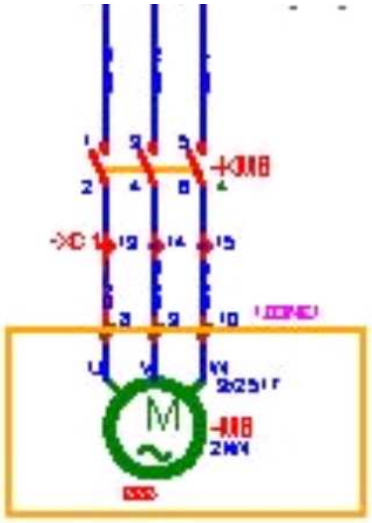


Fig. 2.2 – *Zoom sur bitmap*

Dans le second cas, la description est fournie par un mapping de pixels dont on définit la couleur. Ce mapping donne un rendu de l'image, mais est totalement statique. Les images codées en 24 bits sont décrites avec trois octets par pixel, les images codées en 8 bits le sont avec un. La seule modification que l'on puisse apporter à la source est donc de l'ordre de la modification des propriétés d'un pixel, ce qui n'apporte pas beaucoup de flexibilité.

De cette description physique des formats, on peut déduire que l'image bitmap est limitée dans sa qualité d'affichage par la granularité des pixels de l'écran. D'où l'effet de pixelisation en cas de recadrage de l'image. A l'inverse, l'on constate intuitivement qu'un recadrage dans le cas de l'image vectorisée va simplement induire une transformation affine du dessin. Une fois les proportions calculées, l'image peut être réaffichée à la taille désirée sans avoir été limitée par la définition de l'écran ; l'apparence du dessin restera la même([16]).

De plus, intrinsèquement, on retrouve dans l'image vectorisée une décomposition de l'image en ses différentes parties. Donc, en utilisant cette qualité, on pourrait donner des propriétés particulières aux différents éléments d'un dessin. Dans le cadre de l'application, le rendu de l'intelligence du plan peut exploiter ces possibilités,allant de la définition de couleur jusqu'à l'utilisation du protocole xlink, en passant par l'identification individuelle de chaque élément du plan pour une recherche nominale de ceux-ci. Ceci serait totalement impossible avec un bitmap. En effet, à moins d'utiliser des algorithmes de découpage d'image selon les contours, ce qui ne peut pas être très efficace, il n'est pas possible de donner une propriété particulière à un élément du dessin, et encore moins d'identifier des composants. Tout au plus peut-on donner des propriétés à une zone géométrique du dessin, définie à l'aide des coordonnée. Et même à partir de là, on ne pourrait pas modifier l'une ou l'autre propriété de l'élément du dessin.

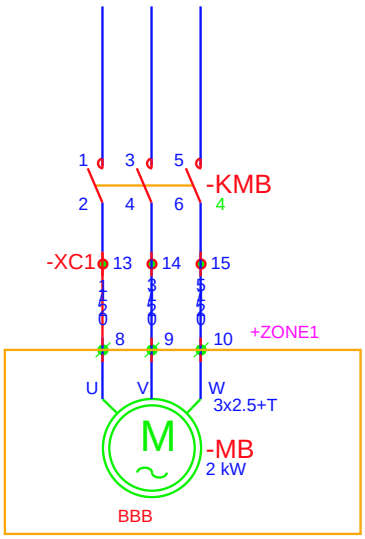


Fig. 2.3 – *Zoom sur image vectorielle*

En effet, le SVG étant issu de la mouvance XML, il a été conçu avec le DOM (Document Object Model). Ce dernier permet l'animation et le traitement des images à l'aide d'un langage qui permettrait d'implémenter les interfaces de cet API. Cependant, s'il existe des langages supportant le traitement du SVG, la médaille à son revers en ceci que la visualisation de telles images dans un browser nécessite également (jusqu'ici) un environnement ou un plug-in particulier. Adobe a été le premier à sortir un plug-in permettant de visionner un SVG qui aurait été inséré dans une page html. Celui-ci permet déjà d'exploiter la fonction la plus représentative du SVG, le zoom. L'insertion de svg dans une page html combinée à l'exploitation la plus répandue du DOM, le script, permet déjà de manipuler des graphiques à l'intérieur d'un browser. Mais cela ne peut suffire au développement d'une application telle qu'elle a été envisagée. Nous en reparlerons plus loin.

En définitive, on peut dire que le dessin vectoriel, s'il n'en est pas réellement à ses débuts, est en tous cas plus neuf que les bitmaps, et a, sans aucun doute, une plus grande marge de progression([14]) et donc un avenir plus certain. Le seul avantage des bitmaps face au SVG est leur facilité d'utilisation et leur côté déjà très standard (à venir pour le SVG).

Choix et implémentation d'une solution

Chapitre 3

Spécifications et orientation

Sommaire

3.1 Déploiement général de l'application

Pour concevoir le déploiement d'une telle application, il a fallu imaginer le fonctionnement global d'une entreprise développant des plans électriques et voulant les publier sur internet. Un certain nombre d'hypothèses peuvent être formulées :

- Les concepteurs de plan travaillent sur des stations reliées à un réseau interne.
- Ce réseau interne possède une station connectée à Internet
- L'entreprise héberge son site chez un ISP (Internet Service Provider) et ce serveur fournit des services tels que FTP (file transfer protocol), SMTP (simple mail transfert protocol)
- On peut éventuellement trouver une base de données reliée à la station "gateway"

Ces hypothèses ne sont pas restrictives dans la mesure où, si l'entreprise héberge elle-même son site, elle est à même de se fournir les protocoles indispensables. Alors que si elle est hébergée chez un ISP, les services en question sont des plus standards. On peut donc supposer que c'est dans ces conditions que l'application sera installée.

Dans le cadre de ce travail, le déploiement dont il est question permet d'envisager les fonctions qui sont nécessaires au niveau du serveur web pour implémenter une application permettant de visualiser, voire manipuler, des plans électriques. Cette application doit être développée comme un noyau potentiel d'une application de plus grande

envergure qui couvrirait chacune des étapes du processus.

En effet, dans une vision collaborative de l'ensemble, on peut penser qu'il est possible

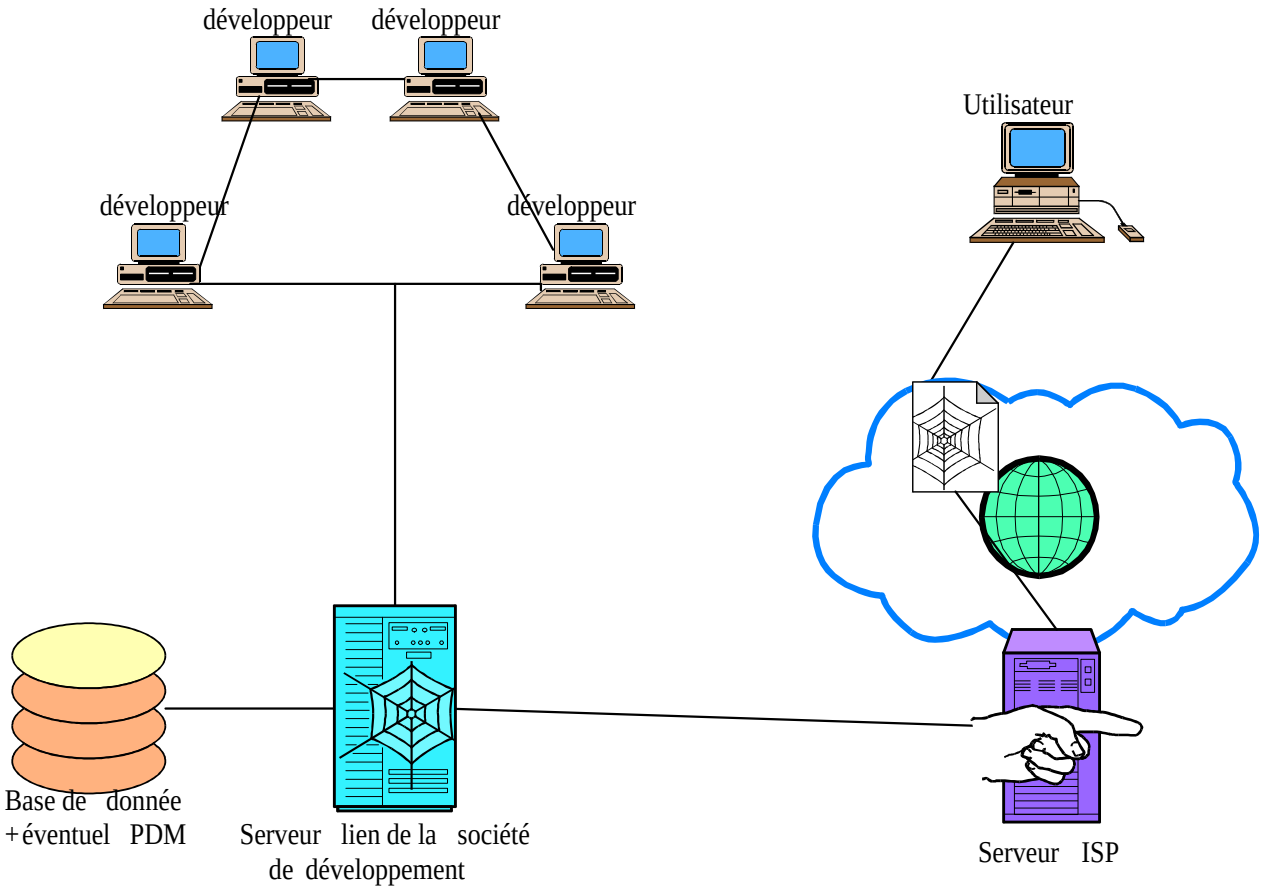
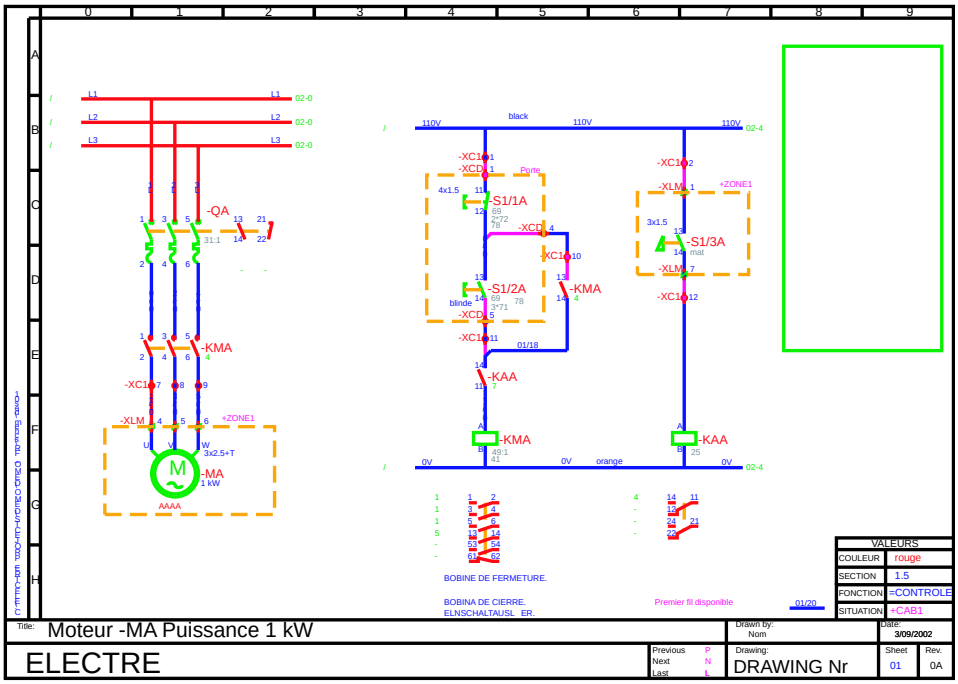


Fig. 3.1 – *Schéma de déploiement*

d'automatiser tout le processus, depuis l'export au format graphique voulu, jusqu'à son accessibilité sur Internet, en passant par la gestion des versions de fichiers à travers le programme concerné ainsi que la communication de remarques par un autre biais que le mail. Mais ceci dépasse le cadre de ce travail, pour lequel nous nous attarderons sur la partie application disponible sur internet.

