David MENANT
apprenti
david.menant@imt-atlantique.net

Steven YVINEC
apprenti
steven.yvinec@imt-atlantique.net

Rapport de projet MGP 320

Interface tangible domotique

Formation d'Ingénieur en Partenariat

Semestre 5

Année scolaire 2017-2018

Encadrants :

Jérôme KERDREUX

Christophe LOHR

IMT ATLANTIQUE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Sommaire

1. INTRODUCTION

Dans le cadre de notre formation d'ingénieur à IMT Atlantique, l'école nous propose un projet de développement de 63 heures en 3ème année. Réalisé en binôme, il permet aux élèves de mettre en application les éléments vus en cours et de se confronter à des technologies différentes de celles abordées en entreprise.

Pour cette UV, nous avons choisis de travailler sur le développement d'une « Interface tangible domotique ». Ce projet consiste à développer une interface utilisateur dite « tangible », c'est-à-dire « une interface qui permet d'interagir avec de l'information numérique par le moyen de l'environnement physique » (source Wikipédia). Elle permettra, à partir de différentes actions exécutées sur un table interactive, de commander les équipements domotiques d'un laboratoire (lampes, chauffage) et de récupérer des informations en provenance de certains capteurs comme l'humidité ou encore la température.

Ce rapport présente dans un premier temps la version définitive du cahier des charges. Cette partie permet de préciser le contexte du projet, la formulation du besoin et détaille ensuite les différentes étapes de planification, de l'élaboration du diagramme de Gantt jusqu'à la répartition des tâches entre les membres du binôme. Dans un second temps, il aborde la partie d'études et de développement. Enfin, il propose le bilan du projet, en exposant les points difficiles rencontrés et les solutions mises en œuvre pour les résoudre.

2. CAHIER DES CHARGES

2.1 CONTEXTE DU PROJET

Pour contribuer à la recherche dans le domaine de l'assistance à la personne, l'équipe HISEV travaille sur les interactions entre les utilisateurs et les systèmes. Pour cela, une reconstitution d'un appartement miniature (contenant une cuisine, salle de bain, salon, entrée, etc.) a été réalisée au sein même de l'école. Elle est pourvue d'équipements domotiques connectés et permet à l'équipe de réaliser un certain nombre d'expérimentations et de travailler sur les utilisations futures des équipements domotiques.

Plusieurs projets ont déjà été développés, comme par exemple la librairie xAAL qui permet de faire communiquer un ensemble d'équipements provenant de constructeurs différents selon un seul et même langage. Cette technologie a déjà été utilisée pour commander certains appareils à partir d'une application WEB (allumage et extinction des lampes, réception d'informations de certains capteurs d'humidité ou de température, etc.).

Le but du projet, proposé dans le cadre du module MGP 320, est de développer une interface utilisateur tangible afin de piloter ces mêmes équipements. L'interface sera déployée sur une table interactive récemment acquise nommée Reactable. Les commandes seront envoyées aux équipements grâce à la librairie xAAL.

2.2 FORMULATION DU BESOIN

Le besoin initialement exprimé par le client est le suivant :
« *Concevoir et implémenter une interface tangible sur la table Reactable pour la solution domotique xAAL* »

Suite aux précisions apportées par le client, nous pouvons reformuler le besoin de la manière suivante :

- L'application utilisera le protocole xAAL pour communiquer avec les équipements ;
- L'interface sera affichée à partir d'une table Reactable interactive ;
- L'interface de l'application sera innovante ;
- L'application permettra de commander des équipements ;
- L'application permettra de recevoir des informations des capteurs ;
- L'interface permettra d'allumer ou d'éteindre les luminaires de l'appartement.
- L'interface permettra d'afficher la température renvoyée par le capteur de température de l'appartement.

2.3 EXPRESSION FONCTIONNELLE DU BESOIN

Pour l'expression fonctionnelle du besoin, les notations suivantes ont été utilisées :

- FP : Fonction de services Principale ;
- FC : Fonction de Contrainte.

La priorité des fonctions est notée sur 10.

2.3.1 FP : Fonctions de services Principales

Fonctions	Sous-fonction	Priorité
FP1 : Communiquer avec les équipements domotiques de la salle.	FP1.1 : Commander l'allumage et l'extinction d'une lampe.	10
	FP1.2 : Afficher les informations de température reçues par le capteur de température.	10

Tableau 1 : Liste des fonctions de services principales

2.3.2 FC : Fonctions de Contraintes

Fonctions	Priorité
FC1 : Utiliser la librairie xAAL pour communiquer avec les équipements de la salle.	10
FC2 : Utiliser la table interactive Reactable.	10
FC3 : Proposer une interface tangible innovante.	8

Tableau 2 : Liste des fonctions de contraintes

2.4 LIVRABLES ET PLANIFICATION DU PROJET

À partir de l'expression fonctionnelle du besoin et des discussions avec le client, nous avons tout d'abord découper notre projet en lots et tâches. Ensuite, nous avons procédé à la planification et à la répartition des tâches entre les contributeurs du groupe. Enfin, nous avons réalisé un récapitulatif des livrables demandés par le client.

2.4.1 Lotissage du projet

Voici le découpage du projet en lots et tâches associées :

