Université de Nice - Sophia Antipolis

Polytech Nice Sophia

Sciences Informatiques 5ème année

Projet de fin d'études
Conception assistée d'IHM distribuées

Alix Humbert, Valentin Ah-Kane, Baptiste Frere

Encadrant : Anne-Marie Dery Pinna

Table des matières

1 Identification du projet **2**

2 Introduction **2**
- 2.1 Contexte . . . 2
- 2.2 Problématiques . . . 3
  - 2.2.1 Répartition des tâches . . . 3
  - 2.2.2 Conception assistée . . . 5
- 2.3 Objectif . . . 5

3 Travail préliminaire **6**
- 3.1 Analyse de l'application existante . . . 6
  - 3.1.1 Fonctionnement . . . 6
  - 3.1.2 Limitations . . . 7
  - 3.1.3 Points positifs . . . 7
  - 3.1.4 Bilan . . . 8
- 3.2 Problématiques liées au développement . . . 8
  - 3.2.1 Interactions et visualisations . . . 8
  - 3.2.2 Extensibilité et déploiement . . . 9
  - 3.2.3 WEB (titre à modifier pour concepts plus proche du métier) . . . 10
- 3.3 Planning . . . 10

4 Travail réalisé **11**
- 4.1 Scénarios d'utilisation . . . 11
- 4.2 Solutions apportées . . . 11
  - 4.2.1 Interactions et visualisations . . . 11
  - 4.2.2 Extensibilité et déploiement . . . 11
  - 4.2.3 WEB (titre à modifier pour concepts plus proche du métier) . . . 11

5 Conclusion **13**
- 5.1 Rétrospective . . . 13
- 5.2 Perspectives d'amélioration . . . 13

6 Annexes **15**

1 Identification du projet

Intitulé du sujet : PFE2019#028 Conception assistée d'IHM distribuée
Type de projet : Développement
Encadrant : Anne-Marie Dery Pinna
Composition de l'équipe :

- Alix Humbert : **majeure WEB**
- Valentin Ah-Kane : **majeure IHM**
- Baptiste Frere : **majeure Architecture Logicielle**

2 Introduction

2.1 Contexte

Dans le cadre de la recherche en Interface Homme-Machine, plusieurs chercheurs du laboratoire I3S tentent d'établir des règles pour aider à la conception d'applications à interfaces utilisateur distribuées (IUD). A la différence d'un logiciel classique disposant d'une unique interface sur laquelle un utilisateur peut entrer et accéder à des informations, une application IUD fait intervenir plusieurs interfaces utilisateur au cours de son utilisation. Le terme d'interface "distribuée" ou "répartie" exprime le fait que l'interaction avec l'application ne s'effectue pas via une interface monolithique mais de façon répartie, à travers les interfaces de plusieurs appareils.

Exemple : On peut voir sur la figure 1 ci-dessous une application mobile "classique" d'édition photo utilisable avec la seule interface d'une tablette. La figure 2 montre l'utilisation d'une application d'un jeu de plateau à interface répartie. L'utilisation de ce jeu fait intervenir ici cinq interfaces sur autant d'appareils différents. Chaque joueur accède à sa main de carte via son smartphone personnel et tous peuvent voir le plateau sur l'écran d'une tablette commune

FIGURE 1 – Application mobile d'édition photo

FIGURE 2 – Jeu de plateau numérique à interface répartie

2.2 Problématiques

2.2.1 Répartition des tâches

L'utilisation d'une application IUD peut être décrite de la même manière qu'une application classique, à travers un ensemble de tâches. Une tâche est une action atomique devant être effectuée soit par l'application, soit par un ou plusieurs de ses utilisateurs, soit par les deux à la fois. Dans le premier cas on parle de **tâche système**, dans le second on utilise le terme de **tâche utilisateur** et dans le troisième celui de **tâche interactive**.