3.2 Choix du format graphique

Nous avons expliqué qu'Electre renferme une bibliothèque d'éléments très vaste. La diversité de ces éléments doit pouvoir être prise en compte. Surtout si l'on veut pouvoir attribuer à certians d'entre eux des fonctions particulières. La distinction des composantes d'un dessin inhérente au SVG se présentait dès lors comme un énorme avantage. De plus, un plan n'a pas toujours la même dimension : le standard est le format A3, mais il est très régulier de rencontrer des formats bien plus larges. Or l'espace dans lequel il est visualisé, lui, reste fixe. Le détail apparent n'est donc pas le même au départ, d'où la



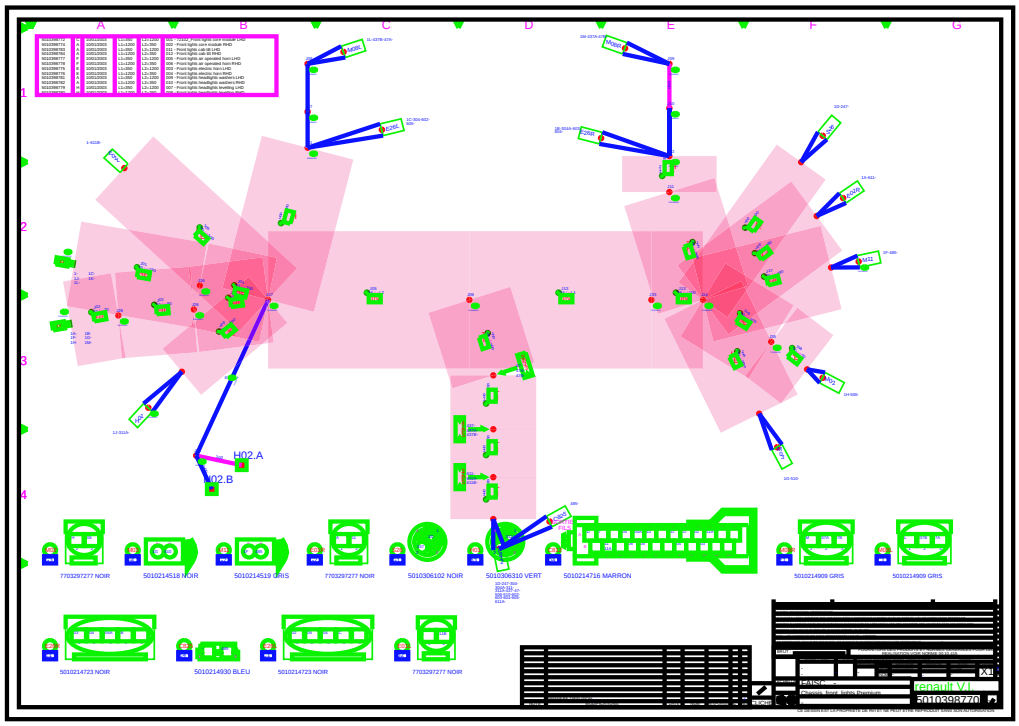


Fig. 3.2 – *affichage d'un plan A3(au dessus) par rapport à un plan A1(en dessous) sur un même espace*

nécessité d'employer un zoom lorsqu'on visualise un plan, tout en gardant le détail des éléments.

Il s'agit donc d'une raison supplémentaire pour préferrer du dessin vectoriel. Le seul inconvénient du SVG réside dans le fait qu'il n'est pas un format déjà produit par Electre, contrairement aux formats bitmap. Cela nécessitait donc l'implémentation d'un convertisseur au format SVG. Cette partie a été prise en charge par Jacques Goldstein qui est parvenu à faire le parallèle entre la description fonctionnelle des graphiques qui existait déjà dans Electre et la norme SVG.

Enfin, le côté modularisable des graphiques SVG ainsi que son API DOM inscrivent ce format dans une perspective de développement à long terme. Non seulement les possibilités graphiques devraient progresser, mais son lien avec des langages de programmation orientés objet laisse imaginer une extension des moyens de traitement du dessin lui-même.

On pourrait encore préciser que cette décision était pressée aussi par le choix, déjà fait, de la concurrence.

3.3 Choix du langage

Plusieurs facteurs sont intervenus dans ce choix. Dans les contraintes de départ, nous avons parlé de l'indépendance de l'application par rapport aux technologies internet. Autrement dit, il fallait parvenir à développer une application capable de tourner à partir de n'importe quel serveur, dans un langage qui permette l'utilisation d'un service de mail ainsi que l'accès au DOM pour le traitement de l'image vectorielle.

Quelques langages ont implémenté des fonctions permettant d'accéder au DOM. On les retrouve en C++, et en Delphi. Des langages de scripting sont aussi utilisés, tels que Perl ou Python, ou même JavaScript([12]). Il existe même un standard, développé par le W3C, appelé le ECMAScript, dont l'intention est de standardiser le scripting, afin de niveler les différences de fonctionnement entre JavaScript et Jscript (la version de Microsoft). Ce langage de script est développé essentiellement pour le traitement des documents XML ou XSL. Certaines applications ont déjà été développées par ces moyens, essentiellement dans le domaine de la CAO d'ailleurs. Le problème de ces langages est qu'ils sont directement interprétés, donc non compilés. Or rappelons qu'il s'agit d'une application qui doit pouvoir être vendue et utilisée sur Internet. Il est donc impensable de laisser le code à la disposition de tout utilisateur. La nécessité d'avoir un code compilé écartait dès lors cette possibilité.

Le dernier grand langage implémentant les fonctions du DOM est Java([9]), et c'est celui là qui a été choisi. Les possibilités qu'offre Java en matière d'applications web sont nombreuses et variées. Dans la mesure où le collaboratif est très vite synonyme de publication sur les réseaux, mieux vaut disposer d'un outil puissant afind'en exploiter les différentes possibilités. Java semble répondre à cette attente dès lors qu'il offre des modules de communication à distance, RMI ou CORBA, des modules pour application serveur, JSP et servlets, et un module d'application en browser, les applets.

Un dernier avantage de Java, et pas des moindres, se trouve dans le fait que c'est le langage par excellence pour les applications qui doivent fonctionner sur n'importe quelle platte forme. La librairie correspondant au traitement d'images vectorielles s'appelle Batik. Elle implémente les interfaces du DOM, et d'un autre côté les fonctions nécessaire au traitement de cet arbre DOM, en proposant une image en objet Java permettant l'exploitation des données pour permettre la visualisation d'un fichier.

Soulignons aussi que le choix, bien que resté objectif, a été facilité par le fait que Java, outre ses nombreux atouts techniques, est un langage qui a beaucoup été utilisé au cours de mes études, et dont j'ai, par conséquent, déjà une bonne pratique de l'analyse des codes. Cette analyse était nécéssaire pour comprendre le mode d'utilisation de Batik, vu le manque de documentation à ce sujet. En effet, cette librairie étant très récente et encore en développement (malgré des versions stables déjà publiées), on ne trouve sur Internet que peu de sites expliquant l'utilisation de ses fonctions. Quant au site de développement d'apache([1]), on n'y retrouve que quelques indications et exemples, mais qui ne s'appliquent qu'à des contextes particuliers, et qui incitent plutôt à l'usage des outils déjà développés en license GPL (ce qui n'est pas utilisable ici). Le support principal pour la recherche à travers cette bibliothèque a donc été la mailing list dédiée à Batik([7] et [8]), et l'outil principal, lui, le code source ainsi que la JavaDoc([2] et [4]) qui lui est attachée.

3.4 Description de l'architecture Batik

La librairie Batik est développée elle même en plusieurs couches, représentées par différents niveaux de modules. En fonction des modules utilisés on personnalise plus ou moins les différentes fonctionnalités implémentées. En effet, les module supérieurs sont en eux-même les applications graphiques permettant la manipulation de documents SVG, mais ce sont les applications sous licence dont nous avons parlé plus haut.

Ces applications se servent d'une série de classes de deux niveaux inférieurs, celui du milieu offrant des composants déjà agrémentés de fonctions, le niveau du bas renfermant les classes de base pour l'analyse des fichiers et l'affichage des graphiques. Dans le cadre de ce travail, nous avons travaillé avec ces deux couches : d'un côté en reprennant certaines propriétés déjà implémentées en les modifiant pour les faire correspondre au besoin, et de l'autre en s'inspirant de la structure des classes existantes pour créer de nouvelles fonctionnalités.

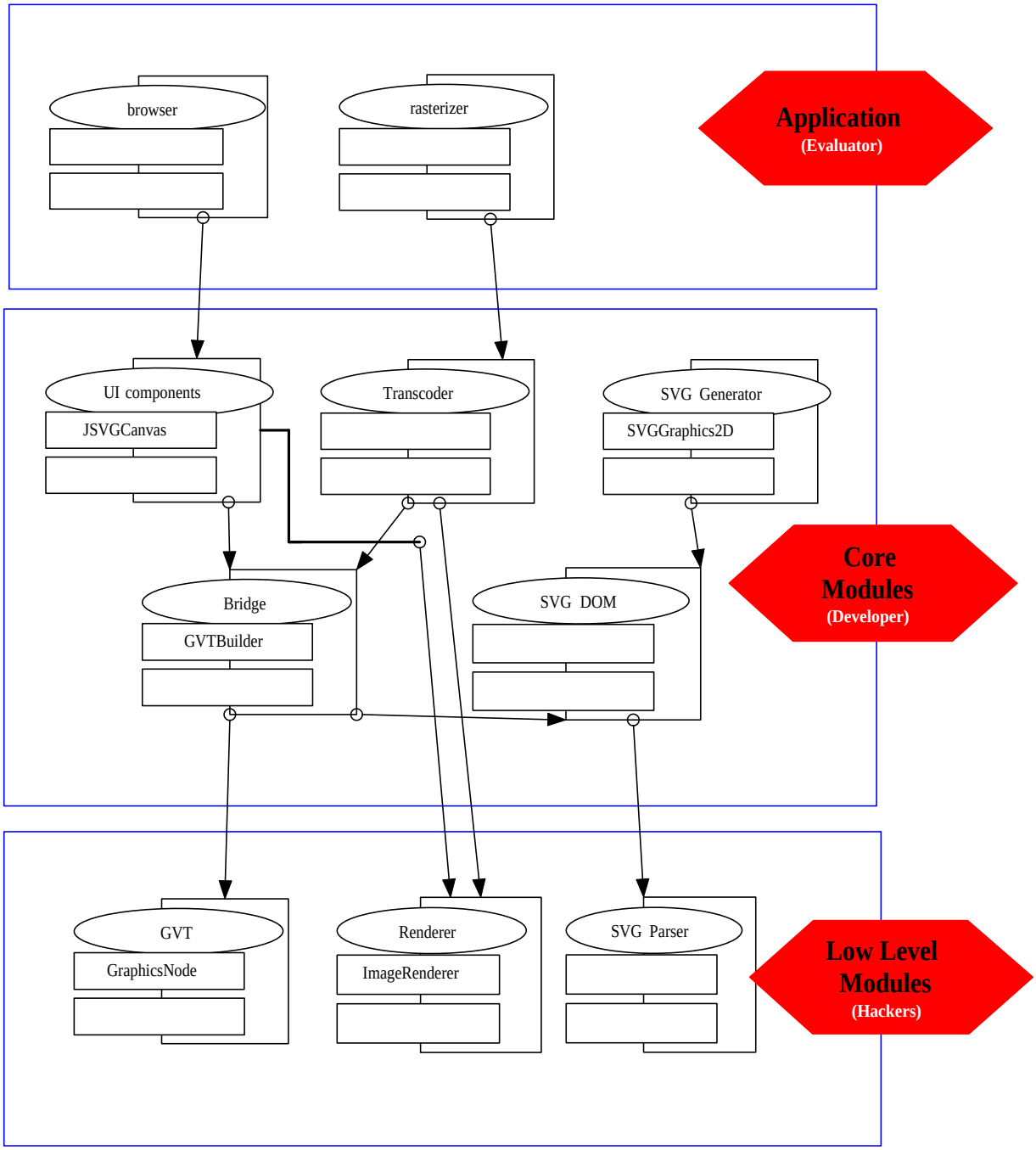


Fig. 3.3 – *Structure de la librairie Batik*

3.5 Le fonctionnement des applets

Les applets sont des programmes pouvant s'exécuter à l'intérieur d'un browser. Ces programmes se doivent d'être sûrs, dans le mesure ou ils sont destinés à être utilisés par beaucoup de personnes différentes. Ce pourquoi le déroulement est continuellement surveillé par un security Manager([11]).

Les restrictions les plus importantes que l'on peut observer pour les applets sont :

- L'applet ne peut pas lire sur le disque local : en effet, on ne veut pas prendre le risque que l'applet puisse accéder à des données privées et les publier sur internet, ou les lancer en circulation. Toutefois, pour certaines applications cela peut se révéler nécessaire. Mais il est possible de contourner ces difficultés. Le moyen pour permettre un comportement moins restrictif est la signature digitale de l'applet offerte par Java. Ceci permet à l'utilisateur de constater l'origine de l'applet, et d'accepter son exécution ou non.

- L'applet doit être entièrement téléchargée avant de se mettre en marche. Dans certains cas, cela peut être long. Pour éviter ce désagrément, rien n'empèche de rassembler les fichiers compilés dans un fichier .jar (Java ARchive). Cela réduit le chargement à un seul accès serveur. Même avec ce procédé, il est toujours possible de signer l'applet.

Toutefois, les applets possèdent un certain nombre d'avantages qui les rends très attractives, d'autant plus que Java offre des possibilités pour combler efficacement les manques de ce type d'application.

Tout d'abord, alors que dans les systèmes client/serveur traditionnels, créer et installer une nouvelle version du logiciel client est souvent un cauchemar, ici cela ne pose pas de problème d'installation. Une applet est réellement indépendante de la plateforme (y compris pour jouer des fichiers audio, etc.) et donc il n'est pas nécessaire de modifier le code en fonction des plateformes utilisées ni d'effectuer des "réglages" à l'installation. En fait, cette dernière est automatique chaque fois qu'un utilisateur charge la page Web qui contient les applets, de sorte que les mises à jour se passent en silence et automatiquement.

De plus, on ne doit pas craindre un code défectueux qui endommagerait le système de l'utilisateur, eu égard aux restrictions citées plus haut. Cela, avec l'avantage cité ci-dessus, fait de l'applet un moyen populaire pour l'installation d'applications sur des réseaux intra ou extra-net.

Pour faire fonctionner une telle application, on introduit une balise HTML dont le nom est simplement : <applet>. Dans un premier temps, les navigateurs Netscape et Internet Explorer ont rapidement intégré les fonctionnalités nécessaires au support des applets. Mais avec la complexification des langages (dont Java) et le rapprochement des philosophies entre JavaSoft et Microsoft du point de vue de la programmation orientée objet (avec ActiveX et C♯), ce fait n'était plus si évident. Pour éviter que certains browsers ne supportent plus le fonctionnement d'une applet, JavaSoft a créé un plug in, qui est contenu dans la JRE (Java Runtime Environment), qui est en réalité la machine virtuelle dans laquelle s'exécute le programme. Cela a aussi complexifié la syntaxe de la page HTML même. En effet, pour renseigner les versions de Java et du plug-in utilisé, et ceci aussi en tenant compte du type de browser, on atteint tout de suite un nombre

impressionnant de paramètres à gérer.

