Version initiale : Peiqi SHI, création le 7 Mars 2015
Encadrant : Fabien DAGNAT
Partenaires extérieurs : Sylvain Guérin (Openflexo)

# Rapport Technique

## P305 : Interface HTML5 pour outils de modélisation logicielle

Version : 7 Mars 2015
Période d'application : à partir de octobre 2014
Diffusion : Ouvert

# Résumé

Dans le cadre de l'ingénierie logicielle par des modèles, le start-up Openflexo a besoin d'un IHM pour leur outils de modélisation. L' objectif de ce projet est donc de développer une interface HTML5 afin de permettre aux utilisateurs d'utiliser un outil de modélisation dans le navigateur.

Dans ce rapport technique, je commence par décrire le contexte de ce projet. J'explique ensuite d'une manière détaillée l'architecture de framework Vaadin, son caractère et le composant logiciel Diana pour lequel nous avons besoin de développer une interface HTML5. Je présente les problématiques et mes déscriptions techniques dans la troisième partie, Je cite les travaux que j'ai effectué pendant ce projet, À l'issu de cette partie, je montre les challenges que j'ai réussi à surmonter et aussi les missions techniques restent à finir. je conclu mon rapport en citant, l'apport positif de ce projet sur mon expérience en informatique et un bilan personnel du travail.



**Mon clé :** Modélisation, Web développement, Vaadin, Java



# *Abstract*

*In the project of the software engineering with modeles, the start up Openflexo needs a IHM for their modelisations tools. The goal of this project is to develop an interface of HTML5 so that the clients can use this modeling tool in a navigator.*

*In this technical report, I start by describing the contexte of this project. I explain then in a detail manner the architecture of the Vaadin framework, its character and the sofeware composant Diana for which we need to develop an interface HTML5. Je present the problemes and my technical discriptions in the third part, I cite the work I have done during this project, at the end of this part, I indicate all the challenges that I managed to overcome and technical mission remains to finish. I conclude ma report in citing the positive contribution of this project on my experience in computer science and an assessment of the work.*



***Key word :*** *Modeling, Web development, Vaadin, Java*

# Table des matières



# Table des figures



# Liste des tableaux

# 1 Introduction

Dans un monde où les terminaux d'interaction se multiplie (tablettes, smartphones et autres). Même un ingénieur qui modélise n'utilise pas forcément une station de travail chez son employeur. Ainsi, il devient indispensable de fournir des interfaces de modélisation multi-plateforme. Une solution est d'utiliser les standards du web et de réaliser une interface dans un navigateur afin de répondre aux exigences.

La startup Openflexo développe des composants logiciels génériques pour que des entreprises puissent construire leurs propres outils de modélisation par assemblage et configuration de composants. Ainsi, ils ont développé des composants permettant de gérer des diagrammes de classe UML ou de lire et sauvegarder des données dans différents formats standards (tableur, XML, modèle au format eclipse, . . .).

L'un des ces composants, DIANA, permet de modéliser les classes de client (les objets abstraits) et de synchroniser les classes java orignaux et leurs modèles graphiques. Il permet aussi d'enregistrer les propriétés statiques et dynamiques des modèles graphiques. Pour l'instant, ce composant ne possède qu'une réalisation en Java Swing et l'objectif de ce projet est de débuter le portage de ses fonctionnalités en HTML5. Ainsi, il sera possible pour le client d'utiliser une IHM dans un navigateur.

Plus concrètement, nous travaillerons sur la construction d'un éditeur simple de diagramme de classe UML. Le framework qu'on utilise est Vaadin

# 2 État de l'art

## 2.1 Vaadin

Parmi les frameworks de développement Web AJAX, Vaadin tient une place particulière en raison d'un choix d'architecture qui le place entre les frameworks Web traditionnels et les frameworks RIA.

Avec les frameworks utilisés dans les architectures Web traditionnelles : JSP, Struts, Spring MVC, la page HTML est construite sur le serveur pour être interprétée et affichée par le navigateur client. Cette approche a évolué avec la standardisation des échanges asynchrones client / serveur et des opérations de transformation du document HTML, avec pour résultat la possibilité d'interagir côté client sur les pages HTML générées avec de nombreux frameworks AJAX tels que Jquery.