Exemple : Dans le cas du jeux de plateau numérique figure 2, distribuer les cartes aux joueurs est une tâche système effectuée automatiquement par l'application et l'utilisateur n'a pas à interagir avec un dispositif pour que la distribution soit menée à bien. Choisir une carte (à différencier du fait de jouer une carte), est une tâche utilisateur où un joueur doit réfléchir à la carte qu'il souhaite jouer. Jouer une carte est une tâche interactive car nécessite que le joueur appuie sur l'écran et que l'application traite cette action (enregistrement la carte jouée et affichage sur le plateau.)

Dans le cas d'une application IUD, la question se pose alors de savoir sur quel appareil effectuer une tâche en particulier. Dans l'exemple du jeu de plateau numérique figure 2, comment a été fait le choix de placer les tâches relatives aux mains des joueurs sur leurs smartphones et celles relatives au plateau sur une tablette tactile partagée ?

On formule alors la problématique suivante :

Problématique n°1 : Comment répartir les tâches d'une application à interface répartie sur un ensemble d'appareils prédéfinis ?

Afin de répondre à cette problématique, il est possible d'établir des critères sur lesquels baser la décision d'attribuer une tâche à un dispositif. Plusieurs cirières de la sorte ont déjà été spécifiés par les chercheurs du laboratoire I3S :

Critère n°1 : Le type de données

Une application à interface répartie manipule des données de la même manière qu'une application standard. Ces données peuvent être de trois types :

- Publique
- Personnelle
- Privée

Les **données publiques** sont des données auxquelles n'importe quel utilisateur peut avoir accès.

Exemple : Dans l'exemple du jeu de plateau figure 2, la disposition des cartes sur le plateau est une donnée publique par nature puisque chaque joueur a besoin d'y accéder pour jouer son tour.

Les **données personnelles** sont des données propres à un utilisateur.

Exemple : Dans l'exemple du jeu de plateau figure 2, chaque joueur possède sont propre nombre de jetons de chaque couleur. Bien que l'image montre des jetons physiques, ils pourraient être intégrés à l'application et affichés sur un écran

Les **données privées** sont à la fois personnelles et confidentielles. Elle ne concernent qu'un utilisateur et ne sont accessibles que par ce dernier.

Exemple : Dans l'exemple du jeu de plateau figure 2, la main de carte d'un joueur est une donnée privée. Elle lui est propre et personne d'autre ne peut la voir

L'information sur le type des données manipulées au cours de la réalisation d'une tâche peut être utilisée afin de choisir quel dispositif associer à cette dernière. Parmi les appareils dont dispose le concepteur de l'application, certains sont plus naturellement aptes à recevoir des données d'un type particulier. Nous avons établies les associations suivantes (liste non exhaustive) :

Appareil	Publiques	Privées	Personnelles
Smartphone		X	X
Tablette tactile	X	X	X
Casque VR		X	X
Mur	X		
Table tactile	X		

TABLE 1 – Type de données privilégiées en fonction des appareils

Critère n°2 : Le type d'interaction

On peut se baser sur le fait qu'une tâche nécessite ou non un affichage afin de réduire le choix des appareils sur lesquels la placer. Lors de la répartition, on privilégiera les dispositifs munis d'un affichage pour les tâches en ayant besoin.

Exemple : Dans l'exemple du jeu de plateau figure 2, les tâches systèmes ou interactives relatives aux mains de cartes des joueurs nécessitent un affichage. En effet, les joueurs ont besoin de voir leurs mains, qui changent au cours de la partie.

Exemple : Dans certains jeux multijoueurs, les joueurs disposant d'une manette avec écran peuvent réaliser plus de tâches que ceux ne disposant que d'une manette classique. On peut voir une telle situation sur la figure 3

FIGURE 3 – Jeu multijoueur jouable avec différents types de manettes

Critère n°3 : Le nombre d'utilisateurs

On s'intéresse ici au nombre d'utilisateurs nécessaires pour réaliser une tâche. Celles n'impliquant qu'un seul utilisateur sont dites individuelles et celle nécessitant la participation de plusieurs utilisateurs sont appelées coopératives. On peut alors décider qu'un tâche coopérative devra plutôt être réalisée sur une même interface, de grande taille, par l'ensemble des utilisateurs. Ainsi, il sera plus facile pour chacun de voir ce que les autres font. La table tactile est un dispositif plutôt adapté pour les tâches coopératives puisqu'elle est de grande taille et permet une interaction tactile simultanée. Au

contraire, une tâche individuelle devrait plutôt être accomplie sur une interface personnelle telle qu'un smartphone ou une manette.