Pour ne pas se perdre dans la confection du fichier HTML, un outil est mis à disposition des développeurs, appelé HTMLConverter. Pour générer le fichier html voulu, on passe au programme la page html de base, ne contenant que les paramètres renseignant le code et les librairies. Un fois passé par le programme, le fichier contient du script permettant l'identification du browser dans lequel la page a été chargée et ainsi, la prise en compte des paramètres sous la forme demandée par ce browser. De cette manière, l'applet parvient à se charger et fonctionner dans le carde de tous les browsers.

A partir de là, la seule contrainte restante afin de pouvoir utiliser une applet est d'avoir ce JRE installé. Il est téléchargeable gratuitement sur le site de Sun, les développeurs de Java. De plus, cet environnement n'est pas à usage unique, il permet donc d'utiliser n'importe quelle applet sur Internet.

3.6 Utilisation de RMI comme canal de communication

Vu les restrictions imposées par le fonctionnement de l'applet, certaines fonctionnalités recherchées pouvaient poser problème. En effet, une nécessité était de pouvoir importer tous les noms des dossiers des projets ainsi que, pour chaque projet, le nom des fichiers qui le composent. Or, s'il est effectivement possible d'explorer des dossiers avec des fonctions Java, il n'est pourtant pas possible de le faire pour les fichiers du serveur depuis l'applet. L'applet peut bien lire un fichier sur son serveur, par l'intermédiaire d'une fonction prenant un objet URL en paramètre, mais pas en reconstituer l'arborescence.

La solution alors envisagée a été de mettre en place un canal de communication entre l'applet et un petit serveur tournant sur l'hôte de l'applet, à l'aide de RMI([13]). Ce petit serveur se charge alors, comme une application tournant localement, de produire le listing des fichiers ou dossiers demandés([10]). Le fonctionnement de RMI permet alors d'appeler depuis l'applet, qui possède une classe implémentant la recherche du serveur, cette fonction de listing qui, à son tour, renvoie en valeur de retour un vecteur de String contenant l'information.

C'est alors que s'ouvre un vaste champ de possibilités. En effet, l'implémentation de ce canal est pratique pour le problème cité, mais on peut même pousser plus loin : le canal établi peut en plus être sécurisé (il s'agit là d'un travail conséquent mais possible). Ce médium pourrait donc également être utilisé pour le développement plus large de l'application : une automatisation complète du procédé de placement des fichiers sur le serveur, ainsi que le renvoi de propositions de modifications, et de cette manière permettre le suivi de l'utilisation et des propositions de modification des différents fichiers.

En outre, c'est ce canal qui est utilisé dans le cadre de la solution implémentée jusqu'ici pour l'utilisation du service mail, ainsi que pour l'import des noms des fichiers

disponibles sur le serveur, pour des raison expliquées plus loin.

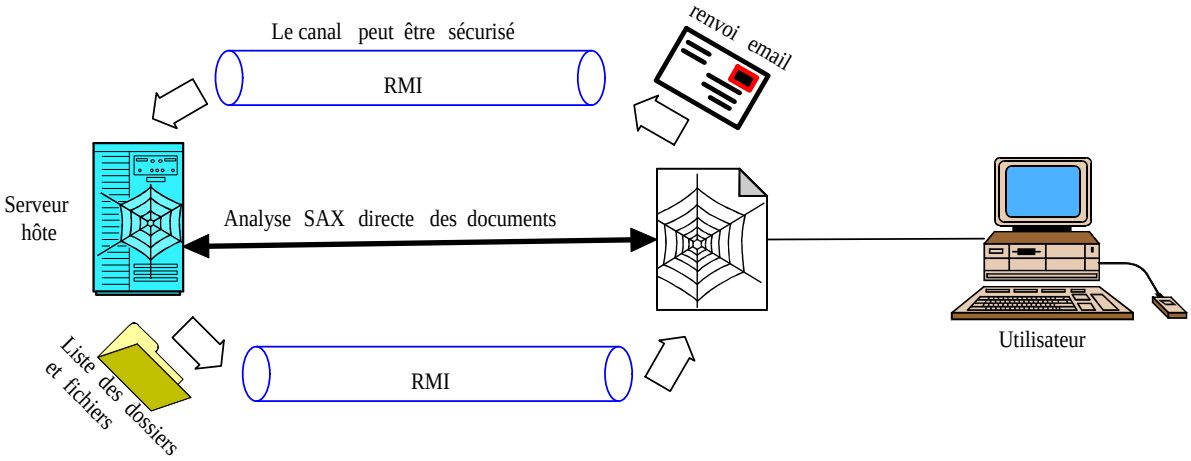


Fig. 3.4 – *Fonctionnement de l'applet avec RMI*

3.7 Possibilités offertes par Batik du point de vue collaboratif

Batik offre un composant permettant d'afficher un SVG. Il s'agit d'un canvas particulier appelé JSVGCanvas.

Précisons tout d'abord le fonctionnement des éléments graphique de Java. Plusieurs objets du langage permettent l'affichage de parties graphiques. Pour expliquer d'où vient celui qui nous intéresse, partons des composants AWT (Abstract Window Toolkit) (3.7).

Dans tous ces éléments graphiques, le seul destiné à offrir un espace personnalisable est le Canvas. Cette classe offre un espace vierge dont on peut fixer la dimension ainsi que des fonctions de dessin basiques, et implémente la méthode paint() pour le rendu des dessins. Cela permet de modifier les graphiques affichés à l'intérieur. Une liste encore un peu plus étendue vient de la bibliothèque Swing de Java, qui avait pour but de remplacer AWT, en permettant une utilisation plus simple. Il s'agissait donc de retrouver ce canvas dans Swing([18]), ce qui n'est pas le cas. Cependant, en repartant de la classe JComponent de swing, on peut facilement recréer un espace ayant les propriétés d'un Canvas, mais du côté de swing. La classe JSVGCanvas étend donc ce JComponent au travers de deux autres classes JGVTComponent et JSVGComponent.

GVT est une boite à outil, Graphic Vector Toolkit, qui permet de faire correspondre le DOM à l'arbre d'objets Java dont on a parlé précedemment, tout en donnant une bien

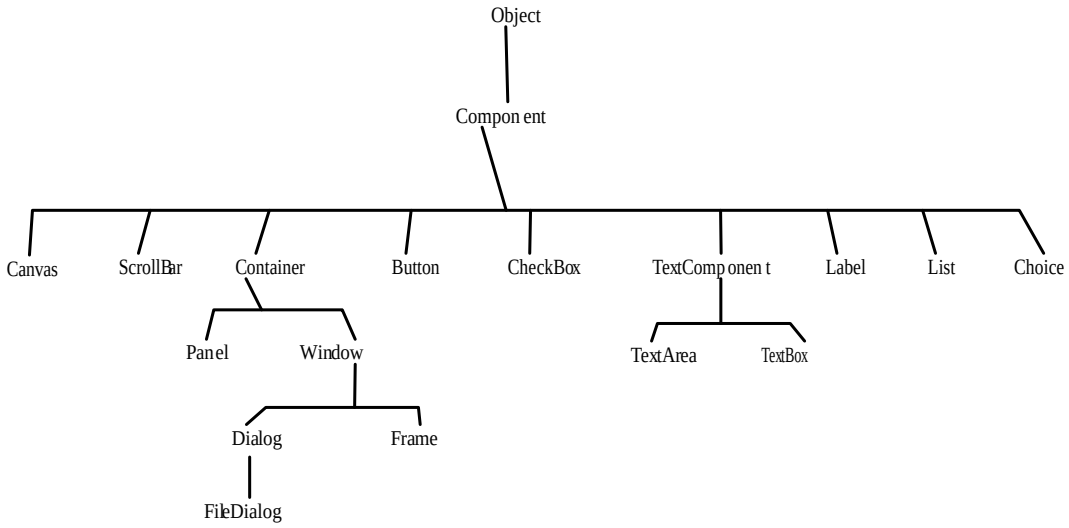


Fig. 3.5 – *Liste des éléments AWT*

plus grande facilité pour l'implémentation du traitement d'événements et pour la gestion du rendu à l'écran. L'élément graphique qui lui correspond, le JGVTComponent, est l'ingrédient de base pour pouvoir afficher un SVG. Il prodigue les fonctions nécessaires à la construction de l'arbre d'objets Java, et celles nécessaires à leur rendu. Il permet donc de générer un objet Image.

La classe JSVGComponent étend le JGVTComponent. Elle permet de gérer l'importation et le traitement des éléments du dessin d'un niveau supérieur. Ainsi on rend plus transparentes pour l'utilisateur les différentes étapes. La fonction la plus représentative de cette classe est la fonction : loadSVGDocument(URL). En effet, à partir de celle-ci, on va pouvoir suivre les 5 étapes de la création du rendu :

- Construction de l'arbre du DOM
- Construction de l'arbre GVT
- Initiation des gestionnaires d'évenement et des scripts éventuellement associés
- Rendu du GVT
- Amorce des parties dynamiques

Enfin, le JSVGCanvas étend le JSVGComponent. Il n'est que la conformation du JSVGComponent aux spécifications des JavaBeans. De plus, il implémente déjà ce que l'on appelle les "Interactors", fonctions dynamiques, et les active par défaut. Ces "Interactors" sont des "super-listeners". Ils implémentent les "event","key", "mouse" et "mousemotion listeners", permettant d'interagir au moyen de la souris et du clavier avec la zone qui affiche le dessin. Il peuvent être activés et désactivés par une fonction du canvas; dans notre cas nous donnerons ce rôle à l'utilisateur. Ces Interactors ont une structure particulière, en plusieurs classes qui s'étendent de l'une à l'autre en implémentant à chaque niveau des fonctionnalités supplémentaires, qui permettent de préciser l'usage. On peut agir à plusieurs de ces niveaux pour modifier l'interaction voulue. Nous en donnerons une description plus précise dans le chapitre sur les techniques

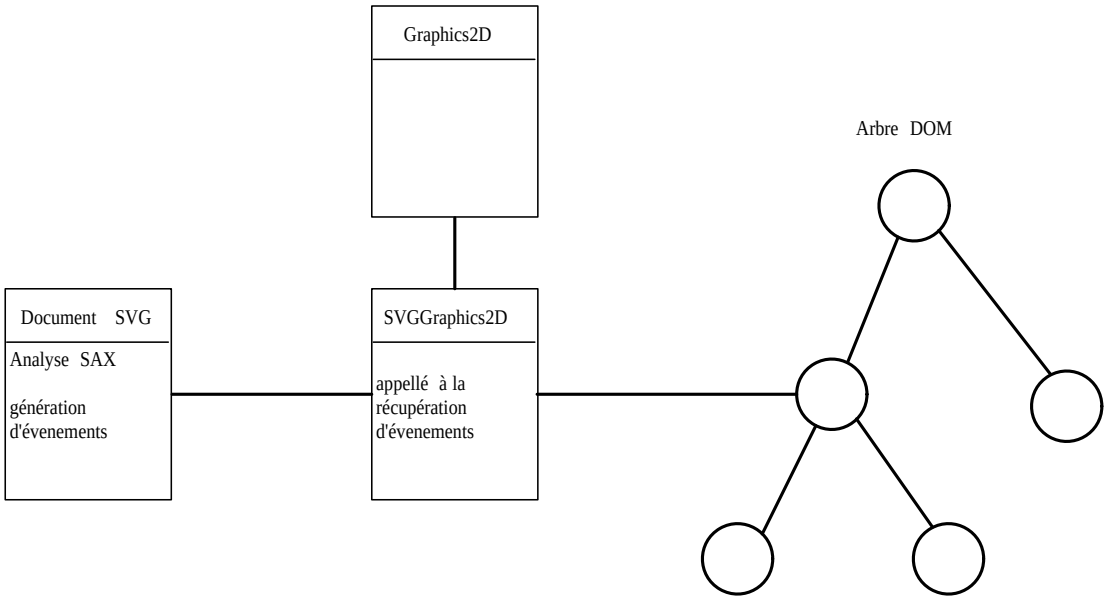


Fig. 3.6 – *Construction de l'arbre DOM*

particulières d'implémentation.

L'accès aux dimensions et proportions définies pour un composant se fait via un UserAgent. Celui ci gère les propriétés additionnelles de l'espace d'affichage du graphique. C'est donc via cet objet que nous pourrons gérer les interactions entre l'utilisateur et les propriétés internes du dessin : gestion des ouvertures de liens ou appréhension des éléments survolés par la souris. Cet objet est déterminé au moment du chargement du fichier svg. Il en existe un, implémenté par défaut, qui gère les interactions standards. Il est également possible de l'étendre, afin de rajouter des comportements de saisie d'évenement sur des éléments.

Ce que l'on constate au vu de l'agencement de ces quelques classes, c'est que l'on est parfaitement équipés pour implémenter un solution ayant les propriétés collaboratives désirées. Tous ces objets offrent la possibilité d'implémenter un espace qui permet :

- de visualiser un fichier
- d'utiliser certaines des propriétés d'un plan qu'on aura laissé accessible
- d'en modifier l'affichage pour l'étudier en profondeur
- d'y rajouter des éléments sans altérer le fichier de base

Tout cela, auquel on rajoute un service de mail, permet bien d'établir une communication centrée sur une production de plans, pouvant contenir des critiques ou des informations sur la pertinence du développement. L'utilisateur aurait la possibilité de manipuler à souhait le plan qu'il consulte pour en retirer autant d'informations qu'il en a besoin.

Chapitre 4

Conception de l'application

Sommaire

4.1 Présentation de l'application

La première contrainte à prendre en compte était le fonctionnement de l'application sur un accès plus ou moins public. Par "plus ou moins public", j'entend que le programme peut-être mis sur un serveur et utilisé sur Internet, mais aussi simplement installé sur un pc et utilisé par son propriétaire, par l'intermédiaire d'un browser tout de même.

