INSA RENNES

Institut National des Sciences Appliquées
Ecole publique d'ingénieurs

TELELOGOS

Manage Yourself

Rapport de spécifications

Projet de 4ème année informatique

Equipe :

Etienne Alibert,
Florian Barbedette,
Pierre Chesneau,
Mathias Deshayes,
Sevan Hartunians,
Mathieu Poignet.

Encadrant :
Laurence Rozé

Le 01/12/09

Table des matières

I. Introduction 3
A. But du document 3
B. Documents de références 3
C. Contenu du document 3
II. Présentation générale du produit 3
A. Contexte et objectifs 3
B. Enoncé du besoin 4
C. Outils 4
1. Langage C# 4
2. Compact Framework .NET 3.5 4
D. Environnement de développement 5
1. Windows Mobile 5
2. Visual Studio 2008 5
III. Expression fonctionnelle du besoin 5
A. Partie serveur 5
1. Interface administrateur 5
2. Format des règles 7
3. Apprentissage 8
B. Partie embarquée 10
1. Système expert 10
2. Application de création de rapports 11
C. Communication 14
IV. Chronologie 15
V. Conclusion 16

I. Introduction

A. But du document

Ce document est destiné à identifier et décrire les besoins de la société Telelogos pour le logiciel Manage Yourself.

B. Documents de références

Le site de Microsoft sur le développement Windows Mobile :

http://msdn.microsoft.com/en-us/library/bb847935.aspx

La documentation de weka :

http://www.cs.waikato.ac.nz/~ml/weka/

C. Contenu du document

Ce document présentera dans un premier temps les objectifs du projet dans le contexte actuel, les besoins qu'il est censé satisfaire et l'environnement dans lequel il pourra être utilisé. Dans un second temps, nous allons décrire les fonctionnalités proposées par le produit final pour répondre aux attentes de la société Telelogos. Pour terminer, ce document présentera les livrables attendus et les délais du projet.

II. Présentation générale du produit

A. Contexte et objectifs

Manage Yourself est un projet de diagnostique et surveillance de plates-formes embarquées, il s'inscrit dans le cadre d'une collaboration entre Telelogos et DREAM. Telelogos est une entreprise éditant des logiciels professionnels qui automatisent, administrent et optimisent les processus d'échanges entre système informatique central et ceux des utilisateurs distants. DREAM est une équipe de recherche de l'IRISA spécialisée dans l'aide à la surveillance et au diagnostic de systèmes évoluant dans le temps.

Le besoin de développer une telle application émane de Telelogos, qui, dans le cadre d'un programme de formation aux technologies d'innovation cherche à développer deux logiciels concepts. Manage Yourself est l'un de ses logiciels, ayant pour but d'explorer les possibilités dans le domaine du MBM (Mobile Device Management).

Le but du projet est de développer une plate forme de surveillance sur des appareils de type Smartphone ou PDA, fonctionnant sous Windows mobile. L'application de surveillance sera embarquée sur l'appareil mobile. Elle sera constamment au courant de l'état du système et sera capable de détecter les situations problématiques et d'y remédier d'une façon complètement transparente pour l'utilisateur.

B. Enoncé du besoin

L'entreprise Télélogos souhaite mettre en place un système de monitoring, d'autodiagnostic et d'autoréparation de PDA pour ses clients.

C'est dans cette optique qu'a été proposé le concept logiciel Manage Yourself. Pour cette application, le principal besoin se situe au niveau de l'intelligence artificielle : le logiciel devra embarquer un moteur d'inférence capable de détecter, diagnostiquer, réparer et reporter les pannes survenues sur le PDA.

Ce système se devra d'être portable (une seule application compatible avec tous les PDA, quelque soit leur processeur : ARM – x86 – …), simple d'utilisation pour les utilisateurs finaux, performant (bien que les PDA aient des ressources limitées).

Le système se devra d'être évolutif : à l'aide des données collectées depuis les PDA, et stockées sur le serveur, le logiciel devra être en mesure d'alimenter une base de données des pannes possibles.

Pour être plus précis : Une fois un certain nombre de rapport de pannes collectées, le logiciel effectuera des traitements statistiques sur les données pour en extraire les raisons des pannes survenues. Une interface devra proposer aux administrateurs une vue synthétique des nouvelles pannes et leur permettre de saisir un correctif.