Exemple de tâche coopérative : Sur la figure 4, on peut voit plusieurs personnes collaborant sur une même table tactile. Si chacun disposait de sa propre interface au cours de la réalisation d'une tâche coopérative, il serait moins facile de savoir quelles actions les autres joueurs font.

FIGURE 4 – Utilisation coopérative d'une table tactile

Autres critères

Il existe d'autres critères pouvant être utilisés pour choisir le dispositif d'une tâche. Par exemple, une tâche devant être réalisée rapidement devra plutôt être attribuée au même dispositif que la tâche précédente afin d'économiser le temps de changement d'appareil.

2.2.2 Conception assistée

Malgré les critères établis précédemment, concevoir une application à interface répartie demeure une tâche complexe. A mesure que le nombre de tâches augmente et le nombre d'appareils disponibles également, le nombre de façons d'associer les premières aux seconds devient trop important pour que le concepteur de l'application puissent toutes les prendre en compte. En effet, il n'est pas rare qu'une application soit composée de plusieurs centaines de tâches. De plus, le concepteur doit faire en sorte que l'expérience utilisateur de son application reste satisfaisante. Celle-ci peut facilement être dégradée si les utilisateurs doivent trop souvent changer de dispositif pendant qu'ils utilisent l'application.

On formule donc une seconde problématique :

Problématique n°2 : Comment guider la conception d'une application à interface répartie ?

2.3 Objectif

Le but de notre PFE fut de répondre à la problématique N°2 en développant un outil d'aide à la conception d'applications réparties. L'application que nous avons développée se base sur les réponses fournies à la problématique N°1. Le développement a été initié par Valentin Ah-Kane et Simon Serrano durant leur stage de quatrième année au laboratoire I3S. Durant notre PFE, notre rôle a été de poursuivre le travail qu'ils ont réalisé en améliorant les fonctionnalités déjà présentes et en y apportant de nouvelles.

3 Travail préliminaire

Cette section est consacrée au travail que nous avons réalisé avant de commencer le développement de l'application. Dans un premier temps, nous avons effectué une analyse approfondie de l'application existante. A la suite de cette analyse, nous avons pu établir une liste d'améliorations techniques et fonctionnelles à apporter. Nous avons ensuite tenté de répondre aux différentes problématiques soulevées par ces améliorations. Nous avons finalement établit un planning ainsi qu'une répartition des tâches au sein de l'équipe.

3.1 Analyse de l'application existante

La première étape de cette phase préliminaire fut de faire des réunions avec les anciens membres chargés du développement de l'application. Ils ont pu nous expliquer de façon globale les fonctionnalités qu'ils avaient déjà intégrées et comment ces dernières étaient implémentées.

3.1.1 Fonctionnement

Le fonctionnement de l'application était en trois étapes majeures. D'abord, le concepteur entre une représentation des tâches de son application et indique les dispositifs dont il dispose. Ensuite, il applique une règle de répartition automatique des tâches sur les dispositifs. Enfin, il visualise la répartition obtenue, la complète manuellement si besoin, et règle les éventuels problèmes. Le fonctionnement est schématisé en figure 5.