Le RIA est l'étape ultime de cette évolution où le document HTML généré par le serveur est réduit à sa plus simple expression pour être enrichi par le navigateur client jusqu'à atteindre sa forme finale. L'essentiel de la logique de présentation (apparence, contrôle et cinématique) est ici hébergée par le client, avec pour conséquence une régression de la sécurité applicative qui va avec puisque le code JavaScript est entièrement lisible sur le navigateuR. GWT est un framework qui illustre parfaitement cette logique. Cette remarque ne vaut pas pour des frameworks RIA comme Flex ou JavaFX qui n'utilisent pas JavaScript et nécessitent des plugins sur les navigateurs pour le rendu graphique, ce qui pose d'autres contraintes.

Une architecture orientée serveur associée à un modèle de composants clients AJAX est utilisée pour simplifier la programmation et augmenter la sécurité des applications Web.

Au niveau du support du navigateur, Vaadin compte sur le soutien de Google Web Toolkit pour une large gamme de navigateurs, le développeur donc n'a pas besoin de prendre en considération des problèmes de compatibilité du navigateur et du type de terminal.

## 2.2 L'architecture de Vaadin

Vaadin offre deux modèles de développement pour les applications web : côté client et côté serveur. le développement du côté serveur est plus puissant, permettant le développement d'applications uniquement sur le côté serveur, en utilisant un moteur du côté client basé sur AJAX. Le modèle du côté client permet de développer des Widgets et des applications en Java, qui seront compilées en JavaScript et exécutées dans un navigateur. les deux modèles peuvent partager dans une même application.

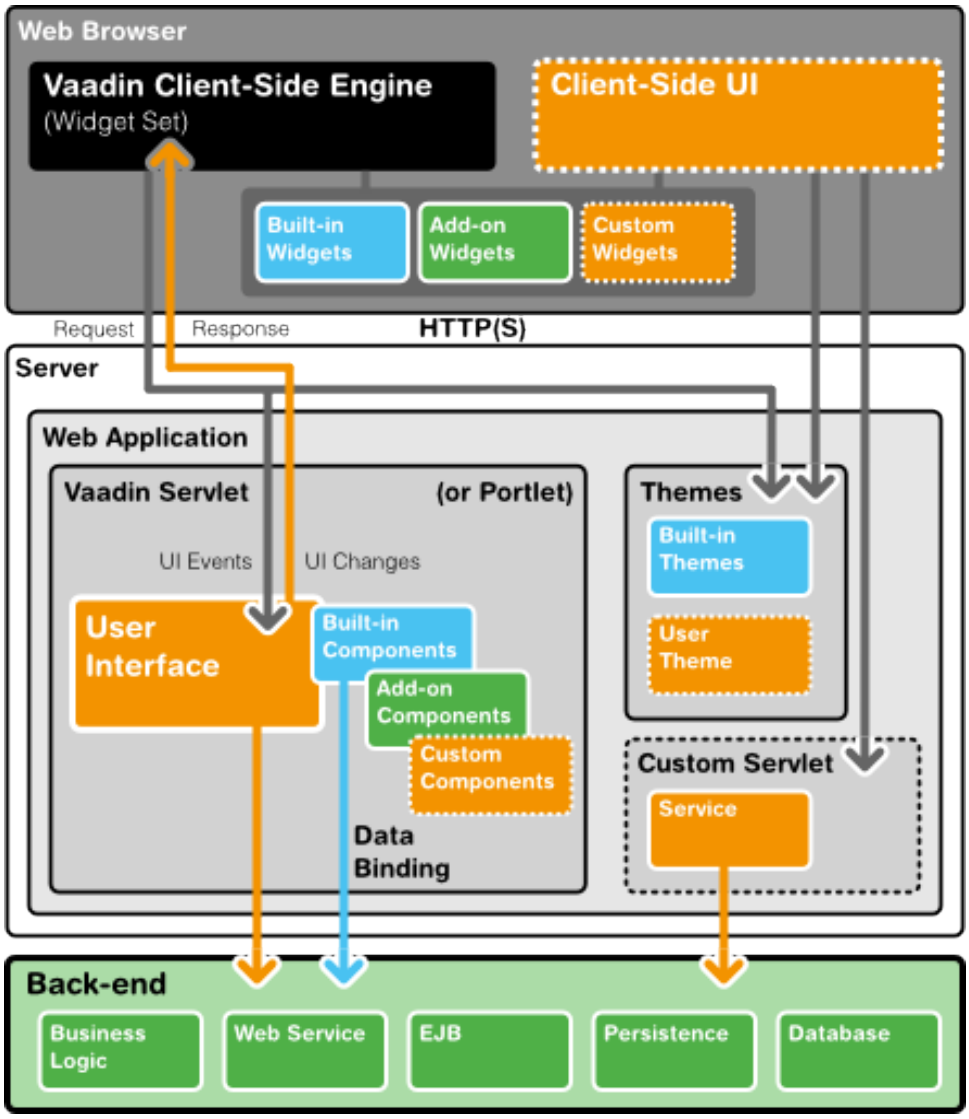


**Figure 1** – Architecture générale de framework Vaadin

## 2.3 Paradigme MVC

Le patron modèle-vue-contrôleur est un modèle destiné à répondre aux besoins des applications interactives en séparant les problématiques liées aux différents composants au sein de leur architecture respective.