Du coté serveur sera conservé une base de données de toutes les pannes survenues, permettant lors de l'abonnement d'un nouveau client de transférer les règles de détection ; tout comme de régénérer l'ensemble des règles de détection (en cas de mise à jour/ d'erreurs).

On notera que l'aspect transfert des données, bien que part importante du logiciel, ne fait pas partie des besoins : en effet, pour l'occasion, la société Télélogos nous fourni des licences de Media contact, logiciel dédié à la synchronisation des données entre PDA et Serveur.

C. Outils

1. Langage C#

Le C# est le langage qui permet d'utiliser au mieux toutes les fonctionnalités apportées par le Framework .NET. Ce langage est très proche syntaxiquement du Java mais il offre cependant plus de possibilités comme par exemple la surcharge d'opérateur (semblable au C++).

Les types natifs correspondent à ceux de .NET, les objets sont automatiquement nettoyés par un ramasse-miettes, et beaucoup de mécanismes comme les classes, interfaces, délégués, exceptions, ne sont que des moyens explicites d'exploiter les fonctionnalités de la bibliothèque .NET.

2. Compact Framework .NET 3.5

Le .NET Compact Framework est une version du .NET Framework conçue spécialement pour les PDA et les Smartphones. Pour exécuter un programme qui fait appel à ce Framework il est nécessaire d'avoir le runtime Compact Framework sur la machine cible. Ce dernier est inclus dans les dernières versions de Windows Mobile.

D. Environnement de développement

1. Windows Mobile

Windows mobile est le nom du système d'exploitation pour Smartphones et PDA de Microsoft. Ce logiciel permet une compatibilité transversale de la plateforme mobile vers le monde PC. De nombreuses versions sont sorties, cependant nous nous intéresserons uniquement à la version 6. En effet cette version est la plus rependue sur le marché actuel des plateformes mobiles.

Ce système d'exploitation existe sous 3 versions différentes :

- Une version standard pour les Smartphones.
- Une version classique pour les PDA sans fonction téléphonique.
- Une version professionnelle intégrant une fonction téléphonique.

2. Visual Studio 2008

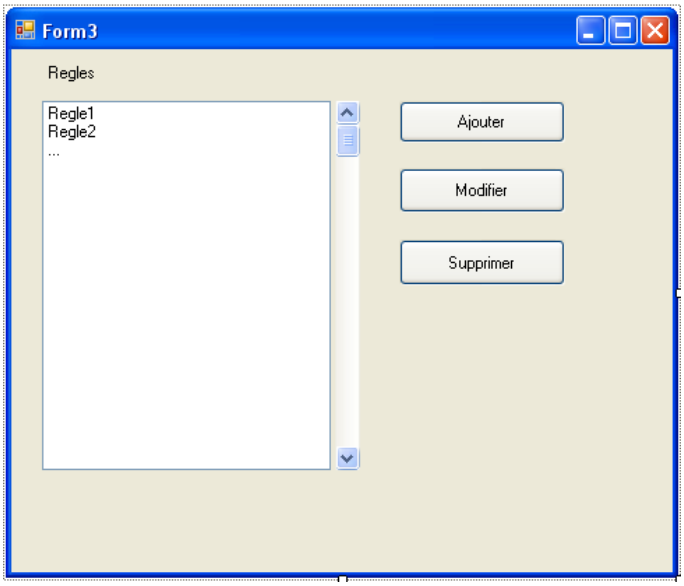
Visual Studio est l'outil de développement de Microsoft qui permet de réaliser facilement des applications avec une interface graphique tout en utilisant toutes les librairies .NET. Pour développer sous Windows Mobile avec Visual Studio nous avons installé le mobile SDK qui fournit le compact Framework ainsi qu'un émulateur pour tester les applications avant de les charger dans le mobile.