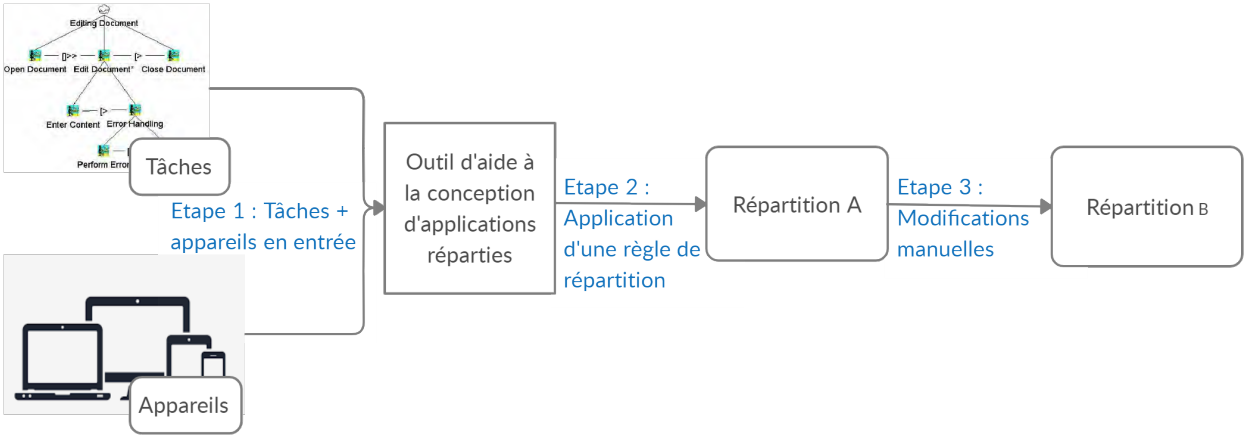

FIGURE 5 – Fonctionnement de l'application au début du projet [1]

Un autre aspect important de l'outil est la possibilité pour le concepteur d'avoir une représentation graphique des tâches de son application au moyen du modèle CTT (Concur Task Tree). Cette représentation a la forme d'un arbre dont les noeuds représentent les tâches et dont les branches modélisent les liens possibles entre ces dernières. On peut voir sur la figure 6 un exemple simple d'arbre de tâches.

Dans la représentation CTT, les tâches sont soit abstraites soit concrètes. Les premières sont représentées par les noeuds qui ne sont pas feuille. Ces tâches ne correspondent pas directement à des tâches comme définies en 2.2.1. Ce sont des abstractions regroupant un ensemble de tâches plus précises et situées sur les noeuds enfant. Ces derniers peuvent être abstraits à leur tour. Les tâches utilisateur, système et interactives n'arrivent qu'en fin d'arbre et constituent les feuilles de l'arbre.

La section 3 de la documentation sur le model CTT **[1]** explique en détail les relations possibles entre les tâches de l'arbre et la signification des symboles sur les branches.

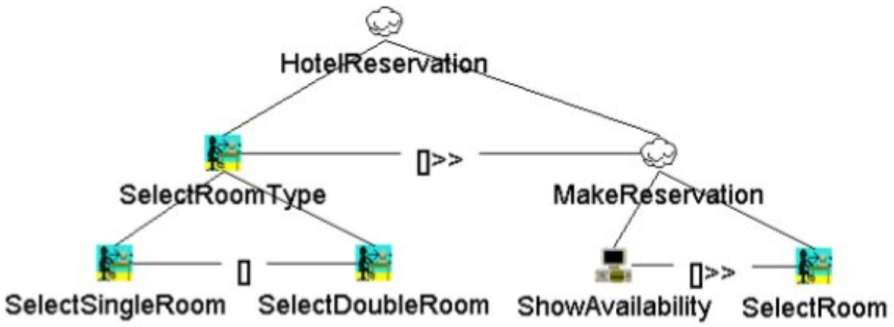

FIGURE 6 – Arbre de tâches décrivant la réservation d'un hôtel en ligne [1]

Une fois la passation de projet effectuée, nous avons pris du temps pour passer en revue les points positifs et négatifs de l'application.

3.1.2 Limitations

a) Lisibilité

Le temps qu'il nous a fallu (environ 1 semaine) pour arriver à comprendre les détails de l'implémentation du serveur témoignait d'un manque de lisibilité. Cela était en partie dû au fait que la structure du code ne respectait pas les bonnes pratiques de conception d'un serveur Web Java utilisant le framework Spring Boot **[2, 3]**. La figure 7 montre la structure en place à ce moment là. La figure 8 montre la structure conseillée. C'est cette dernière que nous avons choisi d'adopter.