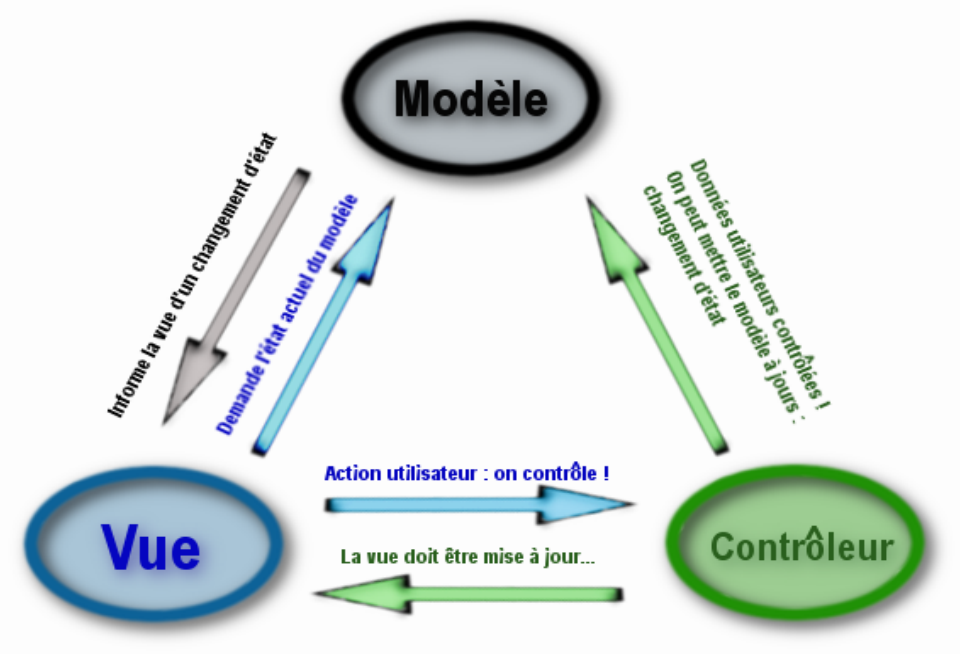


**Figure 2** – paradigme mvc

Ce paradigme regroupe les fonctions nécessaires en trois catégories :

1. un modèle (modèle de données) ;
   Le modèle représente le cur (algorithmique) de l'application : traitements des données, interactions avec la base de données, etc. Il décrit les données manipulées par l'application. Il regroupe la gestion de ces données et est responsable de leur intégrité.
2. une vue (présentation, interface utilisateur) ;
   Ce avec quoi l'utilisateur interagit se nomme précisément la vue. Sa première tâche est de présenter les résultats renvoyés par le modèle. Sa seconde tâche est de recevoir toute action de l'utilisateur.
3. un contrôleur (logique de contrôle, gestion des événements, synchronisation).
   Le contrôleur prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle et les synchroniser. Il reçoit tous les événements de la vue et enclenche les actions à effectuer. Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle afin que les données affichées se mettent à jour.

## 2.4 L'architecture de DIANA

DIANA prend son originalité en mesure d'associer un graphe d'objets interconnectés avec des représentations graphiques adaptés. Les représentations graphiques sont considérés comme des propriétés graphiques tels que la couleur ou la taille, qui sont ensuite associés à des modèles en fonction du contexte. Une mise en oeuvre Java Swing de Diana a été développé pour Openflexo.