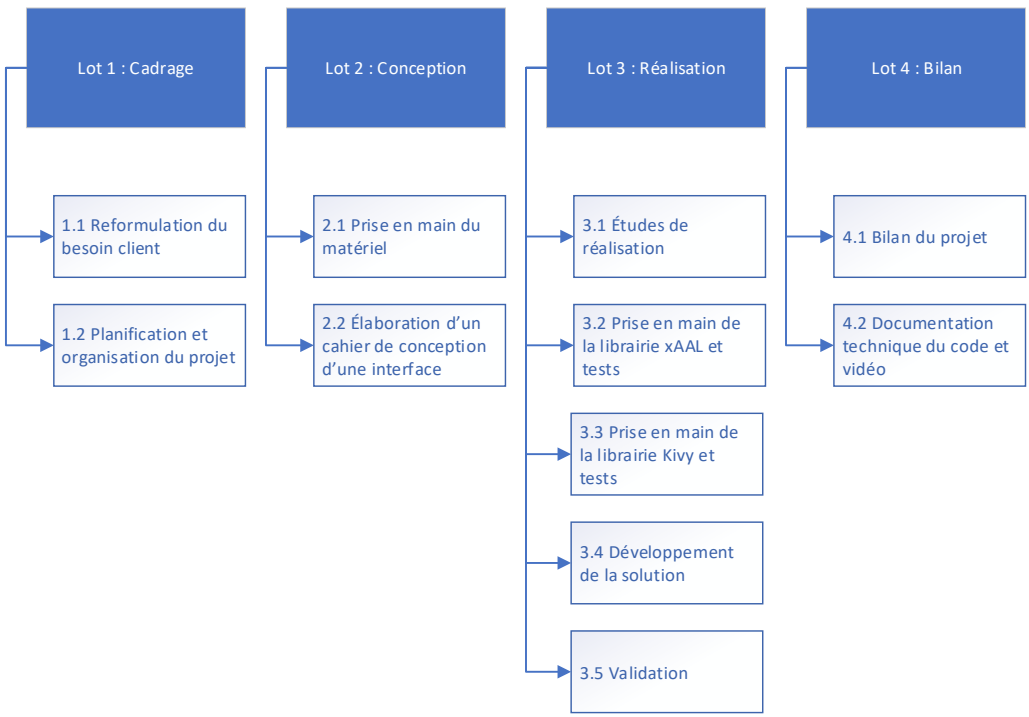

Figure 1 : Lotissage du projet

2.4.2 Diagramme de Gantt

À partir du descriptif du sujet et des échanges initiaux avec les encadrants, nous avons réalisé une première planification du projet, en estimant les tâches présentes ainsi que leur durée. Ce planning, bien que sommaire, nous as permis de nous organiser dès le début du projet malgré notre connaissance limitée du contexte.
Les rectangles verts représentent la charge de travail prévue par tâche.

Figure 2 : Diagramme de Gantt initial

Après avoir commencé à travailler sur le projet, nous nous sommes rapidement rendu compte qu'un certain nombre de tâches n'avaient pas été anticipées lors de la première planification du projet. De même, certaines tâches avaient été sous-dimensionnées.
Nous avons alors établi une seconde version plus complète en utilisant un diagramme de Gantt :

N°	Mode Tâche	WBS	Nom de la tâche	Durée	Début
1		**1**	**Cadrage**	**2 jrs**	**Lun 22/01/18**
2		1.1	Reformulation du besoin client	1 jr	Lun 22/01/18
3		1.2	Planification et organisation du projet	1 jr	Mar 23/01/18
4		**2**	**Conception**	**4 jrs**	**Mer 24/01/18**
5		2.1	Prise en main du matériel	2 jrs	Mer 24/01/18
6		2.2	Elaboration d'un cahier de conception d'interface	2 jrs	Ven 26/01/18
7		**3**	**Réalisation**	**6 jrs**	**Mar 06/02/18**
8		3.1	Etudes de réalisation	1 jr	Mar 06/02/18
9		3.2	Prise en main de la libraire xAAL et tests	2 jrs	Mer 07/02/18
10		3.3	Prise en main de la librairie Kivy et tests	2 jrs	Mer 07/02/18
11		3.4	Développement de la solution	2 jrs	Ven 09/02/18
12		3.5	Validation	1 jr	Mar 13/02/18
13		**4**	**Bilan**	**1 jr**	**Mer 14/02/18**
14		4.1	Documentation technique du code et vidéo	1 jr	Mer 14/02/18
15		4.2	Bilan du projet	0 jr	Mer 14/02/18

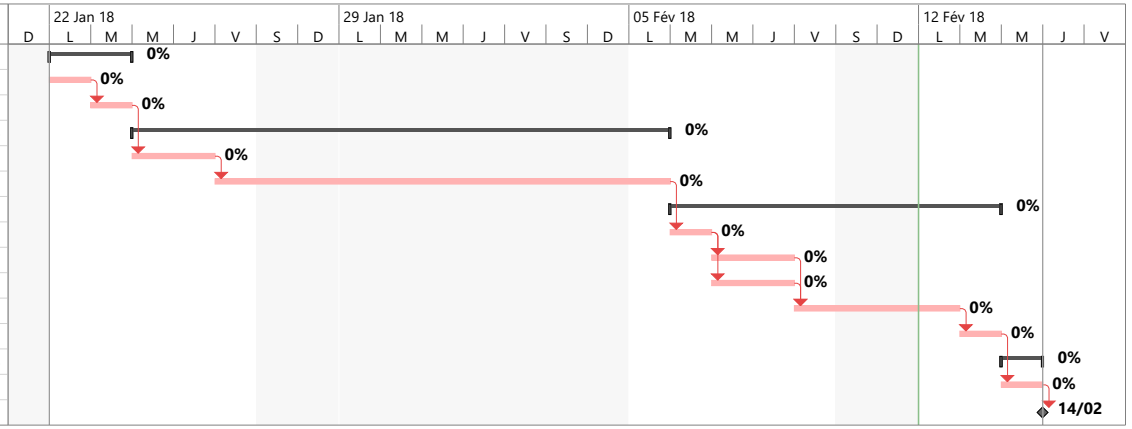

Figure 3: Diagramme de Gantt

2.4.3 Contributions aux tâches du projet

Nous avons répertorié dans le tableau ci-dessous les contributeurs associés à chaque tâche du projet :

	Lot 1 : Cadrage		Lot 2 : Conception		Lot 3 : Réalisation					Lot 4 : Bilan	
Nom du contributeur	**1.1**	**1.2**	**2.1**	**2.2**	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**4.1**	**4.2**
YVINEC	**X**	**X**	**X**		**X**	**X**		**X**	**X**	**X**	**X**
MENANT	**X**	**X**	**X**	**X**	**X**		**X**	**X**	**X**	**X**	**X**

Tableau 3 : Liste des contributions par tâche

2.4.4 Livrables attendus

Les livrables attendus par le client au cours du projet ont été listés dans le tableau ci-dessous :

Nom du livrable	Date de livraison
Rapport de projet	13/02/2018
Cahier de conception d'une interface tangible domotique	14/02/2018
Code d'un prototype implémentant l'interface tangible définie	14/02/2018
Documentation technique du code	14/02/2018
Démonstration sous forme de vidéo	14/02/2018

Tableau 4 : Liste des livrables

3. ÉTUDES ET DÉVELOPPEMENT

3.1 DÉCOUVERTE ET CALIBRAGE DE LA TABLE INTERACTIVE

L'une des premières étapes du projet a été la découverte de l'environnement de travail et plus particulièrement de la table interactive Reactable. La fonction principale de la table est d'être un synthétiseur permettant des performances musicales de DJ. La table interactive Reactable se compose d'une caméra infrarouge, d'un rétroprojecteur et d'une surface ronde. La caméra permet de tracker la position des doigts ainsi que celles de cubes en plexiglas fournis avec la table. Ces cubes possèdent un mécanisme de reconnaissance similaire aux QR-codes qui permet de les identifier de manière unique. Dans le cadre de notre application, ils sont appelés fiducials.