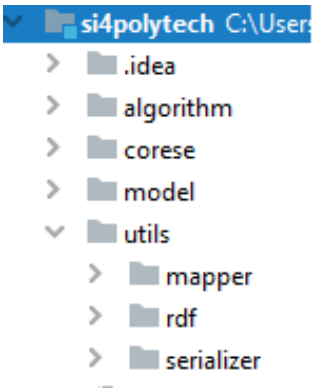

FIGURE 7 – Ancienne structure

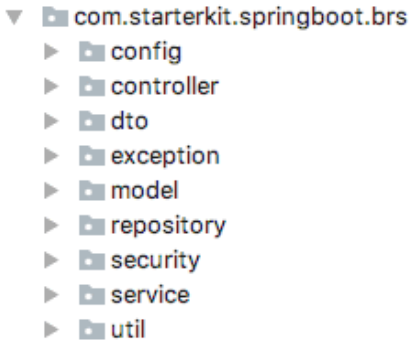

FIGURE 8 – Structure conseillée

b) Utilisabilité

En examinant le code du serveur, nous nous sommes rendu compte qu'il y avait un problème majeur : l'application ne permet pas à plusieurs utilisateurs de s'en servir en même temps. En effet, si cette dernière avait été déployée telle quelle, tous les concepteurs qui l'auraient utilisée auraient manipulé les même données i.e. les même tâches. Si l'application était utilisée localement cela ne poserait pas de soucis mais une fois en production elle serait devenue inutilisable.

3.1.3 Points positifs

L'application existante présente plusieurs éléments positifs que nous avons décidé de conserver.

a) UI

L'interface graphique de l'application, développée avec Angular, ne présentait pas de problème particuliers et avait une base solide à partir de laquelle continuer le développement.

b) Algorithme de calcul des chemins

L'outil permet au concepteur de visualiser les différents enchaînements de tâches, ou chemins de tâches, que peuvent emprunter les utilisateurs de son application. Ces chemins sont extraits de l'arbre des tâches et leur détermination est basée sur les types de transitions entre les tâches. L'algorithme nécessaire pour effectuer ce calcul est assez compliqué c'est pourquoi nous avons jugé qu'il serait une perte de temps trop importante de s'attarder à essayer d'améliorer celui qui existait déjà. Les détails d'implémentation sont fournis par Simon Serrano dans son rapport de stage de SI4, pages 19 à 22.

3.1.4 Bilan

A la suite de cette analyse, nous avons décidé d'extraire l'algorithme de calcul des chemins du serveur puis de le refondre complètement. L'interface a été conservée pour être améliorée.

3.2 Problématiques liées au développement

3.2.1 Interactions et visualisations

a Comment représenter un arbre de tâche + transitions entre les tâches

b Quels peuvent être les interactions possibles entre l'application et l'utilisateur
//TODO Introduction Nous sommes partis du principe que l'outil sera utilisé à partir d'un ordinateur. Par conséquent, les interactions avec le concepteur devront être fait de manière à utiliser les fonctionnalités qu'offre un ordinateur par rapport à une tablette ou un smartphone. Cela comprend des interactions effectuées principalement avec une souris ou un clavier.

c Comment représenter des données sous différentes formes

À partir du moment où l'on compte manipuler une grande quantité de données, il est humainement nécessaire de trouver des visualisations appropriées pour représenter ces données afin d'en faciliter la compréhension et le traitement. On peut représenter des données de beaucoup de façons différentes (ex : D3js [1] : diagramme, carte, graphique, arbres, courbes, tableaux, matrice...). Néanmoins, il faut trouver la bonne forme avec laquelle représenter ces données. Par exemple, si l'on souhaite manipuler des données récursives, une visualisation sous forme d'arbre ou de graphe semble plus adapté. Idem pour des données contenant des informations géographiques ; utiliser un repère visuel tel qu'une carte ou une matrice permet de mieux comprendre la quantité d'information disponible. La carte de John Snow illustre bien