III. Expression fonctionnelle du besoin

A. Partie serveur

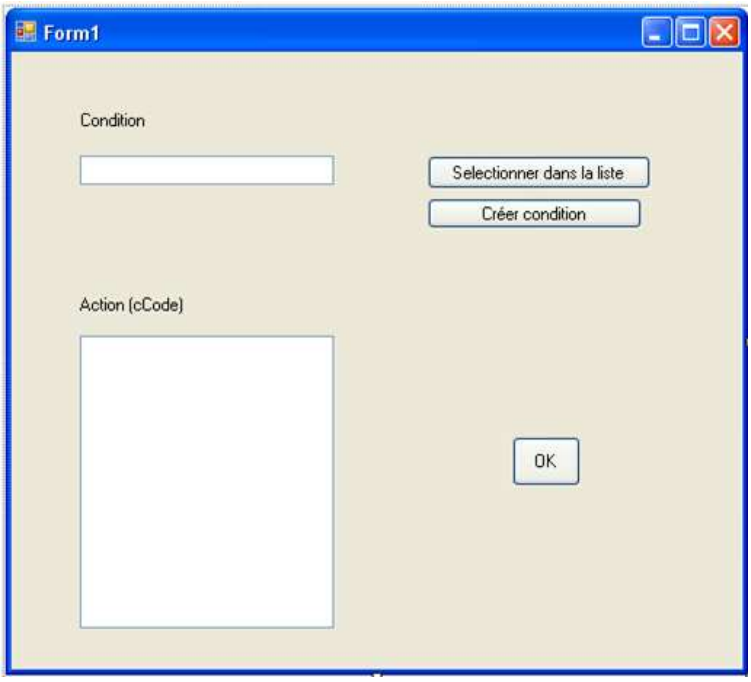
1. Interface administrateur

Une interface administrateur devra être présente sur le serveur, elle permettra de visualiser facilement toutes les règles présentes. Elle permettra également de supprimer des règles ou d'en saisir de nouvelles.



Ebauche du panneau permettant de visualiser les règles

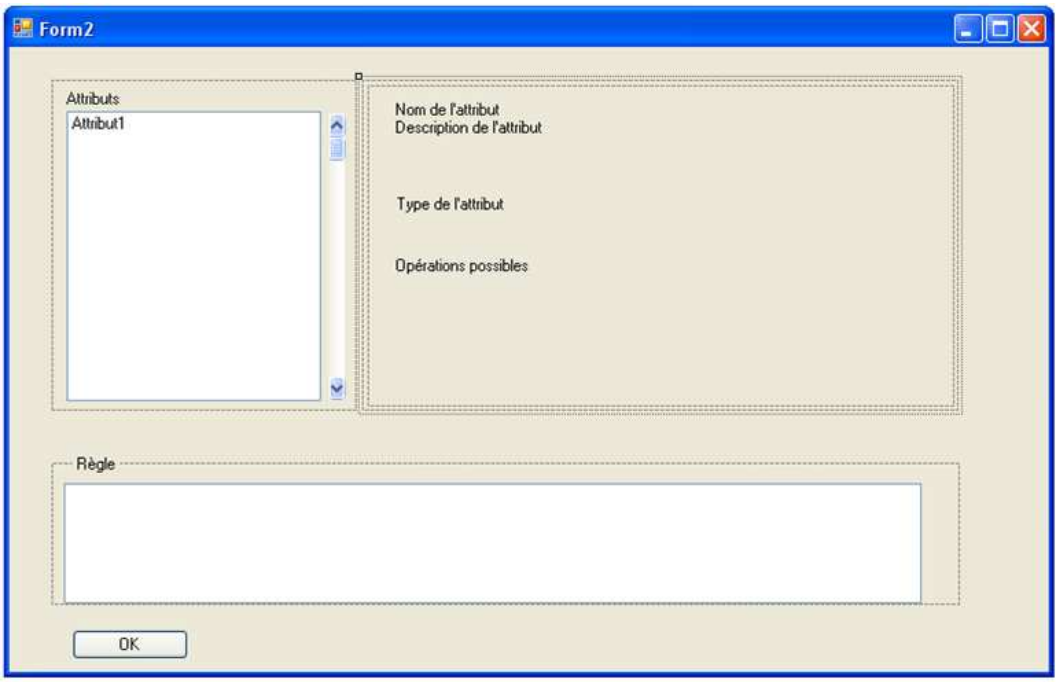
Lors de la saisie d'une nouvelle règle, la partie condition de la règle pourra être saisie par l'administrateur ou sélectionnée dans une liste générée via apprentissage. L'action à faire lorsque les conditions seront vérifiées sera saisis sous forme de code dans une boite de texte.