D'un point de vue logiciel, une table interactive se compose de deux éléments :

- Une application de tracking, qui va analyser les données reçues du capteur pour détecter les objets et leurs positions ;
- Une application « client » qui va gérer l'affichage sur la table en fonction des messages qu'elle reçoit de l'application de tracking.

La communication entre ces deux applications peut se faire à l'aide de différents protocoles. La table Reactable utilise nativement le protocole MIDI[1] entre ses composants logiciels. Ce protocole est adapté à un usage musical de la table. Un autre protocole très répandu dans le cadre d'interfaces tangibles est le protocole TUIO. C'est un framework qui définit à la fois un protocole et une API pour les surfaces tangibles multitouch. Dans notre cas, le protocole TUIO est plus adapté puisque de nombreuses implémentations de client et de simulateur existent dans de nombreux langages de programmation, comme par exemple Java, Python ou encore C.

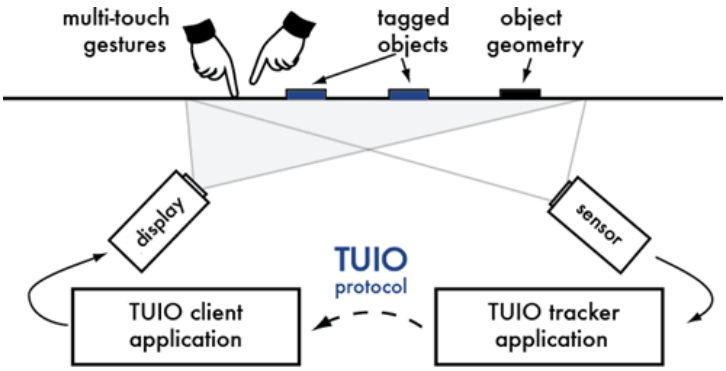

Figure 4 : Architecture d'une table interactive utilisant le protocole TUIO

Ici, l'application de tracking est celle fournie avec la table et se nomme ReacTIVision. C'est un Framework open-source et multi-plateformes permettant un tracking performant des objets de type fiducial, ainsi que des doigts.

Lorsque ReacTIVision détecte l'apparition, le déplacement ou la disparition d'un objet sur la table, il émet des trames TUIO sur le port 3333. Il suffit alors à l'application cliente d'écouter sur ce port et d'interpréter ces messages. ReacTIVision supporte aussi le protocole MIDI, et est utilisé par le logiciel fourni avec la table Reactable pour gérer la partie tracking.

[1] MIDI : Musical Instrument Digital Interface est un protocole de communication et un format de fichier dédiés à la musique, et utilisés pour la communication entre instruments électroniques, contrôleurs, séquenceurs, et logiciels de musique.

Afin de découvrir le fonctionnement de la table, nous avons dans un premier temps utilisé ReacTIVision couplé à une implémentation basique d'un client TUIO en Java.

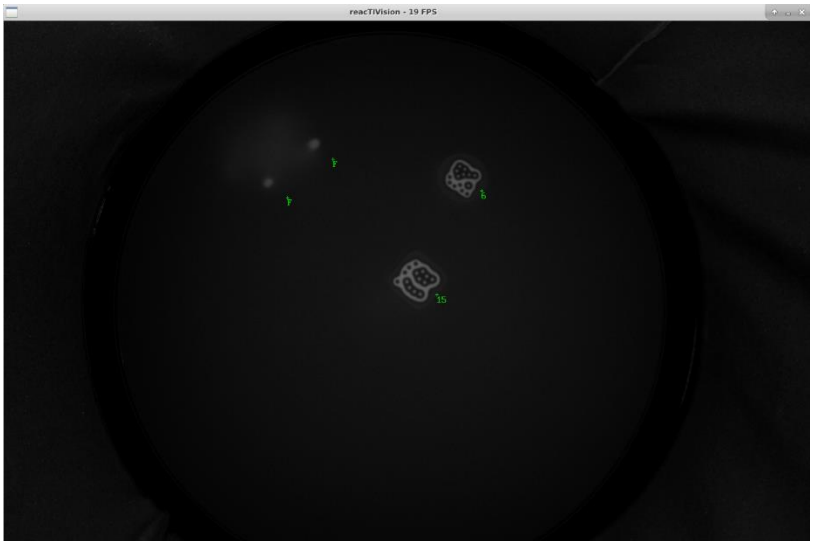

Figure 5 : Interface de reacTIVision

La capture ci-dessus (voir *Figure 5 : Interface de reacTIVision*) présente la vue caméra obtenue à partir de l'utilitaire ReacTIVision. Ici, on peut voir que la caméra infrarouge détecte deux fiducials, ayant pour identifiant 6 et 15, ainsi que deux doigts signalés par un « F » signifiant finger.
Avant d'obtenir ce résultat, nous avons dû procéder à la calibration de la caméra, de manière à ce qu'elle puisse détecter au mieux les objets sur la table.

Pour la partie affichage, nous avons décidé de projeter une interface graphique, développée en Java, permettant d'afficher la position des fiducials et des doigts relevés par la caméra infrarouge (voir *Figure 6 : Interface cliente Java*).
Il a été difficile de régler l'affichage du projecteur. En effet, le but était de projeter, de la manière la plus fidèle possible, l'interface sur la vitre de la table interactive. Seulement, le projecteur n'est pas centré au milieu de la table. Il est donc très difficile de couvrir l'intégralité de la surface vitrée et d'éviter l'effet d'affichage en trapèze. Après de nombreux essais, nous avons réussis à régler l'affichage sans avoir à réaliser de matrice de correction de position.
Réglages choisis :

- Dupliquer l'affichage du PC outillage sur le projecteur, en 1440x900.
- Réduire l'effet trapèze du projecteur au maximum grâce aux réglages.
- Redimensionner la fenêtre de l'interface de manière à couvrir toute la table.