Sa fonction est avant tout de permettre la consultation de plans. Elle ne s'adresse donc pas uniquement à des spécialistes de la CAO, et encore moins à des spécialistes de l'informatique. Il a donc fallu développer une application qui soit la plus intuitive possible à son utilisation. L'idéal était donc d'implémenter un GUI (Graphical User Interface), fonctionnant essentiellement en presse bouton.

Le code de l'applet est basé sur le server qui la publie. Or plusieurs paramètres de l'application doivent être précisés en fonction de la configuration de celui-ci. En effet, pour implémenter la solution d'une communication RMI entre l'applet et le serveur, ainsi que la solution de service mail, il faut utiliser les noms de serveur locaux. Un petit programme d'installation est donc nécessaire pour déterminer les constantes locales. C'est

ainsi que nous déterminerons le nom du répertoire racine où se trouveront les différents dossiers concernant les différents projets, mais aussi l'adresse email par défaut qui sera utilisée pour l'envoi des messages.

A partir de là, une fois l'applet installée sur le serveur, elle sera accessible à toute personne autorisée sur cette page, et aura un fonctionnement tout a fait transparent pour l'utilisateur. A cette fin, les fonctions sont bien en vue, les différents projets sont disponibles dans un espace bien visible de l'applet, et l'utilisateur pourra donc importer tous les folios d'un projet pour ensuite les visualiser dans l'ordre qu'il veut, en les sélectionnant dans une liste située juste en dessous de celle des projets, par un simple clic sur un bouton.

On peut donc constater que le nombre de composants apparents a été réduit au maximum, afin d'éviter toute ambiguïté d'usage. Le service mail, lui, apparaît en pop up, de manière à ne pas encombrer l'espace et à pouvoir proposer un agencement standard des champs à remplir.

En définitive, l'application se veut "user-friendly" mais efficace en même temps, et ne demandant surtout pas trop d'interventions supplémentaires de la part d'un gestionnaire de serveur, du point de vue des pages HTML.

4.2 L'affichage d'un plan

D'un point de vue plus matériel, les dimensions des plans et la dimension de la zone d'affichage, dont on a déjà parlé, sont des paramètres qu'il a fallu concilier. En effet, non seulement la taille originale des plans peut varier, alors que la taille de l'espace d'affichage est fixée, mais en plus, la définition de l'écran de l'utilisateur influence la taille de l'affichage de l'applet elle-même au sein du browser. Il était donc nécessaire de trouver un bon compromis entre l'espace potentiel fourni par un browser (la taille de l'écran qui, elle même, peut évidemment être variable), la définition de l'écran de l'utilisateur et l'espace nécessaire pour afficher un plan de n'importe quelle taille, de manière à garder une bonne vue globale du dessin.

Le plan affiché dans son espace peut être traité avec plusieurs outils. En cas de zoom, il est possible de ne plus trop savoir où l'on se situe sur celui ci. Dès lors l'application telle qu'elle a été implémentée propose un "overview" un espace d'affichage supplémentaire, affichant le même schéma que son grand frère, mais beaucoup plus petit donnant une idée de la forme et des couleurs présentes sur le dessin. Cet espace offre la possibilité de faire le lien entre les deux affichages pour implémenter carrément une coloration, sur l'overview, de la partie visualisée sur le canvas principal. Ce lien n'a pas encore été implémentée faute de temps, mais son fonctionnement devrait rejoindre celui des interactors qui seront expliqués plus tard. Il s'agira en fait de parvenir à intercepter l'évènement d'un changement de cadrage, récupérer les paramètres de la transformation du premier canvas et en déduire les coordonnées de la surface affichée sur un plan général.

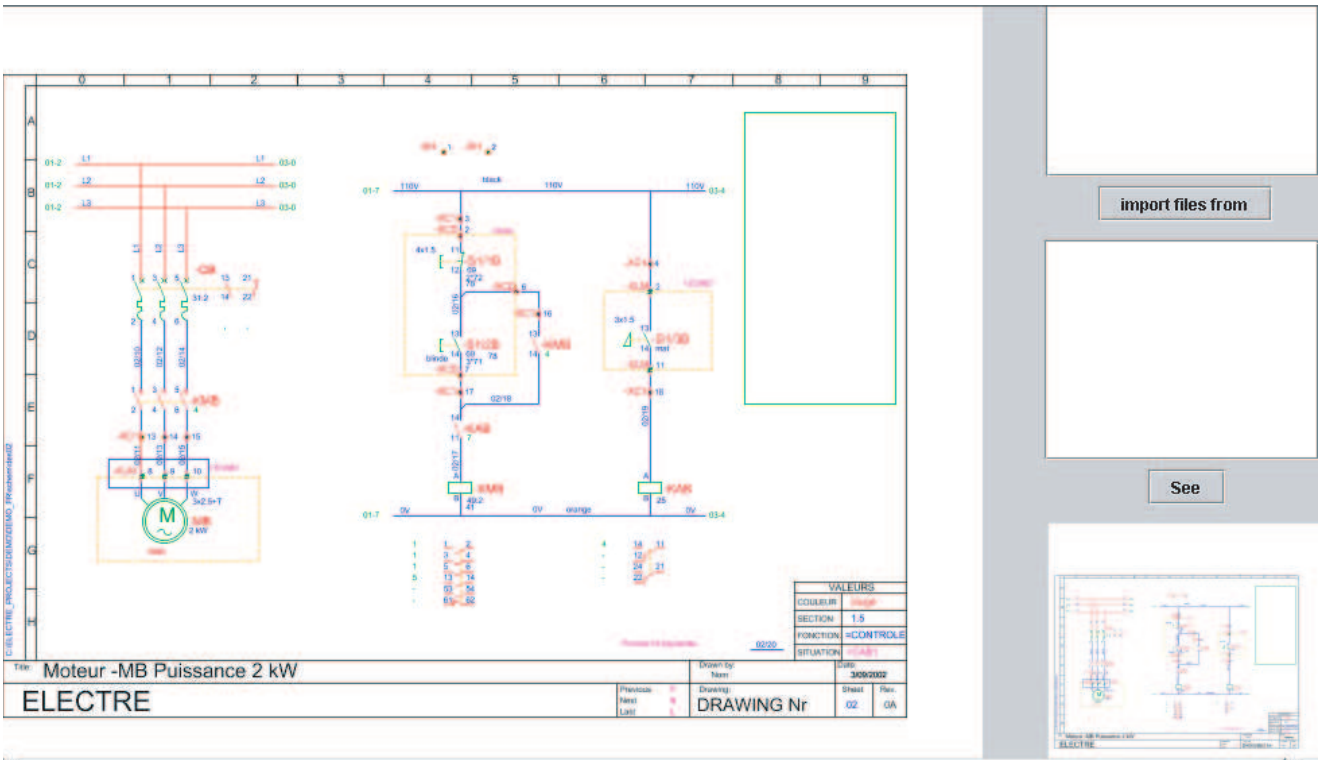


Fig. 4.1 – *Présentation de l'affichage général*

Le plan apparaît par défaut dans son entièreté et rempli l'espace alloué à l'affichage. Comme expliqué plus haut, les parties dynamiques sont chargées à ce moment. Ce qui rend l'espace dynamique, c'est l'activation des interactors. Dans le cadre de l'application, il a été définit que ceux ci le seront par l'utilisation des boutons par l'utilisateur.

4.3 Les fonctionnalités

4.3.1 Le zoom en fenêtre

Cet outil activé, l'action de la souris sur le graphique est de définir une fenêtre de zoom. Le côté le plus grand du rectangle ainsi tracé définira le rapport de zoom. La partie du dessin sélectionnée est alors réaffichée dans le canvas et la transformation est prise en compte dans un objet AffineTransform, qui provient des composants classiques de java.swing. On peut aussi retrouver l'original via cet objet, qui est un attribut du composant de base pour l'affichage : le JGVTComponent.

Le dessin du prérectangle est assuré parun interface qui offre un couche de superposition à la classe qui l'implémente : l'Overlay. Il ne possède qu'une méthode paint(Graphics), méthode qui sera utilisée pour donner un rendu du dessin que l'on veut afficher par dessus le composant SVG. En effet, pour dessiner un composant temporaire sur un espace en Java quel qu'il soit, nous pouvons passer par un Objet Graphics qui reprend la configu-

ration actuelle de l'objet par dessus lequel on veut peindre, et sur lequel on peut dessiner des formes simples. L'interface Overlay a donc comme utilité d'offrir la méthode "paint" sur cet objet Graphics que l'on extrait grâce à une fonction basique des JComponents.

Précisons que cet Overlay est utilisé chaque fois qu'un composant graphique étranger temporaire doit être dessiné par dessus le canvas, par exemple pour un surlignage de texte.

Le bouton activant l'outil est le premier de la barre. Une fois le bouton activé, l'action de presser le bouton de la souris enclenche la saisie de l'espace à réafficher. Le déplacement de la souris entraine quant à lui l'affichage d'une fenêtre, et c'est l'espace contenu dans cette fenetre qui est réaffiché au moment ou l'on relache le bouton.

Les deux phases d'utilisation de l'outil de zoom en fenêtre sont représentées, par les deux captures suivantes. Le premier temps, la délimitation de la zone, et le deuxième, l'apparition de la zone délimitée dans tout l'espace d'affichage.

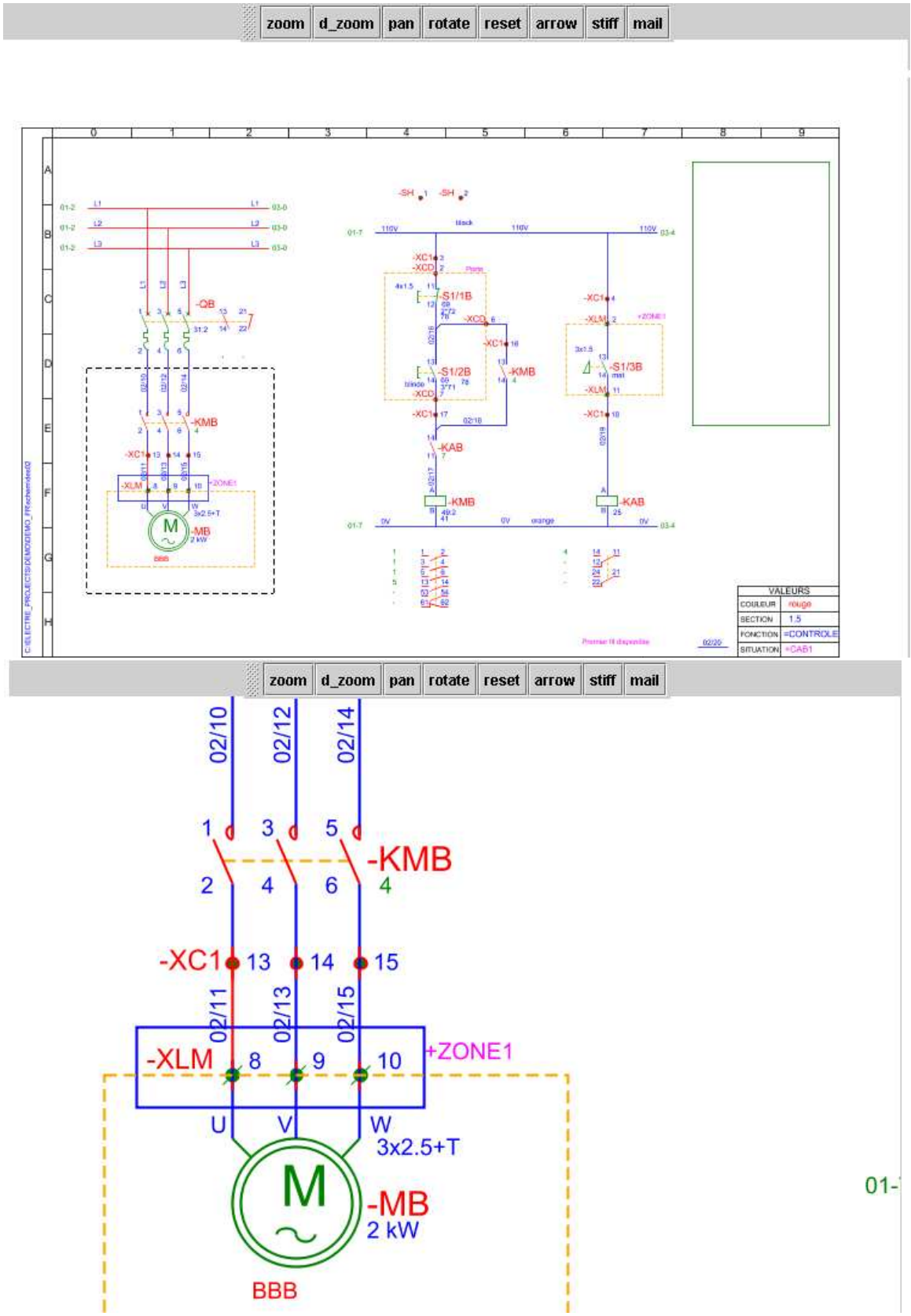


Fig. 4.2 – *Délimitation et affichage d'une zone*

4.3.2 Le zoom dynamique

Cet outil a pour but permettre de réajuster l'affichage d'un plan que l'on aurait zoomé en fenêtre, pour repréciser ou rélargir le point de vue. Cette action se produit en pressant la souris et en lui faisant faire un mouvement vertical. Le rapport de zoom est alors fonction du déplacement du curseur. Le dessin est alors modifié directement. Le seul problème de cet outil est qu'il doit utiliser une méthode des JComponents pour repeindre en direct. L'affichage, à ce moment précis, n'est plus alors celui d'une image vectorielle. Ce n'est qu'une fois que la souris est relâchée que les dimensions sont prises en compte pour recalculer l'affichage. De la même manière, si l'image est déjà zoomée et que l'on veut élargir le champs, on ne verra que la partie du dessin affichée au départ changer de taille. Mais cela est déjà suffisant pour avoir une bonne idée de ce qui sera affiché après le relachement du bouton.

