PRESENTATION DU PROJET DE RECHERCHE ET DEVELOPPEMENT

Table des matières

Présentation de l'organisme

La société KAES a pour activité la conception, le développement et la maintenance d'applications informatiques :

- soit en propre,
- soit pour le compte de tiers en tant que prestataire de service.

Les applications développées et commercialisées directement par KAES sont à destination principalement du grand public et des TPE afin de leur rendre accessible, tant économiquement que technologiquement, des outils modernes et performants.

Dans son activité de prestation de service, les clients visés sont également des TPE innovantes qui souhaitent utiliser ou fournir des solutions basées sur des applications efficaces et élégantes.

Personnels

Les personnels affectés aux travaux de R&D sont ses 2 fondateurs :

- Quentin VUILLIOT, Ingénieur généraliste en génie informatique, diplômé de l'Université de Technologie de Compiègne (UTC)
- Sylvain VUILLIOT, Ingénieur généraliste en agriculture, diplômé de l'Institut Supérieur d'Agriculture de Beauvais (ISAB) et d'un Mastère Spécialisé en Management des Systèmes d'Information Répartis de l'ESSEC et de Télécom Paris

Autres indicateurs de R&D

Nous avons obtenu un avis favorable de la direction générale des finances publiques à la date du 4 mars 2014 au titre de notre premier exercice concernant la qualification en tant que Jeune Entreprise Innovante.

Contexte

Dans le contexte où nous souhaitons proposer des applications pour des clients aux équipements variés et les rendre utilisables dans les différentes situations de leur vie quotidienne, nous devons utiliser des technologies les plus universelles possibles. Dans ce domaine, l'utilisation des technologies du web s'impose et en particulier celles normalisées par HTML5 qui permettent d'utiliser les navigateurs web, non plus comme de simples afficheurs de documents dynamiques avec des hyperliens mais bien comme des environnements d'exécution d'applications hautement réactives et connectées. C'est grâce à cela que l'on peut espérer développer une seule application et la rendre disponible sur la grande majorité des appareils quel que soit leur système d'exploitation (ex: iOS, Android, Windows Phone, Windows, MacOS, Linux, ...).

Cependant, les caractéristiques physiques de ces appareils sont très différentes, tant en terme de taille d'écran (du téléphone 3" à la TV 32"), que de mode d'interaction (tactile, clavier, souris, télécommande), de connectivité (du réseau haut débit permanent à l'absence totale de connexion en passant par la connexion mobile intermittente), de puissance de calcul (du téléphone à l'ordinateur de bureau), ou d'autres encore. Il ne suffit donc pas de développer une application qui fonctionne sur ces différents appareils mais il faut concevoir une application qui s'adapte à ces différents appareils pour rendre son utilisation la plus confortable possible et ainsi la plus efficace possible.

Et ce défi va même jusqu'à concevoir une application qui peut s'adapter à un changement physique de l'appareil y compris en cours de fonctionnement. C'est à dire qu'il ne suffit pas d'être capable de démarrer l'application dans une configuration donnée, il faut aussi s'adapter au changement de cette configuration en cours de vie quand l'utilisateur passe d'une orientation portrait à paysage mais aussi quand il branche une souris sur sa tablette, ou qu'il affiche son téléphone sur un grand écran, quand il passe de sa connection 3G au wifi de son bureau, … et bien d'autres choses que nous n'imaginons pas encore mais qui, dans ce domaine, peuvent apparaître et se démocratiser en seulement quelques années.

L’autre grande ambition de ces applications est que l’utilisateur puisse changer d’appareil quand il change de contexte mais qu’il retrouve la même application, les mêmes données et l’état dans lequel il était sur l’appareil précédent, tout cela sans se soucier des opérations de synchronisation des données. Par exemple, dans une entreprise de pose de fenêtres, le commercial en déplacement peut établir un devis avec son client en utilisant sa tablette, puis planifier la réalisation des travaux de retour au bureau sur son ordinateur avec le détail des stocks, les disponibilités des poseurs, les délais de livraison des fournisseurs, puis être informé de l’état d’avancement par des notifications sur son téléphone.

Un autre domaine d’innovation de nos applications, est de les rendre hautement connectées et capables de dialoguer avec plusieurs services et sources de données simultanément, d’agréger ces contenus et de les mettre à jour pour, par exemple, afficher les fonds de carte de l’IGN, y associer les données communales de l’INSEE et des photos publiques de la zone issues de flickR tout en faisant des relevés de terrain géo-référencés qui, eux, seront partagés uniquement avec les autres collaborateurs de l’entreprise.

Ensuite, nous souhaitons également rendre accessible aux TPE des outils permettant un mode de travail collaboratif en temps réel qui permet à des utilisateurs de travailler sur le même dossier au même moment peu importe leur localisation.

C’est ce type d’applications pionnières et innovantes que la société KAES a pour ambition de concevoir et de développer pour le bénéfice de ses clients. Cela nécessite un effort important en recherche et développement technologique pour transformer ces promesses en outils utilisables simplement et aussi vite que possible par nos clients utilisateurs et nos clients donneurs d’ordre.