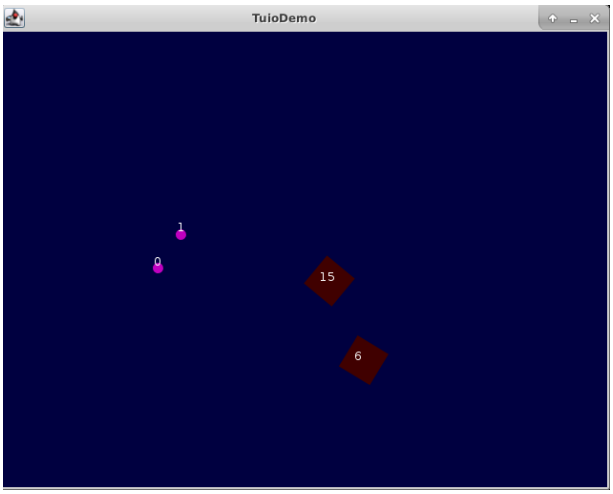

Figure 6 : Interface cliente Java

3.2 CAHIER DE CONCEPTION D'UNE INTERFACE

La réalisation d'une interface graphique dite « tangible » demande un travail de recherche important afin d'obtenir une interface à la fois ergonomique et fonctionnelle. Le temps accordé au projet étant limité, il a été choisi de mettre l'accent sur une interface fonctionnelle, mais simple d'utilisation, permettant de démontrer la possibilité d'interagir avec les équipements domotiques à partir d'une interface tangible, et d'exposer les différents évènements que la table est capable de détecter. En parallèle, un travail de recherche a été réalisé afin de découvrir des exemples de réalisation d'interface tangible.

Afin de pouvoir se concentrer sur l'aspect « Proof Of Concept », il a été décidé que dans un premier temps, l'interface aurait pour mission de contrôler seulement quelques équipements de la salle. Le choix s'est porté sur le contrôle des lampes et sur l'affichage de la température. Ce choix permet de montrer qu'il est à la fois possible d'envoyer un signal « ON » ou « OFF » à un appareil, ici une lampe, et qu'il est possible de recevoir et afficher une information de la part d'un appareil, ici un thermomètre.

Plusieurs prototypes d'interface ont ainsi été réalisés avec différentes approches. Pour le prototype, nous avons choisi d'adopter une approche « par catégorie » d'appareils à contrôler. Une partie de l'interface est ainsi dédiée au contrôle des lampes, tandis qu'une autre partie est utilisé pour le thermomètre. Le laboratoire disposant d'un nombre important d'équipements domotiques, il a été décidé de limiter la place occupée par chaque catégorie d'équipements dans l'interface, afin de pouvoir en ajouter de nouvelles ultérieurement.

Ainsi, l'interface à l'état initial est très sobre et ne comprend que deux boutons. Mais en plaçant un fiducial ou en maintenant une pression sur ces boutons, l'interface affiche davantage d'options et d'informations. On peut alors contrôler individuellement les lampes, et afficher la température. Le schéma de gauche représente l'interface à l'état initial, tandis que le schéma de droite représente l'interface complète une fois les fiducials posés sur les boutons 1 et 2.

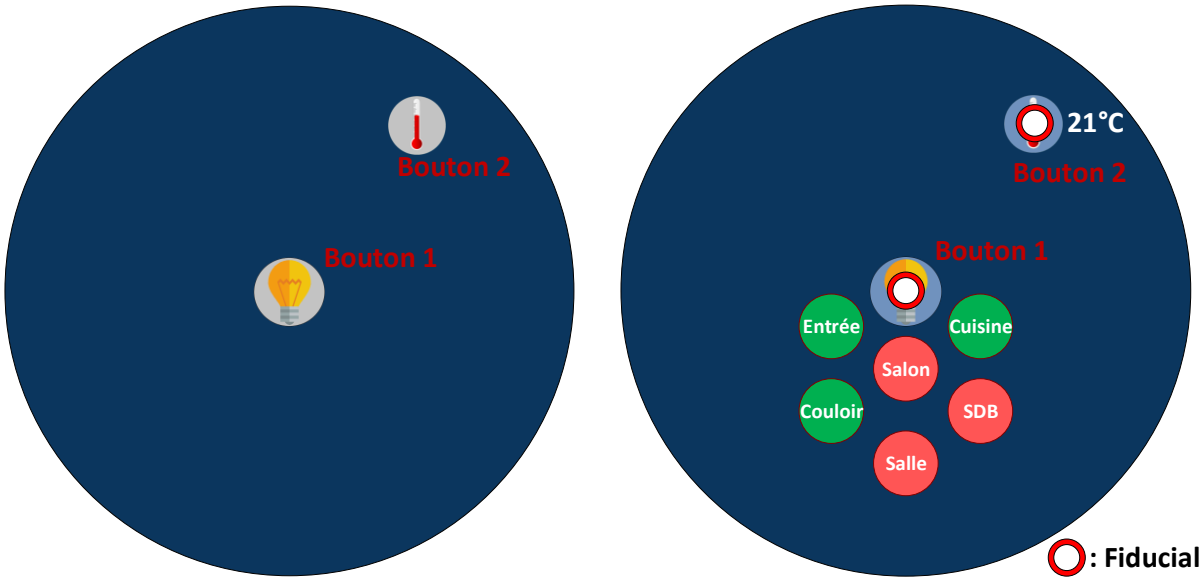

Figure 7 : Interfaces utilisateur

3.3 DÉCOUVERTE ET PRISE EN MAIN DE LA LIBRAIRIE XAAL

Comme dit précédemment, la solution domotique utilisée est xAAL. Elle a été développée pour permettre aux équipements de communiquer selon le même protocole, même s'ils proviennent de vendeurs différents.

