

**Département de génie logiciel et des T.I.**

Rapport de Laboratoire

|  |  |
| --- | --- |
| **Numéro du laboratoire** | 04 |
| **Nom du laboratoire** | Projet de session |
| **Étudiant(s)** | Jean-Simon Bonin  Xavier Duval  Martin Gingras |
| **Code(s) permanent(s)** | BONJ02098701  DUVX20048607  GINM10108708 |
| **Cours** | LOG 430 |
| **Session** | Hiver 2012 |
| **Groupe** | 01 |
| **Chargé(e) de laboratoire** | Samir Djeffal |
| **Date** | 16 avril 2012 |

Table des matières

[Introduction 5](#_Toc322319436)

[Développement 6](#_Toc322319437)

[*Rappel de la nature et la mission commerciale du système* 6](#_Toc322319438)

[*Présentation de l’architecture et des approches architecturales* 6](#_Toc322319439)

[*Vues architecturales* 7](#_Toc322319440)

[*Analyse des approches architecturales* 11](#_Toc322319441)

[*Arbre d’utilité* 12](#_Toc322319442)

[*Scénarios de qualité* 14](#_Toc322319443)

[*Tableau des risques* 17](#_Toc322319444)

[*Tableau des non-risques* 17](#_Toc322319445)

[*Tableau des sensibilités* 18](#_Toc322319446)

[*Tableau des compromis* 18](#_Toc322319447)

[Discussion 19](#_Toc322319448)

[Conclusion 20](#_Toc322319449)

**Liste des tableaux**

Tableau 1 : Description du module libJAUS

Tableau 2 : Description du module LiveUV.Persistence

Tableau 3 : Description du module LiveUV.Controller.WidgetManager

Tableau 4 : Description du module LiveUV.Controller.PerspectiveManager

Tableau 5 : Description du module LiveUV.Controller.AlerteManager

Tableau 6 : Priorités et importances des scénarios de l’arbre d’utilité

Tableau 7 : Détails du scénario U1

Tableau 8 : Détails du scénario M4

Tableau 9 : Détails du scénario M1

Tableau 10 : Détails du scénario U2

Tableau 11 : Détails du scénario M2

Tableau 12 : Détails du scénario I2

Tableau 13 : Risques

Tableau 14 : Non-risques

Tableau 15 : Sensibilité

Tableau 16 : Compromis

**Liste des figures**

**No table of figures entries found.**

# Introduction

Ce laboratoire a pour objectif de nous permettre d’effectuer la conception d’une architecture d’un système en se basant sur un document de vision, un SRS et des cas d’utilisation tout en suivant l’ensemble des méthodologies vues dans le cours d’architecture logicielle. La réalisation des exigences de ce laboratoire mettra en pratique ces diverses méthodologies et servira de récapitulatif complet pour ce cours.

La suite de ce rapport contient trois sections. D’abord, la section principale du document, le développement, contient l’ensemble de notre conception architecturale. La deuxième section est une discussion comparant les quatre laboratoires sur lesquels nous avons eu à travailler cette session et faisant état de notre point de vue sur ce quatrième laboratoire. Une conclusion vient clore le document en faisant un retour sur les objectifs du laboratoire.

# Développement

## *Rappel de la nature et la mission commerciale du système*

Le système est un client de télémétrie qui communique par l’entremise du protocole JAUS et qui gère un ensemble de widgets, perspectives et alertes. Le système permet de créer des widgets qui présenteront les données acquis par les différents senseurs externes au client qui seront interrogés à l’aide de la librairie libJAUS. De plus, un ensemble de widgets dans une configuration donnée peut être enregistré sous forme de perspective. Il y a donc des perspectives qui peuvent être créé pour chaque club étudiant ou utilisateur du client. Le système permet aussi la création d’alertes sur certaines valeurs des senseurs.

La mission du système est de profiter non seulement pour le club étudiant SONIA mais aussi pour tout autre club étudiant pour autant qu’il respecte le protocole de JAUS. De plus, le système doit être multiplateforme et être développé en JAVA.