d Comment gérer une grande quantité de données

Une notion importante à prendre en compte lors de la conception de notre interface est la gestion de la quantité de données affichées en simultané. En effet, une interface qui est surchargée d'information ou dont le style est "cassé" sera moins lisible et compréhensible, d'autant plus que des problèmes de performances peuvent survenir s'il y a trop d'élément à charger. Cela entraînerait une perte d'utilisabilité et pourrait décourager les utilisateurs de se servir de l'outil. Dans le cas de notre application où les concepteurs pourraient manipuler de très grands arbres

1. `https://observablehq.com/@d3/gallery`

de tâche, nous sommes directement concerné par le problème. Par conséquent il faut réfléchir à des méthodes de filtrage et des règles pour limiter le nombre d'élément affiché sur l'interface. Des solutions simples existent tels que des systèmes de filtres pour permettre à l'utilisateur d'afficher ou de rechercher les données qui l'intéressent et cacher le reste (Figure 9 et 10). Il existe aussi la pagination. Le principe de la pagination est d'afficher un nombre limité d'élément à la fois, et accéder au reste des éléments en cliquant sur un bouton (11 et 12)

e Retour utilisateur ?

f Technologiquement parlant ? bibliothèques utilisées ? performance ?

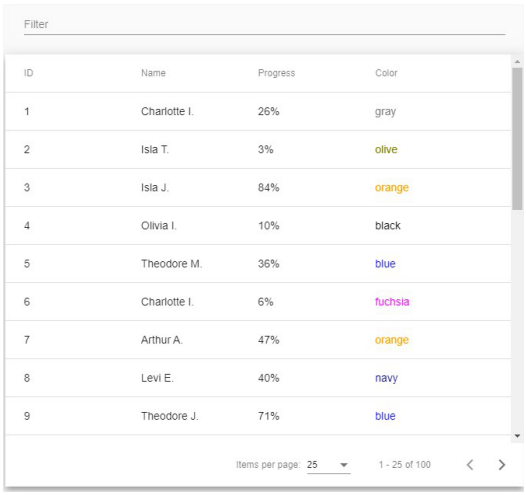

FIGURE 9 – Liste d'élément sans filtre

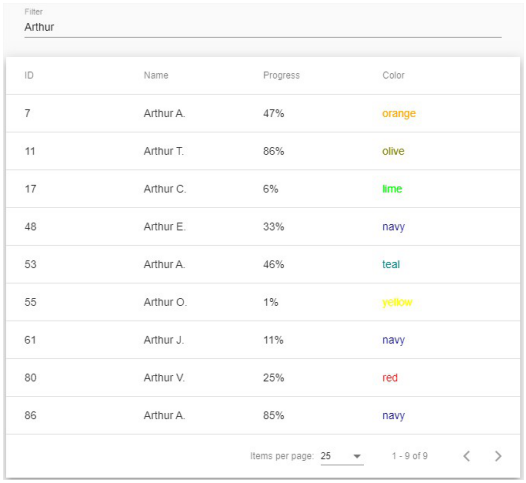

FIGURE 10 – Liste d'élément après l'application d'un filtre

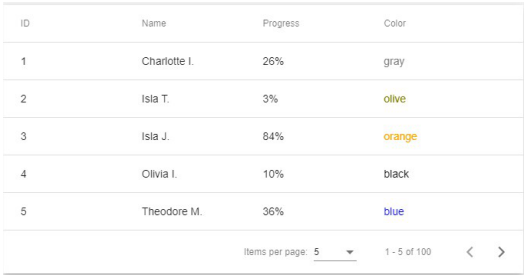

FIGURE 11 – Exemple de pagination page 1

FIGURE 12 – Exemple de pagination page 2

3.2.2 Extensibilité et déploiement

La version de l'application telle que nous l'avions récupéré était constituée d'une partie frontend en angular et d'une partie backend en java. Nous voulions changer le backend qui était un monolithe difficilement maintenable et compréhensible. Une de nos premières problématique a été de trouver une répartition du backend en différents services cohérent pour faciliter son utilisation et son développement tout en essayant d'impacter au minimum le frontend.