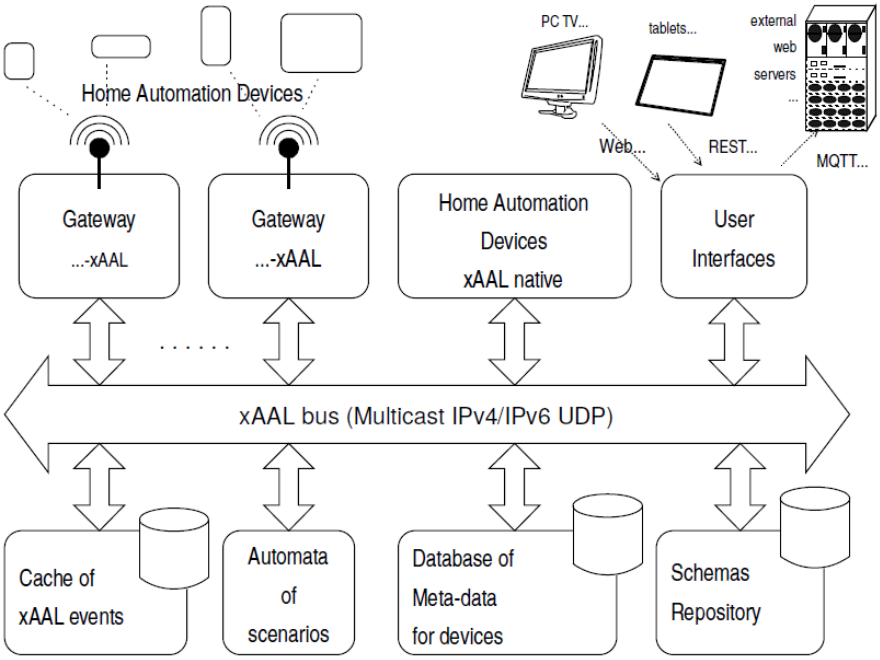

Figure 8 : Architecture de la solution domotique xAAL

Tous les appareils communiquent au travers d'un bus UDP multicast. Sur ce bus, seuls des messages au format JSON, dont le formalisme est implémenté dans la libraire xAAL, sont autorisés à transiter. Une passerelle a été développée pour chaque équipement afin de traduire les messages JSON dans le langage de l'équipement et inversement.
Ainsi, pour communiquer avec des équipements, une interface utilisateur (WEB, Rest, etc..) se connecte sur le bus UDP et envoie des messages au format JSON. C'est la passerelle qui s'occupe de la traduction des messages. L'interface n'a donc pas besoin de connaître les protocoles de communications propriétaires de chaque équipement.

La librairie a été implémentée en C, en Java et en Python. La partie Java n'étant pas à jour, nous avons décidé de travailler en Python puisque c'est le langage que nous maitrisons le plus après le Java.

Dans un premier temps, nous avons donc procédé à l'installation de la librairie et de l'environnement de développement sur notre PC outillage. Pour cela, nous avons suivi les différentes étapes décrites dans le README présent sur le gestionnaire de version SVN.
Pour des raisons de confidentialité, l'ensemble des messages qui transitent sur le bus UDP sont chiffrés. Pour pouvoir les déchiffrer, il est indispensable de générer une clé à partir d'une passphrase (spécifique à la librairie) grâce à l'utilitaire *xaal-keygen*.

Pour pouvoir utiliser correctement la librairie, nous avons d'abord parcouru le code et essayé de réaliser un schéma d'architecture. Ce travail a été difficile puisque la librairie n'est pas très documentée. Après une discussion avec les enseignants, voici le schéma que nous avons pu réaliser :

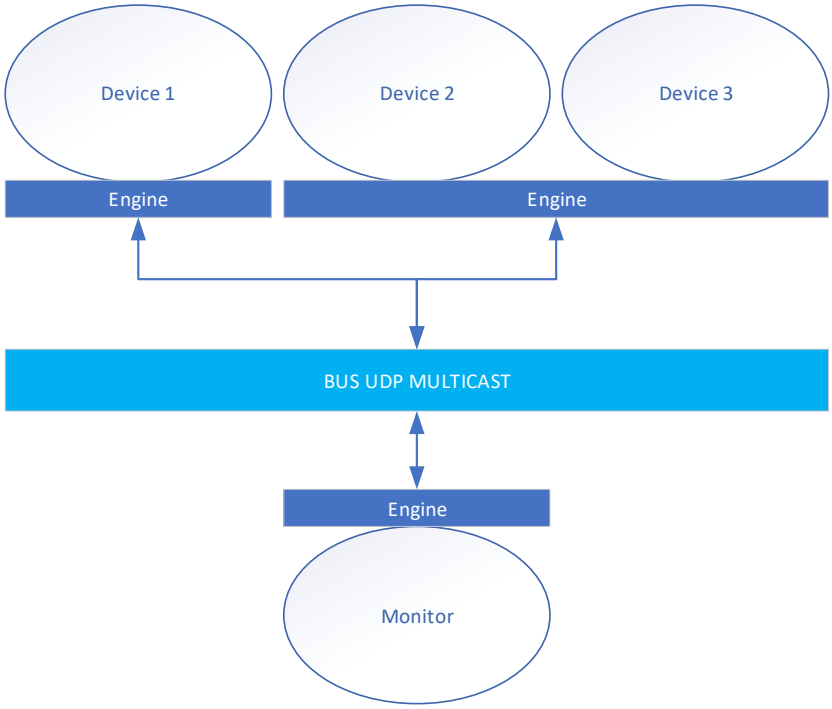

Figure 9 : Architecture logicielle de la solution domotique xAAL

L'ensemble des équipements est monitoré grâce à un « Monitor ». Il permet d'initier des requêtes périodiquement vers les équipements afin d'obtenir les valeurs de leurs attributs, leur description ou encore leur état.
Les requêtes sont transmises sur le bus par l'intermédiaire d'un « Engine », rattaché au « Monitor », en direction des équipements concernés. Chaque équipement est vu comme un « Device » et dispose aussi d'un « Engine » pour la réception des requêtes. De son côté, l'« Engine » initie aussi périodiquement des requêtes vers ses équipements afin d'obtenir des informations. Ainsi, lorsque le « Monitor » initie une requête, il n'interroge pas directement l'équipement mais son « Engine ».

Dans le but de commencer à manipuler les fonctions de la librairie, nous avons essayé d'allumer et d'éteindre une lampe. Pour cela, nous avons d'abord récupéré l'adresse d'une lampe à partir d'un *xaal-walker* (utilitaire permet de récupérer les informations sur toutes les devices de la salle) et créé un « Monitor ». Une fois le « Monitor » créé, il suffit de lancer une requête d'allumage vers la lampe en précisant son adresse. Pour récupérer les changements d'états, il ne faut pas oublier la méthode `loop()` de l'« Engine ».

Ensuite, de façon à satisfaire les besoins clients, nous avons essayé de récupérer la valeur de température mesurée par le thermomètre de la salle. Le principe d'utilisation de la librairie est similaire.

3.4 RECHERCHE ET PRISE EN MAIN D'UNE IMPLÉMENTATION DE CLIENT TUIO

Comme expliqué dans la partie « Découverte de la table », l'aspect logiciel de la table interactive se compose d'une application de tracking et d'une application cliente dialoguant à l'aide du protocole TUIO. Dans notre cas, le logiciel open-source ReacTIVision joue le rôle de l'application de tracking. Côté client, de nombreuses implémentations sont disponibles librement sur Internet, et dans des langages variés.