L’architecture de notre système est déterminée par certains attributs de qualité plus dominants soit la maintenabilité, l’interopérabilité, l’utilisabilité et la portabilité.

## *Présentation de l’architecture et des approches architecturales*

Pour ce qui est de l’approche architecturale, nous avons utilisé certaines tactiques afin de bien soutenir nos attributs de qualités.

**Maintenabilité**

L’architecture proposée utilise le patron architecturale MVC afin de favoriser la maintenabilité du système. Il y a donc séparation de la logique d’affaire des données et des éléments lié à la vue du client. Dans l’architecture, le seul point d’interaction avec la librairie ce produit dans le module de « persistence » soit dans la classe « SensorsData ». De cette façon il sera possible de facilement remplacer la librairie puisqu’il n’y aura qu’un seul point de contact avec celle-ci à vérifier.

**Utilisabilité**

Pour l’utilisabilité, on permet à l’utilisateur de créer ses propres perspectives et de les enregistrés.

**Portabilité**

Pour s’assurer de profiter d’une grande portabilité et de rendre le système compatible pour plusieurs types de véhicules autonomes, on se sert d’une seule interface afin de communiquer avec les capteurs et périphériques externe soit la librairie JAUS.

**Interopérabilité**

## *Vues architecturales*

Vue de type modules - décomposition:

****

|  |  |
| --- | --- |
| **Module** | LibJAUS |
| **Responsabilités** | Ce module est responsable de la communication entre les capteurs et les autres modules du système. |
| **Cas d’utilisations supporté** |  |
| **Interfaces** | Interface qui fournit des services qui permet d’obtenir les valeurs des différents capteurs et périphériques externe. |
| **Scénarios supporté** |  |
| **Contraintes** | Communication à l’aide du protocole JAUS |
| **Données produit** | Retourne la valeur du capteur selon la requête. |
| **Données requis** | Doit connaître le capteur voulu et le type de valeur demandé. |

*Tableau 1*

|  |  |
| --- | --- |
| **Module** | LiveUV.Persistence |
| **Responsabilités** | Ce module est responsable de conserver les données des capteurs acquérir par l’intermédiaire de la librairie JAUS et de conserver les préférences des utilisateurs du client. |
| **Cas d’utilisations supporté** |  |
| **Interfaces** | Fournit le service pour accéder aux données |
| **Scénarios supporté** | U1, U5, I3, P2 |
| **Contraintes** | Utilisation des librairies de communication JAUS (module libJAUS), Langage de programmation JAVA. |
| **Données produit** | Ce module produit les données pour le module LiveUV.Controller |
| **Données requis** | Ce module requiert la librairie JAUS afin de communiquer avec les capteurs pour avoir les données. |

*Tableau 2*

|  |  |
| --- | --- |
| **Module** | LiveUV.Controller.WidgetManager |
| **Responsabilités** | Ce module est responsable de gérer les widget du client de télémétrie. |
| **Cas d’utilisations supporté** | CU01, CU03 |
| **Interfaces** | Fournit les services pour gérer les widgets. |
| **Scénarios supporté** | M1, M2, M3, U2, I3, P1 |
| **Contraintes** | Langage de programmation JAVA. |
| **Données produit** | Ce module produit les widgets à présenter dans le module LiveUV.View |
| **Données requis** | Ce module à besoin des données du module LiveUV.Persistence. |