Bien sûr, les technologies de base existent, et la communauté open-source met à disposition de très nombreux outils pour les utiliser. Mais ce sont des technologies jeunes, naissantes, évoluant rapidement, plutôt instables, parfois incomplètes et pas forcément adaptées aux types d’applications que nous réalisons. D’où la nécessité d’un travail de recherche, d’expérimentation, d’évaluation, de prototypage et de validation par la pratique pour trouver

des solutions aux différentes problématiques rencontrées.

Ce travail, que l'on peut qualifier de "recherche appliquée", aboutit à des résultats que nous utilisons pour notre propre compte mais que nous souhaitons également rendre public dans la philosophie open-source en les publiant sur notre blog et également en mettant sous licence ouverte le code source de notre "cadriciel". En effet, nous sommes amenés à concevoir, développer et mettre au point des techniques, des outils, des briques logiciels, pour la réalisation de nos applications et les fruits de ce travail sont matérialisés par la constitution d'un "cadriciel" et de sa documentation associée qui aident au développement des types d'applications que nous réalisons. C'est un actif technologique que nous allons alimenter et améliorer en continu car il constitue, à la fois le socle technologique des applications que nous développons aujourd'hui et également, dans ses versions non stables, celui de nos prototypes des applications de demain.

Présentation des domaines d'étude et travaux en cours

Dans le cadre de notre projet de R&D, nous allons détailler dans ce chapitre les domaines technologiques pour lesquels nous avons identifié des limites voire des contraintes techniques fortes et dans lesquels nous avons l'ambition de mener des travaux de R&D pour tenter de les lever.
Pour les domaines pour lesquels nous avons commencé des travaux, nous présenterons également l'état d'avancement de ces travaux et, pour les plus avancés, les résultats obtenus.

Stratégie de réutilisation de logique en JavaScript

Contexte

JavaScript (JS) est un langage orienté objet mais il est dynamiquement typé et propose un mécanisme de réutilisation de logique par délégation de prototype, ce qui en fait une exception parmi les langages de programmation utilisés industriellement (Java, C#, Python, Ruby, PHP). De plus, même si JavaScript existe presque depuis le début d'internet, il n'y a vraiment que depuis 3 ou 4 ans qu'il est utilisé réellement comme un véritable langage de programmation et non pas comme un simple langage de script pour animer une page web.

Ainsi il n'existe pas encore de consensus sur la stratégie à employer pour permettre la réutilisation de logique applicative et permettre une utilisation industrielle du langage qui favorise la maintenabilité.

Problématique

La problématique principale de ce domaine d'étude est donc de trouver une stratégie de réutilisation de logique en JavaScript qui, à la fois :

- permette une réutilisation simple des fragments de logique
- tire parti de la délégation par prototype
- soit la plus universelle possible (évite de s'enfermer dans une technologie non pérenne)

Travaux menés

A ce sujet, la communauté des développeurs JavaScript est très productive en terme de solutions, mais cela prouve aussi qu'il n'existe pas de standard établi et que personne n'est réellement satisfait de ce qui existe.

Notre travail a donc consisté à réaliser une étude comparative des solutions les plus populaires en les appliquant à un cas d'exemple récurrent dans nos applications : un composant avec des fonctions de stockage de données, d'observabilité, de gestion de composants et de rendu graphique.

En synthèse, nous avons pu établir que les solutions testées pouvaient être regroupées en 4 catégories :

- celles qui tentent de reproduire le modèle des langages par "classes" tels que Java

ou C# et qui implémentent une fonction d'héritage multiple de classes.
Elles ont l'avantage de présenter une grande simplicité de mixage des fonctionnalités. Mais cette simplicité d'utilisation est rendue possible par l'exécution d'algorithmes complexes de résolution des méthodes/propriétés conflictuelles, où l'ordre de la chaîne d'héritage est important. Ces conflits sont courants puisqu'il s'agit de concilier des classes entières, qui exposent souvent quelques méthodes de base (ex: toString).
De plus, ces solutions sont théoriquement limitées aux cas où la classe héritée doit pouvoir se substituer à ses classes mères, or ces cas sont finalement assez rares dans nos applications.
Enfin, cela nécessite l'utilisation d'une librairie spécifique, comme par exemple : dojo/declare.

- celles qui applique le principe des "mixin", c'est à dire des "morceaux" de classe qui sont fait pour être collés les uns aux autres pour former une nouvelle classe. Ce principe utilise en fait les mêmes mécanismes que l'héritage ci-dessus, mais diffère par le fait que les mixins sont des éléments de composition, et non pas des classes à part entière : un mixin n'est pas utilisable directement.
  Utiliser les mêmes mécanismes que l'héritage pose aussi les mêmes problèmes de conflits, mais sont théoriquement moins fréquents puisque les mixins sont supposés fournir un comportement spécifique, les chevauchements sont donc normalement plus rares. Les mixins doivent être orthogonaux pour bien fonctionner ensemble.

- celles qui privilégient l'approche par composition. Ici, il n'est pas question d'héritage mais de délégation. C'est à dire que si un objet doit implémenter une fonction, au lieu de le faire directement, il peut se contenter de la déléguer à un autre objet spécialisé qui lui est réutilisable. C'est l'approche la plus séduisante d'un point de vue théorique car chaque fonctionnalité est supportée par un objet spécialisé, ces objets sont composés dans des objets plus complexes et ainsi de suite. Il n'y a pas de problème de conflits de nom, ni de problème d'ordre d'héritage.
  Mais d'un point de vue pratique, cette approche nécessite beaucoup de code d'assemblage sans valeur ajoutée (tel que la ré-exposition simple de méthodes et de propriétés) ce qui rend son utilisation fastidieuse dès qu'on veut l'employer à large échelle.