Le bouton activant cet outil est le deuxième dans la barre. Une fois activé, l'image affichée change son niveau d'affichage lors d'un mouvement vertical de la souris sur l'espace du Canvas avec le bouton gauche enfoncé. L'image est effectivement réaffichée, avec une bonne définition, lorsque le bouton est relâché.

Pour illustrer le fonctionnement de l'outil, nous avons capturé différents écrans reflétant les étapes de son utilisation. Ces captures montrent la pixélisation apparente dont on a parlé. Le dernier écran souligne le fait qu'on en revient quand même à un affichage clair et précis, une fois l'outil relaché.

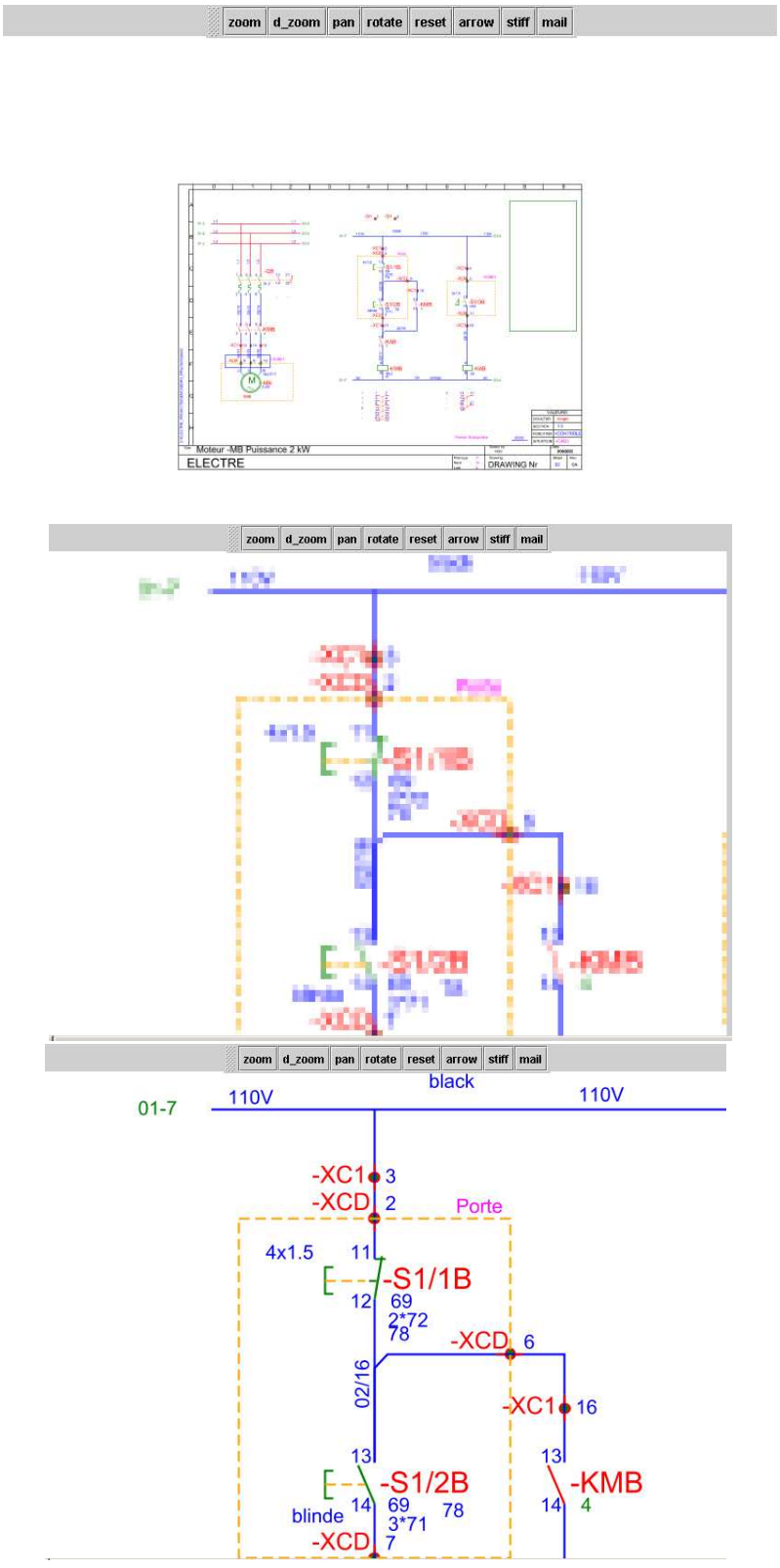


Fig. 4.3 – *Résultat du zoom dynamique*

4.3.3 Le panning

Grâce à cet outil, on peut se déplacer dans un plan sans modifier le niveau de zoom établi. Il s'agit en réalité de faire glisser le plan. On implémente cette solution plutôt que des barres de scrolling pour une maniabilité plus grande. De plus, encore une fois, le moyen utilisé pour l'affichage reste cette fonction de swing qui ne recalcule pas encore l'affichage entier du svg, mais qui ne fait que translater l'image affichée jusque là. Le scrolling n'apporte donc rien en particulier, alors que le panning, lui, permet de bouger directement l'affichage du plan dans toutes les directions.

Le bouton activant cet outil est le troisième dans la barre. Une fois activé, une pression de la souris sur le plan ancre celle-ci dessus, et la translation suit son mouvement. L'image n'est réaffichée entièrement que lorsque le bouton est relaché.

La suite de captures correspondant à cet outil veut montrer le côté pratique de la méthode en traversant en diagonale le plan, qui a été zoomé au préalable, plutôt que de devoir passer par une scrollbar qui oblige à circuler sur le plan en alternant les directions horizontales et verticales.

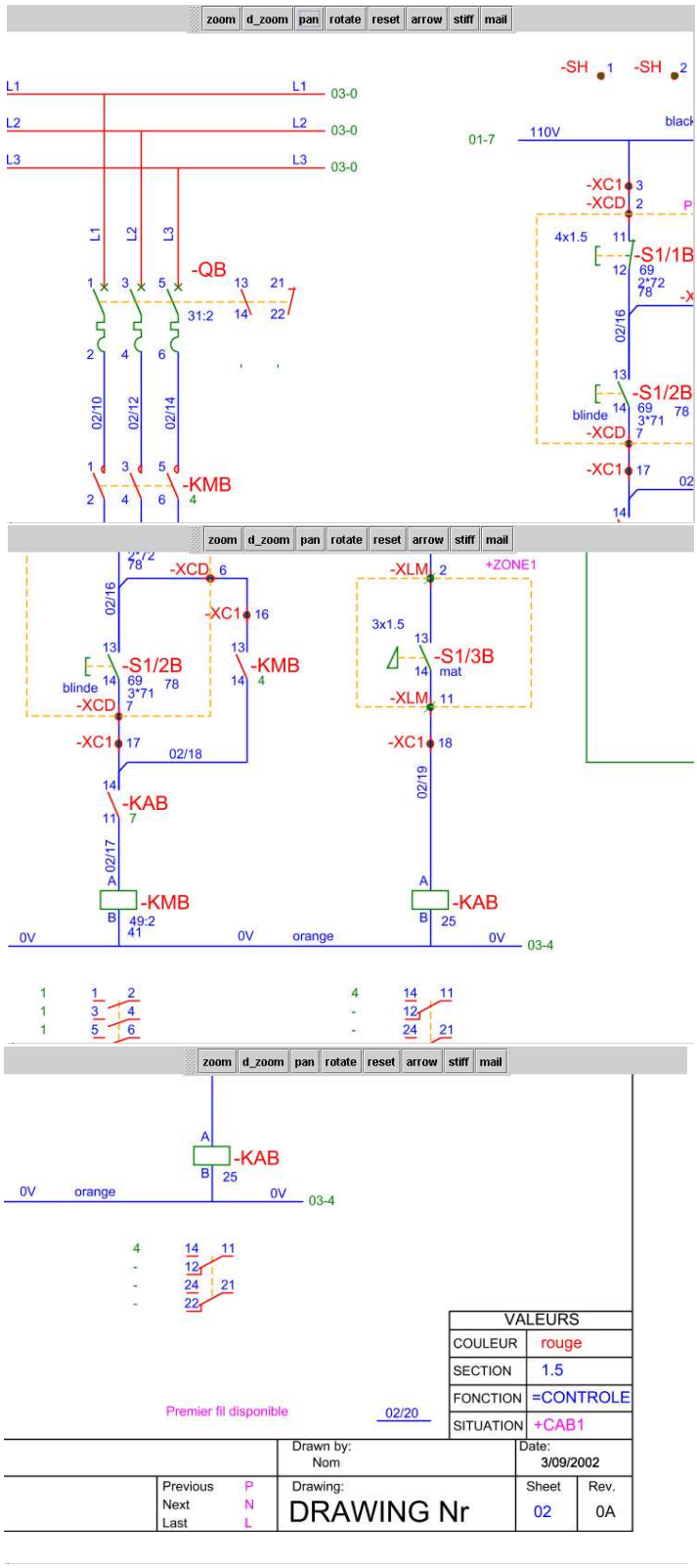


Fig. 4.4 – *Résultat du panning*

4.3.4 La rotation

Cet outil est utile dans le cas où l'on trouve dans le plan des parties écrites sur un autre axe que l'horizontale. Le mode de fonctionnement est intuitif, dans la mesure où l'endroit où l'on presse la souris est enregistré comme le centre de la rotation. L'angle, lui, est calculé comme celui du triangle rectangle formé par les déplacements de la souris en x et y, et dont un sommet est le centre. On peut suivre l'évolution de la rotation de la même manière que pour les interactions précédentes.

Le bouton activant cet outil est le quatrième dans la barre. Une fois activé, il reste à l'utilisateur à presser le bouton gauche de la souris et à observer la rotation entraînée par le mouvement de celle-ci. Il est difficile de trouver une méthode précise pour entrainer une rotation d'angle connu. Toutefois, ce mode d'utilisation est facile et suffisant pour faciliter la lecture de plans.

La capture d'écran pour l'utilisation de l'outil de rotation veut montrer la possible nécessité de faire tourner le plan dans l'espace d'affichage. En effet, certaines parties du plan, qui sont ici assez simple, sont écrites à la verticale. Il est d'autres plans où ces écritures peuvent se retrouver en diagonale.

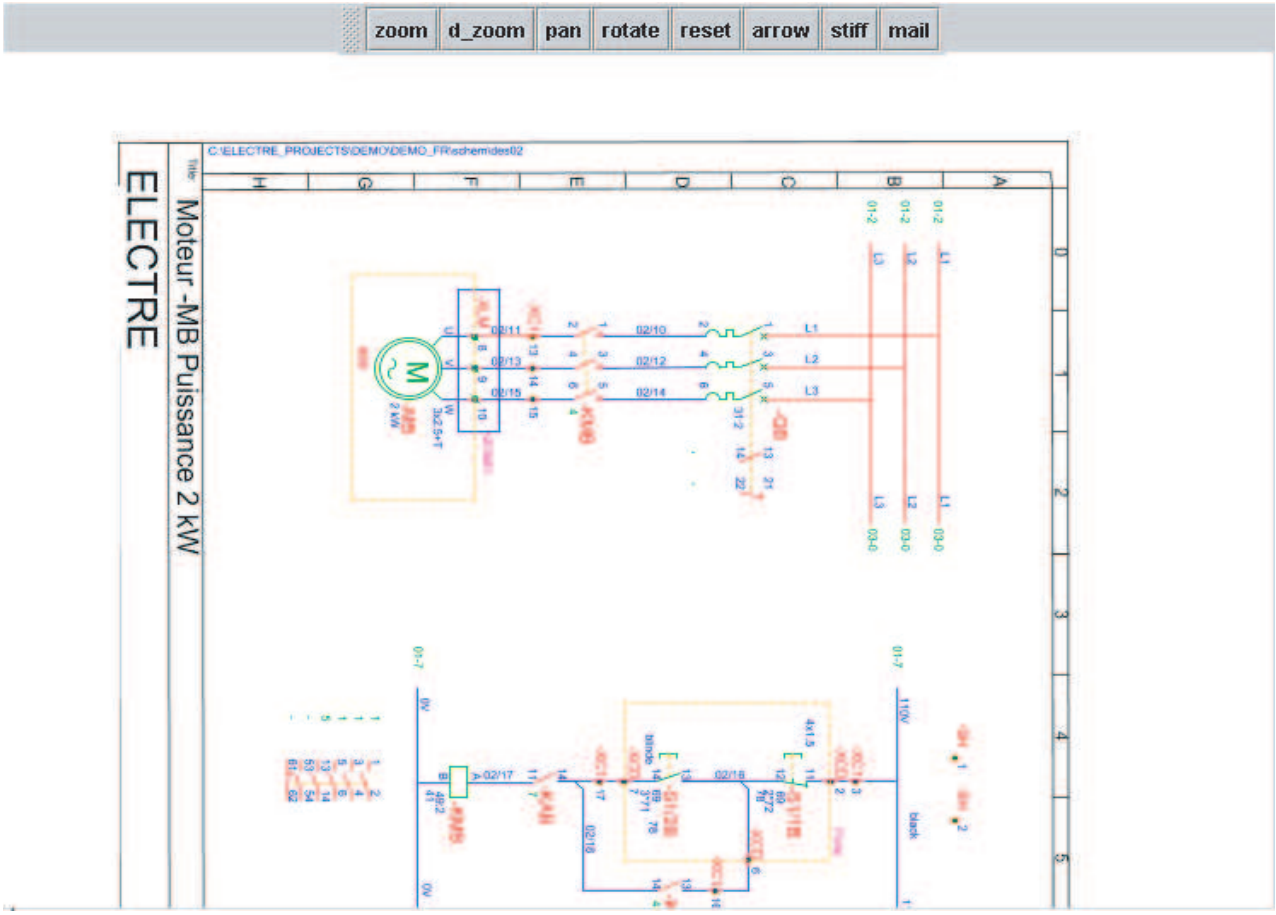


Fig. 4.5 – *Résultat d'une rotation*

4.3.5 Le red-lining

Cet outil-ci ne faisait pas partie des premières fonctionnalités standard que l'on retrouve dans différents éditeurs de SVG. C'est une fonction représentative de la spécificité de l'application. Via ce moyen, on veut pouvoir signaler clairement des zones pour lesquelles on pourrait devoir faire une remarque ou formuler une question.