*Tableau 3*

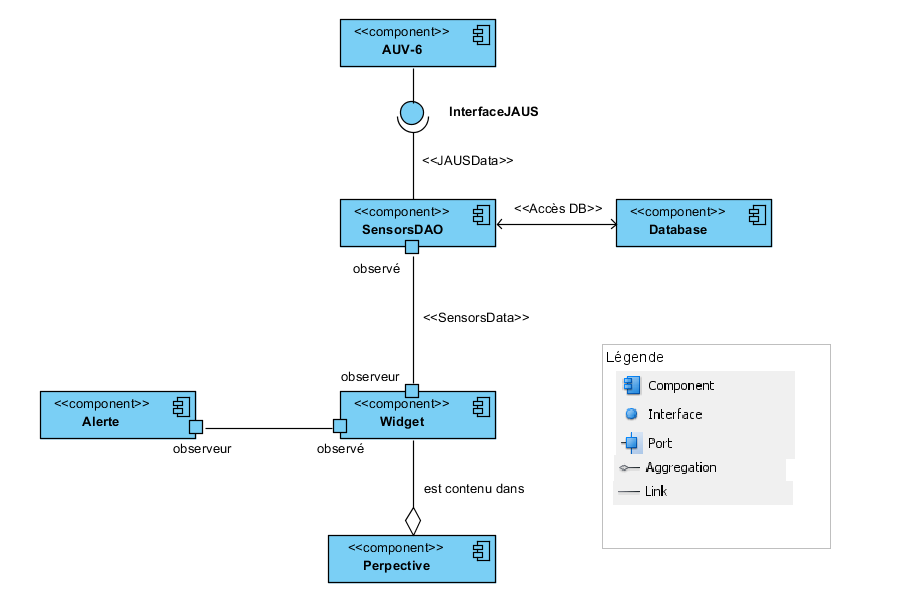
|  |  |
| --- | --- |
| **Module** | LiveUV.Controller.PerspectiveManager |
| **Responsabilités** | Ce module est responsable de gérer les perspectives, soit l’ensemble des widgets et leur positionnement dans l’application. |
| **Cas d’utilisations supporté** | CU02, CU05, CU06, CU07, CU08, CU09. CU12 |
| **Interfaces** | Fournit les services pour gérer les perspectives. |
| **Scénarios supporté** | I3 |
| **Contraintes** | Langage de programmation JAVA. |
| **Données produit** | Ce module produit les perspectives à présenter dans le module LiveUV.View |
| **Données requis** | Ce module a besoin des données du module LiveUV.Persistence. |

*Tableau 4*

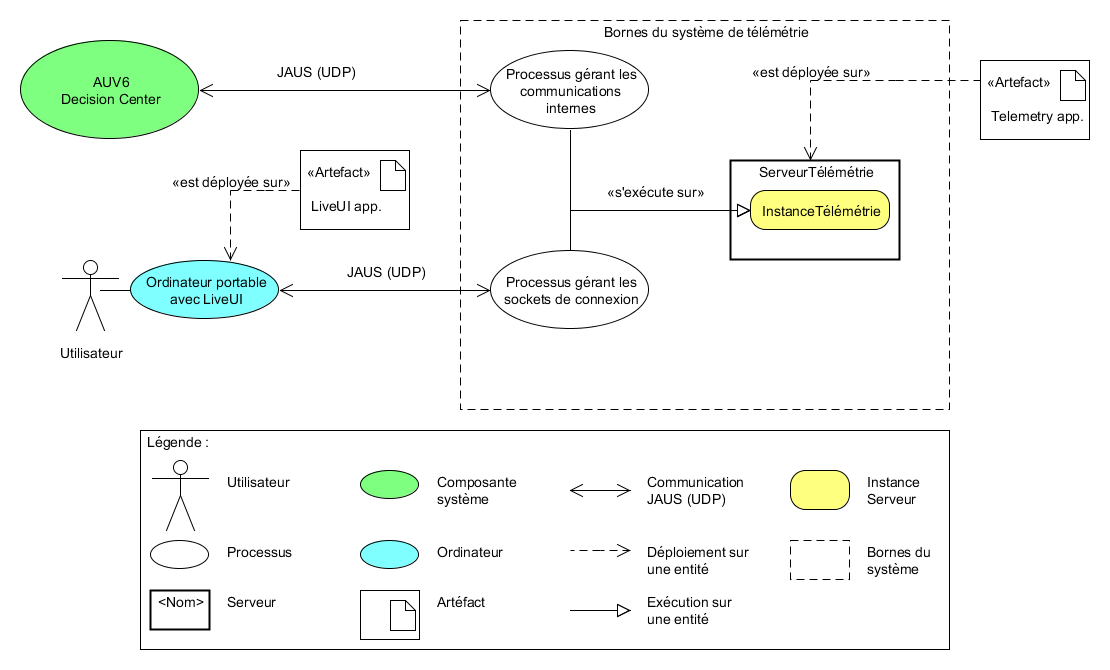
|  |  |
| --- | --- |
| **Module** | LiveUV.Controller.AlerteManager |
| **Responsabilités** | Ce module est responsable de gérer les alertes du client de télémétrie. |
| **Cas d’utilisations supporté** | CU10 |
| **Interfaces** | Fournit le service pour créer des alertes. |
| **Scénarios supporté** | I3 |
| **Contraintes** | Langage de programmation JAVA. |
| **Données produit** | Ce module produit les alertes à présenter dans le module LiveUV.View au moment de leur déclenchement. |
| **Données requis** | Ce module a besoin des données du module LiveUV.Persistence. |