- celles qui implémentent le concept de "traits". Proche du concept de mixins, les "traits" sont des éléments de compositions, mais qui ne présupposent jamais l'existence de méthodes de base (pas de surcharge de méthode), et n'ont pas d'état (propriétés). L'ajout d'un état se fait au moment de la composition de la classe. Les traits peuvent définir des prérequis comme l'obligation de l'existence de méthodes pour accéder notamment à cet état.
  La résolution des conflits se faisant au moment de la composition, le système de composition des traits offre plus de contrôle que l'héritage multiple, et permet donc d'avoir la main sur d'éventuels conflits. On peut par exemple renommer ou exclure une méthode d'un trait lors de la composition. Un des points clés de cette solution est que l'ordre de composition n'a pas d'importance.
  Mais la mise en oeuvre de ce concept, nécessite de recourir à une librairie spécifique,

comme par exemple : trait.js.
Un autre inconvénient est que les prérequis des traits ne sont pas configurables. Si deux traits requièrent une méthode portant le même nom mais pour chacun un comportement différent, il existe toujours un conflit non résolvable.

Résultats obtenus

Les concepts de traits et de mixins nous sont apparus les plus intéressant, car ils apportent des solutions à la problématique de composition en permettant une bonne maîtrise de celle-ci. Nous avons donc cherché à résoudre les problèmes qui nous faisaient encore défaut, à savoir :

- la non-configurabilité des méthodes pré-requises ou de l'accès à l'état de l'objet
- le recours à des librairies spécifiques

Nous nous sommes donc basés sur le concept de traits, en permettant la configuration des prérequis, et en essayant de s'affranchir d'une librairie particulière pour leur utilisation. En s'appuyant sur la souplesse du langage Javascript, nous avons abouti au concept de "générateur de trait", qui est une fonction prenant en argument une configuration, permettant ainsi de changer à la volée le nom des méthodes ou propriétés requises. Le renommage ou exclusion des méthodes pouvant être reporté au moment de la composition proprement dite.

L'opération de composition peut, elle, être faite en JavaScript natif, ou via un utilitaire simple laissée au choix du composeur.

Conclusion

Cette solution nous permet d'isoler des fonctionnalités dans des fichiers différents, en assurant une réutilisation maîtrisée dans des contextes différents. La maintenabilité, la fiabilité des applications et le temps de développement s'en trouvent améliorés.

Agencement graphique avancé

Contexte

Par "agencement graphique avancé", nous entendons les mécanismes qui doivent nous permettre de créer des interfaces graphiques destinées à un acteur humain, et ceci avec le moins de contraintes techniques possibles pour assurer l'ergonomie souhaitée. L'objectif est d'être capable de reproduire au plus proche, sur l'écran, le croquis d'un designer.
Cela doit évidemment être dynamique, c'est à dire fonction de la taille de la zone d'affichage, et de ses redimensionnements potentiels. Cela signifie aussi que des traitements doivent pouvoir être exécutés au changement de taille d'un composant graphique (ex : une carte qui s'agrandit devra demander de nouvelles informations cartographiques au serveur).

Problématique

La technologie Web souvent considérée responsable de cette tâche est Cascading Style Sheet (CSS). Mais elle ne répond pas à tous les besoins :

- des agencements trop sophistiqués, non prévus par la norme, doivent être faits en JavaScript (JS). La réalité actuelle étant que les agencements prévus par la dernière

version de la norme CSS3 pour réaliser des applications Web ne sont pas tous implémentés dans les navigateurs actuels, et même s'ils l'étaient, ils ne répondent pas à tous les cas de figures possibles. Il n'est par exemple pas possible de spécifier une hauteur en fonction de la largeur.
- il est impossible de détecter un changement de taille, et donc impossible d'exécuter un traitement si besoin

CSS n'étant pas extensible, il faut donc trouver un système plus souple pour répondre à cette problématique, qui n'a d'autre choix que de reposer sur JS.

Etat de l'art

Des librairies de composants graphiques en JS existent, et disposent généralement d'un catalogue bien fourni de composants qu'il serait intéressant de réutiliser tel quel. Mais en pratique dans notre cas, cette possibilité ne s'est pas prouvée être réaliste :

- dojo/dijit : ne prévoit pas que les composants soient retirés de leur conteneur sans les détruire, ce qui le rend incompatible avec notre concept d'interfaces adaptables (cf : "Interfaces graphiques adaptables").
- Sencha ExtJS : même problème que dijit, en plus d'avoir une licence plus contraignante et de ne pas être compatible AMD, donc difficilement réutilisable en association avec d'autres librairies.

Travaux menés

Même si nous ne pouvions pas réutiliser directement les librairies mentionnées ci-dessus, les mécanismes utilisés dans celles-ci ont été des sources d'inspiration.

Le principe général consistant en un arbre de composants graphiques, avant tout en JavaScript, mais pouvant déléguer certaines tâches au CSS pour des questions de performance, nous a semblé être le bon.