Ebauche du panneau permettant de choisir une condition et de saisir une action correspondante

Le bouton « Sélectionner dans la liste » permet d'accéder a toutes les conditions générée par apprentissage afin d'en choisir une.

Lors du clic sur le bouton « Créer condition » un deuxième panneau s'ouvre permettant de saisir une condition. Afin d'aider l'administrateur à rentrer les conditions de la règle, une liste des attributs disponibles sera mise à disposition. Lorsqu'un attribut sera sélectionné, des informations le concernant seront affichées telles que son type ainsi que les opérations disponibles. La condition sera alors rentrée dans le format correspondant au format des règles.



Ebauche du panneau permettant de saisir une condition

2. Format des règles

Pour diagnostiquer correctement les problèmes, le système expert nécessite une base de connaissance qui lui permet de prendre les bonnes décisions. La base de connaissances rassemble la connaissance et le savoir-faire du spécialiste dans son domaine. Le formalisme le plus répandu pour représenter cette basse de connaissance est celui des règles de production. On parle alors de base de règles.

Nous définissons alors une règle comme étant une expression de la forme « SI condition(s) ALORS conclusion(s) », ce qui permet généralement d'ajouter de nouveaux faits. Et une condition étant la description d'une situation représentée dans un formalisme approprié (souvent sous la forme d'une conjonction de faits) ; et une conclusion étant une action à envisager si la situation décrite pas les conditions est réalisée.

Dans le cadre de notre projet, les conditions seront liées aux différents éléments de l'appareil mobile que nous surveillerons (tels que la batterie, les applications lancées, la mémoire…). Les conditions seront donc assez simples. Nous définissons alors un langage prenant en compte les entiers, les flottants, les doubles et les booléens ainsi que les opérateurs suivants : <, <=,>, >=, <>, ==. Nous pourrons de plus accumuler les conditions avec la conjonction « ET » ce qui permettra de tester plusieurs éléments à la fois pour prendre une décision. En ce qui concerne les actions, elles seront, dans cette base de règles, définies de manière assez générale (à l'aide de « noms » qui permettront un traitement dans le code plus précis).

Dans un premier temps, cette base de règles sera stockée sur le serveur sous forme de fichiers simples (tels que des fichiers XML ou Txt) et transmises à chaque appareil mobile. De plus, étant donné que nous souhaitons intégrer de l'apprentissage, les règles contenues dans la base

devront être éditables par l'administrateur par l'intermédiaire d'une interface, mettant ainsi à jour la base de règles.

3. Apprentissage

Dans notre projet, la partie apprentissage est une voix exploratoire. Après avoir étudié deux grands domaines de l'apprentissage : la classification et les arbres de décision, nous étudions l'utilisation d'une bibliothèque java : weka qui implémente la plupart des algorithmes d'apprentissage existants.

Dans notre logiciel final, l'apprentissage en lui même s'exécutera de façon tout à fait transparente. Une condition de déclenchement devra être définie. Dans un premier temps celle-ci sera très naïve :

- Dès que l'on a reçu plus de 100 rapports, relancer l'apprentissage
- Ou dès que l'on a reçu plus de 100 exemples négatifs, relancer l'apprentissage.

Une fois la plateforme mise en place on pourra voir si de telles règles de déclenchement sont trop naïves et voir comment les adapter.

Après l'exécution de l'apprentissage de nouvelles règles sont déterminées. Il faut alors déterminer pour chacune d'entre elle les actions à exécuter pour sortir de la situation critique. Une solution serait de demander à l'administrateur de redéfinir pour chacune des règles obtenues les actions à effectuer. Mais ceci risque d'être beaucoup trop lourd pour l'administrateur et de rendre le logiciel inutilisable. Une autre solution serait de réutiliser les règles déjà en vigueur, de voir si entre les nouvelles règles et les anciennes il existe des règles de généralisation et si oui garder l'action de l'ancienne règle. L'administrateur pourra alors :

- Choisir d'enlever des règles (par exemple s'il ne voit strictement aucune action correctrice à faire)
- Choisir de garder d'anciennes règles car elles détectent un problème connu.
- Associer une action aux règles non encore étiquetées.