Comme expliqué précédemment, il s'agit également de ne pas modifier le fichier original. Et, en observant l'implémentation des fonctions de batik, on a signalé que le fonctionnement est basé sur la conversion de l'arbre DOM en arbre graphique d'objet Java. En conséquence, toutes les modifications passent d'abord par un tampon intrinsèque. On peut dès lors modifier le document affiché sans endommager l'original.

Dans ce but, l'outil développé ici permet de définir une zone en la surlignant avec de la couleur rouge. Son fonctionnement est copié sur celui des précédents, d'un clic de bouton on délimite la zone, et lors du relâchement de la souris, les caractéristiques géométriques de la zone sont utilisées pour créer l'objet graphique que l'on va rajouter à l'arbre en mémoire. Les éléments des matrices de transformation sont utilisées pour placer le rectangle dessiné au bon endroit eu égard aux coordonnées du fichier svg original.

Lors du réaffichage du document entier, pour visualiser la zone ajoutée, le composant reprend une matrice d'affichage vierge. Il faut donc avoir sauvegardé la transformation précédente pour la rééditer. Cela permet à l'utilisateur d'observer l'effet de son action dans les conditions d'affichage où il a dessiné.

Cet outil a été développé selon un modèle général. Son implémentation permet d'en développer des semblables pour éventuellement définir un code couleur d'utilisation. Il est en effet simple, à partir du modèle établi, de définir des zones de différentes formes et de différentes couleurs, les formes pour mieux s'adapter au plan, les couleurs pour signaler différents problèmes, voire différents types de problèmes préétablis. C'est donc une bonne base pour complexifier l'outil dans la perspective d'une augmentation des capacités de communication, et donc de l'aspect collaboratif.

Pour être complet, cet outil doit être accompagné d'une fonction qui permette d'ôter du dessin des rectangles déjà dessinés. Une fois de plus, la structure XML du document vient à l'aide de cette nécessité. En effet, l'élément correspondant au rectangle ajouté est marqué d'une identité spécifique et de plus, il est rajouté en tant que dernier enfant de l'arbre. Il y a donc deux moyens de le retrouver. En extrapolant, on peut facilement penser qu'il est possible d'afficher une liste des rectangles ajoutés et de les traiter individuellement, ce qui augmenterait la flexibilité de l'application.

Le bouton activant cet outil est le troisième dans la barre. Une fois activé, l'utilisateur pourra, de la même manière que dans le cas du zoom en fenêtre, définir une zone rectangulaire. Cette fois-ci, la zone délimite l'espace qui sera occupé par une trace rouge dont la siginfication est de signaler un problème. Il est possible d'en dessiner plusieurs sur un même canvas, et même de varier l'intensité de la couleur. En effet, si l'utilisateur rajoute un surlignage par dessus le précédent, la couleur est intensifiée sur la zone de

superposition.

Ces deux captures montrent, respectivement, un document qui a déjà reçu une trace de red-lining, et sur lequel l'utilisateur en délimite une zone pour une deuxième trace, et dans un deuxième temps l'effet de la superposition de deux traces, qui permet de renforcer la couleur sur une zone critique par exemple.

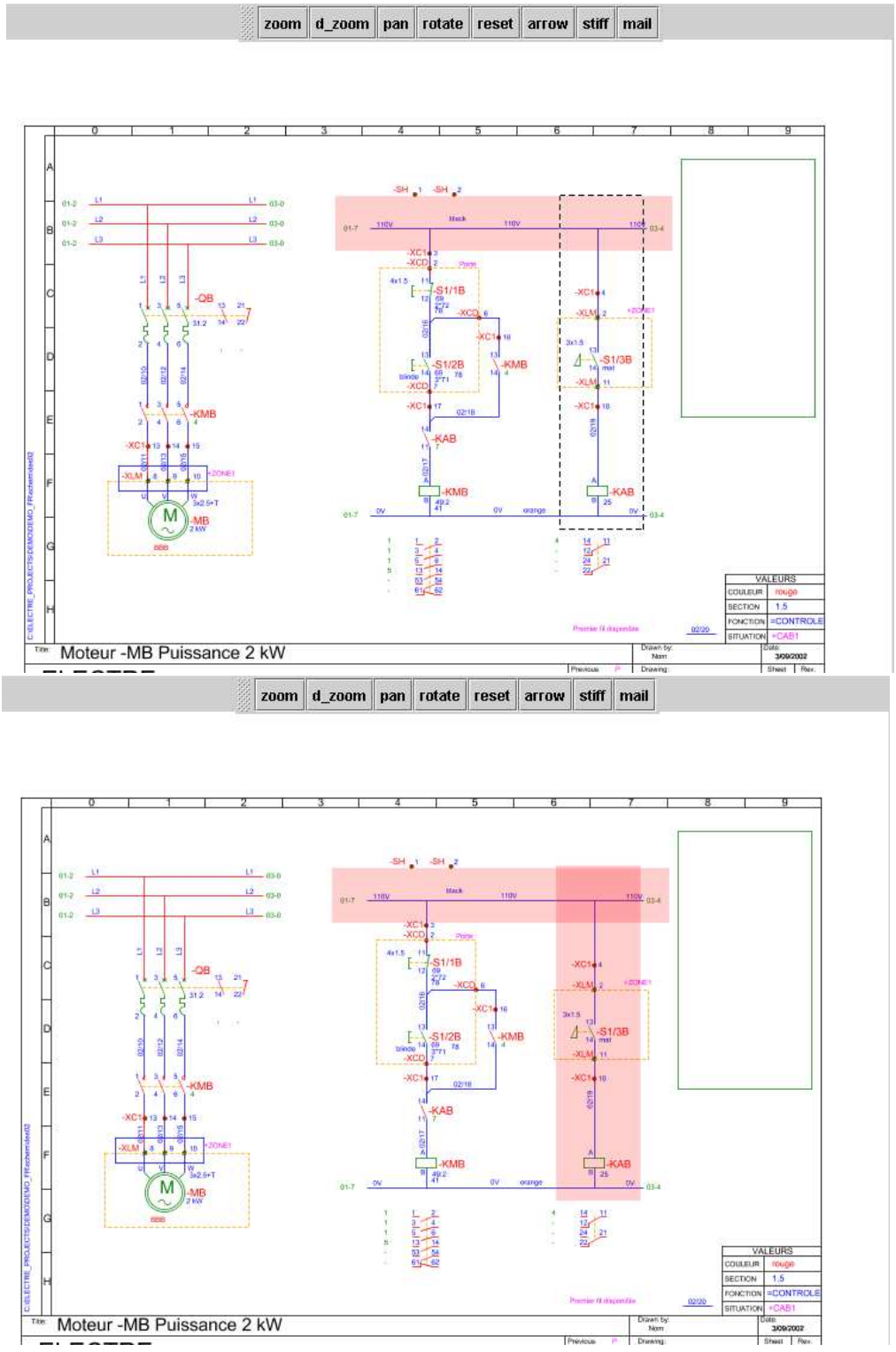


Fig. 4.6 – *Résultat d'un red-lining*

4.3.6 Le renvoi par email

Cet outil devait être implémenté dès lors qu'il constitue la base même d'un instrument de communication, indispensable à une application qui se veut collaborative. Pour cette fonction, une fois encore, Java offre un outil tout trouvé : le JavaMail([17]). Il s'agit d'une bibliothèque de fonctions implémentant les services mail de base que l'on retrouve sur la majorité des serveurs : SMTP, POP, IMAP et MIME. Nous nous limiterons au plus utilisé d'entre eux : SMTP.

Toutefois, le fonctionnement même des serveurs mail complique la structure de l'application dans la mesure où l'applet fonctionne dans le browser. En effet, les serveurs SMTP ne pas peuvent relayer les mails venant de n'importe où, mesure prise sur l'entièreté du réseau Internet pour éviter le phénomène de spam (diffusion excessive de messages). C'est pourquoi, si une demande d'envoi de mail provient de l'applet, elle ne sera probablement pas traitée car non identifiée.

Pour contourner ce problème, on va tenter de transférer la tâche au serveur même. Le moyen pour faire ça est d'avoir une application qui tourne sur le serveur, attendant de recevoir les données d'un mail. Lorsque la fonction de mail de cette application est apellée, elle formatte ces données et les adresse au serveur mail. On voit donc se profiler la structure d'un petit server et d'un client qui serait rattaché à l'applet. Et encore un fois, cela est implémentable grâce aux fonctions de Remote Method Interface.

L'espace de mail prévu permet de rentrer les données nécessaires au formatage d'un email. Celles-ci sont transmises au server accompagnée du document tel qu'il a été modifié par le red-lining. Pour alléger le fonctionnement dans le browser, la tâche de reconvertir le document (qui contient en fait l'arbre graphique) est laissée au server. On prendra juste garde à ne pas écrire ce fichier dans un espace contenant les originaux.

A partir de là, l'origine du message est clairement identifiée par le serveur mail, et il est donc possible de l'envoyer.

Le bouton activant cet outil est le dernier sur la barre. Lors de l'activation apparaît un pop up. Une fois les différents champs remplis, appuyer sur le bouton "Send" transmet un mail de format standard à l'adresse indiquée avec, en attachement, le plan tel qu'il a été modifié du point de vue red-lining. Les transformations géométriques, quant à elles, ne sont pas rendues, dans la mesure où le récepteur pourra visualiser, et donc manipuler le document de la même manière que l'a fait l'expéditeur.

Ici, nous voulons juste montrer la fenêtre qui apparait lorsque l'utilisateur désire envoyer un mail.

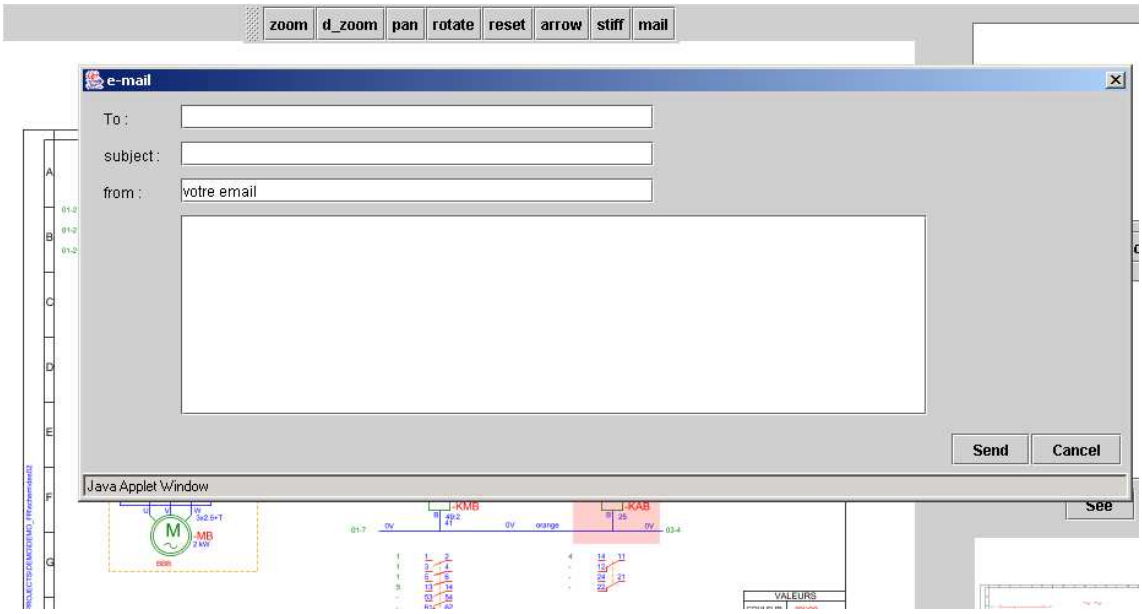


Fig. 4.7 – *Fenetre d'envoi de mail*

4.3.7 La jointure interfolios

Au sein même du programme Electre NT, les projets sont donc divisés en folios, chaque folio représentant une partie d'un plan. On conçoit facilement que certains câblages ne peuvent tenir en un plan, mais que les différentes parties ont des liens entre elles. Ces liens sont représentés par des codes internes à Electre. Ils se situent aux endroits même où, physiquement, les jointures se font.

Il a donc fallu retranscrire ces liens dans le document SVG. Dans le format vectoriel, cette fonction de jointure est remplie par le protocole XLink. Dans la recommandation du W3C, il est expliqué comment implémenter un lien vers une autre source. Dans le cadre d'une application Java, il s'agit encore de parvenir à récupérer l'utilisation de cet élément précis pour, lors de son activation par l'utilisateur, lui donner un comportement semblable mais purement interne à l'application.