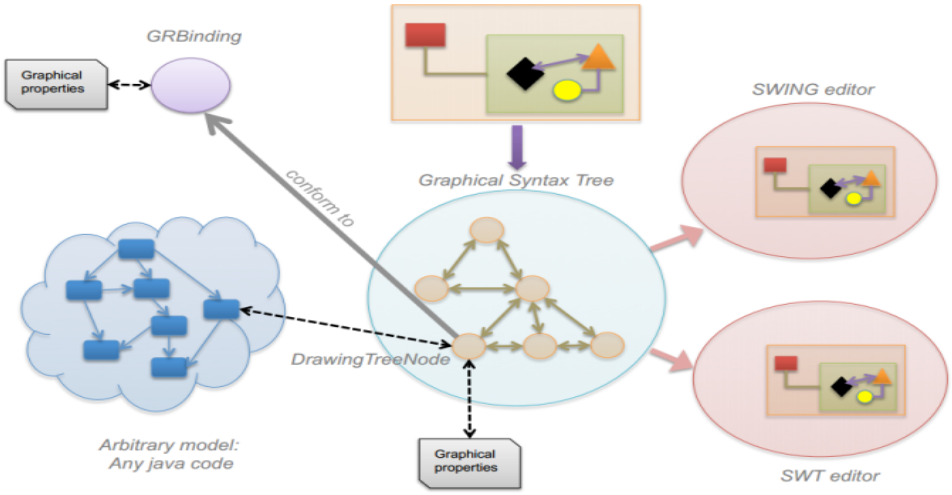


**Figure 3** – L'infrastructure de DIANA

**Arbitrary model :** c'est les codes de notre client. L'objectif de notre travail est de créer des modèles graphiques pour ces codes afin d'éditer ces objets abstraits plus facilement.

**Graphical Syntax Tree :** c'est l'endroit où on conserve les "DrawingTreeNode", c'est la couche modèle pour notre application.

**DrawingTreeNode :** ils sont associés aux modèles graphiques dans l'interface utilisateur. Chaque DrawingTreeNode a deux références (pointeur) pour le "GraphicalBinding" et "Abitrary model". Les modifications des propriétés des modèles graphiques peuvent donc être enregistrés en "GraphicalBinding" correspondant et en "Abitrary model" correspondant.

**Graphical properties :** ce sont des propriétés statiques de modèle graphique qu'on n'a pas besoin de synchroniser avec "Abitrary model". Il y a un autre type de propriétés, propriétés dynamiques sont des propriétés qu'on a besoin de mise à jour dans le "Abitrary model". Les propriétés statiques et dynamiques d'un modèle graphique sont tous conservés dans un "GraphicalBinding" correspondant.

**SWING éditeur ou SWT éditeur :** ce sont des interfaces graphiques permettent d'interagir avec les utilisateurs. Notre objectif est de créer un prototype simple d'éditeur Vaadin.