Dans un premier temps, en attendant la mise en place de la plateforme et la définition exacte des rapports, nous allons travailler sur un exemple jouet. L'exemple utilisé va être celui d'un mobile dont un certains nombre de caractéristiques sont surveillées : la mémoire disponible, l'état de la batterie, 3 applications A, B et C qui peuvent être lancée ou non et dont on sait si on en possède les dernières versions ou non. Les attributs et leurs domaines sont donc les suivants :

- Mémoire $\in$ {normale, presquePleine, pleine}
- Batterie $\in$ {normale, faible, vide}
- ApplicationA $\in$ {lancée, nonLancée}
- DerniereVersionA $\in$ {oui, non}
- ApplicationB $\in$ {lancée, nonLancée}
- DerniereVersionB $\in$ {oui, non}
- ApplicationC $\in$ {lancée, nonLancée}
- DerniereVersionC $\in$ {oui, non}

Cet exemple est un exemple jouet dans le sens ou il n'y a que 486 instances possibles.

On suppose que le téléphone émet à intervalle régulier et lors des plantages des rapports d'exécutions. Ces rapports sont stockés, et à un moment on décide de lancer l'apprentissage à partir de tous les rapports.

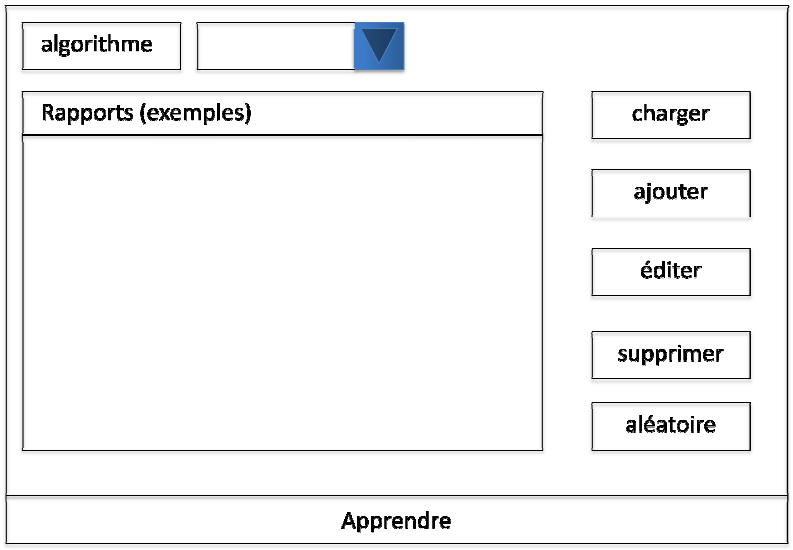
Dans un premier temps pour étudier le problème, on choisit de tester sur un téléphone idéal sans données bruitées : on suppose que le téléphone est régi par les règles suivantes :

- (Mémoire=pleine) $\wedge$ (applicationC=lancée) $\Rightarrow$ (plantage=oui)
- (Mémoire=presquePleine) $\wedge$ (applicationC=lancée) $\Rightarrow$ (plantage=oui)
- (Mémoire=pleine) $\wedge$ (applicationA=lancée) $\Rightarrow$ (plantage=oui)
- (applicationA=lancée) $\wedge$ (applicationB=lancée) $\Rightarrow$ (plantage=oui)
- Dans tous les autres cas on suppose que (plantage=non)

On va dans la suite tirer aléatoirement 30 exemples et essayer d'exécuter des algorithmes d'apprentissage sur ses exemples.

Dans un deuxième temps on choisira de rajouter des données bruitées c'est à dire des données normales mais associées à un plantage (simulation d'une panne matérielle).

Afin de faciliter le travail sur l'exemple jouet, nous proposons un petit outil, comme le montre la figure ...



Outil permettant de tester les algorithms d'apprentissage

Le menu déroulant étiqueté algorithme nous permet de choisir l'algorithme d'apprentissage que l'on souhaite utiliser. La fenêtre rapports permet de visualiser tous les exemples utilisés pour l'apprentissage, l'idéal étant de faire apparaître en rouge les données bruitées. Le bouton charger permet de charger un ensemble d'exemples à partir d'un fichier. Le bouton ajouter permettra d'ajouter manuellement un exemple à l'ensemble d'exemples déjà traités. Le bouton éditer permettra d'éditer et de modifier un exemple. Le bouton supprimer permettra de supprimer un exemple. Le bouton aléatoire permettra de tirer aléatoirement un certain nombre d'exemples, nombre demandé à l'utilisateur. Enfin de le dernier bouton apprendre permettra de lancer l'apprentissage et d'afficher le résultat de ce dernier dans une nouvelle fenêtre.