Cet arbre est composé de noeuds conteneurs et d'enfants pouvant être à leur tour conteneurs. Les conteneurs dimensionnent leurs enfants, à moins que les enfants ne soient responsables eux-mêmes de leur taille. Cela forme donc des chaînes entières de composants graphiques contrôlés, pour chaque maillon, par du code JS. Si cette chaîne est rompue, il devient impossible pour les composants suivants ou précédents d'être avertit d'un changement de taille.

La maturation de ce mécanisme nous a amené à discerner notamment les notions de taille, contraintes de taille (min/max), dans/hors DOM (Document Object Model). Nous avons aussi introduit la possibilité de désactiver la mise à jour du rendu, pour les composants non affichés, par exemple dans un onglet de navigateur inactif, ou dans une zone hors écran, afin d'économiser du temps de calcul.

Conclusion

Sur ce sujet, nous étions d'abord partis avec l'idée reçue que le CSS et ses dernières évolutions pouvaient satisfaire les besoins inhérents aux agencements graphiques requis pour nos applications. Mais nous nous sommes assez rapidement rendus compte que la réalisation d'agencement graphiques même courants allaient bien au-delà des possibilités

actuelles de CSS.

La base technique que nous avons acquise actuellement a fait ses preuves, mais le tour du sujet n'a pas été fait. Des questions restent en suspens en autres pour le glisser/déplacer, les panneaux glissants ou les affichages temporaires en surimpression (info-bulle, liste déroulante).

Interfaces graphiques réactives

Contexte

Le principe d'une interface graphique réactive est de réagir automatiquement et en temps réel aux modifications qui ont lieu sur les données, qu'elles soient issues de l'utilisateur lui-même, d'une autre application connectée aux mêmes données, d'un changement de l'environnement (passage de en-ligne à hors-ligne ou inversement, …). Le principal avantage est que l'utilisateur n'a pas besoin de demander explicitement le "rafraîchissement" de son écran ou bien que celui-ci soit fixé à un intervalle de temps donné. Lorsque les données changent, leur traduction graphique change également. Par exemple, lorsque l'utilisateur change le titre d'un document dans un champ de saisie, celui-ci est mis à jour en temps réel dans le menu qui liste ce document et partout où ce titre est affiché, sans attendre une validation par un bouton 'enregistrer'.

Cela nécessite des mécanismes d'observation qui n'existent pas nativement en JavaScript et donc nécessitent l'implémentation d'un outillage spécifique.

Mais il faut également prendre en compte les problématiques de performance lors de la mise en oeuvre de ce type d'interface car elles entraînent des recalculs très fréquents. C'est pourquoi il faut s'arranger pour limiter ces calculs à ce qui est strictement nécessaire et ne pas mettre à jour par exemple des éléments graphiques qui sont hors de la zone d'affichage ou réduits : dans notre exemple, si le menu était un panneau rétractable en position "replié", il faudrait éviter de le mettre à jour en temps réel et ne le faire que lorsqu'il serait déplié.

Etat de l'art

Comme pour les autres sujets, il existe des solutions libres pour traiter ce sujet. Notre premier travail a donc été d'en faire l'inventaire puis de réaliser une étude comparative des solutions les plus adaptées à notre domaine.

Voici le résultat de cette étude :

- AngularJS
    - observation des changements par "dirty checking", c'est à dire qu'à chaque action qui a un impact potentiel sur les données, tous les observateurs sont notifiés, charge à eux de ne pas réagir si la donnée qu'ils observent n'a pas changé effectivement. Cette technique est simple mais elle devient peu performante dès que le nombre d'observateurs augmente.
    - la technologie d'observation ne peut pas être utilisée de façon indépendante en dehors du framework
- KnockoutJS

    - déclaration en html via des attributs spécifiques (nécessite donc d'écrire des trames en html)
    - observation et réactivité possible uniquement entre le DOM et un objet JS (pas de réactivité possible entre deux objets JS)
- FRB
    - librairie indépendante
    - liens réactifs possibles entre deux objets JS
    - mise à jour incrémentale des objets (forte optimisation de la performance)
    - uniquement un mode déclaratif (difficulté de créer des liens réactifs à un plus bas niveau d'abstraction)
    - pas de possibilité d'observer deux changements en même temps
    - pas de possibilité de suspendre un lien réactif pendant une période de temps

Problématique

Suite au constat des limitations des solutions disponibles, nous avons décidé de développer notre propre technologie d'observation et de réaction optimisée avec :

- limitation du nombre d'événements (un seul événement 'change' lorsque plusieurs propriétés sont modifiées simultanément)
- observation possible d'un objet en mode différentiel ce qui est particulièrement efficace pour les listes où, le plus souvent, on procède par ajout ou suppression d'un nombre limité d'éléments parmi un grand nombre
- débrayage possible en fonction de critères
- facilité d'utilisation via un mode déclaratif pour les cas simples (mais en ne se limitant pas au mode déclaratif)
- utilisable entre un objet JS et le DOM et entre plusieurs objets JS