La librairie Xaal étant disponible et à jour en Python, nous avons orienté nos recherches vers ce langage. Python a aussi l'avantage d'être un langage multiplateforme et de disposer d'un

grand nombre de librairies facilitant la création d'interfaces graphiques. En effet, la durée du projet ne nous permettait pas de tout développer nous-même.

Notre attention s'est alors portée sur Kivy. C'est un Framework open-source et multi-plateforme Python utile pour le développement rapide d'applications prenant en compte des interfaces utilisateurs innovantes telles que les surfaces tactiles ou les surfaces multi-touch comme les tables interactives. Kivy est donc disponible sur de nombreux systèmes comme Windows, Linux, osX. En plus d'une librairie permettant le développement d'interface, Kivy dispose d'une implémentation de client TUIO, ce qui permet de gérer les évènements en provenance de la table Reactable. Enfin, l'ensemble de la librairie est documenté et de nombreux exemples de création sont présents sur Internet. Pour ces différentes raisons, nous avons décidé d'utiliser Kivy comme application cliente TUIO après avoir vérifié sommairement son bon fonctionnement avec la table.

Pour commencer, nous nous sommes intéressés à l'aspect technique de ce Framework. D'un point de vue fonctionnel, Kivy se compose de multiples blocs travaillant ensemble :

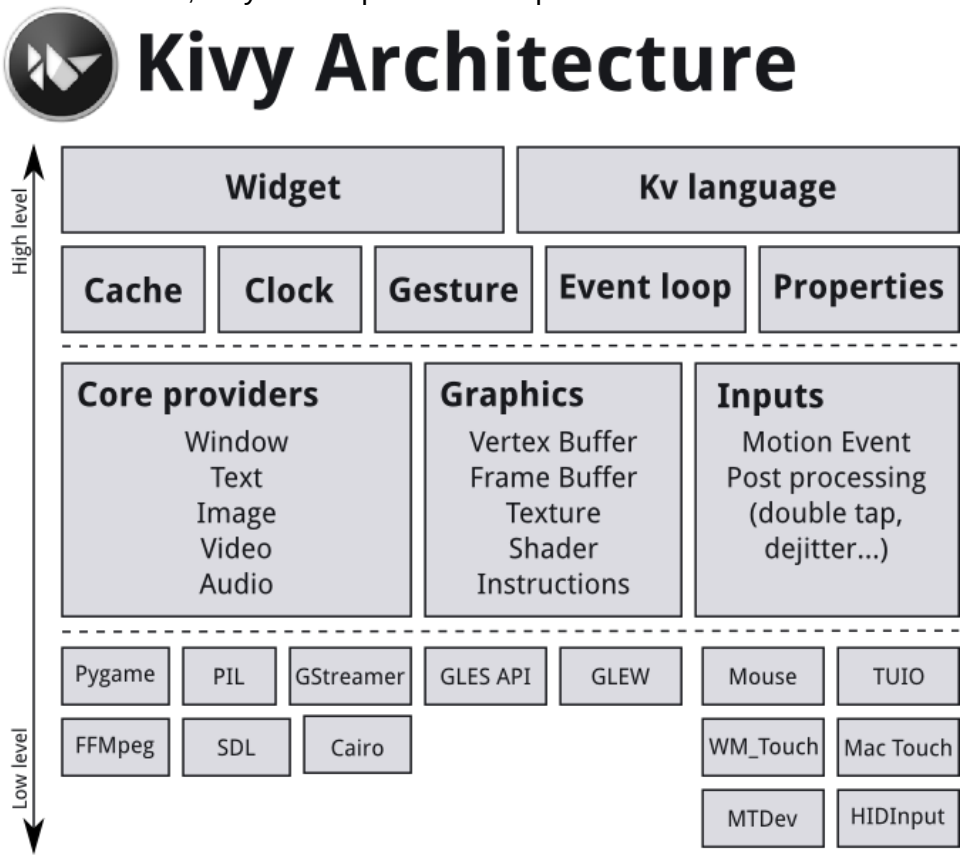

Figure 10 : Schéma d'architecture du Framework Kivy

La brique « Inputs » permet à Kivy de faire abstraction des différents types d'entrée, tels que TUIO ou une interface clavier/souris classique. En effet, pour toutes ces entrées, il est possible de leur associer une position en deux dimensions sur l'interface. Elles sont alors représentées par une instance de la classe Python `Touch()` qui peut posséder trois états : Down lors de son apparition - Move tant qu'elle est maintenue - Up lorsqu'elle disparait.

Lorsqu'une donnée arrive en entrée, Kivy envoie un évènement. Cet évènement va ensuite être traiter par les widgets. Le terme widget désigne dans Kivy un objet recevant des évènements. Cet objet n'est cependant pas forcément visible sur l'interface. Tous les widgets sont ordonnés sous forme d'arbre. Ils peuvent avoir n'importe quel nombre d'enfant et posséder un widget parent, exception faite de la racine de l'arbre. C'est ce widget racine qui va recevoir les évènements en premier. En fonction des caractéristiques de l'évènement, le

widget va le traiter ou l'ignorer. S'il l'ignore, l'événement est transmis aux enfants, et ainsi de suite jusqu'à ce que l'évènement soit traité ou que l'évènement est traversé l'arbre.

La brique « Core Providers » permet quant à elle de faire le lien entre Kivy et les APIs des différents systèmes d'exploitation gérant les « Core Task ». Ces tâches correspondent aux tâches basiques telles qu'ouvrir une fenêtre, afficher du texte, ou encore récupérer l'image d'une caméra. Cette brique permet ainsi de s'affranchir des APIs natives différentes sur chaque système.

Enfin, la brique "Graphics" est une API faisant abstraction d'OpenGL. Cette API permet notamment de créer des formes en utilisant des métaphores simples comme par exemple "Rectangle". Cette API est ensuite utilisée par l'ensemble des widgets et est implémenté en C pour des raisons de performances.

En plus du Python, il est possible de développer une application à l'aide d'un langage propre à Kivy : le langage Kv. Ce langage déclaratif permet de créer plus facilement des arbres de widgets et des liens entre eux. Il est possible de développer une application utilisant à la fois un code Python et un code Kv. Cela permet de séparer d'un côté la logique du programme et de l'autre l'interface.

Après avoir étudié le fonctionnement de Kivy, nous avons exécuté plusieurs démonstrations fournies avec le Framework afin d'observer les possibilités offertes par l'application. En parallèle, la prise en main s'est faite à travers la réalisation et l'étude de tutoriels, comme par exemple le développement d'un jeu de Pong compatible avec la table, ainsi que la recherche et la compréhension de différents exemples.