Notre application s'appuyant sur la recherche en interfaces distribuées il était également très important de penser cette dernière en ayant toujours une approche permettant son extensibilité . En effet il faut que notre application puisse s'adapter aux exigences évoluant au gré de la recherche.

Nous voulions également que l'utilisation de l'application soit la plus simple possible à utiliser en ayant une installation facile. C'est pourquoi nous avons cherché a déployer l'application pour que les utilisateurs n'aient qu'a aller sur le site web dans leur navigateur pour avoir accès à l'application.

Une autre problématique était de permettre la manipulation de données en parallèle car si notre application est hébergée sur un serveur, il y aura plusieurs personnes qui risquent de l'utiliser en même temps. Il faut un système pour que chaque utilisateur ne soit pas impacté et n'impacte pas le travail des autres utilisateurs.

Cela implique aussi une complexité supplémentaire pour gérer la cohérence des informations entre les différents services

3.2.3 WEB (titre à modifier pour concepts plus proche du métier)

- Comment représenter l'abre des tâches dans le back afin de pouvoir facilement manipuler son contenu pour des opérations de lectures et d'écriture : RDF, SPARQL
- Sous quelle forme récupérer l'entrée utilisateur (le CTT) afin de pouvoir facilement le transformer en données manipulables (lecture/écriture) et que l'on peut télécharger en fin d'édition

3.3 Planning

//TODO répartition des tâches + planning prévu dans le DOW comparer avec le planning final

4 Travail réalisé

contenu : travail réalisé pendant la période à plein temps, organisation, planning, scénarios d'utilisation et démonstration des fonctionnalités.

4.1 Scénarios d'utilisation

//TODO contenu : scénarios d'utilisation, personas

4.2 Solutions apportées

4.2.1 Interactions et visualisations

4.2.2 Extensibilité et déploiement

- Trouver une répartition du backend en différents services cohérents
- Permettre l'extensibilité de l'application
- Déployer l'application pour permettre une utilisation simplifié
- Gérer la cohérence des informations entre les différents services
- Comment permettre la manipulation de données en parallèle ? sessions, formulaire caché, IP, login, ...

Nous avons profité des semaines précédent la partie à temps plein pour étudier l'état de l'application telle qu'elle avait été conçue pendant le stage de Simon Serrano et Valentin Ah-Kane. Nous avons pu en déduire les fonctionnalités disponibles ainsi qu'une partie de leur implémentation. Pendant la première semaine du temps plein nous avons réfléchit à la manière de séparer la partie backend de l'application existante en différents services correspondant aux différentes séparation du traitement des données.
blabla premières idées et pk pas pris
Notre architecture à la fin de ce PFE est composé de cinq services. Un service gérant l'arbre de tache dans sa représentation RDF et utilisant des requêtes SPARQL pour effectuer les différentes actions disponibles à l'utilisateur. C'est également au niveau de ce service qu'est géré l'upload de l'arbre de tache ainsi que les sauvegardes. Un deuxième service est dédié à la représentation des "devices" et des futurs "territoires". Un service gère la représentation des différentes taches, il permet d'autres interactions complémentaires à celles effectuées dans le service RDF avec les requêtes SPARQL. Un autre service gère les "chemins" de tâches. Il permet notamment au frontend de ne pas avoir à recalculer les chemins. Un dernier service annexe permet de convertir les fichier d'entrée provenant de Hamsters au format RDF avec le bon schema pour notre application.
//diagramme d'archi ici
-Deploiement continu Jenkins/Docker... (qu'est ce qui marche,mis sur le coté pk)
-Problème single user -> multi users
-cohérence données
-gestion parallèle
-session, pseudo-session
-etat actuel

4.2.3 WEB (titre à modifier pour concepts plus proche du métier)

- Mettre à jour l'architecture du logiciel en "cassant" le monolithe difficilement maintenable et extensible