Travaux réalisés

- Comparaison des stratégies d'observation possibles
    - utilisation d'événements (comme dans BackboneJS)
    - utilisation de getter/setter ES5 qui appellent des listeners (comme dans FRB)
    - utilisation de méthodes "get" et "set" qui appellent des listeners (comme dans dojo/Stateful)
- Prototypage d'une solution de liens réactifs implémentée selon le paradigme FRP ("Functional Reactive Programming", Programmation Réactive Fonctionnelle) pour permettre de créer des liens hautement configurables
- Etude des solutions possibles pour les cas de liens réactifs bi-directionnels. En effet, il est courant de relier 2 objets de façon bi-directionnelle pour que, lorsque la donnée A change, l'objet graphique qui la représente change, mais aussi que si l'objet graphique réagit à une interaction de l'utilisateur, son changement soit reporté sur la donnée A. Cela conduit potentiellement à un effet de boucle infinie qu'il faut donc éviter.
- Etude de l'intérêt d'une solution avec une réaction asynchrone : la réaction aux événements n'a pas lieu immédiatement mais est différée dans la file d'attente (en cours).
- Prototypage d'un mécanisme d'observation incrémental : les observateurs ne sont pas simplement informés qu'un changement général a eu lieu, ils peuvent être

notifiés du détail des changements unitaires qui ont eu lieu
- Recherche d'un algorithme de calcul de l'index final d'un élément d'une collection ordonnée à partir de la collection initiale et d'une liste de changements unitaires (ajouts et suppressions)
- Prototypage de liens réactifs entre N objets avec N > 2
    - un seul objet dans une liste peut avoir une propriété donnée à "vrai", pour les autres elle est à "faux"
    - synchroniser plusieurs objets (dès que l'un change, les autres changent de la même façon)
- Etude des intérêts et contraintes de liens réactifs bidirectionnels préventifs : si un objet A et un objet B sont liés entre eux de façon bidirectionnelle et que A souhaite changer, B pourrait refuser ce changement avant même qu'il n'ait lieu.
- Prototypage d'un mécanisme de lien réactif réflectif, c'est à dire un lien qui modifie l'objet A en réaction à une modification de l'objet A lui-même. Cela est utile pour définir des propriétés calculées (la surface d'un rectangle est recalculée dès que sa largeur et/ou sa longueur change) mais aussi des contraintes du type "la valeur de la propriété 'selected' doit appartenir à l'ensemble A", donc si l'utilisateur vient à spécifier une valeur en dehors de l'ensemble autorisé, il faut réagir et faire un changement qui annule son changement.

Conclusion

Les travaux dans ce domaine nous ont permis de créer des composants applicatifs plus rapidement et plus facilement avec une capacité de configuration des liens réactifs beaucoup plus puissantes que les solutions existantes. En particulier, notre solution permet d'observer plusieurs changements simultanément et ainsi de pouvoir les combiner ce qui n'est pas possible avec les solutions existantes.

Cependant, cette technologie est encore expérimentale et il nous reste à traiter le cas des liens réactifs réflectifs qui semblent prometteurs mais difficiles à traiter.

Interfaces graphiques adaptables

Contexte

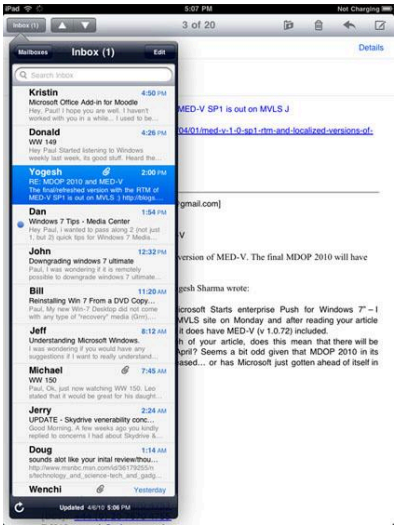
Un autre aspect des interfaces graphiques modernes que l'on souhaite créer est leur capacité à s'adapter dynamiquement à leur contexte :

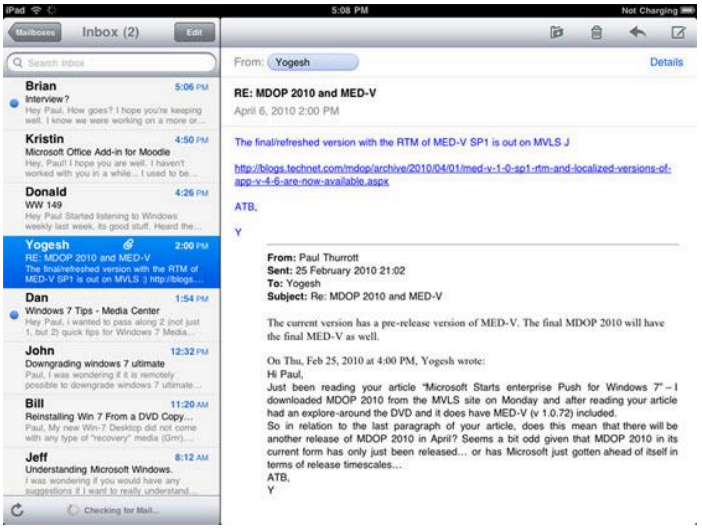
- en terme de taille et de forme d'écran (ratio hauteur/largeur)
- en terme de dispositifs d'interaction et de leurs combinaisons : souris + clavier + écran non tactile, écran tactile seul, écran tactile + clavier, commande vocale, …
- en terme de mode d'utilisation : mode "mobilité" (interface allégée qui ne donne accès qu'aux informations principales), mode "bureau" (possibilité d'accéder à toutes les fonctionnalités, via une interface plus dense), mode "lecture seule" (garantit que l'on ne modifiera pas des données par inadvertance), mode "débutant" (mise en avant des commandes les plus courantes), …

La possibilité de définir ces différents modes et de pouvoir basculer à la demande de l'un à l'autre nécessite de pouvoir réagencer les composants d'affichage dynamiquement, d'en

masquer certains, d'en afficher d'autres et de modifier éventuellement ceux existants.