*Tableau 5*

Vue de type component & connector



Vue de type allocation : Déploiement



Étant donné que l’application à concevoir et implémenter est presqu’en totalité une application d’interface graphique cliente se connectant au système de télémétrie du véhicule, il y avait peu d’éléments à introduire pour une vue de type allocation. La vue de déploiement ci-dessus montre le contexte d’utilisation et de déploiement du projet LiveUV.

L’application LiveUV est déployée sur les ordinateurs des utilisateurs qui prévoient avoir besoin de communiquer avec le système de télémétrie du véhicule. L’application se connecte via le protocole JAUS sur UDP au système de télémétrie pour obtenir les valeurs des diverses propriétés de la télémétrie qui intéressent l’utilisateur. Ces valeurs s’intègrent aux divers widgets faisant partie de la perspective utilisée par l’usager.

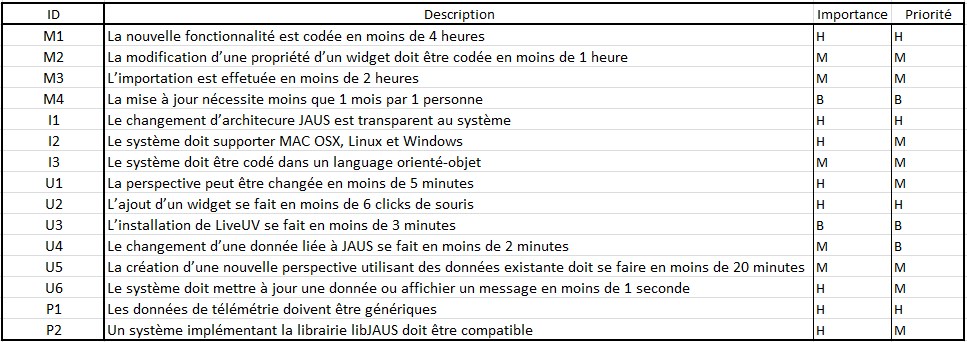
## *Analyse des approches architecturales*

Le style architectural prédominent dans l’application est l’architecture en couche. En utilisant ce style, on permet d’augmenter grandement la maintenabilité du code grâce à la grande modularité de l’architecture.

Ce style favorise aussi l’interopérabilité de l’application car un système externe doit seulement connaître une couche de l’application pour s’interfacer avec le logiciel.

## *Arbre d’utilité*

****

 *Tableau 6*

Pour faire les choix de l’importance et de la priorité de chaque nous nous sommes basé sur les priorités des différentes fonctionnalités dans le document de vision ainsi que dans le SRS. De plus nous avons priorisé attributs de qualités les plus mis en valeur dans les documents : la maintenabilité et l’utilisabilité.