- Proposer de nouvelles règles de répartition pour aider le concepteur de différentes manières. Permettre à un utilisateur de construire au préalable un arbre de tâche sur Hamster, générer un xml et le passer à l'application

- Améliorer l'IHM globale en proposant de nouvelles interactions à l'utilisateur concernant l'arbre de tâches... chemins .... Aider l'utilisateur à "filtrer" des grandes quantités d'informations et pouvoir se concentrer sur l'essentiel

5 Conclusion

5.1 Rétrospective

contenu : prise de recul, analyse du travail accompli objectivement, comparaison avec les objectifs initiaux. feedback lors de l'appel vidéo . Feedback+demarche etudiants ihm

- Les résultats sont plus ou moins satisfaisants, certaines améliorations apportées rajoutent des fonctionnalités à l'application. Cependant l'intégration ne s'est faite que très tardivement et a donc retardé le développement d'autres fonctionnalités.

- La démarche IHM à été peu abordée dans ce projet, il était difficile au vu des contraintes temporaires d'avoir des retours sur notre avancée et nous n'avions que peu de temps pour développer l'application. Nous avons quand même suivi les différents retours que nous avons pu avoir sur ce point et effectuer les changements adéquats.

- Nous avons également passé beaucoup de temps au début de la période à temps plein pour réfléchir à l'architecture du serveur qui allait remplacer la version précédente. Ce temps nous a permis d'éviter plusieurs problèmes que nous aurions rencontrés mais nous a mis en retard par rapport à notre planning initial.

- Pour tester notre application nous avons utilisé deux arbres différents, un arbre très basique composé d'une dizaine de taches et un arbre composé d'une trentaine. Nous avons remarqué que pour effectuer certaines actions sur le deuxième arbre nous avions des problèmes de performances ce qui est problématique car l'application se doit d'être utilisable avec des arbres bien plus complexes.

- De même la visualisation des arbres de taille moyenne est parfois peu lisible, plus particulièrement sur la visualisation des chemins. Cela peut être problématique avec des arbres très complexes comme le jeu de société "7 Wonders" par exemple. Pour cela nous avons mis en place un système de zoom permettant de visualiser un sous-arbre pour plus de clarté.

5.2 Perspectives d'amélioration

- Cette application est développée en parallèle à la recherche sur la conception d'IHM distribuées. Des fonctionnalités potentielles sont donc envisageables et pourront être ajouté à l'application dans le futur. Par exemple la notion de territoires différents sur un même dispositif a été abordé mais n'a pas pu être implémenté par manque de temps.

- Nous avons développé cette application en français mais certains concepts sont restés en anglais ce qui donne une inconsistance à l'application. Il faudrait régler ce problème en gardant un seul langage ou en mettant la possibilité de le choisir.

- Certains calculs, servant pour l'affichage, sont effectués au niveau du front-end et sont à l'origine de baisse de performance pour les arbres complexes. Il faudrait migrer une partie de ces calculs un niveau de back-end pour obtenir un résultat plus performant.

- Dans la version actuelle nous n'avons pas implémenté la possibilité de rajouter un dispositif personnalisé mais uniquement une liste de dispositifs prédéfinis, de même pour les règles de répartitions.

- Le système de warning est visible uniquement sur la visualisation des chemins, il serait intéressant d'avoir ces warnings sur les différentes visualisations pour améliorer l'expérience utilisateur.

Références

[1] L. D. S. Fabio Paternò, Carmen Santoro, "Concur task trees (CTT)." https://www.w3.org/2012/02/ctt/, 2012.

[2] A. Khandelwal, "Spring boot 2.0 — project structure and best practices (part 2)." https://medium.com/the-resonant-web/spring-boot-2-0-project-structure-and-best-practices-part-2-7137bdcba7d3, March 2019. 2600+ claps.

[3] "Spring boot - code structure." https://www.tutorialspoint.com/spring_boot/spring_boot_code_structure.htm.

6 Annexes