Prenons l'exemple d'une application qui en mode portrait n'affiche que le panneau principal et permet à l'utilisateur de faire apparaître le menu soit en appuyant sur un bouton, soit en faisant glisser son doigt. Par contre, en mode paysage, elle affiche les deux panneaux côte à côte : plus besoin du bouton d'affichage du menu ni de l'écoute du geste d'ouverture du menu.





En effet, le fait d'utiliser le même composant dans deux agencements différents, plutôt que deux instances de la même classe même reliées aux mêmes données est que :

- l'état du composant est conservé (par exemple le niveau de zoom et le centre d'une carte géographique ne sont pas réinitialisés)
- cela est plus facile à gérer pour le développeur qui n'a qu'un seul composant à gérer et pas plusieurs à synchroniser
- on économise de la mémoire et du temps de traitement car il n'y a qu'un seul composant à instancier

Problématique

L'enjeu de ce domaine d'étude est de permettre la définition de ces différents modes et des combinaisons de ces modes d'une façon à la fois flexible et simple. C'est à dire qu'il faut pouvoir définir des composants et leurs multiples agencements, ainsi que la logique qui permet de passer de l'un à l'autre d'une façon déclarative et non pas impérative (c'est à dire qui limite le code à écrire au profit d'une convention de déclaration).

Il faut également que la solution permette de déclarer les liaisons ("bindings") à activer dans un agencement pour qu'un moteur de changement d'agencement puisse activer ceux qu'il faut, désactiver ceux qui ne sont plus nécessaires et laisser les autres intacts.

Etat de l'art

A notre connaissance, il n'existe pas de solution qui traite de cette problématique de description de plusieurs agencements avec mutualisation des composants pour un réagencement à la volée sans devoir détruire et recréer des composants.

Les solutions déclaratives actuelles, soit en HTML, soit en JavaScript, soit dans un DSL (“Domain Specific Language”) ne permettent pas de traiter le cas d’un même composant utilisé dans plusieurs agencements.

Synthèse des travaux réalisés

Les travaux de recherche réalisés ont abouti à la conception et au prototypage d’une solution dans laquelle on peut déclarer indépendamment les composants graphiques d’une part et leurs différents agencements possibles d’autre part.

La déclaration des composants consiste simplement à leur attribuer un identifiant textuel dans un registre. Puis, il est alors possible de décrire déclarativement des agencements de ces composants sous forme d’arbres écrits en JavaScript dont les noeuds sont des identifiants de composants.

Cette solution autorise également à utiliser directement des composants dans la description d’un arbre d’agencement, ce qui permet de créer des composants spécifiques à un agencement à la volée sans devoir les enregistrer d’abord mais avec la contrainte qu’ils ne sont pas réutilisables dans d’autres agencements. Cela s’avère pratique pour les composants de type “conteneurs” qui n’ont pas besoin d’être réutilisés et qu’il est fastidieux d’enregistrer avec un identifiant.

Conclusion

Cette solution nous a permis de développer plus vite et plus rapidement des composants adaptables avec l’énorme avantage de pouvoir réutiliser les mêmes composants dans différentes configurations.

Cependant, lorsque l’on souhaite réutiliser directement des fragments d’arbre entre plusieurs agencement, le mode déclaratif n’est plus possible. Heureusement, on peut alors recourir au mode impératif, moins pratique, moins visuel mais plus puissant. Mais c’est parfois le signe qu’il faut re-décomposer le composant pour déléguer certaines fonctionnalités à des composants spécialisés.

Application autonome (modèle de données riche)

Contexte

Traditionnellement, les applications web sont un simple affichage déporté des données accessibles sur un serveur central. C’est ainsi que lorsque l’on veut consulter une information ou réaliser une modification, il y a systématiquement un délai d’attente lié à la latence du réseau et au temps de réponse du serveur.

Pour le type d’applications que nous souhaitons développer, il est possible d’opter pour une autre approche qui consiste à considérer que l’application cliente doit être une application riche et qu’il faut lui permettre de réaliser un maximum de traitements. C’est à dire que l’application cliente doit connaître la logique métier et être capable de l'exécuter en autonomie, indépendamment d’un serveur central.

Dans certains cas, on souhaite même que l’application cliente puisse fonctionner

complètement “hors-ligne”, c’est à dire sans devoir faire appel à un serveur, même pas pour la persistance des données. Et c’est même un gage de robustesse et de confort car cela assure à l’utilisateur de pouvoir utiliser son application en consultation comme en modification quelles que soient les conditions de réseau. Il ne faut pas croire que cela ne soit utile que pour les applications en zones rurales reculées. Dans bien des cas, on rencontre des difficultés de connexion en milieu urbain (dans les transports souterrains, dans certains immeubles, …) et il n’est pas courant d’avoir du réseau dans les avions. Ainsi, l’utilisateur s’affranchit des problèmes de connexion lente ou intermittente, de l’anxiété de savoir si ses dernières modifications vont finalement pouvoir être enregistrées ou pas, et, même en situation de bonne connexion, le confort est amélioré car on élimine complètement l’effet de la “petite roue qui tourne”.