# 3 Description de la partie technique

## 3.1 Conception et l'architecture générale

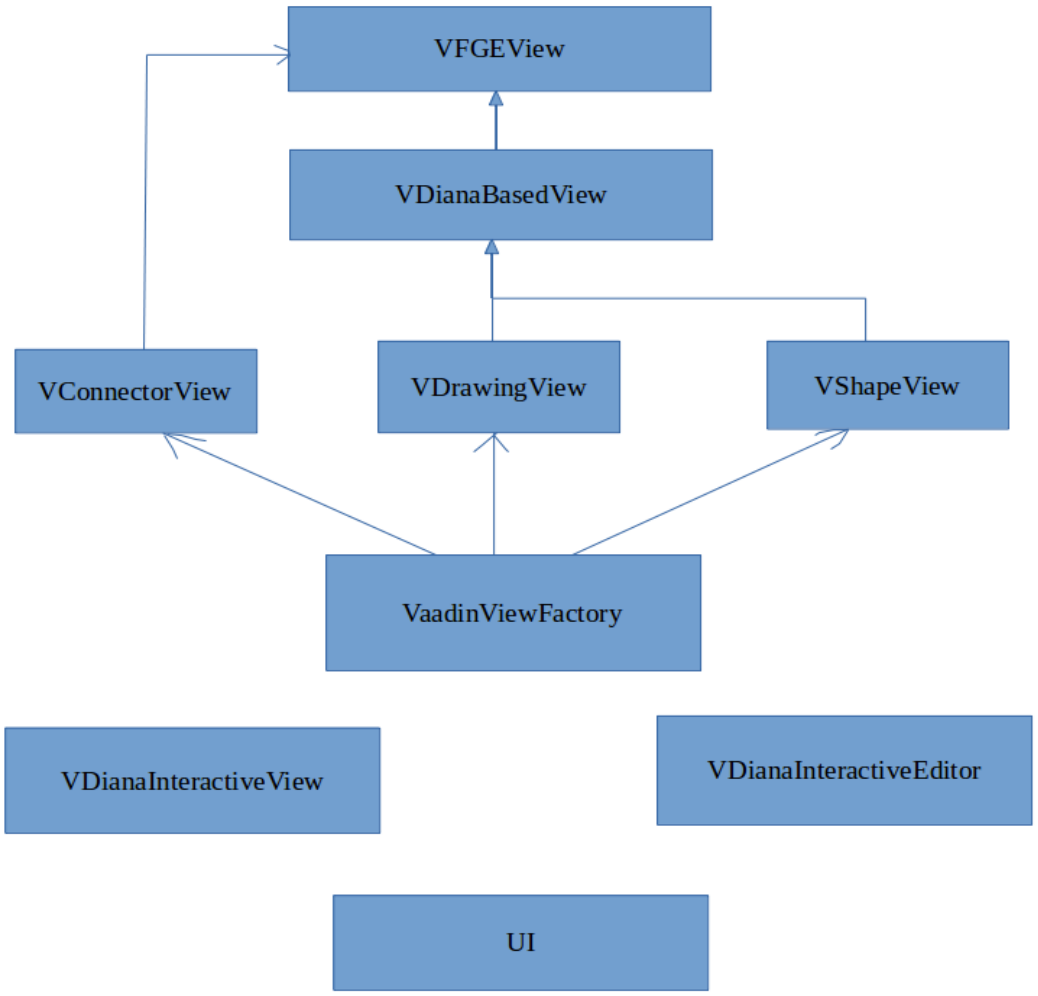


**Figure 4** – L'architecture du projet

**Coté serveur :**

nous créons d'abord une classe d'interface : VFGEView, qui représente tous les signatures d'une view.

une classe abstracte de base qui hérite de AbstractComponent et implémente l'interface VFGEView, dans cette classe j'ai implémenté les méthods communs pour VdrawingView et VshapeView

les classes : VdrawingView, VshapeView héritent de la classe abstracte et utilise la meme valeur de rpc pour connecter et éditeur sur le coté client. VdrawingView est pour rendre le View étirable qui correspond le rootNode de Drawing. Dans la class VdrawingView, nous implémentons aussi les interactions de clicklistener et pressedlistener qui

viennent du coté client afin de rendre le view édirable. VshapeView est pour rendre le view des nodes de fils de Drawing.

Pour les Views, nous créons des classes Graphics qui font des fonctionnalités pour dessiner les views. Donc c'est les classes graphcis qui est responsable pour piloter au client et nous devons donc passer la réference de rpc aux classes graphics, car rpc est un attribut de class AbstractComponent.

**Coté client :**

nous déclarons d'abord des methods qui sont déclarés dans VdianaGraphics qui hérite de FGEGraphicsImpl dans la classe VdrawingViewClientRpc et implémentons ces methods dans la class ClientConnector. Ensuite, dans le VdianaGraphics, nous construisons ces fonctionnalités.

De l'autre coté, la class Client Connector, nous ajoutons les listeners afin d'écouter les commands.