3.5 DÉVELOPPEMENT

Après avoir pris en main la librairie xAAL et le Framework Kivy, nous avons débuté le développement de l'application. Ce dernier s'est fait dans un premier temps au travers de deux objectifs indépendants. Le premier objectif était la réalisation d'une interface graphique conforme au prototype décrit dans le cahier de conception. Le second était quant à lui la création d'un programme Python contenant un ensemble de fonctions afin d'interagir avec les équipements. Ce programme permet notamment de gérer l'envoie des requêtes vers les équipements domotiques.

Le développement du programme de l'interface s'est fait de manière incrémentale. Dans un premier temps, il a été décidé de développer la partie de l'application répondant à la fonction de service FP1.1, c'est à dire la gestion des éclairages de l'appartement. L'accent a plus précisément été mis sur le contrôle individuel des lampes. La seconde étape répond à la fonction de service FP1.2. Cela correspond à l'affichage de la température obtenue à partir d'un thermomètre. Les étapes trois et quatre du développement correspondent respectivement à la mise en place de fonctionnalités avancées de l'interface, et à l'ajout d'un outil permettant de calibrer l'affichage de l'application sur la table. Ces fonctionnalités comprennent notamment la possibilité de déplacer l'affichage de la température sur la table, ou de contrôler de manière global les lampes de l'appartement. Pour des raisons de temps, la quatrième étape n'a, à ce jour, pas été traité.

L'utilisation d'un développement incrémental nous as permis de gagner du temps et de paralléliser davantage la charge de travail autour de la programmation de l'interface. En effet, il a été possible d'intégrer dès la fin de la première « itération » le code de l'interface et celui gérant les interactions xAAL. Ainsi, il a été possible de détecter très rapidement les erreurs dans le code d'un des deux composants, et de tester le bon fonctionnement de l'application de bout en bout, c'est à dire de la table TUIO jusqu'à l'allumage d'une lampe.

Cela a notamment permis de découvrir un problème de concurrence entre les deux composants. En effet, récupérer les requêtes ainsi que les changements d'états des équipements nécessite d'appeler en permanence la méthode `loop()` de l'engine associé à notre Monitor. Seulement, la méthode `run()` de l'application Kivy est aussi une opération bloquante, qui attend en permanence des évènements. Pour pallier à ce problème, le Framework Kivy dispose d'une fonction d'horloge permettant de réaliser un callback périodique sur une méthode. Comme la partie network de xAAL intègre déjà un timer lorsqu'elle accède aux données du buffer, nous avons pu fixer le timer de Kivy à 0. On utilise alors le timer (de 0.03 secondes) de la méthode `network.__get_data()` pour réaliser le callback.

Afin de faciliter le développement de l'application, différents outils ont été utilisés. Ainsi, pour le stockage et le suivi des modifications des fichiers sources, nous avons décidé d'utiliser un gestionnaire de version Git hébergé par la plateforme Redmine de l'école.
Pour le développement de l'interface, un simulateur TUIO, permettant d'afficher les objets traqués, a été utilisé.

3.6 VALIDATION ET TESTS

Notre application répond aux exigences du client. Nous couvrons l'ensemble des fonctions de services principales et des fonctions de contraintes exprimées dans le cahier des charges.

Pour valider le bon fonctionnement de notre application, nous avons procédé à différents tests :

- Les tests de performance : nous avons vérifié que le temps de réponse de l'application était acceptable. Les lampes s'allument et s'éteignent instantanément et les informations de température sont remontées aussi rapidement. En revanche, le temps de réponse des composants de l'IHM est parfois assez lent (plus ou moins une seconde) ce qui n'est pas optimal pour avoir une bonne interactivité. De plus, la fonction première de ReacTIVision est la détection de fiducial. Malgré qu'il supporte aussi la détection des doigts, celle-ci est plus laborieuse à calibrer, ce qui engendre notamment des problèmes de reconnaissance en fonction de la taille des doigts de chacun.
- Les tests d'endurance : nous avons fait fonctionner notre application pendant plusieurs heures en la sollicitant régulièrement. Aucun problème de surcharge mémoire ou CPU n'est à signaler.

Vous trouverez en 5.1 une photographie de l'interface tangible déployée sur la table interactive.

4. BILAN DU PROJET

En comparant les deux graphiques, nous pouvons voir que le planning initial n'a effectivement pas été parfaitement respecté. En effet, nous n'avions pas anticipé tous les problèmes que nous avons pu rencontrer pendant ce projet. Le retard a en revanche été compensé par une réorganisation du temps de travail. Nous sommes passé de journées de 7 heures à des journées de 9 heures et avons travaillé les week-ends. Ainsi, nous avons pu livrer les documents demandés en temps voulu.

N°	Mode Tâche	WBS	Nom de la tâche	Durée	Début
1		**1**	**Cadrage**	**2 jrs**	**Lun 22/01/18**
2		1.1	Reformulation du besoin client	1 jr	Lun 22/01/18
3		1.2	Planification et organisation du projet	1 jr	Mar 23/01/18
4		**2**	**Conception**	**4 jrs**	**Mer 24/01/18**
5		2.1	Prise en main du matériel	2 jrs	Mer 24/01/18
6		2.2	Elaboration d'un cahier de conception d'interface	2 jrs	Ven 26/01/18
7		**3**	**Réalisation**	**6 jrs**	**Mar 06/02/18**
8		3.1	Etudes de réalisation	1 jr	Mar 06/02/18
9		3.2	Prise en main de la libraire xAAL et tests	2 jrs	Mer 07/02/18
10		3.3	Prise en main de la librairie Kivy et tests	2 jrs	Mer 07/02/18
11		3.4	Développement de la solution	2 jrs	Ven 09/02/18
12		3.5	Validation	1 jr	Mar 13/02/18
13		**4**	**Bilan**	**1 jr**	**Mer 14/02/18**
14		4.1	Documentation technique du code et vidéo	1 jr	Mer 14/02/18
15		4.2	Bilan du projet	0 jr	Mer 14/02/18