Notre ambition est donc de permettre à nos applications clientes d’inclure une vraie couche métier pour les rendre autonomes :

- pouvoir travailler complètement localement si besoin
- pouvoir exécuter localement les traitements (autant que possible) afin de gagner en réactivité (pas de latence réseau et déchargement des serveurs)
- pouvoir travailler même sans connexion internet

Problématique

La conception des outils permettant de développer un modèle de données riche dans une application web cliente est déjà un sujet de recherche en soi car aucune solution ne semble exister pour le moment. Mais ce qui nous occupera le plus dans ce domaine d’étude est la recherche et la mise au point d’une solution robuste de réplication de données différée dans le temps et en particulier la problématique de la réconciliation de modifications divergentes.

Car, bien sûr, l’application cliente sera autonome, mais la volonté sera de reporter les modifications faites localement sur un serveur central pour les partager avec d’autres utilisateurs et/ou les autres appareils du même utilisateur.

Il s’agit, ici, de travailler en particulier sur les stratégies de gestion des conflits de données qui arrivent inévitablement lorsque plusieurs utilisateurs travaillent simultanément sur les mêmes données mais qui ne peuvent être résolus par un mécanisme de sémaphore dès lors que les données sont répliquées et modifiées en dehors de tout contrôle central.

Synthèse des travaux réalisés et prévisionnels

- Conception et prototypage d’une architecture avec un modèle de données riche, gestion des relations entre entités métier et persistance locale
- Conception et prototypage d’un système de réplication bi-directionnelle asynchrone
    - bi-directionnelle, c’est à dire que des modifications peuvent être faites d’un côté ou de l’autre et doivent être propagées dans un sens et dans l’autre
    - asynchrone, c’est à dire qu’une modification effectuée d’un côté n’est pas immédiatement répercutée de l’autre car les deux systèmes ne sont pas toujours connectés. Et même lorsqu’ils le sont, on souhaite s’affranchir de la latence du réseau.
- Recherches sur l’optimisation des échanges via une mise à jour incrémentale
    - Dans certains cas, les entités métier ne seront modifiées que partiellement

(par exemple ajout d'un élève à un cours) et il sera alors plus opportun de n'envoyer que cette information au lieu de renvoyer la liste complète de tous les élèves.

- Conception et prototypage d'une technologie de gestion des conflits de données adaptée à nos cas d'applications
    - Outils de détection de conflits et de mise en évidence des écarts
    - Algorithmes de résolution automatique de conflits adaptés aux domaines métier concernés
    - Outils d'aide à la résolution de conflits
- Recherches sur la mise en oeuvre de métadonnées de réplication
    - L'utilisateur doit être informé de l'état de fraîcheur des données qu'il consulte ou qu'il modifie. En d'autre termes, il doit savoir si ce qu'il voit à l'écran représente l'état des données telles qu'elles sont sur le service distant à l'instant T ou, au contraire, si ce sont des données qui datent de plusieurs jours éventuellement car il n'y a pas eu de réplication depuis. De même il doit être informé quand les modifications qu'il a réalisé localement ont été poussées vers le serveur distant et le cas échéant si elles ont été acceptées ou non. Ces informations doivent être structurées selon une convention qui les rend compréhensibles et utilisables par plusieurs types de vues.
- Auto-génération de certains composants de la couche de données à partir de la déclaration de schémas
    - Pour faciliter et accélérer le développement d'application, il faudrait pouvoir se contenter de décrire la structure des données principales via des "schémas de données" et que cela soit interprété par l'application pour auto-générer les composants nécessaires : constructeurs, sérialiseurs, synchroniseurs, …

Conclusion

Ce domaine est un axe prioritaire de notre investissement en R&D, car il devrait nous permettre de proposer des fonctionnalités fortement différenciantes et à forte valeur ajoutée dans nos applications. Mais c'est également un domaine complexe et risqué, où la robustesse et la fiabilité des technologies est primordiale et, donc, nécessite un investissement lourd.

Interaction et intégration de plusieurs services et sources de données

Contexte

L'intérêt des applications développées avec des technologies web est qu'elles ont une capacité naturelle à être connectées et sont donc capables de dialoguer avec au moins un service de donnée via le web. Notre ambition est de permettre à nos applications d'être hautement connectées et donc capables de dialoguer avec, non pas un, mais plusieurs services web simultanément, d'agréger ces données et de les présenter de façon uniformisée à l'utilisateur.

Problématique

La difficulté pour ce domaine d'étude consiste principalement dans le fait de synchroniser les différents services entre eux : par exemple, lors de l'enregistrement d'une ressource, il

faudra potentiellement pousser des données vers plusieurs services en même temps avec le risque que pour l'un d'entre eux, on ait une erreur (erreur réseau, données erronées, conflit, …). On se retrouverait donc à avoir une sauvegarde partielle de la ressource. C'est ce genre de cas qu'il faudra tenter de résoudre pour permettre la mise en place de cette fonctionnalité de façon robuste.