## 3.2 Développement

### UML1 : L'interaction client et serveur

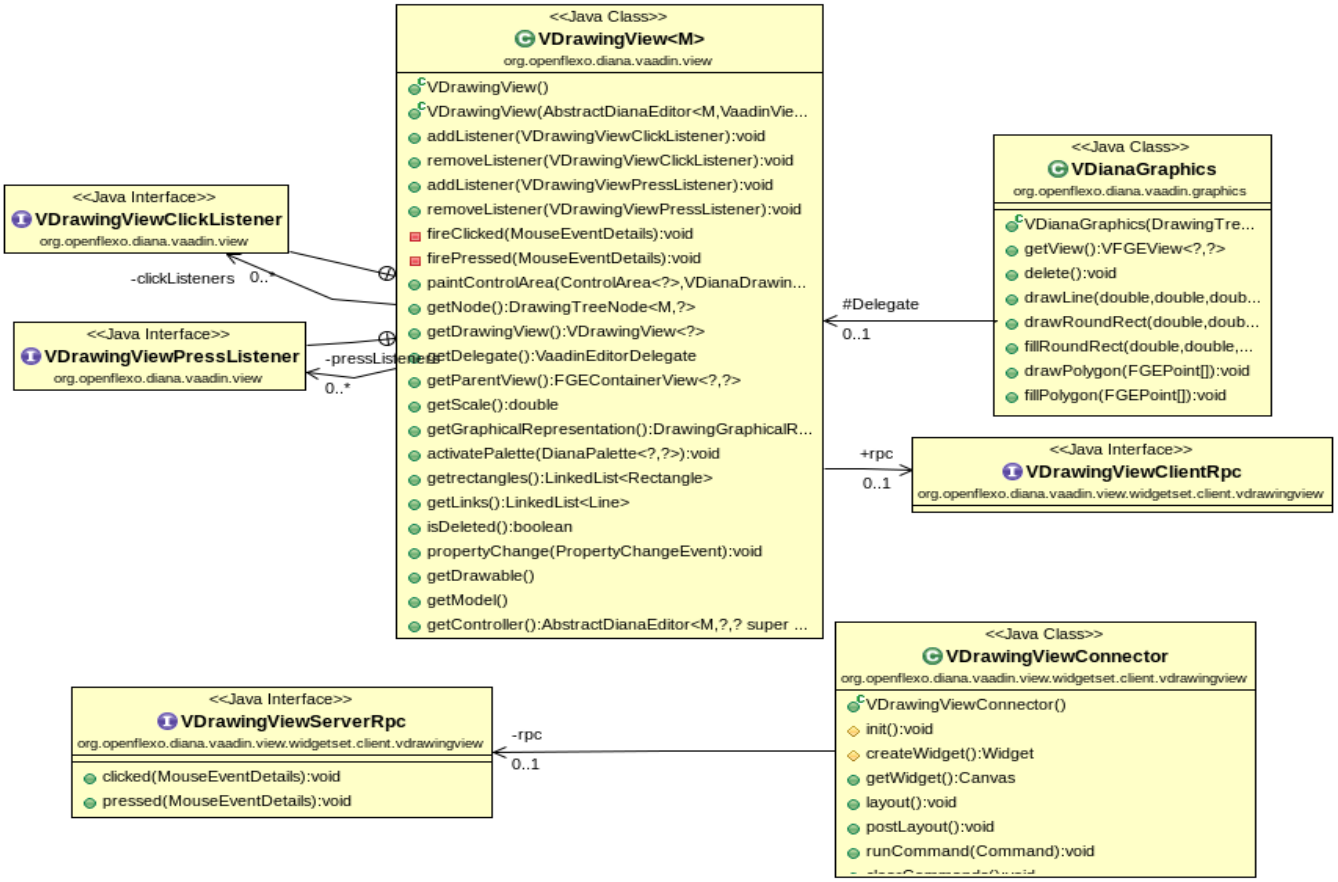


**Figure 5** – l'interaction client et serveur

L'interaction se fait sur le composant drawingView : les méthods d'appele rpc sont implémentées dans la classe drawingGraphics qui est une délégate de drawingView. Les listeners sont implémentés directement dans la classe drawingView.

Dans la classe drawingViewConnector, nous avons implémentées les méthods d'action et écoutés les actions d'utilisateur, les manières d'écoute sont déclarées dans la classe VDrawingViewServerRpc.