Ces éléments, en particulier ceux de type XLink, possèdent déjà des propriétés qui, lors de leur survol, permettent de changer l'aspect du curseur. En cliquant sur celui-ci, l'évènement est pris en charge par le UserAgent et la référence signalée dans le tag est récupérée. On aura pris soin de ne laisser dans la référence que le nom du fichier, sans en mentionner d'adresse complète. L'emplacement des dossiers des différents projets est une constante mémorisée dans le programme, et on peut raisonnablement supposer que des folios ne sont liés à d'autre qu'au sein d'un même projet. Il ne reste donc qu'à concaténer les différents noms de dossiers avec le nom du fichier, et à relancer pour le canvas, un chargement de fichier avec le string créé juste avant.

Ici, il ne s'agit pas d'un outil au sens entendu précédemment. Son utilisation est suggérée par un changement de forme de la souris (la forme standard d'ouverture de lien), au survol des éléments concernés. Un simple clic sur l'élément et le folio concerné

est immédiatement affiché dans le canvas. La forme de ces éléments de lien est la même que celle de Electre NT.

Afin de montrer le standard de lien d'Electre, nous reprenons ici une extrémité de câblage du plans affiché en début de chapitre. Il s'agit des chiffres en vert, indiquant l'extension du folio correspondant.

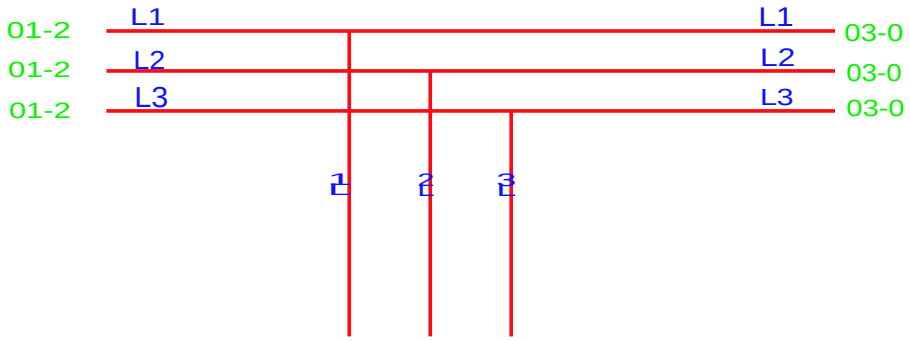


Fig. 4.8 – *Format des liens Electre*

4.3.8 Possibilités d'outils supplémentaires

Grâce à l'objet UserAgent dont on a déjà parlé, il est donc possible de récupérer les éléments survolés par la souris. Ceux-ci ont également une structure en plusieurs extensions de classes successives pour spécifier au fur et à mesure les attributs particuliers de l'élément. Cette structure permet d'intervenir, sur les attributs qui le forment, à plusieurs niveaux.

La structure en arbre d'objets et le fonctionnement de Java permettent donc, dans la mesure où tout élément est objet, de leur rajouter des caractéristiques Java, telle que un MouseListener. De plus, il est possible d'analyser un nœud en fonction de ses éventuels enfants, ce qui permet donc de pré-traiter un arbre en vue d'imputer des caractéristiques à différents niveaux. D'où la possibilité d'avoir un affichage différent pour différents utilisateurs.

En effet, la structure d'un document SVG est calquée sur celle de tout document XML. On peut donc retrouver diverses informations à différents niveaux. On retrouve également cette stratification dans des plans électriques, où des informations se retrouvent à différents niveaux. Ces informations étant utilisées différemment d'un utilisateur à l'autre suivant son profil, il peut être plus efficace, après identification de

l'utilisateur, de personnaliser certains détails d'affichage supplémentaires.

Toutefois, cela n'est possible que si l'on établit une structure du document svg correspondant à celle du plan dès sa génération dans Electre. De la sorte, on augmenterai d'un côté la dépendance de l'application envers Electre (ce qui ne pose pas forcément de problème), mais de l'autre cela renforcerait l'aspect collaboratif de l'application. En effet, les messages arrivant d'un utilisateur ou d'un autre seront d'autant plus précis quant à leur contenu que la possibilté de prendre en compte des données qui ne le concernent en rien ne leur est pas laisée .

De la même manière, il est possible d'imaginer que les textes soient mis en évidence ou réécris dans un espace indépendant, de manière à ce qu'ils soient lisibles plus facilement. Cela permettrait une lecture plus rapide et moins fastidieuse de plans particulièrement grands.

Chapitre 5

Techniques particulières d'implémentation

Sommaire

5.1 Chargement d'un fichier

Nous avons déjà décris la stratification du composant qui permet l'affichage le SVG. Rappelons que la deuxième couche, le JSVGComponent, offrait une fonction de chargement direct en passant à une de ses fonctions un url référençant l'adresse du document voulu. Nous allons examiner d'un peu plus près le fonctionnement de ce mécanisme.

Lors de l'appel de cette fonction, deux objets complémentaires sont créés : le DocumentLoader et le SVGDocumentLoader. En effet, ce dernier nécessite l'intervention du premier pour son fonctionnement.

Le DocumentLoader prend en charge le chargement d'un document depuis une URI et le garde en cache dans une HashMap. C'est au moment du chargement que le document est analysé pour savoir quel parseur sera utilisé. Le SVGDocumentLoader, qui est un thread, appelle la fonction de chargement du premier. De plus, ce thread permet de charger un document de manière asynchrone. Ce thread possède en plus des fonctions génératrices d'événements signalant l'évolution du chargement du fichier. Cela permet de contrôler le chargement et de signaler une erreur, en cas de document non valide par exemple, sans interrompre l'application en cours.

Une fois le document chargé, il est parsé, c'est à dire analysé. Pour cela, les classes de SAX (Simple API for XML) sont utilisées. Celles-ci sont essentiellement des interfaces à

implémenter, mais qui permettent d'obtenir les fonctions nécessaire à l'analyse d'un document XML (ici, SVG). Cette technique d'analyse, en concurrence avec celle du DOM, est basée sur les évènements : le début ou la fin d'un nœud génère un événement. Cela permet donc au parseur de transmettre à l'application un élément après l'autre. C'est à la récupération de ces éléments que l'arbre DOM est reconstitué du côté de l'application. Lors du rendu de l'image, cet arbre est alors lié à un arbre graphique virtuel. Ensuite, ce sont les fonctions de DOM, implémentées dans Batik, qui sont utilisées pour le traitement de l'arbre, la méthode étant plus simple et la plus directe. On tente donc, en définitive, d'utiliser les avantages de chacune des deux techniques pour chaque étape du processus.

5.2 SAX versus DOM

Le *Document Object Model* est une représentation en arbre d'un document SVG. Ce modèle est une recommandation du World Wide Web Consortium (W3C), qui a également publié une API pour le traitement de ce DOM dans le cadre du SVG, Batik proposant une implémentation de celle-ci. Pour la circulation dans l'arbre, on utilise XPath ou des mécanismes assimilés. Ces mécanismes permettent de parcourir un arbre au moyen de quelques commandes qui représentent les circulations verticale et horizontale. Il est également possible de sélectionner un nœud en fonction de ses différents attributs, voire de son type de descendance. Ce modèle convient particulièrement à une implémentation orientée objet. En effet, il est très facile d'implémenter la structure d'un arbre à l'aide d'objets.

Le *Simple API for XML* permet quant à lui d'implémenter des fonctions qui écoutent des évènements générés par le parseur. Ces évènements surviennent en début de document, en début et en fin de chaque nœud, mais aussi à la lecture de valeurs ou d'attributs d'un nœud. L'avantage de cette technique est sa légèreté. En effet, si l'application concernée implémente les classes nécessaires à l'écoute de ces évènements (soit une implémentation de SAX), elle peut aussi transposer presque nœud par nœud la forme de chaque élément dans un objet ayant la forme que l'on désire pour la suite du programme. L'énorme avantage pour la lecture via un tel moyen est la légèreté au niveau utilisation mémoire. En effet, toute la structure ne doit pas être gardée en mémoire dès lors que l'on peu directement créer les objets.

Par exemple, les évènements générés à la lecture de ce fichier :

```
<?xml version=“1.0”?>
<doc>
<para>Hello, world!</para>
</doc>
```

Donnerait, comme succession d'évènements :

```
start document
start element: doc
start element: para
characters: Hello, world!
end element: para
end element: doc
end document
```

On peut aisément interpréter cette série comme un document, le premier enfant étant un élément ¡doc¿. Le deuxième élément rencontré suivant arrive avant la fermeture du premier, il s'agit donc un enfant. Ensuite, on lit des caractères, avant l'ouverture d'un nouveau tag, c'est donc la valeur de l'élément. Dès que l'on retrouve la fermeture de l'élément, on sait que l'objet correspondant est terminé. On peut donc reconstruire l'arbre sans être obligé de le mettre entièrement dans la mémoire, ce que l'on devrait faire en utilisant le DOM.

Le parseur utilisé ici s'appelle Crimson. Les évènements sont récupérés par un objet SAXDocumentFactory, qui implémente le LexicalHandler de SAX. Cette méthode est donc utilisée pour alléger le chargement de l'image dans la mesure où tout ce qu'il faut, c'est lire le document tout en générant des évenements, là où l'analyse en utilisant le DOM nécéssite la création de l'arbre.

5.3 Détail de l'architecture d'un élément

Le W3C propose donc l'API pour le DOM. Celle-ci est représentée par une série d'interfaces qui, pour les éléments, commence par celle de Node. Cette interface propose des méthodes de manipulation des nœuds et de parcours à travers un arbre essentiellement. On peut également déjà récupérer une liste des attributs, mais sans identification directe. Le type de nœud y est représenté par une des constantes définies pour la classe. Cette couche représente l'interface générale pour tous les types de nœud d'un fichier XML :

- root
- attribut
- élément
- processing instruction
- namespace
- comment
- texte

Les nom et la valeur de chacun de ces types peuvent être retrouvés, étant entendu que les nœuds de types comment, text et root n'ont pas de nom.

Les premières spécificités permettant de distinguer chacun de ces types se retrouvent à la couche suivante, où l'on retrouve des interfaces pour chacun d'eux. Et c'est dans la couche suivant celle implémentant l'élément standard que l'on retrouve tous les attributs spécifiques à chacun des éléments propres au SVG.

Ces interfaces sont implémentés dans Batik sous les noms SVGOM-"nom de l'élément", et permettent alors d'utiliser les méthodes à partir d'éléments pris en charge dans le fonctionnement de Batik même.

L'API du W3C propose encore des objets indispensables au traitement via un langage de programmation d'un document SVG. Parmi eux on retrouve le "Document", permettant de retrouver le fichier SVG dans son entièreté, mais sous forme d'objet, et l'objet "NodeList" qui permet de lier les nœuds entre eux.

Les autres parties de l'API concernent les feuilles de style (que nous n'utilisons pas ici) et les fonctionnalités des aspects dynamiques d'un graphique vectoriel. Ces différentes fonctionnalités sont toujours développées selon le même modèle : on démarre d'une interface général, au niveau de laquelle on retrouve les spécificités de base, et des interfaces successives amenant chacune leur spécificités, de manière à pouvoir modulariser le plus efficacement possible le comportement de chaque élément. En effet, les objets des classes de Batik implémentant les interfaces du DOM n'implémente pas seulement l'interface de l'élément leur correspondant, mais aussi toutes les interfaces définissant avec précision leur comportement potentiel.

Dans cette optique, même la saisie d'évènements a du être transformée par rapport à la gestion d'événements classiques. En effet, ceux-ci peuvent être générés par différents éléments du graphique, entraînant une nécéssité d'identification de leur source. C'est à ces fins que l'on retrouve et une interface désignant le fait d'être éventuellement une cible pour la souris, et une autre reprenant les différents évènements de la souris. Enfin pour compléter ces fonctions, un interface permet même de créer des évènements particuliers. De cette manière, on peut imaginer presque toutes les interactions nécessaires pour créer un environnement pleinement collaboratif.

L'observation de la structure de cette API, recommandée par le W3C rappelons le, permet d'avancer l'idée que le SVG, dans le cadre de ce travail, est le support principal de l'aspect collaboratif. En effet, ce sont les notions de découpage du dessin en éléments, ainsi que les attributs particuliers imputables à chacun de ceux ci qui permettent d'envisager un travail complexe sur base de plans déjà produits ou même à développer.

5.4 Processus de rendu d'une image

L'on a expliqué comment se passait le chargement d'une image, par le SAXDocumentFactory, qui permet de reconstituer l'arbre DOM du document. A partir de là, les

fonctions de Batik permettent de lier cet arbre DOM à l'arbre graphique virtuel (GVT tree), à travers toutes les fonctions de "Bridge". Ce sont les objets de l'arbre graphique virtuel qui sont utilisés pour toutes les manipulations.

La librairie Batik fournit aussi les fonctions qui permettent de lire ces objets, et, à partir de leur état, utilisent les fonctions classiques de dessin de Java pour dessiner une Image. Cette image est placée dans un tampon, et c'est celle ci qui est utilisée lors de l'affichage.

Pourquoi alors avoir des composants particuliers pour permettre d'afficher ces images ? Tout simplement parce que c'est à ces composants que sont passés ces différents objets permettant de dessiner les graphiques voulus. De cette manière, on peut récupérer directement tout événement se passant sur le canvas. Ces évènements sont alors passés à l'agent qui gère les services offerts par le composant, et c'est de là que le mécanisme de dessin des différents éléments est mis en marche. Les fonctions de Batik permettent de dessiner toutes les formes dans un contexte graphique, et ce contexte graphique est lui même peint sur le composant (c'est le rôle standard des contextes graphiques : pouvoir peindre sur n'importe quel composant).

5.5 Les Interactors

Ce moyen de dynamiser l'utilisation de l'espace d'affichage du svg a une structure bien particulière. La structure de ce "super-listener" est imposée par Batik : il existe un interface, "Interactor", qui est implémenté par une classe "InteractorAdapter", décrivant les fonctions par défaut d'un tel objet. L'implémentation d'un tel objet permet, en le rattachant au composant concerné, de lui octroyer une propriété particulière.