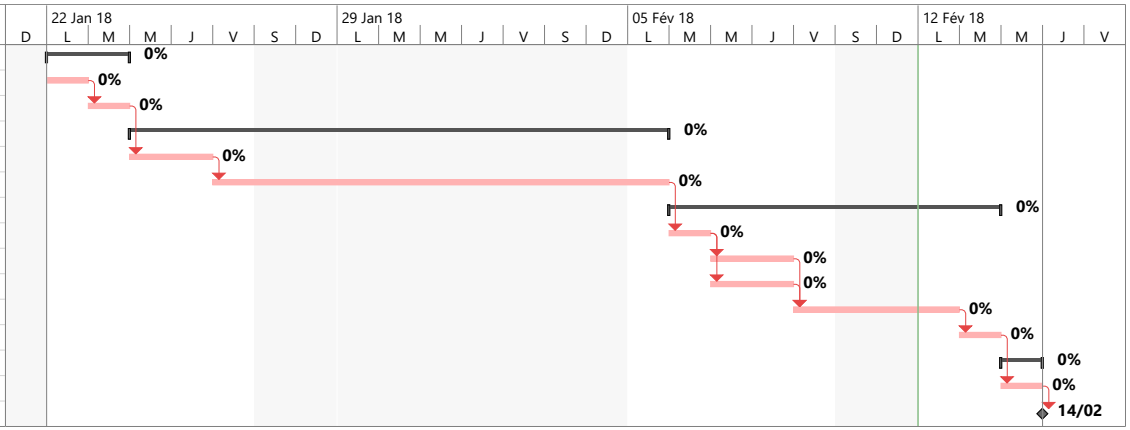

Figure 11 : Planning initial

N°	Mode Tâche	WBS	Nom de la tâche	Durée	Début
1		**1**	**Cadrage**	**2 jrs**	**Lun 22/01/18**
2		1.1	Reformulation du besoin client	1 jr	Lun 22/01/18
3		1.2	Planification et organisation du projet	1 jr	Mar 23/01/18
4		**2**	**Conception**	**5 jrs**	**Mer 24/01/18**
5		2.1	Prise en main du matériel	3 jrs	Mer 24/01/18
6		2.2	Elaboration d'un cahier de conception d'interface	2 jrs	Lun 05/02/18
7		**3**	**Réalisation**	**8 jrs**	**Mer 07/02/18**
8		3.1	Etudes de réalisation	1 jr	Mer 07/02/18
9		3.2	Prise en main de la libraire xAAL et tests	4 jrs	Jeu 08/02/18
10		3.3	Prise en main de la librairie Kivy et tests	3 jrs	Jeu 08/02/18
11		3.4	Développement de la solution	2 jrs	Mer 14/02/18
12		3.5	Validation	1 jr	Ven 16/02/18
13		**4**	**Bilan**	**1 jr**	**Lun 19/02/18**
14		4.1	Documentation technique du code et vidéo	1 jr	Lun 19/02/18
15		4.2	Bilan du projet	0 jr	Lun 19/02/18

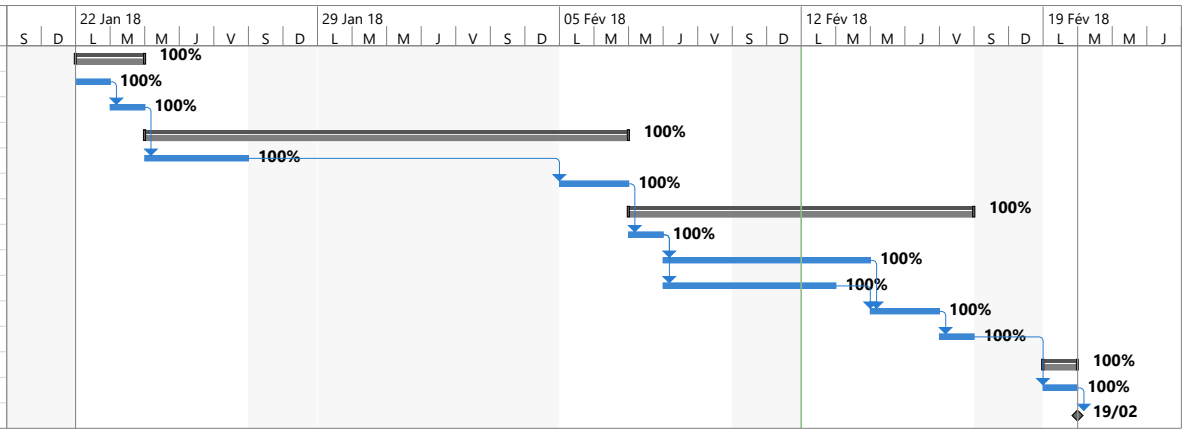

Figure 12 : Planning final

La prise en main de la librairie xAAL a pris deux fois plus de temps que prévu. Nous avons par exemple perdu du temps sur la configuration de la clé pour la partie chiffrement/déchiffrement des messages sur le bus. En effet, nous ne savions pas qu'il y avait une passphrase prédéfinie pour le projet et avons perdu quasiment une journée de travail à essayer de résoudre le problème.

De plus, nous avons eu des difficultés lors de la prise en main de reacTIVision. Lors des premiers tests réalisés, nous nous sommes rendus compte que les fiducials livrés avec la table n'étaient pas tout le temps ou alors mal reconnus (mauvais numéro). Nous avons d'abord pensé à un mauvais calibrage de la caméra et avons essayé de modifier l'ensemble des paramètres de calibration pour améliorer la détection. A la fin de la journée, nous nous sommes rendus compte que le problème ne venait pas de la caméra mais du fichier de configuration des fiducials enregistré dans reacTIVision qui ne correspondait pas du tout aux empreintes de ceux livrés avec la table. Nous avons donc récupéré le fichier de configuration des empreintes contenu dans l'installeur de Reactable (logiciel de musique fourni avec la table) pour le rapatrier dans reacTIVision. ReacTIVision étant open-source, il n'utilise pas les mêmes empreintes de fiducial pour fonctionner.

Ce projet de 63h a été l'occasion de travailler avec des technologies et des équipements que nous n'avions pas l'habitude de manipuler. Il nous a permis de faire face à divers problèmes et de mettre en œuvre les compétences de l'ingénieur, notamment l'adaptabilité. En effet, il a fallu s'adapter à un nouvel environnement de travail, et collaborer ensemble.

Pour terminer, nous souhaiterions remercier nos encadrants Jérôme Kerdreux et Christophe Lohr, pour leur disponibilité tout au long du projet.

5. ANNEXES

5.1 ANNEXE 1

Figure 13 : Interface déployée sur la table

Campus de Brest
Technopôle Brest-Iroise
C S 83818
29238 Brest Cedex 03
France
Tél. : + 33 (0) 2 29 00 11
11

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom