Nom'ad : une solution de télégestion sur Nokia 6131 NFC

Philippe Moulin

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

PARIS

Mémoire

présenté en vue de l'obtention

du DIPLOME d'INGENIEUR CNAM

Spécialité: Informatique

par

Philippe MOULIN

NOM'AD: Une solution de télégestion sur Nokia 6131 NFC

soutenu le 1er Octobre 2010

Jury

PRÉSIDENT : E. Métais

MEMBRES : E. Gressier-Soudan

S. Bouzefrane

F. Sailhan

U. Haberman

B. Lemaire

B. Le Chaux

TABLE DES MATIÈRES

REMERCIEMENTS **4**

INTRODUCTION **5**

1 LE CONTEXTE **7**

1.1 QU'EST-CE QU'UN SERVICE À DOMICILE ? 8
1.2 L'ORGANISATION DU SECTEUR D'ACTIVITÉ 9
1.3 LES ENJEUX 10
1.3.1 L'optimisation du planning *10*
1.3.2 Le compte-rendu d'activité *11*
1.4 LES LIMITES DU SYSTÈME ACTUEL 11
1.4.1 La lourdeur administrative *12*
1.4.2 La véracité des informations *12*
1.5 LA TÉLÉGESTION 13
1.5.1 Définition *13*
1.5.2 Fonctionnement d'un système de télégestion *14*
1.5.3 L'intérêt de la télégestion *14*
1.5.4 Cahier des charges d'un système de télégestion *15*
1.6 LES ACTEURS EN PRÉSENCE 17
1.6.1 Comparatif des acteurs : *17*
1.6.2 Comparatif des solutions *21*
1.6.3 Les méthodes d'identification *21*

2 OBJECTIFS **23**

2.1 L'IDENTIFICATION PROBANTE 23
2.2 LA VISUALISATION DU PLANNING 24
2.3 LA SAISIE DU COMPTE-RENDU D'ACTIVITÉ 25
2.3.1 L'écran d'identification du bénéficiaire *27*
2.3.2 L'écran de saisie de la date et de l'heure de début *27*
2.3.3 L'écran de liste des services *28*
2.3.4 L'écran de saisie de la date et de l'heure de fin *29*
2.3.5 L'écran récapitulatif *30*
2.4 L'ENVOI DE MESSAGES 30
2.5 L'ENVOI DES MESSAGES EN ATTENTE 30
2.6 L'INITIALISATION DES CARTES NFC 31

3 MÉTHODE ET OUTILS **32**

3.1 UN CONTRE-EXEMPLE : NOKIAEXPLORER 32
3.2 MÉTHODOLOGIE 33
3.2.1 Les phases *33*
3.2.2 Les méthodes *34*
3.3 LES TESTS 39
3.3.1 Les tests unitaires *39*

4 LE PROJET....44

4.1 Choix techniques....44
4.1.1Le téléphone Nokia 6131 NFC....44
4.1.2La technologie NFC....45
4.1.3L'environnement J2ME....46
4.1.4Le réseau GPRS....49
4.2 Les choix architecturaux....50

5 CONCEPTION....52

5.1 La machine à états....52
5.2 Les tâches....54
5.3 La persistance des données....57
5.4 L'interface graphique....58
5.5 Les entrées/sorties....63
5.5.1L'interface réseau....63
5.5.2L'Identification forte de l'utilisateur....64
5.5.3L'Échanges de données entre le serveur et le téléphone....68

6 PROBLÈMES RENCONTRÉS....70

6.1 Le manque d'informations....70
6.2 Les contraintes de sécurité....72
6.2.1L'obtention d'un certificat....73
6.2.2La création d'une chaîne de confiance....74
6.2.3Signatures et permissions....76
6.3 Les contraintes réseau....76
6.4 Les contraintes de ressources système....77
6.5 Les limites de l'interface graphique....79
6.5.1Le choix du Displayable utilisé....80
6.5.2Les limites du menu....82
6.6 Différences entre le téléphone et le simulateur....84
6.7 Absence d'outils de débogage sur cible....85
6.7.1Un exemple de recherche de bogue sur hôte : le problème des en-têtes HTTP....86
6.7.2Un exemple de recherche de bogue sur cible : le problème des jauges dans les alertes....87

7 PERSPECTIVES D'ÉVOLUTION....89

7.1 Une authentification plus sûre....89
7.2 La plateforme Android....92
7.3 Conclusion....93

BIBLIOGRAPHIE....95

INDEX LEXICAL....96

INDEX DES TABLES....97

INDEX DES ILLUSTRATIONS....98

REMERCIEMENTS

Je remercie Eric Gressier-Soudan, enseignant responsable, d'avoir accepté de superviser mon mémoire.

Je remercie Ugo Haberman de m'avoir fourni un sujet passionnant qui m'a permis de découvrir le monde du développement sur téléphone mobile. Je le remercie tout particulièrement d'avoir bien voulu m'accorder un peu de son temps pour me fournir tous les renseignements nécessaires à l'élaboration de ce rapport, Je lui suis également reconnaissant pour la confiance qu'ils m'a réservée, au niveau de l'élaboration du programme.

Je remercie Romain Pellerin, de m'avoir aidé à déboguer l'indéboguable.

Je remercie l'ensemble du personnel d'Hippocad pour son accueil, sa bonne humeur et sa sympathie qui ont permis de rendre ce travail agréable,

Je remercie enfin Sarah Mellet, Sandrine Etchébéhère, Eléonore Labaudinière et Bertrand Lemaire (Chef des informations de CIO) pour leur aide.

INTRODUCTION

La vie d'un intervenant à domicile peut se résumer en un mot: Diversité.

- Diversité des lieux d'intervention: l'intervenant peut être appelé à prodiguer des soins au domicile ou sur le lieu de travail du bénéficiaire, à accompagner des enfants à l'école...

- Diversité des bénéficiaires: L'aide à domicile n'est plus réservée aux seules personnes âgées

- Diversité des tâches effectuées: soins, réparations, cours particuliers...

À la fin de sa journée, l'intervenant doit rédiger un compte-rendu détaillé de ses interventions, en précisant le lieu, la durée, la nature et le bénéficiaire de chacune d'entre elles. En cas de contestation, il n'a aucun moyen de prouver l'effectivité de ses interventions.

Heureusement, une autre solution existe.

Cette autre solution se base sur deux outils:

- Le téléphone mobile, qui permet à l'intervenant de garder le contact avec sa structure, et de valider ses interventions.

- La carte à puce sans contact, qui permet de prouver l'effectivité des interventions.

La société Hippocad fournit une solution de télégestion composée d'un logiciel serveur qui assure la gestion des plannings det de la facturation des interventions, et d'un logiciel client, embarqué sur le téléphone mobile de l'intervenant.

Le développement de ce logiciel embarqué est la tâche qui m'a été confiée, au sein d'Hippocad. C'est également le sujet de ce mémoire.

Ce mémoire est organisé en six chapitres:

- Le premier chapitre décrit le monde des services à domicile, les difficultés liées à la coordination d'une équipe de travailleurs nomades, et les solutions actuellement sur le marché.

- Le deuxième donne le cahier des charges de la solution Nom'ad, objet du présent mémoire.

- Le troisième décrit les outils et méthodologies employées pour développer Nom'ad.

- Le quatrième décrit et justifie les choix techniques et architecturaux de cette solution mobile.

- La conception de l'application Nom'ad est décrite dans le cinquième chapitre.

- Le sixième décrit les nombreuses difficultés rencontrées lors du développement.

- Nom'ad est une application destinée à évoluer. Le dernier chapitre cite quelques perspectives d'évolution.

1 LE CONTEXTE

Les conséquences du baby-boom ont entraîné un accroissement inéluctable de l'âge moyen de la population. De ce fait, la demande de maintien à domicile de personnes âgées favorise un nouveau secteur en développement.

Avec un taux de croissance annuel moyen de 5,5 % depuis 1990, les services à la personne constituent le secteur de l'économie française dont la croissance est la plus forte en terme de créations d'emplois.

Ces travailleurs nomades sont confrontés à des problèmes très différents de ceux d'un employé sédentaire. Ce dernier effectue en général des tâches de longue durée, clairement définies, et reste en contact direct avec sa hiérarchie.

Par contre, les tâches de l'intervenant à domicile ont une granularité nettement plus fine. Ce dernier peut intervenir chez plus de dix bénéficiaires différents chaque jour, et effectuer chez chacun d'eux plusieurs tâches facturées différemment. La gestion du planning et des comptes-rendus d'intervention de ces travailleurs représente donc un surcoût non négligeable.

L'efficacité d'une structure de services à domicile est directement liée à sa capacité à rester en contact avec ses intervenants tout en maitrisant ses coûts.

Nous verrons en quoi la solution de télégestion Hippocad tente de répondre à cette problématique.

L'étude du fonctionnement du secteur du service à domicile, puis l'examen du positionnement d'Hippocad, doivent permettre de décrire les spécifications de celui-ci, les contraintes techniques rencontrées lors du développement de Nom'ad, ainsi que l'architecture qui en a découlé.

Ce chapitre est divisé en quatre parties:

- La première décrit le monde des services à domicile: le sous-chapitre 1.1 définit ce dernier, et le 1.2 décrit l'organisation du secteur d'activité.

- La deuxième décrit le système de gestion actuel, et ses limites.

- La troisième décrit la solution aux limites du système actuel.
- La quatrième décrit les solutions de télégestion actuellement sur le marché. Un sous-chapitre est consacré à la solution d'Hippocad, Prox'ad.

1.1 Qu'est-ce qu'un service à domicile ?

Les services d'aide à domicile se traduisent par l'intervention d'une personne « l'aide à domicile », auprès d'un bénéficiaire ayant besoin d'un tiers pour la réalisation d'actes de sa vie quotidienne. Les services rendus sont extrêmement variés :

- accompagnement à l'école
- aide ménagère
- bricolage, jardinage
- courses
- livraison de repas
- soins à domicile

Les bénéficiaires de ces services sont majoritairement des personnes en situation de perte d'autonomie, qui pour diverses raisons (âge, handicap...) ne peuvent plus assumer seuls certaines tâches de la vie quotidienne. Pour ces personnes, l'aide à domicile représente la seule alternative à une mise en maison de retraite, ou à une hospitalisation. Elles peuvent ainsi conserver leur vie sociale. Bien que les personnes dépendantes constituent encore la majeure partie de la clientèle de l'aide à domicile, de nouveaux marchés commencent à apparaître, car l'évolution des modes de vie a tendance à accentuer les besoins ponctuels de services à la personne.

1.2 L'ORGANISATION DU SECTEUR D'ACTIVITÉ

Le secteur du service à domicile fait intervenir un nombre important d'acteurs. Ce sont :

L'unité de coordination (aussi appelée back-office)

Elle s'occupe de la gestion des intervenants et du suivi de la clientèle. Elle assigne les interventions aux intervenants, génère les comptes-rendus d'activité, facture les interventions, et rémunère les intervenants.

Les intervenants (ou front-office)

Ce sont des salariés ou mandataires de l'unité de coordination. Ils effectuent les interventions au domicile des bénéficiaires.

Les financeurs (utilisateurs/patients/organismes publics)

Ils financent les interventions. Une même intervention peut être cofinancée par plusieurs tiers.

Le nombre d'acteurs impliqués dans une intervention rend sa gestion extrêmement complexe : le financeur et le bénéficiaire ne sont pas forcément la même personne, une intervention peut être cofinancée par plusieurs financeurs, un intervenant peut travailler pour le compte de plusieurs structures.

Pour l'instant, ces différents acteurs communiquent de manière administrative via des comptes-rendus d'activité.

1.3 LES ENJEUX

Les services à domicile brassent une masse d'argent extrêmement importante (15,6 milliards d'euros en 2008[1]) avec le vieillissement de la population, les perspectives de croissance dans les années à venir sont certaines. De nouveaux entrants arrivent chaque année sur le marché. Face à un tel succès, les bénéficiaires comme les financeurs demandent une transparence accrue et une qualité des soins effectués.

Le secteur face à cette demande légitime est donc obligé de s'organiser de manière plus efficace. La réponse apportée pour garder une marge bénéficiaire importante réside dans une optimisation du planning des intervenants et un meilleur suivi des actes effectivement réalisés via des comptes rendus d'activité.

1.3.1 L'optimisation du planning

Chaque intervenant reçoit régulièrement un planning détaillé des actes à effectuer.

Un intervenant peut effectuer de nombreuses interventions, pour le compte de nombreux bénéficiaires. Les modifications de planning sont fréquentes. Pour l'entreprise de service à domicile, la gestion efficace des changements de planning est donc un avantage concurrentiel.

Aujourd'hui, les intervenants sont informés individuellement des changements de planning (par SMS, téléphone...) ce qui implique un coût de gestion important et une efficacité limitée.

[1]Source: http://www.servicesalapersonne.gouv.fr/chiffres-cles-%282064%29.cml? [GOUV01]

1.3.2 Le compte-rendu d'activité

Il s'agit d'un rapport détaillé décrivant les bénéficiaires, la date et la nature de chaque intervention. Il permet la facturation de l'intervention, et donc la rémunération de l'intervenant et de sa structure.

Lors de l'intervention, l'intervenant fournit un ou plusieurs services au bénéficiaire. Chaque type de service est facturé différemment, en fonction d'un nombre déterminé de paramètres. Ce nombre varie de zéro à quatre. Les paramètres utilisés sont :

- La distance (nombre de kilomètres effectués dans le cadre de l'intervention)
- La durée de l'intervention
- Le nombre d'exemplaires (par exemple le nombre de livraisons de repas)
- Le montant

Prenons l'exemple d'une intervention qui consiste à faire des courses pour une personne âgée. Le déplacement jusqu'au supermarché sera facturé au kilomètre, le temps des courses sera facturé à l'heure, et le montant des courses sera refacturé à l'euro près.

1.4 LES LIMITES DU SYSTÈME ACTUEL

Le système actuel oblige l'intervenant à rédiger un compte-rendu détaillé de chaque intervention, en mentionnant chaque service effectué, et ses paramètres (nombre d'heures, de kilomètres, etc.)

Les intervenants peuvent être amenés à réaliser des tâches nombreuses et variées au cours d'une même intervention. La rédaction de ce compte-rendu pose donc deux problèmes : une lourdeur administrative importante et un problème de fiabilité de l'information.

1.4.1 La lourdeur administrative

Pour l'intervenant :

Si l'intervenant effectue dix tâches de 5 minutes chacune (changer une ampoule, arroser les fleurs...) il lui est difficile de s'arrêter après chaque tâche pour chronométrer le temps passé, la distance parcourue...

Pour l'unité de coordination :

Le traitement de chaque intervention est lourd pour l'unité de coordination (pointage manuel, vérification successive feuille par feuille...). Plus l'intervention réalisée est fragmentée, plus le traitement administratif qui s'ensuit est important.

Réduire cette lourdeur administrative est donc source de gain de temps, donc d'amélioration de la qualité de service et de gain financier pour l'ensemble de la filière.

1.4.2 La véracité des informations

L'information recueillie n'est pas toujours fiable ou complète.

Le compte rendu d'activité est un document administratif qui doit être rempli dans le cadre de procédures opérationnelles. L'oubli d'une signature, un document mal rempli, des retards dans les transferts de documents nuisent à la bonne gestion du processus. Simplifier le système de compte-rendu d'activités en réduisant le temps de saisie permet d'améliorer la qualité globale des services.

Lutter contre la fraude :

Sachant qu'il est inenvisageable d'installer une pointeuse au domicile de chaque bénéficiaire, comment vérifier l'exactitude du compte-rendu d'intervention ?

Les associations d'aide à domicile sont confrontées à un très fort taux d'absentéisme de la part des intervenants : ces derniers ne déclarent pas leurs retards, et dans certains cas, facturent des interventions non effectuées.

Et ce sont essentiellement les personnes les plus vulnérables qui en subissent les conséquences : Il est facile pour un aide soignant de contester la parole d'un bénéficiaire qui se plaint de ses absences répétées, surtout quand par exemple le bénéficiaire est atteint de la maladie d'Alzheimer.

La situation actuelle est parfois incroyable quant aux carences de contrôle.

Ce système « tout papier » n'est pas satisfaisant. Les intervenants doivent être libérés de leurs corvées administratives pour se focaliser sur leurs interventions (ce qui est tout de même le cœur de leur métier), et les unités de coordination doivent pouvoir améliorer leur contrôle sur l'effectivité des interventions.

La solution trouvée à l'heure actuelle pour répondre à cette problématique est la télégestion des interventions.

1.5 LA TÉLÉGESTION

1.5.1 Définition

La télégestion est le moyen de récupérer en temps réel ou en temps différé, grâce au réseau téléphonique, toutes les informations nécessaires à la justification et à la gestion des interventions effectuées à domicile par les intervenants pour le compte de prestataires de services et/ou d'organismes finançant ces services (Conseils régionaux, Caisses de retraite…).

1.5.2 Fonctionnement d'un système de télégestion

Les informations sont collectées sur un serveur et mémorisées dans une base de données. Elles sont traitées sur le logiciel métier du fournisseur qui a collecté l'information, ou sont routées via Internet vers l'application métier d'un concurrent (logiciels ou applications métier de gestion des plannings, de paie et de facturation des structures). Ce mode de fonctionnement permet d'améliorer la gestion des intervenants, avec notamment la possibilité de :

- Transmettre aux intervenants des messages personnalisés ;
- Contrôler le suivi et l'effectivité des interventions chez les bénéficiaires ;
- S'assurer de ne facturer que les prestations qui ont été effectivement réalisées et de détecter rapidement d'éventuelles dérives par rapport au planning prévisionnel.

1.5.3 L'intérêt de la télégestion

Ce mode opératoire répond aux problématiques rencontrées par le secteur, c'est-à-dire qu'il offre la possibilité de:

- Apporter une réponse concertée aux financeurs : effectivité des prestations, maîtrise des temps et des coûts d'intervention (transports et autres…) ;
- Optimiser les modes de gestion des structures et leur processus métier : automatisation de la saisie des interventions effectuées, des facturations et des paies, restitution immédiate des données d'activité, création de tableaux de bord et statistiques… ;
- Améliorer la qualité du service rendu aux clients : meilleur respect des plannings, parfaite lisibilité des interventions, plus grande réactivité… ;
- Faciliter l'action des intervenants à domicile : perfectionnement et valorisation de leur activité, simplification des tâches administratives ;
- Augmenter la productivité des prestations grâce à la possibilité pour les intervenants d'échanger des messages avec l'unité de coordination.

La télégestion constitue donc un système en pleine expansion qui offre des garanties indéniables en termes de fiabilité, de traçabilité et d'adaptabilité. À la date d'aujourd'hui, il existe peu de solutions de télégestion sur le marché.

1.5.4 Cahier des charges d'un système de télégestion

Définissons tout d'abord les critères permettant d'évaluer la qualité d'un système de télégestion.

Le contrôle de l'effectivité d'une intervention

On l'a vu, le taux élevé de fraude est l'une des principales raisons de passer à un système de télégestion. Pour que ce dernier soit efficace, il doit être :

Peu contraignant

L'intervenant doit pouvoir signaler le début et la fin de son intervention le plus facilement possible.

Utilisable partout

Les interventions ont essentiellement lieu au domicile des bénéficiaires. Un système de télégestion ne doit pas imposer la disponibilité de certaines ressources. Prenons l'exemple de l'usage du téléphone du bénéficiaire.

Les systèmes utilisant un serveur vocal identifient le bénéficiaire en utilisant l'identification de l'appelant. Ce dernier doit donc posséder un téléphone, accepter de le prêter à l'intervenant, ne pas masquer son numéro, ni être abonné à un opérateur ne supportant pas la présentation du numéro.

En outre, le bénéficiaire peut très bien être en train de téléphoner au début de l'intervention. L'intervenant doit donc attendre la fin de la conversation de son bénéficiaire pour pouvoir signaler son début d'intervention. Cela représente une contrainte pour les deux protagonistes. De plus, l'intervenant se verra attribuer un retard d'une durée égale à la durée de la conversation téléphonique restante de son bénéficiaire !

Sûr

Il ne doit pas être possible de falsifier une intervention. Un contrôle d'effectivité est considéré comme sûr s'il assure que la bonne personne s'est rendue au bon endroit, et au bon moment.

- La bonne personne : Comment l'intervenant est-il authentifié? Ses données d'identification peuvent-elles être partagées, falsifiées ou volées ?

- Au bon endroit : Comment s'assure-t-on que l'intervenant s'est bien rendu chez le bénéficiaire ? Peut-il avoir envoyé le compte-rendu de chez lui, sans que la fraude soit détectée ?

- Au bon moment : Comment le système horodate-t-il le début et de la fin de l'intervention ? Si l'heure du pointage fait foi, un intervenant n'ayant pas pu pointer son début d'intervention immédiatement (suite à un oubli, ou à l'indisponibilité du téléphone du bénéficiaire) sera-t-il pénalisé ? Et inversement; l'intervenant pourra-t-il masquer un retard au système ?

Le planning

La solution de télégestion doit permettre à l'intervenant de consulter un planning constamment à jour.

Les acteurs du marché ne proposent pas tous cette fonctionnalité. Et ceux qui la proposent se différencient par leur niveau de réactivité.

Une bonne gestion de planning doit être capable de prendre en compte le plus rapidement possible les modifications du planning.

Par exemple, si l'intervenant est en retard, la solution de télégestion doit lui donner la possibilité de signaler son retard, donc permettre au coordinateur d'ajuster immédiatement le planning, et de signaler les modifications à l'intervenant et au bénéficiaire.

1.6 LES ACTEURS EN PRÉSENCE

1.6.1 Comparatif des acteurs :

Domiphone

C'est une filiale d'Accor Services, créée en 1999. Le Groupe Accor met en avant sa notoriété, sa puissance économique, financière et politique afin de promouvoir Accor Services dont dépend Domiphone[2].

Domatel

Domatel fait partie du groupe Apologic, qui se présente comme « Leader dans l'informatisation des structures d'aide et de soins à domicile », et développe depuis plus de 16 ans une gamme complète de produits et de services dédiés à ces domaines[3].

Apologic Applications compte près de 2000 structures clientes sur toute la France et les Dom-Tom et près de 6500 logiciels installés. C'est la première société informatique du secteur de l'aide et des soins à domicile certifiée ISO 9001-2000.

Prylos

Créé en 2003, Prylos se présente comme spécialiste des applications mobiles pour les utilisateurs professionnels et le grand public avec le concept du « rendez votre mobile vraiment intelligent »[4].

Incubé par Paris Développement, structure de la Ville de Paris spécialisée dans les secteurs de haute technologie, Prylos a obtenu en 2006 le soutien d'OSEO Anvar dans sa démarche d'innovation.

[2]Source http://www.domiphone.com/ [DOMIPHONE]
[3]Source http://www.apologic.fr/domatel/ [DOMATEL]
[4]Source http://www.prylos.com/ [PRYLOS]

Hippocad

Hippocad Telesystems est fournisseur de solutions communicantes à destination des acteurs de la prise en charge à domicile.

Ayant obtenu le Label Anvar Entreprise Innovante, Hippocad bénéficie également d'avances remboursables de l'ANVAR.

Hippocad est devenue Lauréat LMI (Lille Métropole Innovation) début 2005 et a été également soutenue dans sa démarche par le Conseil Régional du Nord-Pas-de-Calais.

Hippocad a par ailleurs mis en place des partenariats stratégiques clés avec Bouygues Telecom et Nokia portant sur le développement de l'ensemble des outils de communication et de mobilité permettant de répondre aux besoins des opérateurs et intervenant, mais également d'élargir l'offre de service à destination des bénéficiaires, de leur famille…

Hippocad est éditeur et intégrateur de logiciels et de solutions de communication centrées sur la coordination et l'efficience de la prise en charge à domicile. La solution Hippocad est constituée d'une offre double de progiciels (plateforme web 2.0, logiciels embarqués) et de services de télégestion.

PROX'AD est un ensemble de progiciels ayant pour vocation de permettre la coordination du maintien à domicile. Destinée aux structures de prise en charge et de gestion du « A domicile », sociétés privées ou organismes associatifs ou publics, quelle qu'en soit la taille, PROX'AD est une plateforme Web centrée sur le plan de services du bénéficiaire à domicile. Ces applications permettent de gérer en temps réel les processus de planification, de coordination, de suivi des interventions et de gestion d'alertes[5].

[5]Source http://www.hippocad.com/ [HIPPOCAD]

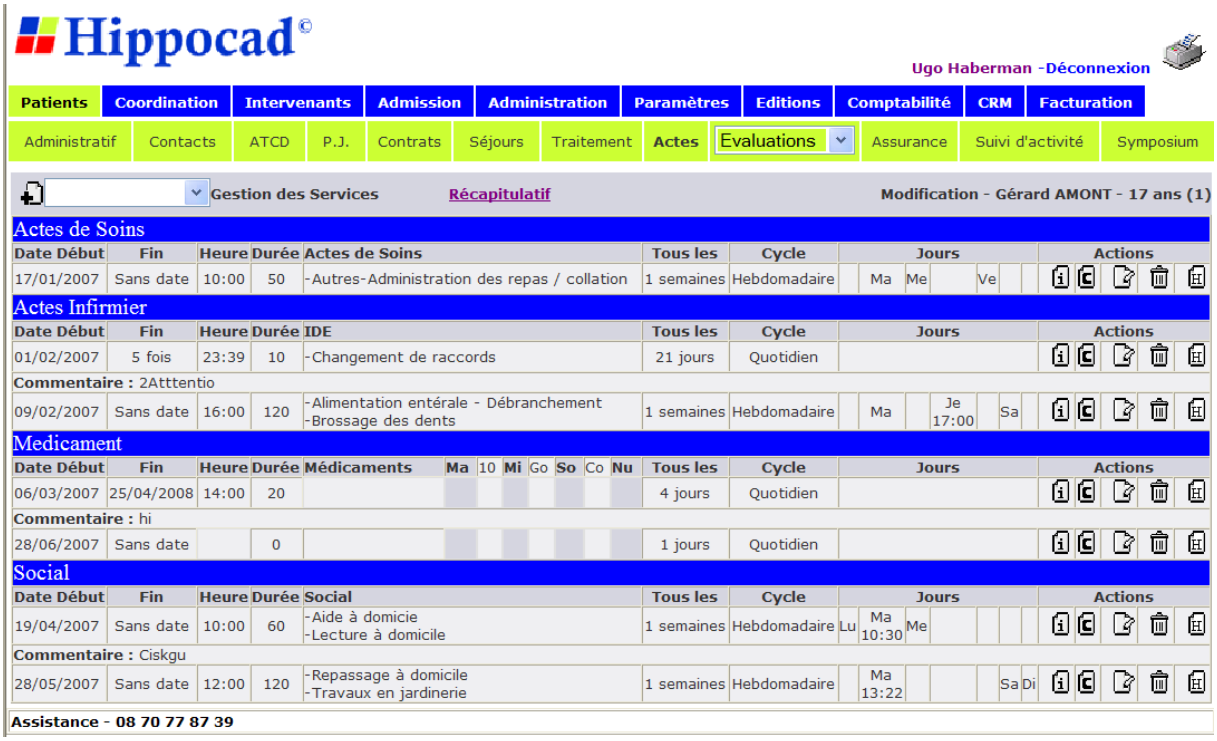

Illustration 1: Capture d'écran du formulaire de gestion des services de PROX'AD

Pour le développement de ces applications, Hippocad a mis en œuvre des partenariats avec des organismes lui permettant de bénéficier dès à présent d'une avance technologique significative : Laboratoire CEDRIC / CNAM, pour ce qui concerne les algorithmes d'optimisation des tournées, EDISANTE pour l'ensemble des éléments concourants à la promotion des normes et nomenclatures.

	Domiphone	Domatel	Prylos	Hippocad
Type	Acteur spécialisé dans la télégestion Filiale du Groupe Accor Services	Acteur spécialisé dans la télégestion Filiale du Groupe Apologic	Éditeur et intégrateur d'applications embarquées sur smartphone	Fournisseur de solutions communicantes à destination des structures de services et de soins à domicile
Cible	Structures d'aide à domicile, de services à la personne et organismes tiers financeurs	Structures d'aide à domicile, de services à la personne	Entreprises de SAV, de sondage...	Structures de services à la personne et d'aide à domicile, soins infirmiers à domicile, hospitalisation à domicile
Solutions	Solution sur téléphone filaire du bénéficiaire	Solution sur téléphone filaire du bénéficiaire	Utilisation du téléphone de l'intervenant, utilisation de visual Tags, et depuis avril 2008, utilisation d'étiquettes RFID	Téléphone avec lecteur RFID, ou terminal RFID mobile (Solem SD05)

Tableau 1: Comparatif des solutions de télégestion

1.6.2 Comparatif des solutions

Domatel et Domiphone proposent des systèmes de télégestion par téléphone. En début et en fin de visite, l'intervenant compose un N° vert sur le téléphone du bénéficiaire, et s'identifie au moyen de son code personnel.

En fin d'intervention, l'intervenant peut indiquer au serveur vocal quelles prestations il a effectuées.

Avantages

C'est une solution très simple à mettre en œuvre, et peu coûteuse, car l'intervenant n'a besoin d'aucun matériel, tout se fait via le serveur vocal.

Inconvénients

Domatel contrôle l'effectivité de l'intervention en utilisant l'identification de l'appelant. Cette solution pose des problèmes : l'intervenant doit utiliser le téléphone du bénéficiaire et ce dernier ne doit pas masquer son N°.

1.6.3 Les méthodes d'identification

Quatre méthodes d'identification sont actuellement utilisées par les acteurs du marché

- L'identification de l'appelant :
  Domatel et Domiphone demandent à l'intervenant d'appeler un serveur vocal depuis le téléphone fixe du bénéficiaire. Cette technique permet uniquement d'identifier le bénéficiaire. Ce dernier doit donc mettre à disposition de l'intervenant un téléphone fixe.

- Le code personnel :
  Les acteurs du marché utilisant un serveur vocal identifient leurs intervenants par leur code personnel. Cette technique ne peut pas être considérée comme probante, car un intervenant peut très bien confier son code à une tierce personne, qui pourra alors remplir un compte-rendu d'intervention à sa place.

- Le code-barre, ou étiquette visuelle :
  Prylos identifie les protagonistes grâce à des « Visual Tags » ces codes-barres 2D peuvent être lus par l'appareil photo du téléphone mobile de l'intervenant.
  Ils ont l'inconvénient de pouvoir aisément être copiés : une simple photocopie suffit.

Illustration 2: Une carte NFC

Illustration 3: Un QR Code[QRCODE]

- Les cartes NFC [NFC01]
  Cette dernière méthode d'identification est la plus sûre : une carte NFC contient une carte à puce sans contact qui offre les mêmes fonctionnalités qu'une carte à puce classique. Chaque carte NFC possède un identifiant unique et non modifiable.

L'utilisation d'une carte NFC est donc la meilleure réponse à la problématique du secteur, puisqu'elle permet l'identification la plus probante.

La société Hippocad a choisi d'utiliser cette solution pour développer sa solution de télégestion appelée Nom'ad.

Les besoins du secteur de l'aide à domicile sont complexes, et les réponses apportées par les acteurs du domaine sont très variées. La solution Nom'ad ne se limite pas au pointage, mais propose d'autres services essentiels: comme le planning mis à jour automatiquement et la rédaction simplifiée des comptes-rendus d'intervention.

Ces nouveaux services sont décrits en détail dans le chapitre suivant.

2 OBJECTIFS

L'application embarquée Nom'ad doit fournir les services suivants :

- L'authentification de l'intervenant et du bénéficiaire,
- La visualisation du planning des intervenants,
- La saisie du compte-rendu d'activité,
- L'envoi de messages par mail ou par SMS,
- L'initialisation des cartes NFC (Near Field Communication).

Elle assure également l'identification de l'intervenant et du bénéficiaire.

Ce chapitre décrit le cahier des charges de Nom'ad. Chaque fonctionnalité fait l'objet d'un sous-chapitre.

2.1 L'IDENTIFICATION PROBANTE

Le système Nom'ad doit permettre de s'assurer de l'effectivité des interventions et de la confidentialité des informations mises à disposition par l'application.

Pour cela, nous avions choisi d'utiliser la technologie NFC.

Comme vu précédemment, les cartes NFC sont très difficilement falsifiables et possèdent un identifiant unique qui ne peut pas être copié.

Chaque intervenant, ainsi que chaque bénéficiaire, sera équipé d'une carte NFC personnelle.

La lecture des cartes NFC par le téléphone trace de manière probante l'identification de l'intervenant et du bénéficiaire.

2.2 LA VISUALISATION DU PLANNING

L'application doit permettre la visualisation du planning de l'intervenant pour les prochains jours. À chaque envoi de données, l'application doit analyser la réponse du serveur. Si cette dernière contient des informations de planning, ce dernier doit être automatiquement mis à jour.

Illustration 4: Visualisation du planning

Chaque intervenant possède son propre planning. Si le téléphone est partagé entre plusieurs intervenants, chacun d'eux ne doit pouvoir accéder qu'à son planning. L'application doit stocker les informations de planning de manière persistante, pour ne pas avoir à les retélécharger à chaque lancement.

L'écran de planning doit afficher la liste des interventions du jour. Ceci doit être fait de manière ergonomique afin de faire adhérer l'utilisateur au système. L'intervenant doit pouvoir faire défiler les interventions avec les touches haut et bas, afficher le détail d'une intervention avec la touche centrale, et passer aux jours suivant et précédent avec les touches gauche et droite.

2.3 LA SAISIE DU COMPTE-RENDU D'ACTIVITÉ

La spécification phare du téléphone est la saisie du compte-rendu d'activité par les intervenants.

Une saisie de compte-rendu d'activité comporte plusieurs étapes :

- L'identification du bénéficiaire
- La saisie des dates et heures de début
- La liste des services réalisés
- L'écran de saisie de la date et de l'heure de fin
- Le récapitulatif avant validation finale

L'application de saisie des comptes rendus est constituée de cinq écrans successifs. A chaque étape, l'intervenant peut revenir à l'écran précédent, ou passer à l'écran suivant après vérification des données saisies.

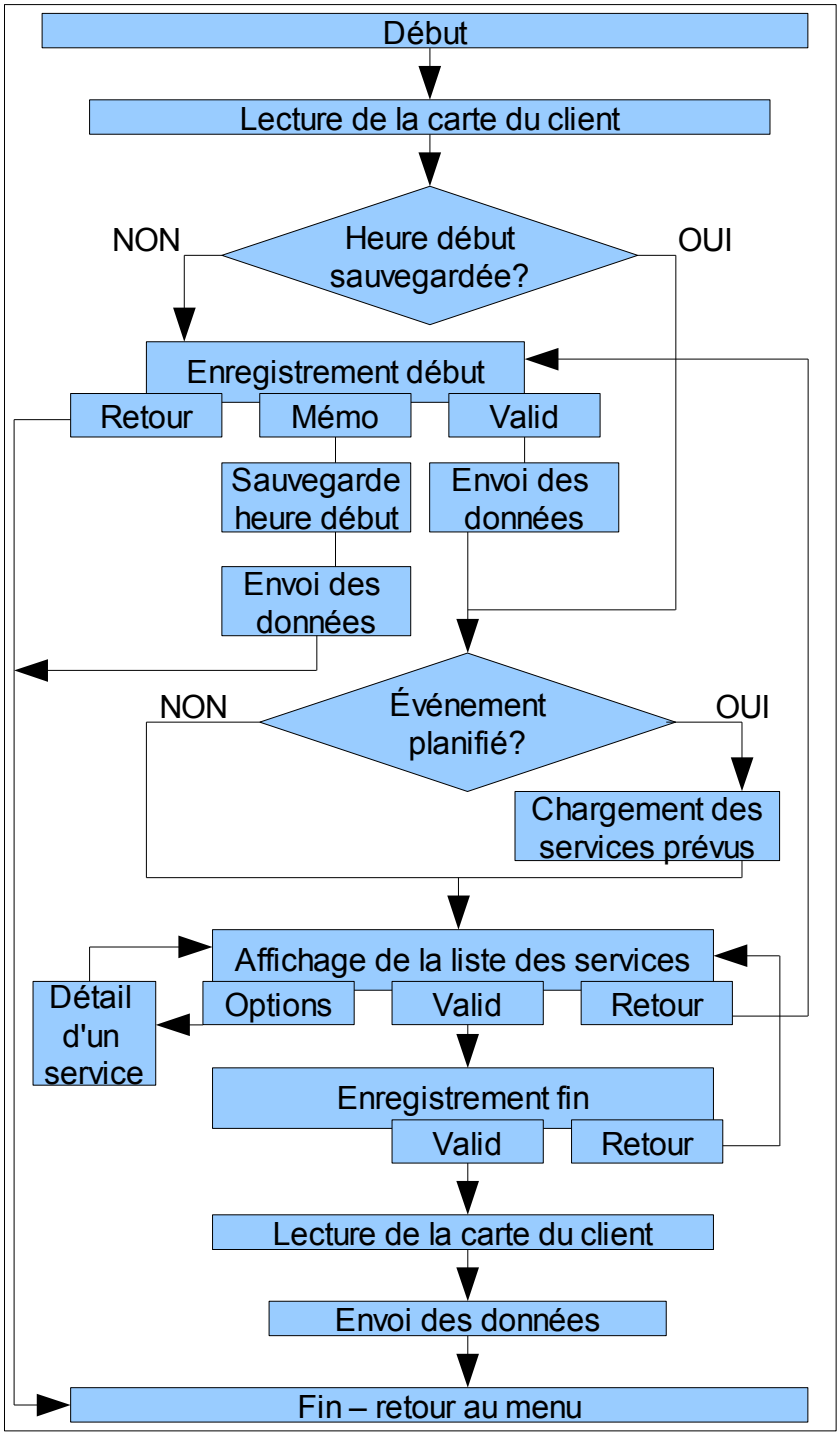

Illustration 5: Saisie d'un compte-rendu d'activité

L'intervenant doit pouvoir envoyer un compte-rendu d'intervention. Cela se fera donc via une option de menu appelée « CR Intervention »

Ce menu lui permet donc d'accéder à l'application de saisie des comptes-rendus d'intervention. Deux options doivent pouvoir s'offrir à lui :

- Saisir l'ensemble des informations en une seule fois (à la fin de l'intervention)

- Enregistrer uniquement l'heure de début d'intervention, puis, en fin d'intervention, relancer l'application : cette dernière doit se souvenir de l'heure de début précédemment mémorisée, et passer automatiquement à l'écran suivant.

Les cinq écrans qui constituent le processus de saisie du compte-rendu d'activité sont les suivants :

2.3.1 L'écran d'identification du bénéficiaire

Le premier écran doit demander à l'intervenant de présenter la carte du bénéficiaire de l'intervention. Une fois la carte reconnue, le nom du bénéficiaire doit s'afficher.

2.3.2 L'écran de saisie de la date et de l'heure de début

L'application doit afficher un écran permettant la saisie de la date et de l'heure de début. Le contenu de cet écran doit être initialisé avec la date et l'heure du téléphone.

L'intervenant doit alors avoir deux possibilités :

- Mémoriser l'heure de début, et revenir au menu principal

- Continuer la saisie du compte-rendu

Dans les deux cas, l'heure de début doit être immédiatement envoyée au serveur, dans un message contenant également les identifiants des cartes de l'intervenant et du bénéficiaire, ainsi qu'un indicateur précisant s'il s'agit de l'heure du téléphone, ou d'une heure entrée manuellement.

2.3.3 L'écran de liste des services

Illustration 6: Affichage de la liste des services

L'intervenant doit pouvoir consulter, saisir et modifier la liste des services qu'il a réalisés. Le programme recherche dans le planning résident sur le mobile s'il existe un événement correspondant à la date et à l'heure de début précédemment saisie, et à la carte bénéficiaire précédemment présentée. Si pour un client, la date et l'heure de début ou de fin d'un évènement planifié se situent entre les dates et heures de début et de fin de l'événement en cours alors nous considérons que l'intervention concerne cet événement, et les données de l'évènement sont préchargées.

Il existe deux types d'événements : L'événement connu (qui correspond à une tâche préalablement définie dans le planning : elle est donc déjà préchargée dans le téléphone) et l'événement inconnu (qui ne correspond pas à une tâche prévue initialement).

Une marge de temps doit être provisionnée par le système. Cette marge de temps doit permettre de reconnaître et donc de précharger les événements connus.

Si l'heure de début d'intervention est comprise dans l'intervalle de temps d'un événement présent dans le planning, la liste des services prévus s'affichera.

La marge de sécurité doit pouvoir être paramétrée en fonction de chaque utilisateur, et doit permettre de rapprocher une intervention d'un événement connu.

Par exemple, si l'intervenant saisit 9H55 comme heure de début, et que son planning contient un événement commençant à 10H, l'application doit considérer que le compte rendu en cours de saisie correspond à cet événement, et en précharger les données.

Si l'heure de début ne correspond à aucun événement du planning, l'application doit considérer qu'il s'agit d'un événement imprévu, et afficher une liste initialement vide.

Dans l'écran de liste des services, l'intervenant doit pouvoir :

- Ajouter de nouveaux services, et supprimer des services existants
- Consulter et modifier les détails (nombre d'heures, de kilomètres effectués)

Ces services doivent être choisis dans une liste téléchargée sur le mobile. En aucun cas l'utilisateur ne doit avoir la possibilité de créer de nouveaux services.

2.3.4 L'écran de saisie de la date et de l'heure de fin

L'intervenant doit arriver dans l'écran de saisie de la date et de l'heure de fin d'intervention. Comme pour l'heure de début, ce sont la date et l'heure du téléphone qui s'afficheront par défaut dans cet écran.

2.3.5 L'écran récapitulatif

A la fin de l'intervention, l'intervenant se verra présenter le dernier écran : le récapitulatif de l'intervention contenant les heures de début et de fin, ainsi que la liste détaillée des services saisis.

Pour valider définitivement l'intervention, l'intervenant doit passer la carte du bénéficiaire. L'application doit alors envoyer l'ensemble des informations saisies au serveur.

2.4 L'ENVOI DE MESSAGES

L'application doit fournir un système permettant à l'intervenant d'envoyer des messages textuels à une liste prédéfinie de destinataires.

Ces messages doivent pouvoir être envoyés par SMS, ou directement en utilisant la connexion GPRS de l'application.

L'écran d'envoi de messages doit permettre de choisir le ou les destinataires dans une liste fournie par le serveur, et de saisir un message en s'aidant éventuellement d'une liste de phrases prédéfinie. Cette liste est fournie par le serveur.

2.5 L'ENVOI DES MESSAGES EN ATTENTE

Tous les messages à destination du serveur doivent être placés dans une file d'attente persistante. En cas d'échec dû par exemple à l'absence de couverture GPRS, l'option de menu « Envoyer N messages » affichera le nombre de messages en attente, et la sélection de cette option déclenchera une nouvelle tentative d'envoi de l'ensemble des messages en attente.

Pour chaque message, l'application maintiendra un compteur de tentatives d'envoi, et stockera la date et l'heure de la première tentative. Ces informations seront ajoutées aux messages lors de leur envoi. Le serveur pourra ainsi savoir si un message donné est envoyé en temps réel ou non.

2.6 L'INITIALISATION DES CARTES NFC

Cette fonction sera réservée à l'administrateur.

L'écran d'initialisation des cartes proposera quatre fonctions :

- La lecture d'une carte : Si la carte présentée est une carte intervenant, ou une carte bénéficiaire, son identifiant unique s'affichera, ainsi que le nom de son propriétaire, et le type de carte (Intervenant ou bénéficiaire).

- La création d'une carte bénéficiaire : L'application enregistrera dans la carte le nom du bénéficiaire, ainsi que le type de carte.

- La création d'une carte intervenant : Dans ce type de carte, le nom de l'intervenant sera enregistré sous la forme d'une URL, ce qui permettra d'utiliser la carte pour lancer le navigateur web et se connecter au serveur.

- L'effacement d'une carte.

Les exigences décrites dans le présent chapitre ont évolué tout au long du projet. Les méthodes utilisées pour faire face à ces changements de spécifications sont décrites dans le prochain chapitre.

3 Méthode et Outils

Passer de l'expression des besoins du client au produit livré ne s'improvise pas. Des logiciels très simples, comme par exemple NokiaExplorer, peuvent être développés sans utiliser de méthodologie, mais au dela d'un millier de lignes, l'usage d'une méthodologie devient obligatoire, et le choix de cette dernière a une influence cruciale sur la suite du projet.

Certains outils, comme par exemple le logiciel de gestion de versions Subversion, sont également essentiels au bon déroulement d'un projet logiciel.

Pour mieux expliquer l'utilité de ces méthodes et outils, je commencerai par une démonstration par l'absurde : que se passe-t-il lorsqu'on tente de développer un logiciel sans méthode ni outils?

3.1 Un contre-exemple: NokiaExplorer

Pour explorer les différentes fonctionnalités du 6131, j'ai écrit un petit logiciel nommé NokiaExplorer. Il affiche diverses informations sur l'état du téléphone (langue sélectionnée, présence ou non d'une carte SD...) et permet également de tester le la connectivité au réseau GPRS, de lire des cartes sans contact NFC, etc.

NokiaExplorer a été développé sans passer par les phases de spécifications, conception, et tests. Au départ, il fournissait des services très simples: affichage de quelques propriétés système, détection de la carte SD, etc. L'ajout de nouvelles fonctionnalités, comme la lecture des cartes NFC a augmenté sa complexité, et il est maintenant très difficile de le faire évoluer.

Par exemple, NokiaExplorer est capable de lire des cartes NFC et de communiquer par GPRS, mais il n'est pas facile de regrouper ces deux fonctionnalités pour l'utiliser comme un lecteur de cartes NFC utilisable en réseau: pour cela il faudrait ajouter une tâche d'envoi et une machine à états, et donc augmenter encore sa complexité.

L'augmentation de la complexité rend les modifications de plus en plus coûteuses, et risquées. Pour maîtriser cette complexité, il faut utiliser une méthodologie.

3.2 MÉTHODOLOGIE

La méthodologie définit l'ensemble des étapes permettant de passer du besoin au produit fini.

3.2.1 Les phases

Tout projet (informatique ou autre) s'effectue en plusieurs phases:

L'analyse

Cette phase a pour objectif de définir le besoin du client. Son objectif est de produire un modèle conceptuel décrivant ce que le système doit faire, mais sans décrire comment il doit le faire.

Ce détail est très important, car, à la question « que voulez-vous ?» le client va généralement répondre par une description de la solution telle qu'il l'imagine, et non du problème.

Une bonne compréhension du problème permet de définir une solution plus adaptée.

La conception

Cette phase a pour objectif de répondre à la question « comment » et donc de définir l'architecture du logiciel. Elle est généralement divisée en deux phases: la conception architecturale, et la conception détaillée.

L'implémentation

C'est la phase de codage proprement dite.

Les tests

Ils permettent de valider la conformité du logiciel avec le besoin du client tel qu'il a été défini en phase d'analyse.

3.2.2 Les méthodes

Examinons les méthodes les plus répandues:

Le modèle en cascade

C'est la méthode la plus simple: on effectue une seule itération des phases définies précédemment.

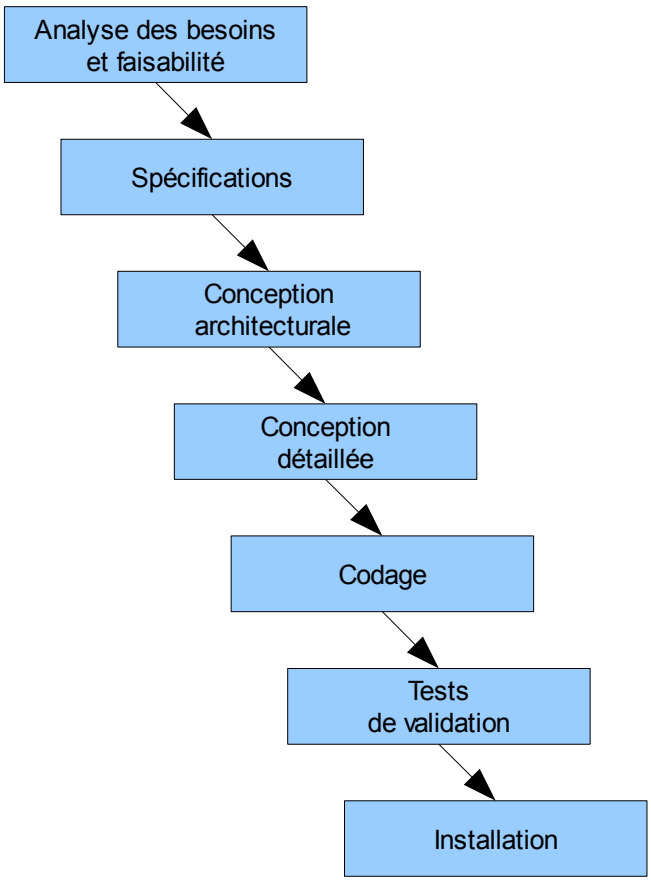

Illustration 7: Modèle en cascade

Ce modèle est hérité de l'industrie du BTP. Son principal inconvénient est le manque de réactivité, par exemple face à une évolution des besoins pendant la phase de codage.

Les éventuelles erreurs commises dans les premières phases (Analyse des besoins et spécifications) peuvent n'être détectées que dans les phases suivantes (conception et codage). Cette faible visibilité est surnommée « effet tunnel »

Ce type de cycle est donc à réserver aux projets à faible incertitude, ce qui n'est clairement pas le cas du projet Nom'ad

Le cycle en V:

Il a été imaginé pour pallier les limitations du modèle en cascade, notamment son manque de réactivité.

En cas d'anomalie, il permet de limiter le retour aux étapes précédentes. Les phases de la partie montante renvoient l'information aux phases en vis à vis.

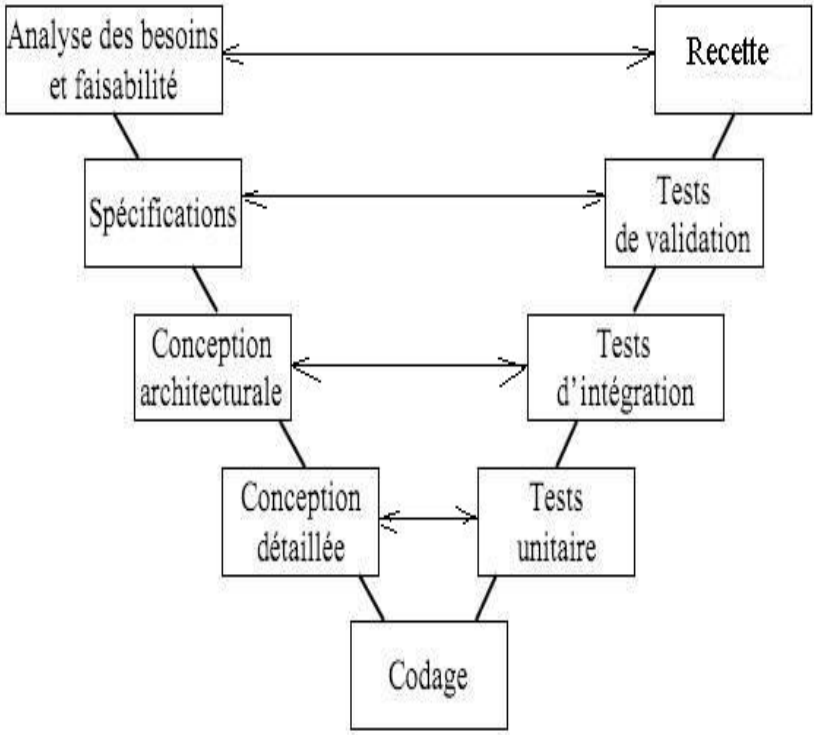

Illustration 8: Cycle en V

Par exemple, si un défaut est détecté en phase de tests unitaires, cela n'impacte que la conception détaillée, et le problème peut être corrigé sans avoir à reprendre le cycle depuis le début.

Ce type de cycle offre donc une plus grande réactivité, mais n'est pas la solution idéale pour faire face à une évolution continue des besoins.

Le cycle en spirale

Il reprend les étapes du cycle en V. A chaque version le cycle recommence en proposant un produit de plus en plus complet.

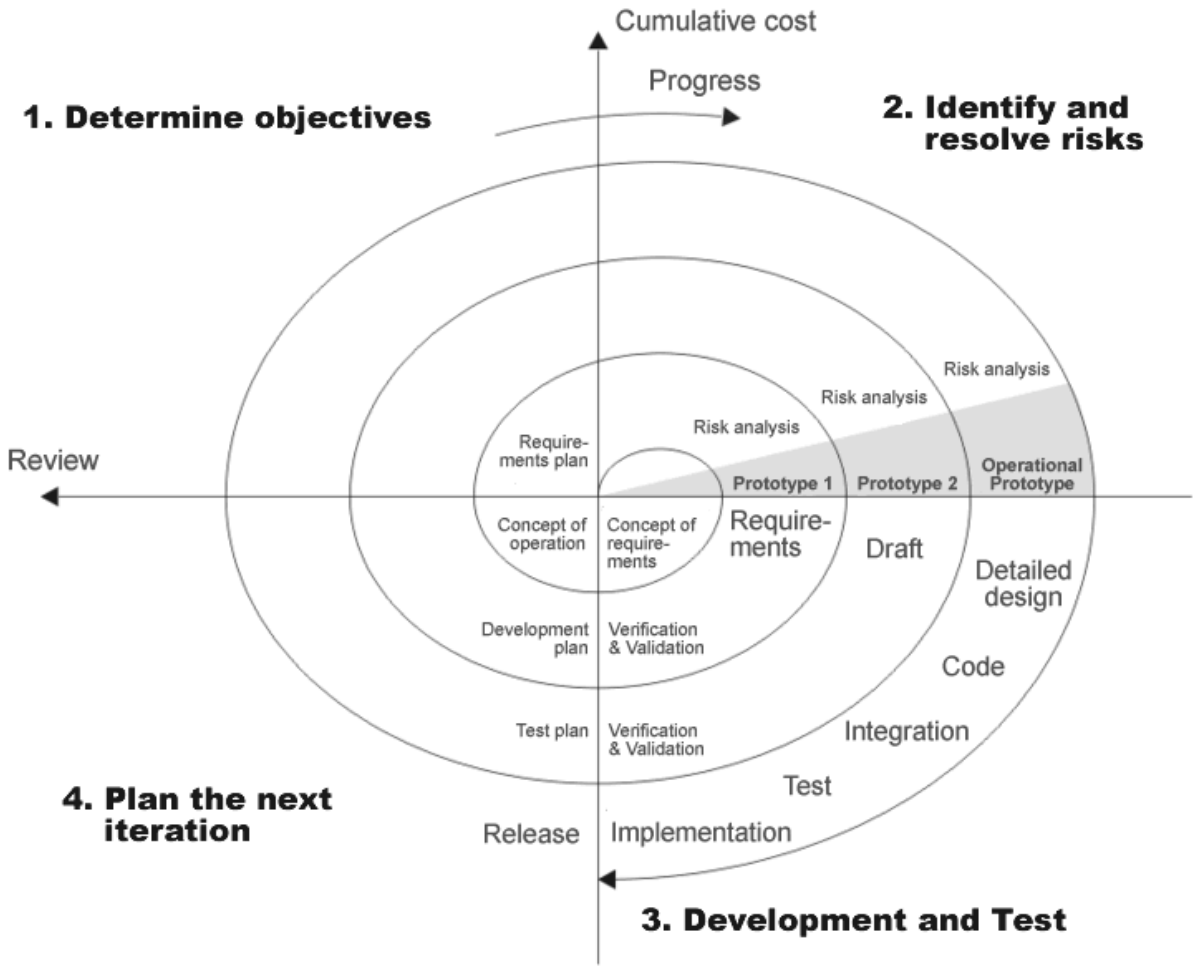

Illustration 9: Le cycle en spirale

Ce cycle met plus l'accent sur la gestion des risques que le cycle en V. Il est adapté au développement de très grands projets.

Le cycle itératif

Il permet de développer un logiciel de manière incrémentale. On commence par créer un logiciel très simple répondant à un sous-ensemble des exigences, puis on le fait évoluer par itérations successives, jusqu'à ce que toutes les exigences soient satisfaites.

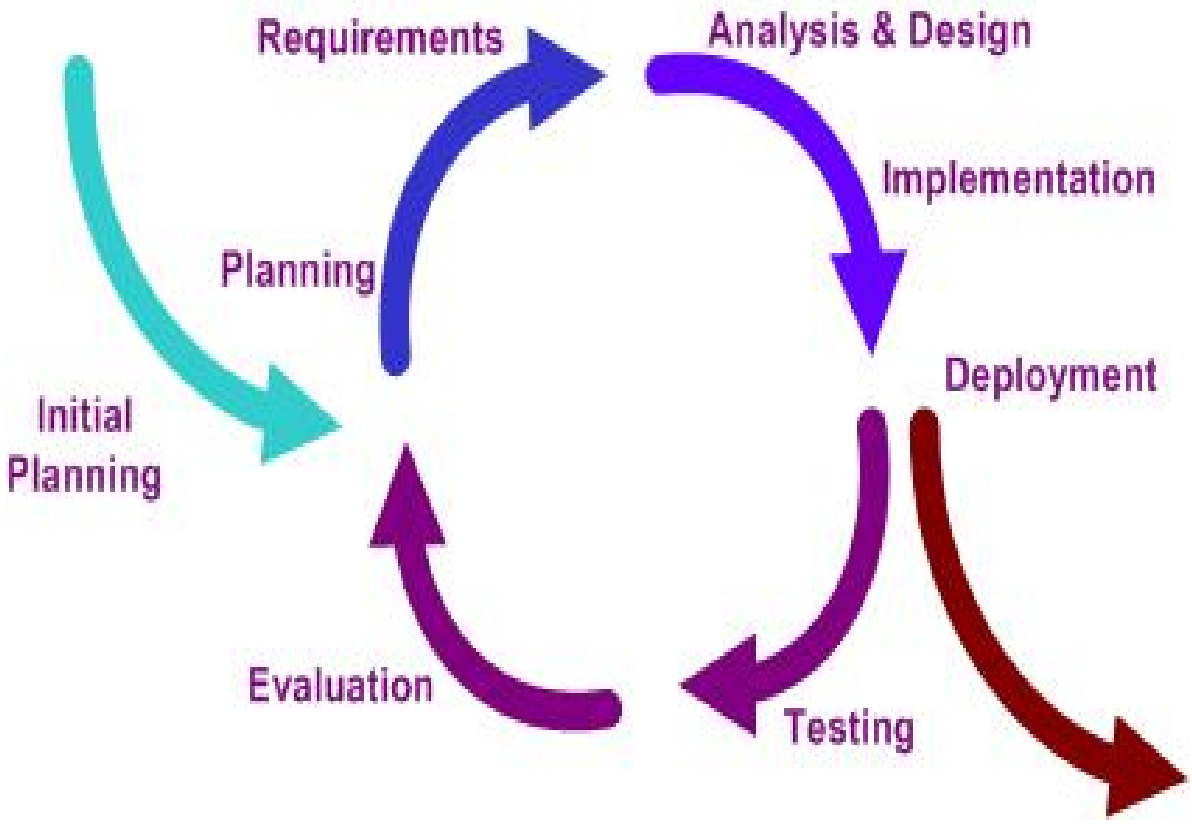

Illustration 10: Le cycle itératif

La méthode choisie

Pour le projet Nom'ad, le cycle en V n'apportait pas assez de souplesse, notamment vis-à-vis des nombreuses évolutions de spécifications en cours de projet.

Le cycle en spirale était trop complexe à gérer pour un projet comportant un seul développeur.

C'est donc le cycle itératif qui a été utilisé.

Ce modèle de développement, allié à une conception très modulaire, a permis de faire face aux nombreux changements de spécifications imposés par les clients.

3.3 LES TESTS

Les tests sont une partie incontournable du développement de toute application. Sur le Nokia 6131, leur mise en oeuvre pose quelques problèmes.

L'intérêt des tests:

Un logiciel de la taille et de la complexité de Nomad ne peut pas être validé par le simple examen de son code. Pour s'assurer de la conformité du logiciel avec ses spécifications, une suite de tests doit être effectuée.

Dans le cycle en V, les tests sont effectués après la phase de codage. Les tests unitaires valident la conception détaillée, les tests d'intégration valident l'architecture, et les tests de validation valident les spécifications.

Cette façon de faire a l'inconvénient de placer les tests en fin de projet. Si ces derniers échouent, un deuxième cycle en V long, coûteux et surtout imprévu sera nécessaire.

Pour éviter ce risque, une autre méthode est possible: l'extreme programming.

Avec cette méthode, les tests sont mis en place avant le développement du logiciel, ce qui permet de détecter les défauts immédiatement.

3.3.1 Les tests unitaires

Ces tests ont la granularité la plus fine. Chaque test unitaire valide une méthode.

Exemple

Cette classe représente un abonnement (à un journal, ou autres) Après l'avoir initialisée en indiquant le cout et la durée de l'abonnement, il est possible d'obtenir le prix par mois en appelant pricePerMonth().

```
public class Subscription {

   private int price ; // subscription total
price in euro-cent

   private int length ; // length of subscription
in months

   // constructor :

   public Subscription(int p, int n) {

     price = p ;

     length = n ;

   }

  /**

   * Calculate the monthly subscription price in
euro,

   * rounded up to the nearest cent.

   */

   public double pricePerMonth() {

     double r = (double) price / (double)
length ;

      return r ;

   }

  /**

   * Call this to cancel/nulify this
subscription.

   */

   public void cancel() { length = 0 ; }
```

```
}
```

Pour tester unitairement les méthodes de cette classe, on peut utiliser le framework Junit, et créer la classe de test suivante:

```
import org.junit.* ;

import static org.junit.Assert.* ;

public class SubscriptionTest {

   @Test

   public void test_returnEuro() {

      System.out.println("Test if pricePerMonth
returns Euro...") ;

      Subscription S = new Subscription(200,2) ;

      assertTrue(S.pricePerMonth() == 1.0) ;

   }

   @Test

   public void test_roundUp() {

      System.out.println("Test if pricePerMonth
rounds up correctly...") ;

      Subscription S = new Subscription(200,3) ;

      assertTrue(S.pricePerMonth() == 0.67) ;

   }

}
```

Un test Junit est associé à chaque méthode de la classe testée. À première vue, l'écriture de ces classes de tests peut sembler une perte de temps (le code de tests est aussi long que le code à tester) mais ces tests permettent de vérifier systématiquement la non-régression du code, et ce à chaque modification.

Le code de la classe Subscription est extrêmement simple. La méthode pricePerMonth n'effectue qu'une seule opération: une division. La rédaction de tests unitaires peut donc paraître superflue: une simple relecture de code permet de constater l'absence d'erreurs. Et pourtant...

Supposons qu'un développeur récupère cette classe, et tente de s'en servir pour gérer des abonnements à vie. Le champ durée n'a donc plus lieu d'être, et le développeur a donc plusieurs possibilités pour gérer sa disparition:

- Modifier Subscription, en y ajoutant un constructeur prenant uniquement le prix comme paramètre

- Décider que, par convention, les abonnements a vie auront une durée nulle

- Décider que, par convention, les abonnements à vie auront une durée de Integer.MAX_VALUE

Dans les deux premiers cas, l'appel à pricePerMonth provoquera une division par zéro. Le troisième cas est nettement plus gênant, car il ne provoquera pas d'erreur. PricePerMonth renverra une valeur extrêmement petite, ce qui n'est certainement pas le comportement désiré.

Pour qu'un test unitaire soit efficace, il doit tester:

- Quelques cas nominaux: Dans notre exemple, on initialisera Subscription avec des valeurs courantes de durée et de coût.

- Tous les cas limites, c'est-à-dire les valeurs maximales et minimales admissibles par les méthodes testées

- Les cas hors limites: Si une méthode est sensée générer une erreur si ses paramètres ont des valeurs aberrantes, on vérifiera que c'est bien le cas.

Dans notre exemple, le fait d'essayer d'initialiser Subscription avec une durée nulle est un test hors limites: cette classe est manifestement destinée à gérer des abonnements dont la durée n'est ni nulle, ni illimitée.

Suite à l'échec des tests hors limites, le développeur de Subscription modifiera sa classe pour que ces cas soient traités correctement. Au lieu de fournir des résultats aberrants quand elle est initialisée avec des valeurs aberrantes, la classe Subscription refusera ces dernières, évitant ainsi que des valeurs aberrantes soient propagées dans tout le logiciel, ce qui aurait sans doute occasionné une recherche de bogue longue et coûteuse.

Les tests unitaires sur le Nokia 6131 NFC

La mise en oeuvre de tests unitaires sur le 6131 pose de nombreux problèmes: il est difficile de lancer un framework de tests dans l'environnement limité du 6131, et l'émulateur (qui permet de tester les Midlets dans un 6131 virtuel, sur PC) est tellement lent que le lancement d'une suite de tests peut facilement prendre plusieurs heures.

La solution retenue a donc été d'effectuer des tests unitaires sur cible, en utilisant NokiaExplorer comme environnement d'exécution: le code de chaque méthode à tester était inséré dans NokiaExplorer, puis associé à une méthode de tests. Pour l'enregistrement des résultats, j'ai utilisé le système des logs sur carte SD: Au démarrage, NokiaExplorer détecte la présence d'une carte SD contenant un dossier de logs, et si c'est le cas, redirige tous les logs dans un fichier présent sur la carte.

4 LE PROJET

Ce chapitre décrit l'environnement du projet, et justifie l'ensemble des choix techniques. Nous commencerons par la description de la plate-forme cible, puis nous aborderons les technologies utilisées: Java ME, NFC, et GPRS. La dernière partie sera consacrée aux choix architecturaux.

4.1 CHOIX TECHNIQUES

4.1.1 Le téléphone Nokia 6131 NFC

Nous avons choisi d'utiliser ce téléphone car il répond parfaitement aux besoins de l'application Nom'ad. Au démarrage du projet, c'était le seul téléphone équipé d'un lecteur NFC intégré de série.

Ce téléphone multimédia est particulièrement complet. Voici un petit rappel de ses nombreuses fonctionnalités :

- Compatible avec le réseau GPRS (General Packet Radio Service)
- Nombreuses fonctionnalités multimédia : Appareil photo, caméscope, lecture des fichiers audio MP3 et de certains fichiers vidéo, radio FM, gestion des messages MMS
- Compatible Bluetooth et IRDA
- Lecteur NFC (Near Field Communication) intégré
- JavaCard intégrée

- Compatible J2ME (Java 2, Mobile Edition)
- Compatible avec les cartes MicroSD

Trois de ces nombreuses fonctionnalités nous intéressaient tout particulièrement, et seront décrites en détail ci-après : le lecteur NFC, l'environnement Java, et la connectivité GPRS.

4.1.2 La technologie NFC

NFC est l'acronyme de Near Field Communication, ou communication en champ proche.

Ce protocole permet des communications à très courte distance (jusqu'à 3 cm, dans le cas du Nokia 6131 NFC). Les antennes NFC émettent et reçoivent sur une fréquence de 13,56 MHz. Il existe deux catégories de périphériques NFC : les périphériques actifs, qui possèdent leur propre source d'énergie, et les périphériques passifs, qui sont alimentés par la porteuse émise par le périphérique actif avec lequel ils sont en train de communiquer.

Les périphériques RFID passifs sont constitués d'une simple puce reliée à une bobine. Leur faible coût (0.5 $) et leur petite taille (de l'ordre du centimètre) les rendent pertinents pour une très large gamme d'applications.

Pour la traçabilité :

Contrairement à une étiquette classique, une balise NFC peut être lue facilement et sans risque d'erreurs par un lecteur moins coûteux qu'un lecteur de code-barres. Le lecteur NFC n'utilise ni optique, ni laser, ni mécanisme, mais une simple bobine. Le contenu de l'étiquette NFC peut être protégé contre la lecture et/ou la modification par des personnes non autorisées.

En remplacement des cartes à puce :

Les cartes NFC fournissent toutes les fonctionnalités des cartes à puce classiques : identification forte, paiement, etc.

4.1.3 L'environnement J2ME

Java 2 Mobile Edition est une plate-forme Java adaptée aux appareils mobiles (téléphones, assistants numériques...).

Write Once, Run Anywhere

Le langage Java a été créé par SUN pour répondre au problème des coûts de portage : la modification d'un logiciel destiné, par exemple, au monde Windows pour en faire une version adaptée au Macintosh, ou au monde Linux représente un coût très important pour l'éditeur. Ce dernier, s'il veut proposer une application multiplateforme, devra faire face à des coûts de portage et surtout de maintenance proportionnels au nombre de plates-formes supportées.

Java propose de mettre fin à ce gaspillage : Le slogan de SUN « Write Once, Run Anywhere » (« Écrivez-le une seule fois, utilisez-le partout ») est emblématique de cette philosophie.

Les programmes Java sont développés sur une seule plate-forme : la « machine virtuelle » Java. Cette machine virtuelle sert d'intermédiaire entre l'application et son environnement, et permet aux développeurs de faire abstraction des différences entre plates-formes. Une application Java fonctionnera sans recompilation sur toutes les plateformes pour lesquelles une machine virtuelle Java est disponible.

Cette généricité a un coût : les applications java peuvent difficilement tirer parti de fonctionnalités qui ne sont pas disponibles sur toutes les plateformes.

Mais globalement le « Write Once, Run Anywhere » fonctionne, et le développeur d'une applet Java verra son code fonctionner à l'identique sous Mac, sous Windows et sous Linux... même si le reste de la page risque de s'afficher différemment suivant le navigateur utilisé.

Write Once, Run Anywhere : Et les mobiles ?

Dans le monde mobile, le « Write Once, Run Anywhere » s'avère être un problème nettement plus complexe que dans le monde des machines de bureau.

Les machines virtuelles Java pour Windows, Mac et Linux fonctionnent dans des environnements ayant de nombreux points communs : beaucoup de mémoire, des écrans dont la résolution est au moins de 640 par 480, et des périphériques d'entrée quasi identiques d'une plateforme à l'autre.

Les plateformes mobiles proposent des ressources nettement plus limitées, et en général nettement inférieures au minimum nécessaire pour faire fonctionner une machine virtuelle Java standard.

Il a donc été nécessaire de déroger à la règle d'or du « Write Once, Run Anywhere » et de créer une version de Java adaptée à l'environnement mobile : J2ME.

Mais la très grande variété du monde de l'embarqué fait qu'il est impossible d'appliquer tels quels les principes qui ont fait le succès de Java dans le monde des machines de bureau. En effet, J2ME ne s'adresse pas qu'aux téléphones mobiles, mais cible toutes les plateformes à ressources limitées, par exemple les récepteurs de télévision numérique.

On l'a vu, les machines virtuelles Java permettent de développer des programmes fonctionnant de manière quasiment identique sur Mac et PC. Mais est-il possible de développer une machine virtuelle permettant de s'abstraire des différences entre des plates-formes aussi éloignées qu'un téléphone et une télévision numérique ?

A cette question, Sun répond en proposant une architecture modulaire : la machine virtuelle est toujours présente, mais réduite au strict minimum, et accompagnée d'une « configuration », et d'un « profil ». Ces deux derniers éléments peuvent varier en fonction de l'environnement cible.

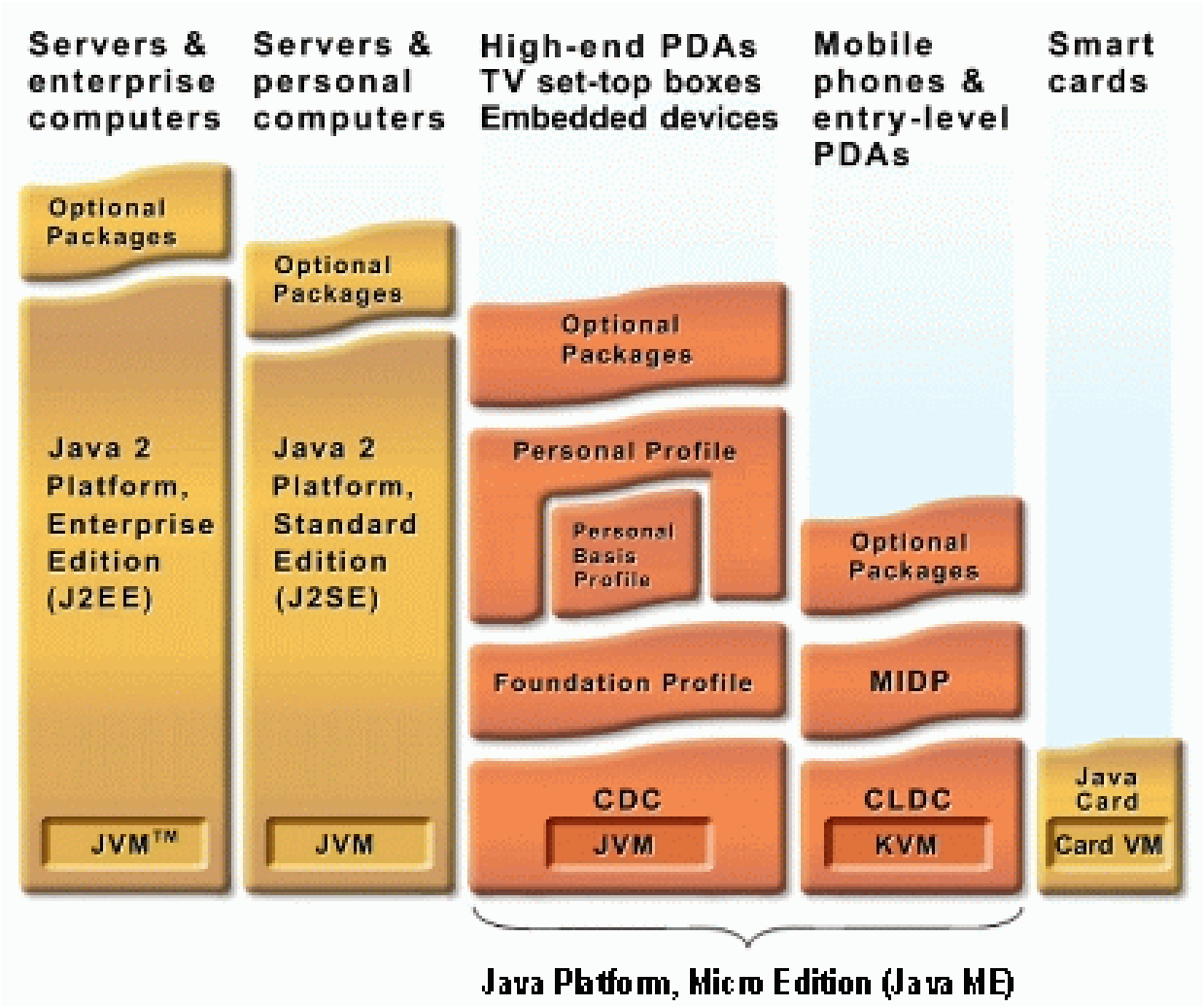

Illustration 11: Les différentes plates formes Java (source: http://java.sun.com [J2ME01])

La configuration

C'est une API donnant accès aux fonctions de base du système. Les configurations les plus courantes sont :

- CLDC (Connected Limited Device Configuration), que l'on retrouve par exemple dans les téléphones mobiles
- CDC (Connected Device Configuration), qui est utilisée dans des décodeurs de télévision numérique

Le Nokia 6131 NFC utilise la configuration CLDC 1.1. Cette configuration contient :

- Le langage Java, et les fonctionnalités de la machine virtuelle KVM

- Les bibliothèques Java de base (java.lang.* et java.util.*)
- Les entrées/sorties
- Le réseau
- L'internationalisation (non disponible sur le 6131)

Ce que le CLDC ne gère pas :

- La gestion du cycle de vie de l'application (installation, lancement et effacement)
- L'interface utilisateur
- La gestion des événements

Le profil

C'est une API donnant accès aux fonctions spécifiques de la plate-forme.

Les profils les plus courants sont :

- MIDP (Mobile Information Device Profile), dont sont équipés les téléphones WAP J2ME
- DoJa, développé par NTT DoCoMo pour les téléphones i-mode J2ME

Le profil du Nokia 6131 NFC est le MIDP 2.0

4.1.4 Le réseau GPRS

Le General Packet Radio Service est une extension du protocole GSM qui ajoute à ce dernier la transmission par paquets, plus adaptée à la transmission de données.

Le réseau GSM fonctionne en mode « circuit » : des ressources sont allouées pendant toute la durée de la communication. Ce mode de fonctionnement est adapté aux communications vocales, et limite le risque de coupures intempestives.

Le mode paquet du GPRS permet de fournir une connectivité IP permanente au téléphone mobile. Les ressources sont allouées uniquement lorsque des données doivent être transférées, ce qui permet d'économiser la bande passante radio.

L'utilisation du GPRS est facturée au débit, avec un minimum de facturation d'un kilo-octet.

Nom'ad doit donc être une solution de télégestion fiable, ergonomique, fonctionnant dans un environnement restreint, tout en continuant d'apporter les mêmes fonctionnalités que les systèmes centralisés classiques.

Devoir développer cette application sur un téléphone mobile a donc été un véritable défi.

4.2 LES CHOIX ARCHITECTURAUX

Une architecture modulaire

L'application doit pouvoir aisément être adaptée à d'autres téléphones que le Nokia 6131. Comme expliqué précédemment, une bonne partie de l'environnement J2ME peut varier d'une plate-forme à l'autre.

On s'orientera donc vers une architecture modulaire : toutes les fonctionnalités dépendantes de l'environnement sont gérées dans des modules spécialisés.

L'adaptation aux ressources

Les ressources disponibles dans l'environnement J2ME sont très limitées, et peuvent varier d'une plate-forme à l'autre. Des traitements qui seraient quasi instantanés sur une machine de bureau peuvent prendre plusieurs secondes sur une plateforme portable, et donc causer un blocage de l'interface graphique.

Tous les traitements potentiellement longs doivent donc être effectués dans une autre tâche que celle qui a la charge de l'interface utilisateur.

La résilience

De nombreux événements imprévus peuvent entraver le bon fonctionnement de l'application. Les communications réseau peuvent échouer à tout moment, l'application peut s'interrompre de manière inattendue, par exemple en cas de panne de batterie, ou suite à certaines bogues de l'interface graphique.

L'application doit donc être conçue pour résister à tous les événements imprévus.

5 CONCEPTION

L'application est composée d' un ensemble de modules, décrits ci-après :

Cette conception modulaire nous permet d'en maîtriser la complexité, de localiser et de corriger les défauts plus efficacement, et d'être plus adapté aux demandes d'évolution, et notamment au portage vers d'autres plateformes.

5.1 LA MACHINE À ÉTATS

L'application est architecturée autour d'une machine à états. A chaque écran correspond un état. Trois types d'événements sont susceptibles de provoquer un changement d'état :

- L'appui sur une touche
- La présentation d'une carte NFC
- La réception d'une réponse du serveur ou un échec réseau

A chaque combinaison état-événement correspond une action à effectuer :

- Si l'événement n'est pas sensé se produire dans l'état actuel, l'application l'ignore, et affiche éventuellement une alerte d'erreur (par exemple, le fait de présenter une carte intervenant alors que l'application n'est pas dans l'état « en attente de carte intervenant » provoquera l'affichage du message « vous êtes déjà connecté »)
- Si l'événement est attendu, l'application effectue l'action demandée, et éventuellement change d'état.

Exemple :

		Événements	
		Présentation d'une carte bénéficiaire	Présentation d'une carte intervenant
États	Dans l'écran d'authentification	Affiche le message « ceci n'est pas une carte intervenant »	Affiche le nom de l'intervenant, et passe dans le menu principal
	Dans le menu principal	Aucune action	Affiche le message « vous êtes déjà connecté »

Tableau 2: Extrait de la machine à états

L'usage d'une machine à états apporte de nombreux avantages :

- Le code est nettement plus lisible. Toute la logique applicative est rassemblée en un seul endroit,

- Le risque d'imprévus est diminué : tous les événements susceptibles de concerner l'application sont réceptionnés par des modules spécialisés, puis transmis à la machine à états. Par exemple, si l'on désire savoir comment réagira l'application si, au moment de signer un compte-rendu, l'intervenant passe sa carte, au lieu de passer celle du bénéficiaire, il suffit de consulter la machine à états. L'événement « présentation d'une carte intervenant », s'il survient dans tout autre état que « dans l'écran d'authentification » provoquera l'affichage du message « vous êtes déjà connecté »,

- Les modifications de la logique applicative sont facilitées, et leur impact mieux maîtrisé. Dans l'exemple précédent, si l'on désire autoriser la signature d'un compte-rendu avec la carte intervenant, il suffira d'ajouter le traitement approprié dans la combinaison état-événement concernée.

5.2 LES TÂCHES

Le choix du multitâches a un coût important en terme d'augmentation de la complexité et du risque de pannes dû, par exemple, à la modification d'une variable par deux tâches différentes.

Mais avant de définir les tâches créées par l'application, répondons à une question importante :

L'application n'est-elle pas déjà multitâches ?

Même la plus simple des Midlets verra son code appelé par le système au moins trois fois :

- Au démarrage de la Midlet, le système appellera la méthode startApp().
- La méthode startApp() n'est pas statique, on peut donc en conclure qu'avant de l'appeler, le système a nécessairement dû instancier la Midlet, et donc exécuter tout son code statique.
- Pour signaler un événement, le système d'exploitation du téléphone appelle une méthode de l'application. On appelle cette méthode « callback ». Tous les événements reçus par l'application le sont par le biais de callbacks.

Rien ne nous garantit que le système utilisera la même tâche pour appeler startApp(), et pour appeler les callbacks.

Par exemple, supposons que l'on veuille modifier le mécanisme d'authentification, pour permettre une authentification par mot de passe ou par carte NFC, au choix :

- Après avoir entré son mot de passe, l'utilisateur appuie sur OK.
- Le système d'exploitation du téléphone appelle la callback du bouton OK.
- Le code de la callback ouvre une connexion GPRS, et interroge le serveur pour valider le mot de passe.
- Pendant toute la durée de la transaction réseau, la tâche système qui a appelé la callback du bouton OK reste indisponible : L'utilisateur ne peut donc pas utiliser les autres boutons, l'application semble bloquée.

Supposons maintenant que l'utilisateur présente sa carte. La détection d'une carte NFC provoque elle aussi l'appel d'un callback. Si le système utilise la même tâche pour appeler la callback du bouton OK et pour appeler la callback de lecture NFC, alors la lecture NFC échouera. Et si le système appelle la callback NFC depuis une autre tâche, on se trouvera confronté à un problème bien plus gênant : Deux tâches différentes seront en train de s'authentifier de deux manières différentes, en même temps !

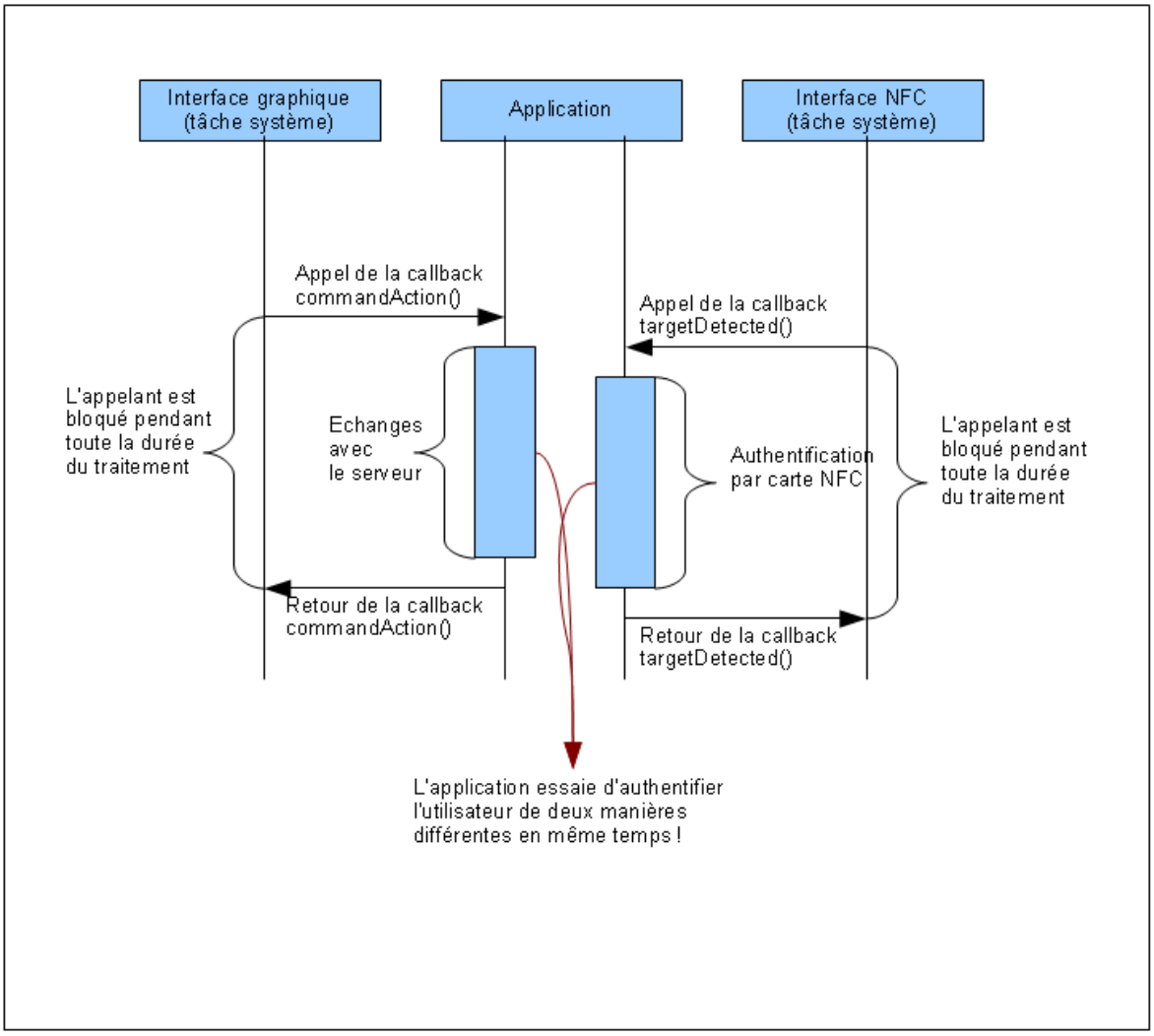

Illustration 12: Multitâches sans file d'attente

Pour éviter cela, l'application utilise une architecture multitâches simple et robuste :

Tous les événements reçus par l'application sont mis en file d'attente.

La tâche principale traite ces événements séquentiellement, dans l'ordre de leur arrivée.

Si l'on reprend l'exemple précédent, voici ce qui se passe :

- Après avoir entré son mot de passe, l'utilisateur appuie sur OK.
- Le système d'exploitation du téléphone appelle la callback du bouton OK.
- Le code de la callback crée un événement « appui du bouton OK », le met en file d'attente, réveille la tâche principale, et rend immédiatement la main à la tâche système.

La tâche principale traite l'événement. Pendant ce temps, d'autres événements, par exemple la présentation d'une carte NFC, peuvent être mis en file d'attente. La tâche principale les traitera dans l'ordre d'arrivée, et seulement après avoir terminé le traitement de l'événement « appui du bouton OK ».

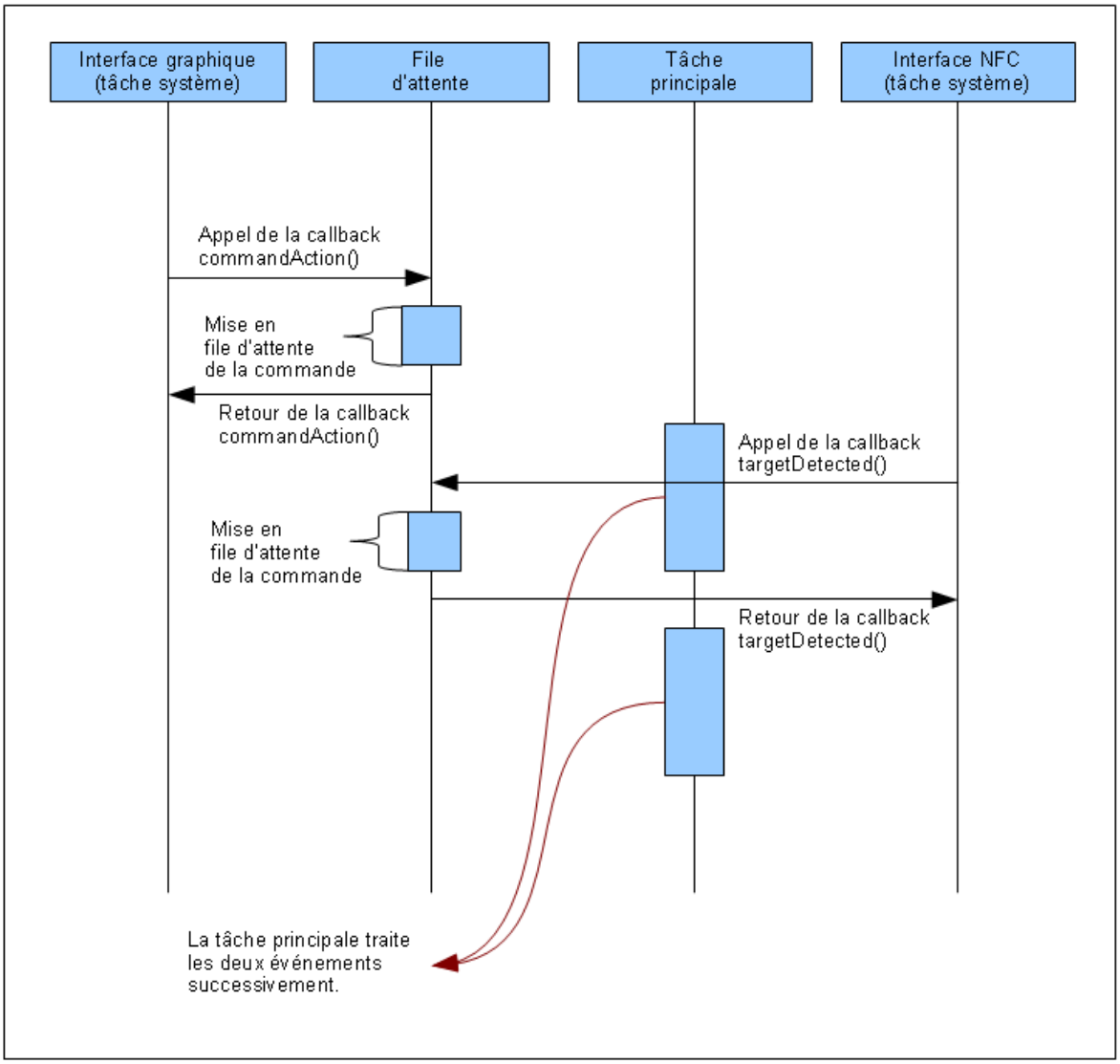

Illustration 13: Multitâches avec file d'attente

5.3 LA PERSISTANCE DES DONNÉES

L'application doit pouvoir être utilisée en l'absence de couverture réseau. En particulier, si l'intervenant remplit un compte-rendu d'intervention dans une zone non couverte par le GPRS, le compte-rendu ne doit pas être perdu, même s'il ferme l'application avant de revenir en zone de couverture.

Nous avons donc besoin d'une zone de stockage persistant pour les messages en attente d'envoi et pour le planning.

Le stockage de ces données dans des fichiers texte pose deux problèmes :

- Toute personne ayant accès au téléphone peut lire et éventuellement modifier ces fichiers.

- En cas d'extinction brutale du téléphone pendant l'écriture du fichier texte, ce dernier peut être laissé dans un état incohérent.

J2ME fournit un autre mécanisme de stockage de données persistantes : le RecordStore.

Les données présentes dans un RecordStore ne sont pas accessibles autrement qu'au travers de l'application qui a créé ce RecordStore. Les impératifs de confidentialités et de non falsification des données sont donc respectés.

De plus, l'implémentation du RecordStore nous garantit que les écritures sont « atomiques » c'est a dire insécables : en cas d'extinction brutale du téléphone, on n'a aucun risque de trouver une entrée du RecordStore à moitié modifiée.

Tous les échanges avec le RecordStore sont gérés par la classe RecordStoreInterface.

Comme tous les objets destinés à être stockés dans le RecordStore sont également destinés à être échangés par GPRS, on utilise le même format et les mêmes méthodes d'encodage pour la transmission et pour le stockage.

En Java standard, on utilise en général des outils de sérialisation pour encoder la représentation d'un objet dans un format adapté à la transmission et au stockage. Ces outils étant nettement trop lourds pour pouvoir être utilisés sur un téléphone mobile, tous les objets pouvant être transmis ou stockés possèdent leurs propres méthodes de sérialisation.

5.4 L'INTERFACE GRAPHIQUE

Pour gérer une interface graphique en J2ME, plusieurs techniques sont disponibles :

L'utilisation du Canvas : Cette technique a l'avantage de permettre un contrôle très précis de l'apparence des écrans, mais au prix d'une plus grande complexité. Les positions de tous les éléments graphiques doivent être calculés par l'application, et certaines fonctions, par exemple la saisie prédictive T9 sont inaccessibles.

Les écrans classiques : (classe Screen et ses dérivés) Avec cette technique, l'aspect de l'écran est contrôlé par le système, ce qui simplifie la programmation, mais apporte aussi de nombreuses limitations.

L'application Nom'ad utilise une combinaison de ces deux techniques : Les écrans non interactifs (le planning, et le récapitulatif du compte-rendu d'intervention) utilisent des Canvas, et les écrans interactifs utilisent les classes dérivées de la classe Screen.

Du point de vue de l'interface graphique, le planning peut être considéré comme une application indépendante :

- Le planning est une application de présentation des informations : l'utilisateur ne peut rien modifier.

- Les écrans classiques sont par contre très liés à la logique applicative : à chaque écran présenté correspond un état de l'application. Par exemple, l'application de saisie des compte-rendus refusera de présenter son écran de récapitulatif tant que la règle métier qui impose une date de fin postérieure à la date de début n'aura pas été respectée.

Pour cette raison, les interfaces graphique du planning et du reste de l'application seront présentées dans deux sous-chapitres distincts.

L'interface graphique du Planning

L'objet Canvas fonctionne suivant une logique très différente du reste de l'application :

Avec l'interface graphique classique, l'application déclare des objets graphiques et laisse le système se charger de leur présentation. Quand l'utilisateur passe d'un écran à l'autre, ou fait défiler un formulaire trop grand pour tenir sur l'écran, c'est le système qui se charge d'afficher, d'effacer ou de déplacer les composants graphiques définis par l'application.

Avec un Canvas, le système ne se charge plus que d'une seule chose : signaler à l'application qu'il est temps de redessiner ses composants graphiques.

Un objet dérivé de Canvas doit donc définir une callback nommée Paint, qui sera appelée à chaque fois que l'écran devra être redessiné.

Seule la callback Paint à le droit de dessiner des composants graphiques. Quand le système appelle cette callback, il lui fournit un objet de type Graphics qui donne accès à la portion d'écran à redessiner.

L'application de doit pas conserver cet objet pour tenter d'accéder à l'écran en dehors de la callback Paint. Le système peut accéder à l'écran à tout moment, par exemple pour signaler un appel entrant. Si l'application tente d'accéder à l'écran depuis sa propre tâche, on a un risque d'interblocage.

Pour modifier ce qui est affiché à l'écran, il faut donc demander au système d'appeler la callback Paint.

L'écran de planning affiche des informations nombreuses et complexes : chaque événement peut contenir, en plus des informations de base comme l'heure et le nom du bénéficiaire, une liste de services et des informations annexes (digicode, chien méchant...)

Le code de la callback Paint est pourtant particulièrement court, car elle délègue l'affichage des informations aux objets concernés.

Les événements à afficher sont stockés dans un tableau d'objets appartenant à la classe Event. Cette classe contient une méthode PaintInBox, très semblable à la callback Paint de l'écran de planning.

La callback Paint se contente de calculer les positions des différents événements visibles à l'écran, puis elle appelle la méthode PaintInBox de chaque événement, en lui fournissant l'objet Graphics qu'elle a reçu du système, et en lui indiquent les coordonnées de la portion d'écran qui lui a été assignée.

L'objet Event contient une collection d'objets de type Service. Ces objets implémentent également la méthode PaintInBox. Pour afficher ses services, l'objet Event va donc définir, dans la portion d'écran qui lui a été assignée, une sous-portion dédiée à chaque service, et leur en déléguer l'affichage.

La gestion des actions de l'utilisateur dispose elle aussi d'un mécanisme différent de celui en vigueur dans le reste de l'application. Six commandes sont disponibles :

- Les touches gauche et droite permettent de faire défiler les jours.
- Les touches haut et bas permettent de faire défiler les interventions.

- La touche centrale permet d'entrer et de sortir du mode détaillé.

- Une option de menu permet de quitter l'application Planning.

Il est impossible de définir plus de deux options de menu sans que cela génère un sous-menu, et une application de planning qui imposerait le passage par un sous-menu pour passer d'un jour ou d'une intervention à l'autre serait particulièrement peu pratique. Heureusement, l'objet Canvas nous offre un accès direct aux touches fléchées et à la touche centrale.

Il suffit pour cela de définir la méthode callback keyPressed. Cette dernière est appelée par le système à chaque appui d'une touche. Seuls les appuis sur des touches non assignées à des options de menu sont signalés.

L'application Planning définit donc une seule option de menu : celle qui permet de quitter le planning, et utilise keyPressed pour toutes ses autres interactions avec l'utilisateur.

L'interface graphique du reste de l'application

A chaque écran correspond un objet dérivé de la classe Screen. On peut découper les écrans en trois catégories, en fonction de leur niveau d'interactivité :

- Le contenu de la plupart des écrans est statique: il suffit dont de définir l'ensemble des composants graphiques contenus dans un écran donné dès l'initialisation de cet écran. Par exemple, l'écran de saisie de la date de début contient uniquement un champ date.

- Certains écrans ont un contenu qui peut varier en fonction du contexte : l'écran de saisie de messages propose une liste de destinataires qui peut être mise à jour par le serveur. Mais si une telle mise à jour a lieu pendant la saisie du message, la nouvelle liste de destinataires n'entrera pas immédiatement en vigueur, mais seulement à la prochaine utilisation de cet écran.

- Certains écrans ont un contenu qui peut varier automatiquement, en l'absence de toute action de l'utilisateur: c'est le cas du menu principal. L'entrée « envoyer N messages en attente » est automatiquement mise à jour à chaque changement du nombre de messages en attente.

Les changements d'écran

Les changements d'écran sont tous gérés par la machine à états. A chaque événement, la logique applicative de la machine à états décide de l'action à effectuer, et, en cas de changement d'état, du nouvel écran à afficher.

Le mécanisme de mise à jour automatique du contenu des écrans

Il est inspiré du mécanisme des PropertyChangeListeners utilisés en Java standard.

Comme l'environnement J2ME ne fournit pas ce mécanisme, j'en ai développé une version simplifiée :

La file d'attente des événements possède une propriété « nombre de messages en attente » et une liste d' « abonnés » aux changements de cette propriété. A chaque fois qu'un message entre ou sort de la file d'attente, tous les abonnés sont informés.

Pour l'instant, seul le menu principal est abonné à la propriété « nombre de messages en attente ». Cela permet de maintenir à jour le nombre affiché dans l'option d'envoi des messages en attente.

Ce mécanisme n'apporte donc pas encore de gain au niveau de la complexité de l'application, car on a une seule propriété et un seul abonné.

Les prochaines versions de l'application devront probablement gérer de nombreuses propriétés visibles et modifiables en plusieurs endroits. Ce mécanisme permettra alors de maîtriser la complexité, en découplant les propriétés de leur affichage.

Les actions de l'utilisateur

Chaque écran possède un menu permettant de le valider, de passer à l'écran suivant ou précédent...

Pour qu'un menu fonctionne il faut :

- Définir les options de menu et les ajouter à l'écran.
- Définir le « Listener » : c'est l'objet qui doit être prévenu quand une option de menu est séléctionnée.
- Dans le Listener, définir le callback commandAction.

Dans l'application Nom'ad, chaque écran est son propre Listener. Cela permet de simplifier la gestion des menus, en laissant le système faire un premier tri. En effet, certaines options de menu sont communes à plusieurs écrans : par exemple, presque tous les écrans possèdent une option de retour à l'écran précédent.

Comme chaque écran est son propre listener, il n'y aucune ambiguïté possible, un écran ne peut pas recevoir des commandes ne le concernant pas.

Quand un écran reçoit une commande, il se contente de la transmettre à la machine à états, en précisant le contexte. Quand l'utilisateur sélectionne « retour » dans l'écran « heure de début », la méthode commandAction de cet écran reçoit l'événement, et envoie à la machine à états un événement « retour sélectionné dans l'écran heure de début »

5.5 Les entrées/sorties

L'application dialogue avec l'utilisateur, avec le réseau, avec les cartes NFC, et avec le mécanisme de persistance des données.

Toutes ces entrées-sorties se font en utilisant des méthodes fortement dépendantes de l'environnement. Il est donc hors de question de mélanger le code d'entrée-sorties avec la logique applicative. Chaque type d'entrée-sortie est donc géré par un objet spécialisé.

5.5.1 L'interface réseau

Le 6131 est compatible avec le réseau GPRS. Pour accéder au serveur, on utilise un code très proche de celui qu'on utiliserait sur une plateforme de bureau, en java standard.

Le module « NetworkInterface » se charge du traitement des demandes d'envoi de données provenant de la machine à états, et de lui transmettre les réponses du serveur. Pour éviter tout blocage du reste de l'application, ce module gère une file d'attente d'envoi. Cette file d'attente est gérée par le module de persistance des données, ce qui nous garantit qu'aucun message à envoyer ne sera perdu, même en cas d'extinction brutale du téléphone.

Pour envoyer un message au serveur, on appelle la méthode postAsynchronously, qui effectuera les tâches suivantes :

- Le messages est encapsulé dans un objet de type OutgoingMessage, qui contient un compteur d'essais, et la date de la dernière tentative d'envoi.
- Cet objet est inséré dans la file d'attente d'envoi.
- La tâche d'envoi est réveillée.

La tâche d'envoi va alors tenter d'envoyer tous les messages en file d'attente, en respectant l'ordre « premier entré – premier sorti »

Le recordStore n'offre normalement aucune garantie quant à l'ordre de présentation des éléments qu'il contient, sauf si on demande à les obtenir dans un ordre précis, en utilisant un recordComparator.

L'application fournit au recordStore une méthode permettant de comparer deux enregistrements, et de lui indiquer lequel des deux doit être considéré comme le prédécesseur de l'autre. Grâce à cette méthode, le recordStore est en mesure d'énumérer les enregistrements dans un ordre donné.

Les méta-données contenues dans les OutgoingMessages permettent de les trier par ordre d'arrivée en file d'attente. On fournit donc au recordStore un recordComparator capable d'identifier le plus ancien de deux OutgoingMessages, ce qui nous garantit que la liste de messages obtenue sera bien triée dans l'ordre chronologique d'arrivée.

A chaque tentative d'envoi, le message sera soit retiré de la file, soit modifié (incrémentation du compteur de tentatives d'envoi).

Le compteur de tentatives d'envoi est ajouté au message transmis au serveur. Cela permet de différencier les messages envoyés en temps réel, et qui ont donc un compteur de tentatives d'envoi à zéro, des messages envoyés en différé.

5.5.2 L'Identification forte de l'utilisateur

L'identification est dite « forte » quand elle apporte un haut niveau de protection contre l'usurpation d'identité. Examinons les différentes méthodes d'identification :

Comparatif des méthodes d'identification

Trois types d'information permettent d'identifier une personne :

- Ce qu'elle est :
  Les méthodes biométriques vérifient l'identité de la personne en contrôlant un ou plusieurs paramètres physiques : Empreinte digitale, Empreinte vocale...

- Ce qu'elle possède :
  La personne est munie d'un document lui permettant de prouver son identité.

- Ce qu'elle sait :
  La personne prouve son identité à l'aide d'une information qu'elle est seule à connaître.

Examinons des avantages et inconvénients de ces trois moyens d'identification :

L'identification biométrique de la personne

Elle permet une identification nettement plus sûre qu'avec les méthodes exploitant ce que la personne connaît ou possède, mais pose de gros problèmes de respect de la vie privée, car elle permet d'identifier une personne sans son consentement.

L'usage de méthodes biométriques est soumis à autorisation de la CNIL (Commission Nationale Informatique et Libertés).

L'identification de la personne par ce qu'elle possède

La personne prouve son identité à l'aide d'un document personnel.

Cette méthode pose moins de problèmes de respect de la vie privée : la personne ne peut pas être identifiée contre son gré ou à son insu. Mais elle est également moins sûre, car le document identifiant peut être volé, ou partagé entre plusieurs personnes.

L'identification de la personne par ce qu'elle sait

La personne s'identifie avec un mot de passe, un code secret, ou toute autre information connue uniquement d'elle-même.

En pratique, pour limiter les risques d'usurpation d'identité, on utilise souvent une combinaison d'au moins deux de ces trois types d'information :

- La carte de crédit combine le « ce qu'elle sait » (son code) avec le « ce qu'elle possède » (sa carte)

- Certains lecteurs d'empreintes digitales combinent les trois éléments : l'empreinte digitale à vérifier est stockée sur une carte à puce, et l'utilisateur doit entrer un code personnel pour autoriser la vérification de son empreinte digitale.

La méthode d'identification choisie : la carte NFC

Elle est basée sur ce que la personne possède (la carte) et peut éventuellement être renforcée en demandant quelque chose que la personne sait (un code).

Cette méthode a de nombreux avantages :

Simplicité d'emploi

Il suffit d'approcher la carte du téléphone.

Robustesse

La carte résiste à l'eau, aux champs magnétiques et à l'électricité statique. Les cartes à puces classiques peuvent devenir illisibles si leurs contacts sont sales ou endommagés.

Sécurité

Pour l'instant, le logiciel identifie les cartes grâce à leur identifiant unique.

La fabrication d'une fausse carte NFC possédant le même identifiant unique qu'une carte existante nécessite des compétences en électronique, du matériel, et la « fausse carte » ainsi créée sera nettement plus volumineuse qu'une vraie.

Pour nous prémunir face à ce risque (qui est tout de même nettement plus faible qu'avec les étiquettes visuelles) nous avons prévu d'améliorer le module d'authentification pour qu'il tire parti des techniques d'authentification sécurisées fournies par les cartes NFC.

L'interface NFC

Quand une carte NFC est détectée, le système d'exploitation appelle la methode targetDetected().

Cette dernière reçoit un objet lui permettant d'accéder au contenu de la carte, mais les traitements effectués doivent être les plus rapides possibles, car cet objet cesse d'être utilisable dès que la carte se trouve hors de portée du lecteur.

Pour que l'utilisateur ne soit pas obligé de maintenir la carte près du lecteur trop longtemps, on commence par faire une copie des données à valider. Si la validation prend trop de temps, on ne sera donc pas affecté par la perte du contact avec la carte en cours de validation.

Pourquoi avoir choisi le format NDEF Record ?

Une carte NFC contient entre 1 et plus de 64 kilo-octets de données accessibles par plusieurs méthodes. Il est possible d'accéder directement aux blocs de données : la carte NFC se présente alors sous la forme d'un ensemble de secteurs, chaque secteur étant protégé par deux clés, l'une protégeant la lecture et l'autre l'écriture.

Le format NDEF record nous permet de manipuler des données structurées nettement plus simples d'emploi que les données brutes présentes sur la carte :

Lors de l'écriture, l'application crée un objet au format NDEF record, puis laisse le système traduire cet objet en un format compatible avec la carte.

Lors de la lecture, le système lit les secteurs de la carte, et s'ils contiennent des données au format NDEF record, assure la traduction.

Les NDEF records ont également un autre avantage : on peut y stocker des structures immédiatement reconnaissables par le téléphone. Par exemple, une URL peut être stockée dans un NDEF Record. Si on présente au téléphone une carte contenant un tel NDEF Record, le téléphone proposera alors automatiquement le lancement du navigateur, pour visiter l'URL.

Chaque carte intervenant créée par l'application Nom'ad contient une URL qui pointe vers le serveur d'Hippocad, et contient l'identifiant de l'intervenant.

Ce dernier peut donc utiliser la même carte pour s'authentifier dans l'application, et pour se connecter au serveur d'Hippocad à l'aide du navigateur du téléphone.

5.5.3 L'Échanges de données entre le serveur et le téléphone

Il y a deux types d'échange :

Du serveur vers le téléphone : planning, liste des services, coordonnées des destinataires de messages et messages préparamétrés.

Du téléphone vers le serveur : compte rendu d'intervention.

En GPRS, les communications ont toujours lieu à l'initiative du terminal mobile. Pour se maintenir à jour, ce dernier doit donc régulièrement interroger le serveur.

On utilise le protocole HTTP en mode POST.

Chaque objet susceptible d'être transmis au serveur possède une méthode permettant d'obtenir sa représentation sous la forme d'une chaîne de caractères pouvant être transmise par HTTP, et chaque objet pouvant être reçu du serveur possède une méthode permettant de décoder une telle chaine de caractères.

On utilise donc le même format pour la réception des objets et pour l'envoi des réponses.

L'envoi des messages au serveur :

L'application compose une chaîne de caractères contenant les variables à envoyer. La couche réseau met cette chaine dans la file d'attente d'envoi. Si le serveur est accessible, toutes les chaines en file d'attente sont envoyées. Si le serveur n'est pas accessible, la partie réseau attendra la prochaine demande d'envoi de message pour refaire une tentative. Il est également possible de forcer une tentative d'envoi en utilisant l'option de menu « envoyer les messages en attente ».

Après chaque envoi, la couche réseau reçoit une réponse du serveur, sous la forme d'une chaîne de caractères.

Cette chaîne peut contenir un certain nombre d'objets de divers types.

Comme la réception d'une réponse du serveur est considérées comme un événement, cette chaîne est envoyée à la machine à états, qui, après avoir vérifié que cet événement était pertinent (c'est à dire qu'on était dans l'état « en attente d'une réponse du serveur ») va décoder la chaîne.

La première étape du décodage consiste à identifier les objets qui s'y trouvent: à chaque type d'objet correspond un paragraphe, qu'on envoie à la classe concernée pour qu'elle le décode et instancie les objets décrits dans le paragraphe.

6 Problèmes Rencontrés

La conception de l'application n'a pas été extrêmement compliquée en elle-même, la mise en œuvre de celle-ci lors du codage fut quant à elle beaucoup plus difficile.

Les contraintes de l'interface graphique, tout comme l'instabilité chronique de l'environnement et la difficulté de résolution des bogues du téléphone font que le codage de l'application Nom'ad fut une tâche particulièrement ardue.

6.1 Le manque d'informations

Par rapport au monde J2ME, la recherche d'informations nécessaires au développement d'une application Java classique est une tâche relativement facile. De nombreux ouvrages sont disponibles sur le sujet, et la plupart des questions que peuvent se poser les développeurs admettent une et une seule réponse, qui est généralement facile à trouver.

Par exemple, un développeur désirant aligner trois boutons dans un formulaire trouvera des exemples de code dans la littérature spécialisée, et des informations détaillées sur l'ensemble des fonctionnalités de la classe Button.

Grâce à ces informations, il pourra écrire un programme simple et efficace effectuant exactement la tâche demandée, avec parfois de très petites différences d'apparence entre les plates-formes. Par exemple, sur une machine disposant d'une police de caractère un peu plus grande que prévu, le label du bouton ne s'affichera pas entièrement.

Si l'on tolère ces petites différences d'affichage, la programmation Java peut se faire en toute confiance.

Avec J2ME, on quitte le monde des certitudes, du « Write Once, Run Anywhere » garanti, pour entrer dans un monde d'essais et d'erreurs, et de recherche empirique.

Un monde où la quasi-totalité des fonctionnalités du langage sont optionnelles, implémentées différemment d'un modèle de téléphone à l'autre, et parfois d'une version à l'autre d'un même modèle de téléphone.

Un monde où la source d'information la plus fiable est rarement la documentation « officielle » de la plate-forme, et souvent le blog d'un autre développeur, ou un forum d'entraide entre développeurs.

Voici un exemple typique et représentatif des problèmes rencontrés lors du développement d'une fonction a priori simple : le lancement du navigateur depuis l'application.

Vouloir lancer le navigateur Web du téléphone depuis une Midlet (application Java pour mobile) ne semble a priori pas irréaliste : sur PC, une telle fonctionnalité ne demande que quelques lignes de code.

Avec J2ME, il faut commencer par chercher dans la documentation de la plate-forme.

Le « Nokia 6131 NFC Programmer's Guide » semble être un point d'entrée idéal pour une recherche sur ce sujet. Mais malheureusement, et comme son titre ne l'indique pas, ce document ne contient que peu d'informations expliquant comment écrire des programmes pour ce mobile. La majeure partie des 28 pages est consacrée aux aspects NFC.

Le « Nokia 6131 NFC User's Guide » est un peu plus long : 52 pages. Il décrit le fonctionnement des outils de développement et de l'émulateur et donne des informations utiles concernant la signature du code, mais toujours pas d'informations sur la programmation proprement dite.

Les seules informations utiles dans ce domaine se trouvent dans les Javadoc fournies par Nokia. Or ces documentations techniques ne sont utiles qu'à partir du moment où l'on connaît les noms des classes à utiliser.

La recherche sur Internet nous mènera vers un fil de discussion du forum Nokia. Là, il nous faut constater que les avis divergent :

- D'après certains développeurs, le lancement du navigateur depuis la Midlet ne pose aucun problème.

- D'après un autre développeur, c'est une chose impossible.

- D'après un troisième, c'est possible, mais il faut que la Midlet demande la permission de le faire.

- D'après un quatrième (qui a essayé tous les exemples de code fournis par les autres développeurs) cela dépend du modèle de téléphone.

La résolution du problème se fait donc de manière empirique. En cas de réponse insatisfaisante, une solution palliative est à trouver.

6.2 LES CONTRAINTES DE SÉCURITÉ

De très nombreux sites internet proposent des Midlets à télécharger. Comme pour toute autre application, l'installation et l'utilisation d'une Midlet exposent l'utilisateur à divers risques : l'application peut modifier, détruire ou divulguer les informations qui lui sont accessibles. Ces problèmes sont plus préoccupants sur les plateformes mobiles que sur les ordinateurs de bureau, pour diverses raisons :

- L'utilisateur d'un téléphone mobile n'a pas nécessairement le même niveau de connaissances en informatique qu'un utilisateur d'ordinateur de bureau. Contrairement à ces derniers, l'utilisation d'un téléphone ne nécessite aucune connaissance particulière. On peut donc raisonnablement conclure que de nombreux utilisateurs n'ont pas été sensibilisés aux risques liés à l'utilisation de logiciels provenant de sources inconnues.

- Du fait des ses ressources limitées, un téléphone mobile ne possède pas les outils de sécurité classiques d'un ordinateur de bureau que sont les firewalls, antivirus et anti-spywares.

- Par sa nature, un téléphone mobile contient essentiellement des données privées, et donc confidentielles.

- Certaines actions effectuées par les Midlets peuvent se révéler très coûteuses pour l'abonné : imaginons, par exemple, qu'un éditeur peu scrupuleux diffuse un jeu envoyant discrètement des centaines de SMS à un N° surtaxé...

Pour toutes ces raisons, les Midlets sont entourées de tout un ensemble de sécurités.

Par défaut, une Midlet a accès a un ensemble très limité de ressources.

Pour avoir accès à certaines capacités, par exemple la connectivité Bluetooth, l'appareil photo, l'envoi de SMS, la Midlet va devoir obtenir des permissions.

6.2.1 L'obtention d'un certificat

Une Midlet non signée sera par défaut considérée comme hautement suspecte par le téléphone. Si cette Midlet a été installée manuellement par l'utilisateur, en utilisant un câble ou une liaison Bluetooth, le téléphone lui accordera certains droits, par exemple celui d'utiliser la liaison GPRS, mais imposera un message d'alerte avant la première tentative d'utilisation du réseau.

La même Midlet, si elle est téléchargée « over the air » c'est-à-dire en utilisant le navigateur web du téléphone, aura besoin d'être signée numériquement. Une recherche sur Internet montre que de nombreux développeurs J2ME découvrent ce « détail » au dernier moment...

La signature numérique d'une application est un moyen simple de garantir à l'utilisateur que l'application qu'il vient de télécharger provient bien de l'organisation à qui il pense avoir affaire.

Pour signer une application, on utilise la cryptographie à clé publique : Le développeur (Hippocad) crée une paire de clés cryptographiques :

- Une clé privée, qui lui permet de crypter des messages, et qui ne doit en aucun cas être divulguée.
- Une clé publique, qu'il peut librement diffuser auprès de tous les destinataires des messages qu'il a crypté avec sa clé privée.

Ce système permet également de signer des documents : si vous arrivez à décrypter un document avec une clé publique, vous avez alors la preuve que la personne qui a crypté ce document possède la clé privée correspondante, et donc que c'est bien elle qui l'a généré.

Quelqu'un qui voudrait usurper l'identité d'Hippocad n'a donc que deux moyens de parvenir à ses fins :

- Soit voler la clé privée du développeur

- Soit générer une autre paire de clés : S'il parvient à persuader les destinataires que sa clé publique est celle du développeur dont il veut usurper l'identité, il n'aura plus alors qu'à crypter ses faux messages avec sa clé privée.

Le premier problème se règle facilement : il suffit de conserver sa clé privée en lieu sûr...

Le deuxième problème est plus complexe : des gens mal intentionnés pourraient très bien créer de très nombreux faux sites Hippocad, et en remplaçant notre clé publique par la leur, diffuser des documents semblant provenir d'Hippocad.

Pour éviter cela, il faut trouver un moyen de prouver aux destinataires que la clé publique qu'ils viennent de récupérer provient bien d'Hippocad. Pour cela, on va faire établir un « certificat d'authenticité » de notre clé publique par un « tiers de confiance ». Il existe pour cela des entreprises mondialement connues, comme Thawte ou VeriSign.

6.2.2 La création d'une chaîne de confiance

La chaîne de confiance est constituée d'un ensemble de certificats établissant un lien entre l'application et une autorité de certification reconnue par le téléphone.

Les autorités de certification possèdent également des paires de clés, mais leurs clés publiques sont pré-chargées dans de nombreux navigateurs web et téléphones portables : elles sont donc à la racine de la « chaîne de confiance ».

Après avoir vérifié notre identité, l'entreprise Thawte a donc rédigé un document numérique contenant notre clé publique et notre raison sociale, et signé avec sa clé privée.

La vérification de l'identité de l'auteur d'une application se fait en remontant la chaîne de confiance des certificats :

- L'application Nom'ad est chargée sur le téléphone. Le serveur qui a fourni l'application fournit également la clé publique d'Hippocad.

- Le téléphone vérifie la signature : la clé publique appartient bien au signataire de l'application... qui peut être n'importe qui.

- Heureusement, cette clé publique est également présente dans un certificat qui dit en substance « Nous, Thawte Consulting Inc. déclarons sur l'honneur avoir vérifié l'identité de la société Hippocad, et avoir reçu en mains propres de leur PDG la clé publique ci-dessous... »

- Le téléphone consulte sa liste de certificats racine (un ensemble de clés publiques dont l'authenticité est indiscutable, et qui appartiennent toutes à des entreprises auxquelles Nokia fait confiance)

- Et le téléphone n'y trouve pas le certificat de Thawte.

Et c'est là que les problèmes commencent : Le téléphone possède bien le certificat de VeriSign, mais pas celui de Thawte !

Cela signifie donc que Nokia fait confiance à VeriSign, mais pas à Thawte. Étrange...

Nokia considère-t-il que les contrôles d'identité effectués par Thawte ne sont pas assez sûrs ? Thawte nous a accordé sa « confiance » sous la forme d'un certificat, dans lequel ils affirment être certains de notre identité. Mais apparemment, Nokia n'a pas accordé sa confiance à Thawte.

En fait, l'explication est bien plus simple : Thawte est une filiale de Verisign. Les deux entreprises possèdent leur propre paire de clés, et Verisign a publié un certificat dans lequel elle affirme faire confiance à Thawte.

En résumé :

- Nokia a confiance en VeriSign

- VeriSign a confiance en Thawte

- Thawte a confiance en Hippocad

Mais tous les téléphones Nokia ne sont pas au courant de la confiance accordée par VeriSign en Thawte. La chaîne de confiance est donc brisée.

Pour régler ce problème, il nous a suffi d'insérer dans notre certificat le certificat proclamant la confiance de VeriSign en Thawte.

Mais de tels problèmes de signature peuvent être la cause de pertes de temps, surtout quand on est novice dans le domaine de la signature numérique.

Et la détention d'un certificat valide n'est que la première étape...

6.2.3 Signatures et permissions

Notre téléphone a maintenant la preuve que l'application qu'il a téléchargée provient bien d'Hippocad, une société dont l'identité a été vérifiée par Thawte.

Mais ce n'est pas pour cela qu'il va accorder tous les droits à notre application.

Notre certificat (facturé 299 $/an) ne prouve que notre identité. Dans certains cas, un téléphone peut exiger d'autres preuves avant d'accorder certains droits à une application.

Par exemple, certains opérateurs téléphoniques distribuent des téléphones modifiés : certaines ressources de ces téléphones ne sont accessibles qu'aux applications signées avec le certificat de l'opérateur.

D'autres ressources ne sont accessibles qu'avec la signature du constructeur du téléphone. C'est apparemment le cas du N° de la carte SIM, qui est apparemment une donnée trop personnelle pour que des applications non signées par l'opérateur et/ou le constructeur y aient accès.

6.3 LES CONTRAINTES RÉSEAU

L'application doit être optimisée pour tenir compte des contraintes du réseau GPRS :

- Le trafic GPRS est facturé au volume. L'application doit donc optimiser ses échanges réseau, et éviter toute communication inutile, en mettant en cache les données reçues du serveur. Le minimum de facturation étant d'un Ko par transaction, on limitera également le nombre de communications.

- Seul le téléphone peut initier une communication GPRS avec le serveur. Pour se maintenir à jour, l'application doit donc régulièrement consulter le serveur, car les mises à jour ne se font pas à l'initiative de ce dernier.

- La connectivité GPRS peut être perdue à tout moment, il faut en tenir compte en contrôlant la cohérence des données reçues du serveur.

6.4 LES CONTRAINTES DE RESSOURCES SYSTÈME

L'environnement embarqué du téléphone mobile est particulièrement limité tant au niveau de la puissance de calcul (un processeur ARM9 cadencé à 229 Mhz) que de la mémoire disponible (11 Mo). Or le langage Java est très gourmand en ressources.

J2SE ne fournit pas de méthode de désallocation des objets qui ne sont plus utilisés. Ces derniers sont désalloués automatiquement par un processus nommé « garbage collector ».

Un programme J2SE qui crée régulièrement de nouveaux objets consommera donc beaucoup de mémoire. Le garbage collector sera donc invoqué plus souvent, d'où une baisse de performances. L'usage de la classe StringBuffer, par exemple, peut être particulièrement couteux.

L'exemple du StringBuffer :

StringBuffer est une classe permettant des manipulations de chaînes nettement plus souples qu'avec la classe String.

Par exemple, supposons qu'on veuille se connecter à un serveur web pour y récupérer une page. La javadoc de la classe HttpConnexion nous fournit un programme d'exemple complet. Il n'y manque que le traitement des données reçues.

Sans StringBuffer, on sera tenté d'écrire :

```
int ch;

String result;

while (ch = inputStream.read() != -1)

{

     result = new String(result + (char)ch);

}
```

A chaque tour de boucle, la chaîne des caractères reçus jusqu'à présent sera concaténée avec le dernier caractère reçu, et le résultat sera stocké dans une nouvelle chaîne.

Cette boucle instanciera donc une chaîne de 1 caractère, puis une de deux, puis une de trois... la consommation mémoire sera donc proche de la moitié du carré du nombre de caractères lus.

La lecture d'une page web de seulement 10 kilo-octets par cette méthode suffira probablement à bloquer l'application, ou à la ralentir fortement, si le garbage collector parvient à libérer la mémoire à temps.

Pour éviter un tel gaspillage de mémoire, on peut être tenté d'utiliser la classe StringBuffer :

```
int ch;

StringBuffer result = new StringBuffer();

while (ch = inputStream.read() != -1)

{

     result.append((char)ch);

}
```

On pourrait croire que cette deuxième méthode est plus économe : un seul objet est créé, alors que la méthode précédente créait un objet par caractère lu.

En fait, rien n'a été optimisé. Cette deuxième technique n'a fait que déplacer le problème.

Le code source de Java ayant été publié, on peut consulter le code de la méthode append() de StringBuffer. A l'initialisation, les StringBuffers sont créés avec une petite marge de manoeuvre. On peut donc y ajouter quelques caractères sans avoir besoin de créer de nouveaux objets, mais dès que la taille de la chaîne atteint la taille initialement allouée, StringBuffer utilise la méthode décrite précédemment : pour ajouter un seul caractère, la méthode append() copie l'ensemble de la chaîne dans une nouvelle chaîne de N+1 caractères, sans ajouter la moindre marge. On consomme donc presque autant de mémoire que dans la première méthode.

Le code source de la classe StringBuffer pour J2ME n'étant pas disponible, on ne sait pas si son fonctionnement est plus économe que la version J2SE.

Pour le savoir, il faudrait rédiger un programme de test qui mesure la consommation mémoire de StringBuffer.

6.5 Les limites de l'interface graphique

Pour afficher un écran en J2ME, il faut créer un objet de type Displayable. J2ME fournit deux types de Displayables : la famille des Screens, et le Canvas.

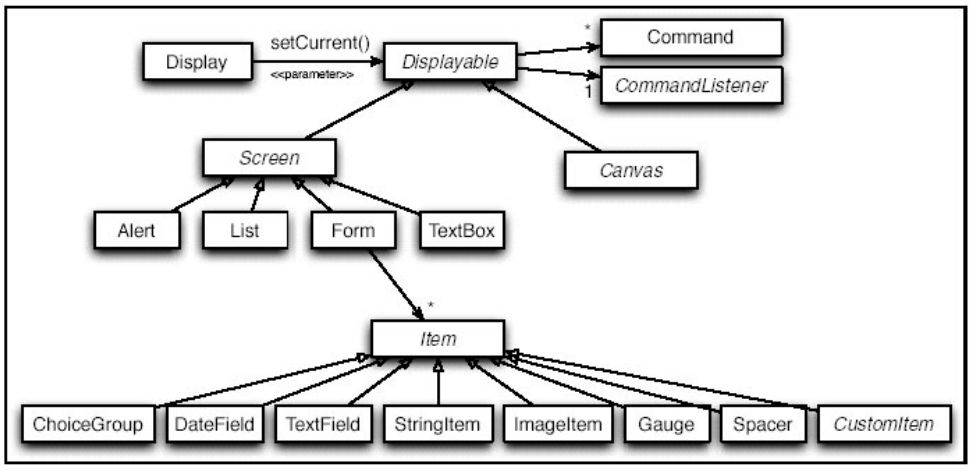

Illustration 14: Les objets disponibles pour la génération d'écrans J2ME

Pour afficher une interface graphique sur l'écran du téléphone, deux possibilités sont disponibles :

- Utiliser un Displayable qui hérite de Screen. C'est la méthode classique.
- Utiliser l'objet Canvas. Cette méthode est essentiellement utilisée par les jeux.

Examinons les avantages et inconvénients de ces deux méthodes.

6.5.1 Le choix du Displayable utilisé

Méthode classique

L'écran est constitué d'une sous-classe de Screen, généralement Form, car c'est la seule classe qui permette la juxtaposition de plusieurs éléments différents sur un même écran.

Cette limitation est plutôt étonnante : Si l'on désire afficher une liste, l'objet List semble être le choix le plus pertinent. Cet objet nous permettra d'afficher une liste, et rien d'autre.

Impossible donc d'avoir à l'écran une liste accompagnée d'un champ date, d'une case à cocher, ou même d'une simple ligne de texte autre que son titre.

Pour afficher une liste accompagnée d'autres éléments, on peut utiliser u Form contenant un ChoiceGroup.

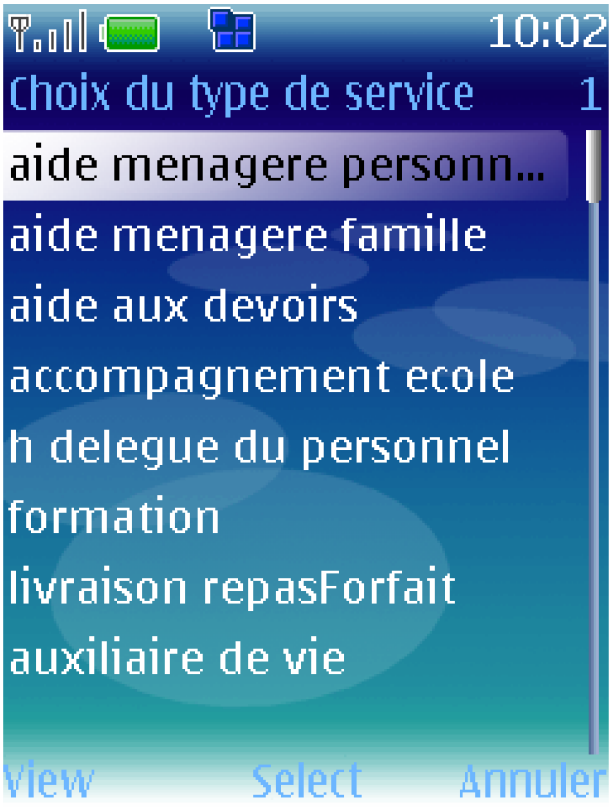

Illustration 15: Une Liste

Illustration 16: Un ChoiceGroup

Mais comme le ChoiceGroup n'est pas destiné à remplacer la liste, il ne fournira donc pas les mêmes services que cette dernière. Le mode de choix « implicite » n'est pas autorisé pour un ChoiceGroup. Pour déclencher une action associée à un élément d'un ChoiceGroup, il faut donc cocher son bouton radio, puis utiliser une autre commande pour déclencher l'action.

Il existe heureusement une autre méthode de création d'écrans : utiliser un Displayable de type Canvas.

Utilisation du Canvas

Le Canvas est une classe dérivée de Displayable qui permet un contrôle total de l'aspect de l'écran, mais ce contrôle a un coût : le développent d'application utilisant un Canvas est nettement plus complexe (par exemple, pour afficher une liste, il faudra calculer les positions à l'écran de chaque élément de la liste).

Et surtout, l'usage des composants graphiques de type Screen n'est alors plus possible. Si la création de composants simples comme des listes reste faisable dans un Canvas, les composants complexes, par exemple une zone d'entrée de texte, ont un coût prohibitif en termes de temps de développement.

Pour gérer une zone d'entrée de texte avec un Canvas, l'application doit détecter tous les appuis sur les touches numériques, et implémenter, entre autres, son propre algorithme de saisie prédictive T9 !

Le coût d'un tel composant, tant en terme de temps de développement que de ressources système utilisées, est prohibitif.

6.5.2 Les limites du menu

En bas de chaque écran (qu'il soit réalisé en Canvas, ou avec la méthode classique) se trouve un menu indiquant les actions associées aux deux touches bleues et à la touche centrale.

L'application n'a qu'un contrôle partiel sur ce menu. Les décisions concernant le positionnement des options de menu sont prises par le téléphone, d'une manière qui est souvent impossible à prédire :

D'après la Javadoc, si les options de menu sont trop nombreuses, un sous-menu est automatiquement créé. L'application peut assigner un niveau d'importance à chaque option de menu. Dans la mesure du possible, les options les plus importantes seront directement accessibles, et les autres le seront via le sous-menu.

En pratique, le téléphone gère les menus d'une manière non conforme à ce qui est annoncé dans la javadoc : dans le cas d'un menu avec trois options un sous-menu est créé, bien que le téléphone ait la place d'afficher les trois options. Pour activer la troisième option, l'utilisateur doit donc ouvrir le sous-menu, puis sélectionner l'option parmi une liste proposant un seul choix possible.

Il est difficile d'expliquer au client qu'on est incapable de proposer trois options de menu en accès direct à cause d'une limitation du téléphone.

Le seul moyen d'éviter l'apparition d'un sous-menu proposant un seul choix est de ne faire que des écrans proposant un nombre d'options différent de trois. Et trois n'est pas le seul nombre d'options qui pose problème : quand on affiche une alerte, il n'est pas possible d'avoir une barre de menu vide. L'alerte est automatiquement accompagnée d'un bouton de fermeture. Ce bouton ne peut pas être supprimé, il ne peut qu'être redéfini.

Pour obtenir une alerte sans bouton de fermeture, on devra donc y ajouter une commande par défaut qui ne fait rien. Et pour que cette commande ne soit pas visible, on remplace son label par une chaîne vide.

Malheureusement, les développeurs de Nokia ont manifestement décidé que la présence d'une chaîne vide ou ne contenant que des espaces ne pouvait être due qu'à un oubli de la part des développeurs, et non à une réelle intention d'obtenir une option de menu invisible.

Toute tentative d'obtention d'un bouton invisible se soldera à l'exécution par l'apparition d'un label par défaut. La seule solution de contournement consiste à définir un label constitué d'un simple point. Le bouton inactif n'est donc pas totalement invisible, mais nettement plus discret.

6.6 Différences entre le téléphone et le simulateur

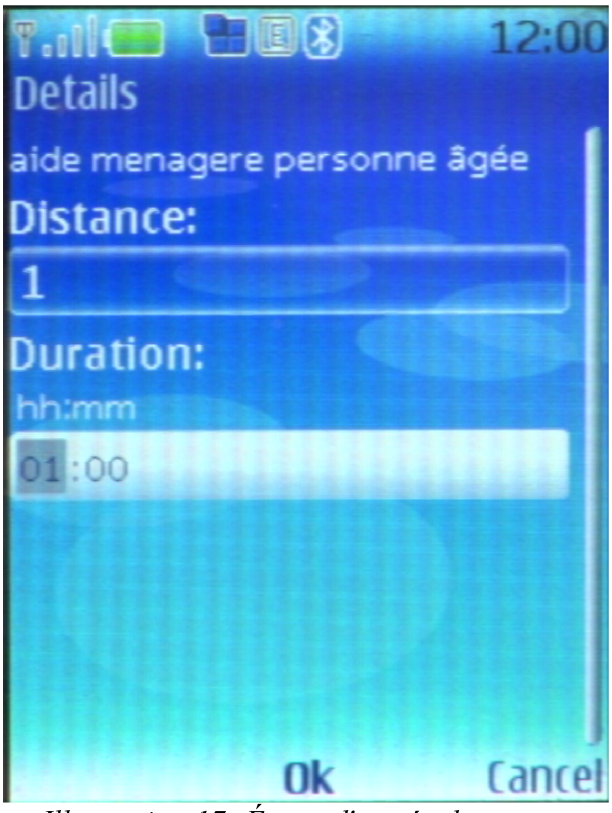

Illustration 17: Écran d'entrée des détails d'un service sur le téléphone

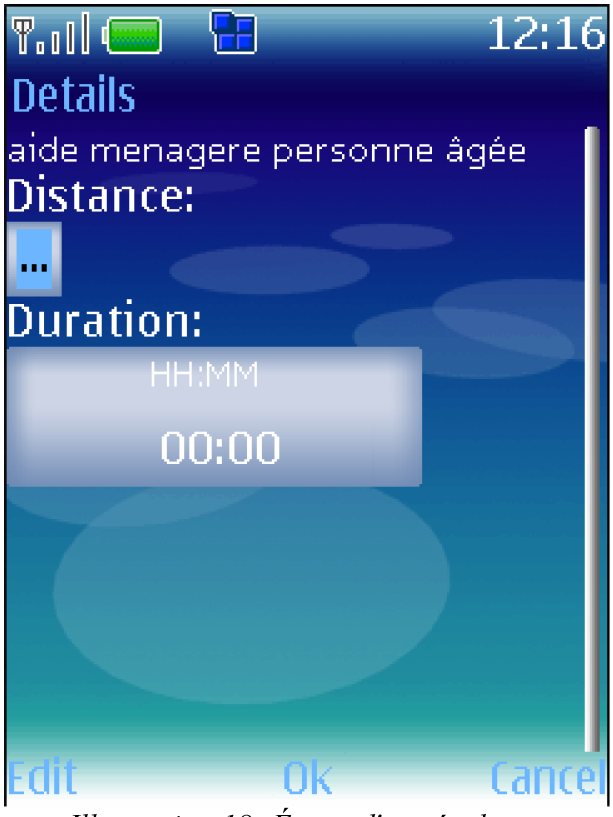

Illustration 18: Écran d'entrée des détails d'un service sur le simulateur

Les tests sur le simulateur ne permettent pas de valider l'aspect des écrans sur le téléphone.

Ces deux captures d'écran illustrent les nombreuses différences entre le téléphone réel et le simulateur :

- Sur le téléphone, la boîte d'entrée de texte est visible, et il suffit de la sélectionner pour pouvoir éditer son contenu. Sur le simulateur, après avoir sélectionné la boîte d'entrée de texte, il faut utiliser l'option de menu « Edit », ce qui a pour effet d'ouvrir un autre écran dans lequel on pourra éditer le contenu de la zone de texte.

- L'aspect et le comportement des zones d'entrées de durée diffèrent également : sur le simulateur, les modifications de leur contenu sont immédiatement prises en compte. Sur le téléphone réel, l'utilisateur doit désélectionner le champ durée pour que sa valeur soit lisible par l'application.

- Sur le simulateur, il est possible de créer un menu avec trois options. Sur le téléphone, une des options sera dans un sous-menu.

6.7 Absence d'outils de débogage sur cible

Ce problème est la principale cause de pertes de temps lors du développement.

Sur hôte, la mise au point et la validation du code posent peu de problèmes. Pour localiser un problème, on insère des instructions println() dans le code. L'ajout d'un println en entrée et en sortie de chaque méthode nous permet d'avoir une trace du déroulement du programme.

Cette technique nous permet de gagner du temps lors des tests. Normalement, une séance de tests se déroule de cette manière :

- Le testeur utilise l'application en s'efforçant de passer au moins une fois par chaque chemin de l'organigramme. Il va donc rédiger un compte rendu d'intervention en deux fois (pour tester la mémorisation de l'heure de début) puis en rédiger un autre en une seule fois, en rédiger un troisième en saisissant des valeurs erronées, etc.

- Quand un dysfonctionnement est constaté, le développeur va commencer par essayer de le reproduire, et de trouver une « recette » permettant d'obtenir le dysfonctionnement en effectuant le moins d'actions possible.

- Une fois cette recette obtenue, plusieurs techniques permettent de localiser la source du problème : on peut examiner l'ensemble des lignes de code exécutées avant que l'erreur ne survienne, ou laisser l'application se charger de cet examen à notre place, en insérant de nombreux println dans le code.

En ajoutant des println préventivement, partout dans le code, on gagne du temps en éliminant les deux premières étapes : quand le dysfonctionnement est constaté, on a déjà la trace de toutes les méthodes appelées avant qu'il survienne.

6.7.1 Un exemple de recherche de bogue sur hôte : le problème des en-têtes HTTP

Par exemple, après une modification du format des données envoyées par le téléphone, nous avons constaté que les données reçues par le serveur n'étaient plus traitées correctement.

Pourtant, aucune ligne de code du mécanisme de réception et de traitement des données n'avait été modifiée.

Les println insérés après chaque étape du mécanisme de réception des données, de la transaction réseau jusqu'à l'extraction des objets, nous ont permis d'isoler la source du problème.

Pour éviter un gaspillage de ressources (voir le chapitre « les contraintes de ressources système ») la méthode de réception des données commence par consulter, dans en-têtes HTTP de la réponse du serveur, le champ indiquant la longueur de la chaîne de caractères que l'application va recevoir. Cela nous permet d'allouer l'espace mémoire nécessaire en une seule fois, et de lire cette chaîne en une seule instruction.

Les println insérés dans cette portion du code nous ont montré que le serveur annonçait une longueur de chaîne nettement plus courte que la longueur réellement envoyée.

Sur le téléphone, il n'existe pas de moyen simple et surtout économe en ressources pour faire de même, un dysfonctionnement survenant uniquement sur le téléphone sera donc nettement plus difficile à trouver.

Pour pallier cette limitation, j'ai installé un écran de logs dans lequel sont consignées toutes les erreurs rencontrées par l'application, ainsi que des messages indiquant, entre autres, le contenu des en-têtes HTTP reçus, le nombre d'événements décodés, etc.

Mais cette technique est très coûteuse : chaque message consigné consomme du temps et de la mémoire. Sur hôte, il est possible d'afficher le contenu de tous les messages envoyés ou reçus sans que cela n'ait d'impact sur les performances. Sur le téléphone, une seule tentative de sauvegarder un message reçu dans l'écran de logs suffit à saturer la mémoire.

Ces limitations se font particulièrement sentir lors de la recherche de dysfonctionnements rares ou difficiles à reproduire. Par exemple, un dysfonctionnement causé par la présence d'une jauge dans une alerte a été particulièrement difficile à trouver.

6.7.2 Un exemple de recherche de bogue sur cible : le problème des jauges dans les alertes

Ce bogue est survenu sans raison apparente, après une modification anodine du traitement des données reçues du serveur.

Le symptôme : dans l'écran d'entrée de l'heure de début, l'application se bloquant parfois pendant l'envoi de l'heure de début au serveur. Ce blocage survenait environ une fois sur cinq, et seulement sur le téléphone. Sur le simulateur, l'application semblait fonctionner parfaitement, et sur le téléphone, le blocage rendait l'écran de logs inaccessible. Nous n'avions donc aucun moyen de localiser précisément la source de l'erreur. Un examen attentif des messages de log sur hôte a permis de constater que, parfois, une exception se produisait pendant la réception des données. Pourtant, le programme continuait à s'exécuter normalement.

Les exceptions sont un mécanisme de débogage intégré au langage Java : même en l'absence de println, certaines erreurs provoqueront l'arrêt du programme, et l'affichage d'un message nous donnant de précieuses informations sur la cause de cet arrêt. Par exemple, une tentative de division par zéro provoquera une exception, et le message d'erreur nous indiquera précisément dans quelle méthode la division par zéro a eu lieu.

Pour éviter l'arrêt du programme, on pourra alors mettre la ligne de code incriminée dans un bloc « try /catch ». L'application pourra alors traiter l'exception.

Curieusement, l'exception affichée dans les logs sur hôte n'était accompagnée d'aucune information permettant de localiser l'erreur.

Pour obtenir plus d'information, j'ai donc inséré la totalité de l'application dans des blocs try / catch, en ajoutant des fonctions d'affichage de l'erreur dans l'écran de logs. En Java standard, le fait d'inclure la totalité d'un programme dans un bloc try /catch garantit qu'aucune exception n'échappera au bloc catch. En J2ME, cette règle a une exception : les violations de la politique de sécurité, qui interrompent le programme sans passer par le bloc catch.

Mais dans le cas présent, il ne s'agissait pas d'une violation de la politique de sécurité, et surtout, l'application ne s'interrompait pas ! Comment peut-on avoir une exception qui n'est pas traitée par un try/catch englobant toute l'application, et qui n'interrompe pas son déroulement ?

La réponse est simple : l'exception n'avait pas lieu dans l'application, mais dans le système d'exploitation du téléphone.

L'ajout d'une pause dans l'application est venu confirmer cette hypothèse : en ajoutant, juste après la dernière action avant l'exception, une instruction demandant à l'application de cesser toute activité pendant trois secondes, on a constaté que l'exception se produisait toujours, et pendant la pause, donc à un moment où aucune ligne de code de l'application n'était exécutée.

La recherche de la cause de cette exception a été longue et fastidieuse. L'action qui avait été à l'origine de cette exception avait forcément eu lieu avant la pause. En insérant des pauses en divers endroits de l'application, on a finalement pu isoler la commande à l'origine du problème : dans certains cas, l'affichage d'une alerte contenant une jauge provoque une exception environ une demi-seconde plus tard.

Cette exception était due à un bogue du système d'exploitation Nokia.

Ces bogues qui sont emblématiques des soucis rencontrés lors du codage ont tous pu être surmontés. L'application Nom'ad a donc pu être commercialisée et est utilisée actuellement par de nombreux intervenants.

7 PERSPECTIVES D'ÉVOLUTION

L'application Nom'ad a été conçue pour être évolutive. De nouvelles fonctionnalités peuvent facilement être ajoutées, mais l'évolution la plus urgente est l'amélioration du processus d'authentification.

7.1 UNE AUTHENTIFICATION PLUS SÛRE

Quand le projet Nom'ad a commencé, la technologie NFC était relativement nouvelle, et l'authentification par carte NFC offrait donc un niveau de résistance à l'usurpation d'identité nettement supérieur à ce que proposaient les concurrents.

C'est aujourd'hui encore le cas, mais peut-être plus pour longtemps.

Comme de nombreux autres systèmes d'identification par cartes NFC, Nom'ad se contente de lire l'identifiant unique de la carte, et de le transmettre au serveur. Un intervenant qui désirerait tromper le système devra donc:

- Se rendre chez le bénéficiaire, et effectuer une copie de sa carte NFC
- Se procurer un émulateur de carte NFC, et y charger l'identifiant unique de la carte du bénéficiaire
- Signer des interventions fictives à l'aide de la fausse carte de bénéficiaire ainsi réalisée.

Au début du projet Nom'ad, j'ai effectué quelques recherches sur Internet dans le but d'énumérer toutes les méthodes permettant de réaliser des fausses cartes NFC.

L'identifiant unique des cartes NFC du commerce ne peut pas être modifié. Pour émuler une carte NFC existante, il faut donc nécessairement utiliser un lecteur configuré en mode émulation de carte. Certains lecteurs (comme par exemple celui du 6131) proposent ce mode, mais à l'époque, aucun ne permettait l'usage d'un identifiant unique copié d'une autre carte.

Les communications entre le lecteur et la carte sont cryptées à l'aide d'un algorithme propriétaire[6]:

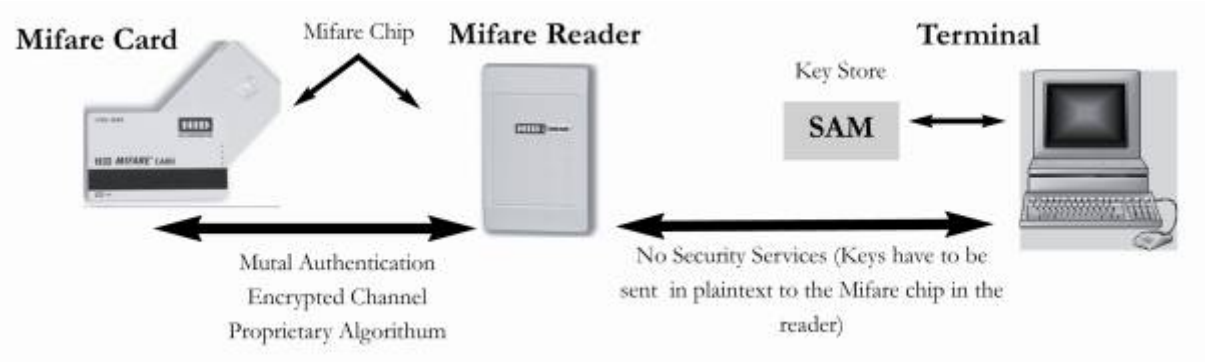

Illustration 19: Le cryptage Mifare

Pour réaliser une fausse carte, il faut donc casser le cryptage de la liaison radio entre la carte et le lecteur. Comme il n'existe pas de version logicielle de cet algorithme, le seul moyen d'y parvenir serait donc d'observer une puce Mifare au microscope, pour analyser son circuit de cryptage.

La puce Mifare est un circuit multicouche d'un millimètre carré comportant plus de 10000 portes logiques. Pour reconstituer le plan du circuit de cryptage, un microscope ne suffit pas: il faut également arriver à séparer les couches!

Cet exploit quasi-impossible a été réalisé en décembre 2007 par deux cryptanalystes hollandais: Nohl et Plotz. Malgré les protestations et actions en justice du fabricant de la puce Mifare, l'algorithme de cryptage a été rendu public. Si cet algorithme avait été suffisamment résistant, cela n'aurait pas posé de problème, bien au contraire:

En publiant un algorithme de cryptage, on permet la recherche de failles par les cryptanalystes du monde entier. Pour qu'un attaquant puisse exploiter une faille, il doit donc être le premier à la trouver. On peut donc en conclure qu'un algorithme de cryptage publié et analysé sans succès depuis plusieurs années présente peu de risques.

[6]Source: http://www.smartcard.co.uk/mifare.html

En ne publiant pas l'algorithme, on pratique la sécurité par l'obscurité: un attaquant motivé finira de toute façon par retrouver l'algorithme, et aura alors beaucoup plus de chances d'y trouver des failles, vu qu'il sera le premier à les chercher.

C'est ce qui s'est passé avec la puce Mifare: Nohl et Plotz y ont trouvé un algorithme assez faible (il utilise une clé de seulement 48 bits) et une faille de sécurité importante: le générateur de nombres aléatoires permettant de créer les clés de sessions n'est pas suffisamment aléatoire, il est donc possible de deviner une bonne partie des 48 bits, et donc de casser la clé en moins d'une demie heure.

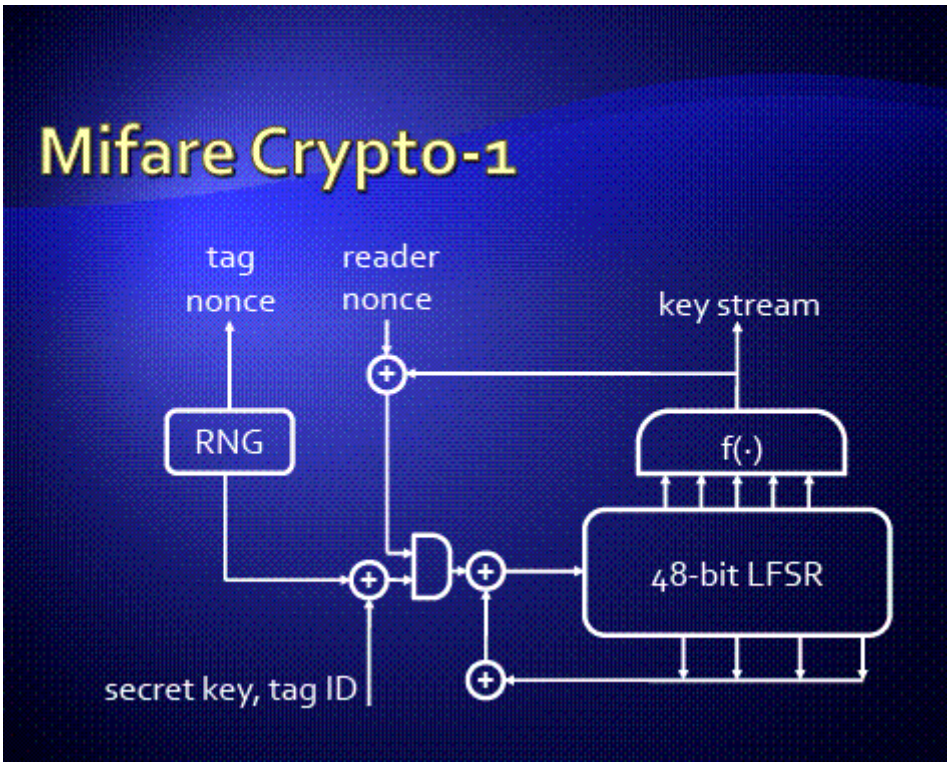

Illustration 20: L'algorithme de cryptage Mifare

On ne peut donc plus compter sur ce cryptage pour garantir l'authenticité des cartes. Pour éviter que les intervenants n'utilisent de fausses cartes patient pour simuler des interventions, il faudra donc modifier la méthode d'authentification. Voici deux propositions permettant d'améliorer la sécurité de l'authentification:

Utiliser des JavaCard

La JavaCard contient un microprocesseur capable d'éxécuter des applications et de mettre en oeuvre des algorithmes de cryptage robustes. Sa mise en oeuvre nécessiterait le remplacement des cartes patient par des JavaCard nettement plus coûteuses, et d'importantes modifications de l'application Nom'ad, mais garantirait alors un niveau de sécurité très élévé.

Écrire sur les cartes

Actuellement, le logiciel Nom'ad se contente de lire la carte patient, et d'envoyer son identifiant au serveur.

La carte patient contient assez de mémoire pour stocker un journal des dernières interventions. L'idée serait donc d'envoyer au serveur le contenu de la carte, puis de mettre à jour cette dernière, en y inscrivant l'heure et l'identifiant de l'intervenant.

Pour pouvoir tromper un tel système, les intervenants devraient donc tous utiliser des fausses cartes, et synchroniser leurs contenus à distance!

Cette dernière solution représente un bon compromis entre les impératifs de sécurité, et le besoin de garder une application simple et fiable, même en l'absence de couverture réseau.

7.2 LA PLATEFORME ANDROID

Android est un système d'exploitation ouvert, ce qui élimine la majorité des problèmes et limitations rencontrés lors du développement sur Nokia 6131.

Malheureusement, aucun des téléphones Android actuellement sur le marché ne possède de lecteur de cartes sans contact. Si cette limitation venait à être levée, il serait alors très opportun de porter Nom'ad vers cette plateforme.

7.3 CONCLUSION

La solution de télégestion d'Hippocad permet de répondre aux nombreuses problématiques du secteur des services à la personne. C'est un système peu contraignant, utilisable à tout moment et qui permet de contrôler de manière sûre l'effectivité d'une intervention.

C'est un système très ergonomique qui apporte de nombreux avantages pour tous les acteurs de la filière.

Grâce à cette solution, les bénéficiaires disposent d'une meilleure prise en compte de leurs besoins, d'une plus grande régularité de service et d'un suivi individuel facilité.

Les intervenants voient leur gestion administrative quotidienne simplifiée et fiabilisée, et peuvent donc se focaliser sur la qualité des prestations apportées.

L'agence, elle, gagne une traçabilité de ses interventions, peut optimiser au maximum la gestion de ses intervenants, et ainsi exercer son activité de manière plus professionnelle.

Après de nombreux tâtonnements, des solutions de gestion fiables commencent à être proposées sur le marché. Bien que ces solutions répondent en partie parfois aux besoins du secteur, elles peuvent encore être optimisées.

La solution Nom'ad fonctionne, elle a été adoptée par 15 entreprises.

Bien qu'il s'agisse actuellement du système qui répond le plus aux attentes du marché, celui-ci peut encore être amélioré. Nom'ad fonctionne avec un téléphone Nokia 6131 NFC, a été porté sur le Nokia 6210, et sera bientôt adapté à d'autres téléphones NFC.

Les principales difficultés que j'ai rencontrées lors du développement sur ces téléphones sont liées au caractère fermé de leur système d'exploitation. De nombreux problèmes bloquants et bogues difficiles à localiser auraient pu être réglés si j'avais eu un accès aux sources. l'idéal aurait été d'avoir un téléphone sous un OS open source.

Trolltech travaille sur ce type de système et pourra être un système de nouvelle génération plus performant et sécurisé.

En l'absence de concurrents réels sur le secteur, Nom'ad peut continuer sa croissance, et même être utilisé par d'autres corps de métiers (livraisons en tous genres, laboratoires, plombiers...)

Nom'ad a donc de beaux jours devant lui.

BIBLIOGRAPHIE

GOUV01 :.....http://www.servicesalapersonne.gouv.fr/chiffres-cles-%282064%29.cml?
DOMIPHONE :.....http://www.domiphone.com/
DOMATEL :.....http://www.apologic.fr/domatel/
PRYLOS :.....http://www.prylos.com/
HIPPOCAD :.....http://www.hippocad.com/
NFC01 :.....http://www.nfc-forum.org
QRCODE :.....http://mobilecrossmedia.blogspot.com/ Jacques-André Fines-Schlumberger 2010
J2ME01 :.....http://java.sun.com/javame/index.jsp

INDEX LEXICAL

Bénéficiaire....**7 sv**
Compte-rendu d'activité....**11 sv**
cycle en V....**35 sv**
cycle itératif....**37 sv**
Domatel....**17, 20 sv**
Domiphone....**17, 20 sv**
Domiphone2....**17**
élégestion....**20**
Financeur....**9 sv**
GPRS....**44 sv, 49 sv, 54, 57 sv, 63, 68**
Hippocad....**18 sv, 22**
Intervenant....**7, 9 sv, 13**
J2ME....**45 sv, 49 sv, 58, 62**
méthodologie....**32 sv**
Méthodologie....**33**
modèle en cascade....**34 sv**
NokiaExplorer....**32 sv, 43**
planning....**7, 10, 14, 16**
PROX'AD....**18**
Prylos....**17, 20, 22**
Service à domicile....**8 sv**
télégestion....**7 sv, 13 sv, 18, 20 sv**
tests unitaires....**36, 39, 42 sv**
unité de coordination....**9, 12**

INDEX DES TABLES

Tableau 1: Comparatif des solutions de télégestion.....20
Tableau 2: Extrait de la machine à états.....53

INDEX DES ILLUSTRATIONS

Illustration 1: Capture d'écran du formulaire de gestion des services de PROX'AD.....19
Illustration 2: Une carte NFC.....22
Illustration 3: Un QR Code[QRCODE].....22
Illustration 4: Visualisation du planning.....24
Illustration 5: Saisie d'un compte-rendu d'activité.....26
Illustration 6: Affichage de la liste des services.....28
Illustration 7: Modèle en cascade.....35
Illustration 8: Cycle en V.....36
Illustration 9: Le cycle en spirale.....37
Illustration 10: Le cycle itératif.....38
Illustration 11: Les différentes plates formes Java (source: http://java.sun.com [J2ME01]).....48
Illustration 12: Multitâches sans file d'attente.....55
Illustration 13: Multitâches avec file d'attente.....57
Illustration 14: Les objets disponibles pour la génération d'écrans J2ME.....79
Illustration 15: Une Liste.....81
Illustration 16: Un ChoiceGroup.....81
Illustration 17: Écran d'entrée des détails d'un service sur le téléphone.....84
Illustration 18: Écran d'entrée des détails d'un service sur le simulateur.....84
Illustration 19: Le cryptage Mifare.....90
Illustration 20: L'algorithme de cryptage Mifare.....91