L'objectif est alors de parvenir à décrire avec précision le comportement souhaité pour l'interaction. Pour cela, il faut préciser ce qu'il se passe lors de l'utilisation de l'outil, ou, en d'autres termes, les séquences qui s'exécutent à l'écoute d'un ensemble d'évènements qui constituent l'interaction. En pratique, il s'agit, dans le cas du zoom en fenêtre, de l'affichage de la fenêtre en pointillé désignant la zone. Cette définition se fait dans une classe qui étend l'"InteractorAdapter", en implémentant les réactions aux évènements de mouvement et d'action de souris, ainsi qu'un objet de la classe Overaly dans l'hypothèse ou l'action devrait afficher un composant temporaire supplémentaire. Le modèle implémenté dans Batik porte le nom de Abstract-"nom de l'outil"-Interactor.

Reste alors à préciser l'action déclencheuse de l'interaction. Celle-ci, toujours dans le modèle, se déroule dans la classe du composant affichant le SVG. A cet endroit, on crée une instance d'un objet Interactor, avec le constructeur particulier de l'interaction voulue (Abstract...Interactor). Dans cette même instanciation on définit la fonction start(), où l'on précise la combinaison de touche et de clic de souris qui enclenchera la manifestation de l'interaction.

Ici, dans la mesure où le contrôle de l'activation des outils est laissé à l'utilisateur et que ces outils sont neutralisés par l'activation d'un seul d'entre eux, l'enclenchement de l'interaction se fait au premier clic. Précisons qu'il existe une différence entre l'évènement produit par un clic de souris et celui d'un bouton pressé et non relaché. Lors de l'activation d'un des outils, le composant d'affichage est continuellement à l'écoute des entrées de l'utilisateur.

Chapitre 6

Conclusions

Sommaire

6.1 Travail accompli

6.1.1 Les choix

Dans ce travail, nous avons recherché des techniques informatiques permettant de promouvoir une philosophie de travail, l'ingénierie collaborative. Cette philosophie se base sur la communication rapide et efficace d'informations ciblées, centrées sur le développement d'un projet commun à différentes équipes ou différentes personnes. Si dans une version complète et aboutie, la majeure partie des tâches de maintien de la cohérence des projets est assurée par l'application elle-même, la base d'un tel programme passe par l'interface utilisée par les différents intervenants sans pour autant agir sur le contenu. Dans un premier temps, ces tâches peuvent largement rester "manuelles".

D'un point de vue pratique, en observant le marché actuel de la CAO, et plus particulièrement de la CAO électrique, on a pu constater qu'il existe deux degrés d'avancement. D'un côté, des gestionnaires de projets déjà aboutis mais très peu flexibles, de l'autre, des applications individuelles rattachées à un programme particulier, mais aussi largement plus adaptables aux besoins de l'utilisateur. Au sein même de ces deux catégories, une seconde distinction peut être faite entre celles basées sur le noyau graphique de l'application mère et celles restées indépendantes. Le choix réalisé à ce niveau a été largement influencé par la philosophie de travail d'Elsys qui se veut proche des besoins du client et indépendante des technologies employées. L'application est donc étroitement liée à Electre NT tout en étant indépendante de son fonctionnement. Un nouveau format graphique devait alors être employé.

A ce stade, il a été intéressant d'observer les différentes solutions implémentées et utilisées sur Internet, afin de pouvoir en comparer les limitations, la comparaison se limitant assez rapidement au bitmap versus SVG. Dès lors, la combinaison des différents facteurs clés dans le cadre de l'application (taille de l'image, espaces d'affichage, temps de calcul en cas de modification ainsi que sauvegarde de ces modifications) nous a assez vite amené au dessin vectoriel, dernière technique en date de rendus de graphiques. En effet, les différentes propriétés observées dans ce cadre nous ont paru très intéressantes dans une perspective d'exploitation dynamique des dessins. Restait à trouver le langage permettant de manipuler un tel format.

Les facteurs qui ont influencé le choix du langage ont été plus divers, la gamme des choix étant plus large, mais le nombre de contraintes, de par la publication de l'application sur un réseau, l'étant également. S'il existe plusieurs langages permettant l'implémentation du traitement SVG, le plus largement utilisé reste le Java. Cet argument appuyé par la bonne connaissance qu'en a le programmeur, ainsi que la perspective d'utiliser une bibliothèque déjà évoluée mais encore en plein développement, Batik, a donné l'avantage à ce langage. De plus, Java propose suffisamment de solutions concernant les réseaux pour en trouver une satisfaisante.

6.1.2 Les fonctionnalités

Cette interface devait être la base d'une application collaborative. Dans cette perspective, elle a été développée selon 4 axes critiques : facilité de publication des travaux, soutien important pour une bonne lecture du plan, communication utilisateur-destinataire précise et enfin, légèreté de fonctionnement.

Publication

Pour son bon fonctionnement, l'arborescence intrinsèque au programme Electre a été réutilisée. L'agencement des projets y est identique, et les liens entre les folios suivent la même logique. Cette réalisation n'entraîne donc pas de changement pour les utilisateur d'Electre.

Lecture

Les premiers outils implémentés sont essentiellement des outils de soutien à une bonne lecture du plan, quelle que soit sa taille. Ce point est critique dès lors qu'une lecture erronée d'un plan fausserait déjà toute la comunication entre les parties. C'est pourquoi tous les détails nécessaires à l'utilisateur sont accessibles via ces outils.

Communication

Une fois la lecture du plan effectuée avec soin, l'utilisateur pourra délimiter avec précision toute zone pour laquelle il aurait à faire une remarque. Le dessin alors transformé ne remplace pas l'original, et ce afin de respecter le processus industriel, mais peut être envoyé par email à une adresse fixée. Le message accompagné du graphique surligné contient alors une information structurée ne laissant que peu de marge quant à son interprétation, tout en laissant la décision au bureau d'étude.

Légèreté

Une application tournant sur un réseau se doit d'être suffisamment légère pour permettre un comportement fluide. C'est dans ce but que différentes techniques ont été combinées, notamment au niveau de l'analyse du fichier SVG. De plus, le dernier traitement de l'arbre, qui est la réécriture d'un fichier SVG pour l'attacher au mail, se fera sur le serveur, afin de soulager un peu l'applet. Le temps d'utilisation pour un travail complet reste donc très raisonnable.

6.2 Les perspectives

Cette application a été développée comme une base pour un développement potentiel plus important. La poursuite des axes cités ci-dessus mènera à une application complète telle que décrite dans l'introduction. En effet, si l'on complète les différentes fonctions implémentées, leur application finit bien par atteindre un comportement proche d'un environnement de développement collaboratif.

Publication

Lors du déploiement, nous avons parlé d'automatiser le processus de placement d'un fichier sur le serveur. Cette automatisation aurait pour conséquence de rattacher un peu plus l'application à Electre NT sans diminuer la flexibilité. Dès lors, pour le contrôle des fichiers en circulation, l'application pourrait être interfacée avec un PDM (Product Data Management). A partir d'une solution de ce type, les différentes versions publiées seraient référencées, et un suivi à long terme des projets pourrait sérieusement être envisagé. De cette manière on diminue les interventions humaines, et donc un certain risque d'erreur au niveau du placement, ainsi qu'une perte de temps dans la démarche.

Ce processus d'automatisation devrait alors être amorcé au niveau d'Electre NT même. Une commande "publication" générerait le format SVG et établirait une connexion avec le serveur ad hoc pour le placement du fichier dans un répertoire image de celui d'Electre. Et ceci en signifiant les attributs du fichier au gestionnaire de fichiers. On voit donc que la structure de fichier sur le serveur a une importance sur le long terme.

Lecture

Celle-ci peut être largement améliorée par le placement de couches d'informations cachées que l'on fait apparaître lors de l'interaction avec l'utilisateur et en fonction du statut celui-ci. Cela implique une découpe des différents utilisateurs potentiels de l'application en ctégories, et leur attribution d'un niveau d'accès. Ce fonctionnement est rendu possible par la structure de SVG et sa maniabilité avec Batik. En effet l'analyse du fichier SVG est faite juste avant l'affichage, par des fonctions de Batik. Dès lors, il est possible, avec une identification préalable de l'utilisateur passée en paramètre à l'applet, de personnaliser les couches d'informations accessibles en personnalisant le traitement de l'arbre.

De cette manière, la lecture que l'utilisateur fait du plan est une lecture qui convient directement à son statut, et qui est donc encore plus précise que précédemment. La communication basée sur cette lecture en sera dès lors améliorée.

Communication

Les améliorations de ce point de vue rejoignent celles la publication pour ce qui concerne l'automatisation de l'envoi de l'information : plutôt que de passer par un service mail, charger le serveur de renvoyer le fichier directement, via un lien FTP par exemple, vers un dossier du bureau d'étude. Cette automatisation permettrai, de la même manière que dans la publication, de passer par l'étape de gestion des fichiers au travers d'un PDM.

Mais en plus, il est largement possible d'améliorer le système de signalisation des erreurs. Nous avons déjà envisagé la possibilité de varier les formes d'affichage, mais il est possible d'aller plus loin. Avec l'automatisation de la communication, les différentes versions d'un même folio seraient répertoriées dans le PDM. De plus, le fonctionnement de Batik permettant la gestion des éléments séparément les uns des autres, on pourrait autoriser certains utilisateurs à rajouter ou enlever des éléments, voire en modifier la forme, dans la mesure où, rappellons le, le fichier original n'est pas touché par les transformations. Ce fichier alors renvoyé vers le bureau d'étude pourrait être affiché par celui qui examine les propositions en le superposant à l'original. Il est alors capable de voir précisément, par transparence, les différences suggérée par l'expéditeur. Ceci est rendu possible par la gestion de l'intensité d'affichage intrinsèque au SVG.

Une fois la manipulation d'éléments électriques envisagée, la marge n'est plus très grande avec l'environnement de conceptualisation en ligne. Bien sûr, cette idée nécessiterait d'implémenter des fonctions d'Electre au niveau de l'application réseau pour le contrôle de la cohérence des plans. Cela représente une grosse quantité de travail, mais reste envisageable pour peu que la demande soit présente.

Légèreté

Dans une perspective générale de complexification de l'application, la légèreté peut être conservée. En effet, la situation physique des différents intervenants dans le processus permet de penser que des traitements intermédiaires sur les différents noeuds de la chaîne pourrait soulager un maximum le traitement au sein du browser. Il s'agirait alors, en quelque sorte, d'un traitement en cascade du fichier SVG, chaque étape pouvant sélectionner un certain nombre de couches ou d'informations à garder ou à ôter en vue de la publication.

6.3 Résultat

On peut donc affirmer que cette application a été developpée dans une perspective de développement itératif. En partant de l'analyse d'une situation, nous avons extrait les technologies les plus appropriées pour répondre au problème, et les avons implémentées dans un prototype de départ. Ce prototype est déjà utilisable, et même utile dans son état actuel. Il propose des fonctions basiques mais importantes dans la philosophie de l'ingénierie collaborative.

Nous avons aussi montré que ce prototype est bien une base pour un développement ultérieur de fonctions bien plus complexes. Le tout s'inscrivant définitivement dans une perspective d'avenir. Le seul frein à un tel développement est son envergure. En effet, il s'agirait alors d'un tout nouveau programme de CAO, ou presque. Bien sûr, il se baserait sur des fonctions déjà pensées au niveau d'Electre, mais le côté spécifique du réseau et du SVG imposeraient de repenser la majeure partie du fonctionnement.

Enfin, précisons que l'application a déjà été baptisée : e-Electre. Elle commence à trouver sa place au sein du programme même, dans la mesure où le programme de setup d'Electre NT v.9 propose l'interface de configuration des fichiers SVG pour leur affichage dans e-Electre.

Bibliographie

[1] Site de présentation de la bibliothèque Batik. http://xml.apache.org/batik/

[2] Javadoc de Batik. http://xml.apache.org/batik/javadoc/index.html

[3] Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation 14 January 2003. http://www.w3.org/TR/SVG11/

[4] UML analysis of Batik. http://opensource.objectsbydesign.com/batik/index.html

[5] Rapport de conférence des développeurs SVG à Zurich du 15 au 17 juillet 2002. http://www.svgopen.org/press_info_sponsoring_f.shtml♯about_svg

[6] J. David Eisenberg *SVG, production orientée XML de graphiques vectoriels*, O'Reilly Edition française février 2003.

[7] Mailing list des développeurs de Batik. http://koala.ilog.fr/batik/mlists/batik-dev/archives/maillist.html♯01960

[8] Mailing list des utilisateurs de Batik. http://koala.ilog.fr/batik/mlists/batik-users/archives/maillist.html

[9] Javadoc Java2 Platform Std. Ed. v. 1.3.1. http://java.sun.com/j2se/1.3/docs/api/

[10] The Java Developers Almanac 1.4. http://javaalmanac.com/index.html

[11] Bruce Eckel *Thinking in Java, second edition, revision 10.* http://penserenjava.free.fr/pens/sommaire.htm

[12] Evolution d'avancement d'un projet d'éditeur graphique SVG. http://www.svg.free.fr

[13] Jim Farley *Java distributed computing*, O'Reilly&associates, 1998.

[14] Articles divers sur les avancées du SVG. http://www.xml.com/pub/q/sacresvg

[15] Site d'Elsys. http://www.e-elsys.com

[16] Outils et méthodes pour les traitement d'images par ordinateur. http://etudiant.univ-mlv.fr/fruitet/Trait_image/an_imag.html#image_raster

[17] The JavaMail API. http://developer.java.sun.com/developer/onlineTraining/JavaMail/contents.html

[18] Conversion d'éléments AWT à Swing. http://java.sun.com/docs/books/tutorial/uiswing/converting/index.html