Ces travaux reposeront en partie sur les standards pour l'interopérabilité des données et en particulier sur ceux concernant le web, que l'on regroupe sous la terminologie de "web sémantique".

Dialogue avec des services web en temps réel

Une tendance forte des applications modernes connectées est de pouvoir écouter le flux de données d'un service en temps réel afin de réagir dès qu'une nouvelle information est diffusée. Dans ce champ d'étude, il n'est pas tant question des technologies à utiliser pour permettre un dialogue client/serveur en temps réel que de structurer et normaliser cet échange pour le rendre interprétable, par les composants applicatifs, à un niveau plus élevé d'abstraction de l'application.

Il serait même souhaitable que l'on puisse concevoir et architecturer l'application sans connaissance à priori de l'existence d'une communication en temps réel ou en temps différé et même de pouvoir utiliser ces deux modes simultanément selon le besoin de réactivité ou les contraintes du réseau.

L'enjeu consiste donc à concevoir une architecture qui permette de faire abstraction de cette différence entre une communication en temps réel ou en temps différé. Ce qui n'est pas évident car, au delà de la technologie de transport utilisé et de la fréquence des échanges, il existe d'autres différences marquantes :

- l'un (le modèle d'échange REST non temps réel) est normalisé au niveau de sa sémantique (requête, réponse, métadonnées, auto-négociation de type de contenu, uri, …), pas l'autre
- l'un (le modèle d'échange REST non temps réel) consiste à envoyer une représentation complète de la ressource alors que l'autre décrit plus facilement la liste des modifications à apporter pour modifier l'état d'une ressource (notion de mise à jour incrémentale).

Modèle de données dynamique et personnalisable par l'utilisateur

Notre souhait, pour les applications qui s'y prêtent, est de permettre à l'utilisateur de personnaliser le modèle de données en général en l'enrichissant avec des éléments qui lui sont propres, un peu de la même façon que l'on peut écrire entre les lignes d'un formulaire ou ajouter des annotations.

En effet, la plupart des applications actuelles de gestion de données structurées ne permettent pas une personnalisation souple du modèle de données, parfois non pas par

dessein mais souvent pour des contraintes techniques.

Or, les technologies actuelles de bases de données de type “document” permettent de décrire des schémas de données moins rigides et de les faire évoluer de façon simple. Il nous semble donc important d’investiguer dans ce domaine pour rendre possible effectivement ce type de fonctionnalités dans nos applications.

Productivité de la réalisation d’applications

S’adressant prioritairement à de petites structures, une de nos ambitions est de réduire le temps de développement et donc le coût de réalisation des applications que l’on proposera à nos clients. Pour cela, la piste principale est tout simplement d’adapter les bonnes pratiques de génie logiciel au domaine du développement web qui est encore peu industrialisé par rapport à d’autres écosystèmes de développement comme C#, Java ou Python. En effet, les technologies du web (Javascript, HTML, CSS) n’ont pas été conçues au départ pour développer des applications mais uniquement pour décrire et présenter des documents liés entre eux. Cela est en train de changer à grande vitesse en particulier avec HTML5 qui incite à ce que ces avancées soient supportées dans les navigateurs modernes mais il reste encore du chemin à parcourir.

Nous chercherons comment mettre en place, avec les technologies web, une architecture applicative qui permette une approche par composition de composants faiblement couplés (graphiques et non graphiques) et facilite ainsi la maîtrise des enjeux suivants :

- encapsulation
- réutilisabilité
- maintenabilité
- testabilité

Tests intégrés automatisés

Actuellement, les solutions pour l’automatisation des tests unitaires sont relativement matures et il est possible de les utiliser presque directement pour nos propres besoins. Cependant, dans le domaine des tests intégrés, c’est à dire des tests qui s’assurent de l’adéquation de bout en bout d’une application aux spécifications métier, l’effort de recherche est important car il n’existe pas actuellement de solution adaptée et encore moins de solutions automatisées.

Expérience utilisateur (User eXperience)

Notre effort de R&D sera également orienté vers des aspects liés à la conception logicielle et en particulier autour de l’expérience d’utilisation. En effet, les applications web sont perçues de par leur nature et leur contexte d’utilisation comme des applications simples et ludiques par opposition aux applications de bureau traditionnelles qui sont plus complexes et souvent plus austères. C’est à dire qu’avec la notion d’application web, le client a également la

volonté d'avoir une application simple, qui ne nécessite pas de phase d'apprentissage de la part des nouveaux utilisateurs, une application intuitive. Cela nécessite donc d'inventer de nouvelles solutions pour masquer à l'utilisateur la complexité de l'application, et lui apporter des fonctionnalités avancées de manière la plus simple possible.

Les travaux porteront sur des questions telles que :

- le retour à l'utilisateur sur l'état de synchronisation de données relationnelles
- l'état de validité des données de façon non intrusive et non bloquante
- des messages d'aide à l'utilisation compatibles avec une interaction tactile (pas de survol de souris)

Cependant, ce domaine est très vaste et il nous sera nécessaire de nous allier avec des spécialistes du sujet au cas par cas en apportant notre vision de concepteurs et de développeurs.