## *Scénarios de qualité*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scénario #** | U1 – La perspective peut être changée en moins de 5 minutes | | | |
| **Attribut** | Utilisabilité | | | |
| **Environnement** | Run time | | | |
| **Stimulus** | Souhaite modifier la perspective actuelle des widgets dans le client | | | |
| **Réponse** | La perspective est changée en moins de 5 minutes | | | |
| **Décision architectural** | | **Risque / Non-risque** | **Sensibilité** | **Compromis** |
| Perspective sont générique | | NR1 |  |  |
| Séparation des éléments de vue de la logique d’affaire | | NR2 |  | C1 |
| **Raisonnement** | | Les perspectives sont génériques pour permettre de créer des types de perspectives spécifiques à un club étudiant ou à un utilisateur. Une modification à la perspective n’affecte en rien la logique d’affaire. | | |
| **Diagramme architectural** | | Voir diagramme de la vue de décomposition | | |

*Tableau 7*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scénario #** | M4 – La mise à jour de l’interface libJAUS nécessite moins que 1 mois par 1 personne | | | |
| **Attribut** | Maintenabilité | | | |
| **Environnement** | Fonctionnement dégradé | | | |
| **Stimulus** | Besoin de mettre à jour la librairie JAUS | | | |
| **Réponse** | Les autres modules ne dépendant pas de la librairie demeurent inaffectés | | | |
| **Décision architectural** | | **Risque / Non-risque** | **Sensibilité** | **Compromis** |
| Architecture modulaire | | NR1 |  |  |
| Interface unique utilisant la librairie | |  |  | C1 |
| **Raisonnement** | | L’architecture modulaire du système et l’interface unique qui utilise la librairie permet de faciliter la maintenabilité de ce module puisque sa modification n’affecte qu’un seul endroit dans le système. | | |
| **Diagramme architectural** | | Voir diagramme de la vue de décomposition | | |

*Tableau 8*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scénario #** | M1 – La nouvelle fonctionnalité est codée en moins de 4 heures | | | |
| **Attribut** | Maintenabilité | | | |
| **Environnement** | Mode d’opération normal | | | |
| **Stimulus** | Le programmeur veut ajouter une nouvelle fonctionnalité | | | |
| **Réponse** | La fonctionnalité est codée et fonctionne adéquatement | | | |
| **Décision architectural** | | **Risque / Non-risque** | **Sensibilité** | **Compromis** |
| Architecture en couche | | R1 | S1 | C1 |
| Programmation modulaire en Java | | NR1 |  |  |
| Séparation de la persistance des données et des contrôleurs | | R2 |  | C2 |
| **Raisonnement** | | Pour faciliter l’ajout de nouvelles fonctionnalités, le système doit être le plus possible extensible et modulaire. Dans l’architecture du système, la séparation des éléments logique de code est un moyen indispensable à la maintenabilité. | | |
| **Diagramme architectural** | |  | | |

*Tableau 9*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scénario #** | U2 – L’ajout d’un widget se fait en moins de 6 clics de souris | | | |
| **Attribut** | Utilisabilité | | | |
| **Environnement** | Mode d’opération normal | | | |
| **Stimulus** | Le programmeur veut ajouter une widget | | | |
| **Réponse** | Le widget est ajouté sans problème | | | |
| **Décision architectural** | | **Risque / Non-risque** | **Sensibilité** | **Compromis** |
| Conception simple de l’interface. | | R1 | S1 | C1 |
| Module de gestion des widgets. | | NR1 |  |  |
|  | |  |  |  |
| **Raisonnement** | | Pour que l’ajout d’un widget soit le plus simple et efficace possible, il faut que l’interface soit simple et que la gestion widgets dans le code soit bien implémentée. | | |
| **Diagramme architectural** | |  | | |