### UML2 : Conception du côté serveur

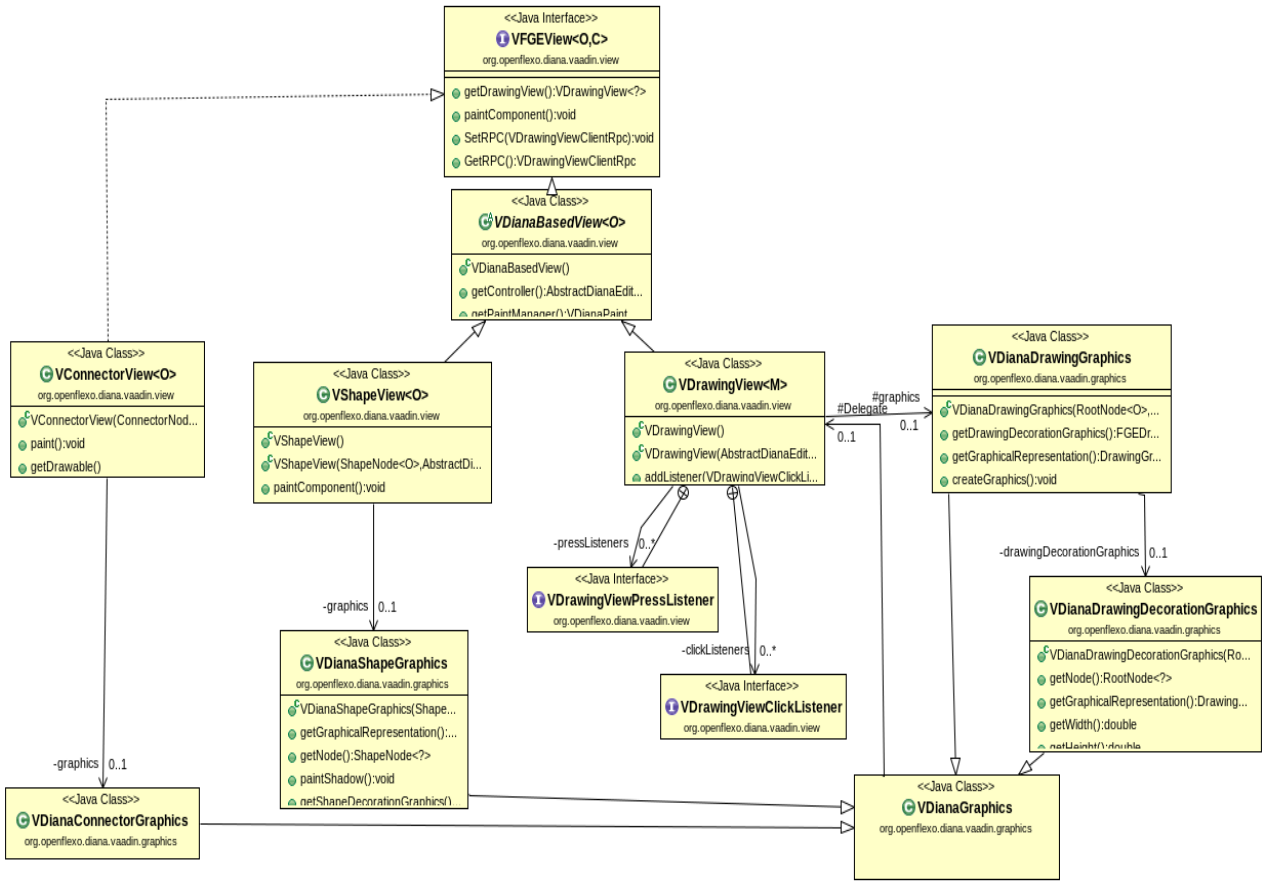


**Figure 6** – l'implémentation du côté serveur

L'architecture générale de côté serveur, la classe VDrawingView et la classe VDiana-Graphics sont deux classes fondamentales.

L'une des difficultés sur ce projet est la synchronisation de rpc entre les views différents. Pour répondre à ce problème, nous avons d'abord déclarés deux méthods dans l'interface VFGEView setRPC et getRPC. Ensuite, dans la classe drawingView, en utilisant la propriété de polymorphisme, nous avons réussi à appeler le SetRPC correctement pour créer proprement le view de chaque node.

## 3.3 Méthodologie de test


**Figure 7** – Méthodologie de test

Afin de rendre des models en graphcis : il faut d'abord tester si des models sont bien fabriqués et se rendre en Drawing, pour cela, nous avons ajoué trois models dans la classe UI. Pour valider la partie entre model et drawing. Ensuite, nous avons besoin de tester si le serveur peut bien interagir avec le client. Pour cela, j'ai implémenté une classe VdianapaintManager et quatre buttons pour valider la partie entre View, Graphics et ClientConnector. Après, nous avons besoin de tester si chaque DrawingTreeNode correspond bien son propre view par générer les traces dessous :

child=Shape-0[312.761599817222 ;222.13812270039085][20.0x20.0][FGEEllips : (0.0,0.0 ,1.0,1.0 type=PIE)] :GraphNode[node1](312.761599817222,222.13812270039085) view=or g.openflexo.diana.vaadin.view.VShapeView@15407ecxxxxxxxxxchild=Shape-1[149.0569224 55735 ;22.791736904618443][20.0x20.0][FGEEllips : (0.0,0.0,1.0,1.0 type=PIE)] :Graph Node[node2](149.05692245957735,22.791736904618443) view=org.openflexo.diana.vaad in.view.VShapeView@18b2573xxxxxxxxchild=Shape-2[379.3682150199206 ;461.3348685443 8195][20.0x20.0][FGEEllips : (0.0,0.0,1.0,1.0 type=PIE)] :GraphNode[node3](379.368 2150199206,461.33486854438195) view=org.openflexo.diana.vaadin.view.VShapeView@5 9213 xxxxxxxxchild=ConnectorImpl-3[Shape-0][Shape-1] :org.openflexo.diana.vaadin. tests.TestEdge@10dccc2 view=org.openflexo.diana.vaadin.view.VConnectorView@b0eb c4ccccccccccchild=ConnectorImpl-4[Shape-0][Shape-2] :org.openflexo.diana.vaadin. tests.TestEdge@fdfb0c view=org.openflexo.diana.vaadin.view.VConnectorView@78d726 ccccccccchild=ConnectorImpl-5[Shape-2][Shape-1] :org.openflexo.diana.vaadin.test s.TestEdge@6d0ddc view=org.openflexo.diana.vaadin.view.VConnectorView@1753145cc ccccccc