*Tableau 10*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scénario #** | M2 – La modification d’une propriété d’un widget doit être codée en moins de 1 heure | | | |
| **Attribut** | Maintenabilité | | | |
| **Environnement** | Mode d’opération normal | | | |
| **Stimulus** | Le programmeur veut modifier une propriété d’un widget | | | |
| **Réponse** | La propriété du widget est modifiée sans problème | | | |
| **Décision architectural** | | **Risque / Non-risque** | **Sensibilité** | **Compromis** |
| Architecture en couche | |  |  |  |
| Programmation modulaire en Java | | NR1 |  |  |
| **Raisonnement** | | Pour faciliter la modification des propriétés des widgets et des éléments de code en général, le système doit être le plus possible extensible et modulaire. Dans l’architecture du système, la séparation des éléments logique de code est un moyen indispensable à la maintenabilité. | | |
| **Diagramme architectural** | |  | | |

*Tableau 11*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scénario #** | I2 – Le système doit supporter Mac OS X, Linux et Windows | | | |
| **Attribut** | Interopérabilité | | | |
| **Environnement** | Mode d’opération normal | | | |
| **Stimulus** | Le programmeur veut utiliser l’application sur le système d’exploitation de son choix. | | | |
| **Réponse** | L’ensemble des fonctionnalités de l’application fonctionne normalement | | | |
| **Décision architectural** | | **Risque / Non-risque** | **Sensibilité** | **Compromis** |
| Conception simple de l’interface. | |  | S2 | C3 |
| Programmation modulaire en Java | | NR1 |  |  |
| **Raisonnement** | | Pour assurer le bon fonctionnement de l’application sur les principaux systèmes d’exploitation, le langage de programmation Java est utilisé. Les éléments de l’interface graphique doivent également demeurer simples et utiliser une librairie « Look and Feel » éprouvée. | | |
| **Diagramme architectural** | |  | | |

*Tableau 12*

## *Tableau des risques*

|  |  |  |
| --- | --- | --- |
| **Scénarios** | **Risque** | **Description** |
| M1 | R1 | Dans notre cas, la programmation modulaire en Java est nécessaire pour implémenter le modèle en couche et pour permettre la modularité et l’interopérabilité de l’application. |
| M4 |  |  |
| U1 |  |  |
| U2 | R1 | Le risque d’une conception trop simple de l’interface est de limiter l’utilisation experte des fonctionnalités pour augmenter la rapidité d’utilisation. |
| I2 |  |  |

*Tableau 13*

## *Tableau des non-risques*

|  |  |  |
| --- | --- | --- |
| **Scénarios** | **Non-risque** | **Description** |
| M1 | NR1 |  |
| M2 | NR1 | Une architecture modulaire favorisera la cohérence dans les fonctionnalités du client. L’endroit où apporter ce genre de modifications sera très clairement identifié et reconnaissable. |
| M4 | NR1 | Une architecture modulaire favorisera la cohérence dans les fonctionnalités du client. |
| U1 | NR1 | Avoir les perspectives génériques permet de créer plusieurs types de perspective spécifique pour plusieurs utilisateurs et donc de sauver du temps dans le changement des perspectives en utilisant d’autres configurations existantes. |
| NR2 | Séparer la logique d’affaire des éléments de vue favorisera l’évolutivité du système. |
| U2 | NR1 | Avec l’implémentation d’un module de gestion des widgets, il devient beaucoup plus facile de gérer efficacement l’ajout ou la suppression de widgets dans l’interface. |
| I2 | NR1 | Comme Java fonctionne à l’aide d’une machine virtuelle et que celle-ci est compatible sur tous les systèmes d’exploitation visés par le projet, la programmation en Java est considérée comme un non-risque en ce qui concerne l’interopérabilité. |

*Tableau 14*

## *Tableau des sensibilités*

|  |  |  |
| --- | --- | --- |
| **Scénarios** | **Sensibilités** | **Description** |
| M1 | S1 | Si l’on ne documente pas assez les échanges entre les couches qui ne sont pas adjacentes aux autres, le code peut devenir difficilement maintenable. |
| M4 |  |  |
| U1 |  |  |
| U2 | S1 | Si les l’interface est trop simple, il peut devenir difficile de gérer les fonctionnalités quand un trop grand nombre de widgets sont affichés en même temps. |
| I2 | S2 | Si l’interface ne peut mettre en pratique des techniques aidant à l’utilisabilité à cause de l’exigence d’interopérabilité, elle pourrait devenir drabe et inélégante. Une telle interface nuirait à l’acceptation du nouveau système par le client. |

*Tableau 15*

## *Tableau des compromis*

|  |  |  |
| --- | --- | --- |
| **Scénarios** | **Compromis** | **Description** |
| M1 | C1 | On doit faire un compromis entre la performance et la modularité et l’extensibilité du code en utilisant l’architecture en couche. |
| M4 | C1 | Le fait de devoir passer par une interface délégué à interagir avec la librairie JAUS au lieu de communiquer avec directement à partir de chacun des contrôleurs peut causer une légère perte de performance. |
| U1 | C1 | L’architecture en couche peut occasionner une baisse de performance du système pendant son fonctionnement. |
| U2 |  |  |
| I2 | C3 | L’interface graphique doit se limiter à des éléments pouvant être affichés correctement et de la même manière sur tous les systèmes d’exploitation, ce qui pourrait limiter un peu le potentiel de maximisation de l’utilisabilité de l’interface graphique. |

*Tableau 16*

# Discussion

Ce dernier laboratoire représente le travail où nous avons pu appliquer le maximum de notions et de méthodologies vues dans lors des séances de cours. Pour cette raison, ce dernier laboratoire mérite une mention d’appréciation particulière. De plus, le fait qu’il soit basé sur des systèmes existants dans les clubs étudiants de l’École est d’autant plus intéressant bien que certains étudiants ayant des connexions avec ces clubs auraient peut-être pu s’inspirer de l’architecture que ces derniers ont choisi d’utiliser pour réaliser ce travail. Nous supposons que cette éventualité n’a pas échappée au chargé de laboratoire.

Ce quatrième laboratoire diffère des trois premiers surtout au niveau de sa nature purement conceptuelle pour ce qui est de l’architecture. Il n’y avait aucun code à programmer ou à prévoir pour ce travail. Bien qu’il fût intéressant de travailler sous un angle différent pour ce dernier laboratoire, le projet sur lequel nous devions nous baser était principalement orienté du côté des interfaces graphiques et il s’agissait d’un projet d’une petite envergure. Ceci a quelque peu réduit les possibilités d’exploration et de conception au niveau des vues architecturales de type allocation.

Afin de maximiser le potentiel d’apprentissage des étudiants et d’encourager ceux-ci à fournir le plus d’effort possible dans leur travail, il pourrait être bien de publier, parmi les étudiants du groupe, les diverses solutions architecturales que chacune des équipes a produit. Cette suggestion s’applique principalement à ce quatrième laboratoire étant donné la nature conceptuelle de celui-ci.

Les premiers laboratoires pourraient bénéficier d’avoir une nature un peu plus conceptuelle. Le travail était beaucoup orienté du côté du code bien que l’élément comparatif entre les laboratoires 2 et 3 était intéressant.

# Conclusion

Ce laboratoire avait pour objectif de nous permettre d’effectuer la conception d’une architecture d’un système en suivant l’ensemble des méthodologies vues dans le cours d’architecture logicielle. Nous devions réaliser au moins une vue architecturale pour chacune des trois catégories de vues (module, composants et connecteurs, allocation). Nous devions également appliquer la méthodologie ATAM (« Architecture Tradeoff Analysis Method ») pour évaluer notre démarche de conception ainsi que le début de notre architecture. Ce quatrième et dernier laboratoire allait donc nous amener à mettre en pratique tous les éléments théoriques majeurs que l’on a vus dans le cours d’architecture logicielle.

Ces objectifs ont été raisonnablement atteints. Nous savons qu’il aurait été utile d’ajouter davantage de détails et même certaines vues architecturales additionnelles. Le travail qui a été produit fût le résultat de compromis par rapport au temps que chacun des membres de l’équipe était en mesure d’allouer envers la réalisation de ce laboratoire.