pour paint chaque view proprement, nous avons besoin de tester si les shapeNode et ConnectorNode et leur views correspondants sont bien fabriqué et si drawingGraphics, shapeGraphics et connectorGraphics sont correctement appelés.

Reste à tester :

le rpc de shapeGraphics partage la même valeur de drawingGraphics afin de paint les shapeNode sur le drawing. Et si le connectorGraphics peut être bien appelés pour chaque ConnectorViewforNode. Le rpc de connectorView est un rpc de Panel, nous avons besoin de tester si les RPC de connectorView et shapeView avec drawingView dans la class drawingView sont synchroniser avant d'appeler la méthode paintComponent.

# 4 Conclusion

Ce projet était pour moi très formateur, j'ai été bien suivi tout au long de mon projet individuel et mon encadreur m'ont fourni toutes les réponses fonctionelles et techniques nécessaires à mon projet.

Le framework Vaadin était tout nouveau pour moi et il était aussi le premier framework Java que j'ai appris. L'intêret de ce framework est qu'il permet de générer le code javascript automatiquement en programmer en java. J'ai passé deux mois pour comprendre l'architecture et le principe de ce framework à travers d'étudier et améliorer un projet implémenté purement en java vaadin.

C'était aussi le premier projet en front-end que j'ai abordé. Un domain assez émergent surtout après la popularité de HTML5. Le pattern MVC est un paradigme populaire en front-end et il est utilisé dans ce projet à fin de répondre aux besoins d'interactions et de séparer proprement les fonctionnalités de chaque composant.

Le composant Diana est un composant complexe, j'ai passé beaucoup de temps à étudier son architecture et les technologies qu'il utilise. L'implémentation d'une interface pour ce composant me demande de programmer dans un design pattern complexe. Comment concevoir et utiliser les classes génériques et comment utiliser des technologies de programmation orienté objet comme polymorphisme. Cette parite améliore fortement mes compétence de programmation en java.

Mise à part la partie technique, j'ai acquis la capacité danalyser les problèmes et concevoir des solutions techniques et j'ai appris aussi à gérer mon stress et la pression d'accomplir mes tâches en parallèle de réussir mes autres cursus.

Enfin, comme c'est mon dernier projet scolaire et mon projet professionnel est de devenir un ingénieur systèmes et réseaux, un titre demande vraiment des connaissances et compétences en informatique et réseaux à la fois très étendus et profondu. Grâce à ce projet, j'ai au la chance de compléter ma vision dans le développement de front-end et cela me permet d'avoir une vision plus concrète pour le monde d'informatique et aussi d'acquérir la capacité de développer en front-end dans mon future carrier. Je voudrais donc adresser tout mes remerciement à mon encadreur Fabien Dagnat et le partenaire extérieur Monsieur Sylvain Guérin, sans qui, je ne pourrais jamais réussir à apprendre tellement de chose dans un semestre.

# Références


**BIBLIOGRAPHIE**

Vaadin 7 UI design by Example — Alejandro Fraude

Learning Vaadin 7,Seconde Edition — Nicola Frankel

Vaadin 7 cookbook — Jaroslav Holan , Ondrej Kvasnovsky

**SITES INTERNET CONSULTES**

Thinking U and I — https ://vaadin.com/home/

Site de Openflexo — https ://openflexo.org/tiki/Welcome+to+Openflexo

www.telecom-bretagne.eu

**Campus de Brest**
Technople Brest-Iroise
CS 83818
29238 Brest Cedex 3
France
Tl : + 33 (0)2 29 00 11 11
Fax : + 33 (0)2 29 00 10 00

**Campus de Rennes**
2, rue de la Chtaigneraie
CS 17607
35576 Cesson Svign Cedex
France
Tl : + 33 (0)2 99 12 70 00
Fax : + 33 (0)2 99 12 70 19

**Campus de Toulouse**
10, avenue Édouard Belin
BP 44004
31028 Toulouse Cedex 04
France
Tl : + 33 (0)5 61 33 83 65
Fax : + 33 (0)5 61 33 83 75