B. Partie embarquée

1. Système expert

L'objectif du logiciel étant de résoudre les pannes du mobile il est nécessaire que celui-ci embarque un système expert. En effet il est impossible de laisser le système expert sur un ordinateur distant car cela signifierai l'obligation de la présence d'une connexion réseau pour diagnostiquer et réparer ; connexion pouvant être perdue si le logiciel du mobile est mal configuré.

Besoins principaux

Le système expert embarqué devra répondre aux besoins suivants :

- Légèreté :

Les mobiles sur lesquels l'application s'exécutera ne disposant pas d'une puissance de calcul importante, il faut que le moteur soit suffisamment rapide pour diagnostiquer la panne sans figer le portable.

- Autonomie :

Bien que dépendant d'une base de données commune, le moteur d'inférence devra être capable de « raisonner » en autonomie, c'est-à-dire de trouver des solutions aux problèmes rencontrés sans avoir accès à cette base de données commune.

- Simplicité d'utilisation

Les interactions entre le système expert et l'utilisateur se devront d'être minimale. La cible étant le zéro interaction entre le système expert et l'utilisateur : Le système expert raisonnant grâce aux seules informations dont il dispose.

Prolog

Pour créer ce système expert embarqué nous avons également réfléchi à l'utilisation de Prolog.

En effet, prolog est basé sur un moteur d'inférence (en chainage arrière, mais transformable en chainage avant) ; est simple à mettre en œuvre ; et est performant.

Cependant, après des recherches il nous est apparu que :

- Il n'existe pas de prolog conçu pour fonctionner nativement sur Windows mobile
- Les versions de prolog existantes ne sont pas portable sous Windows Mobile, ou elles le seraient mais au prix de modifications très lourdes, (aussi lourde que le développement depuis 0). De plus rien ne pourrait nous assurer à priori que la version de prolog que nous choisirions serait suffisamment légère. (Développement à effet tunnel)

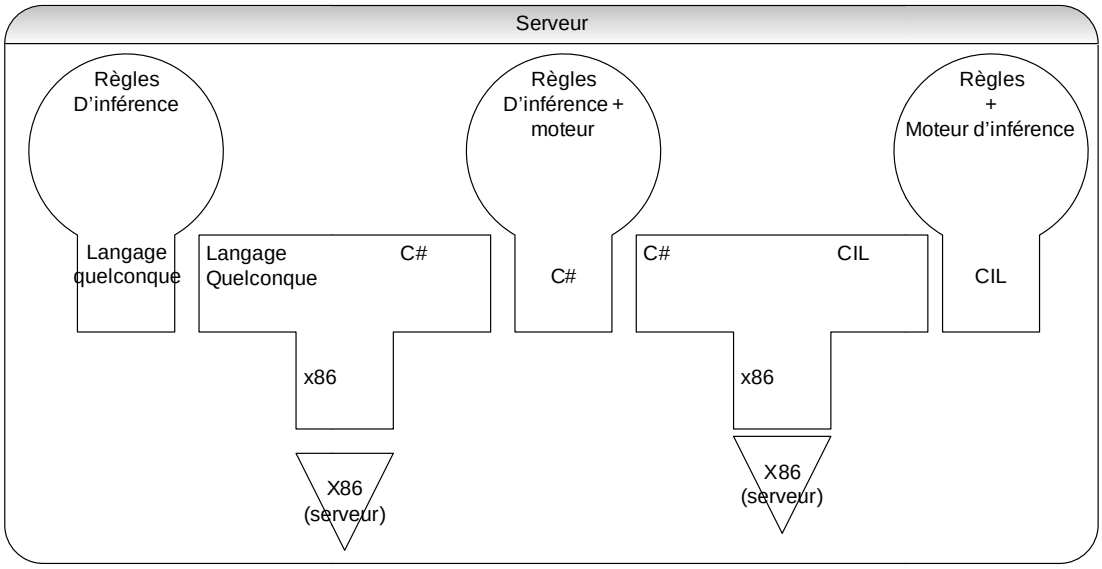
P#

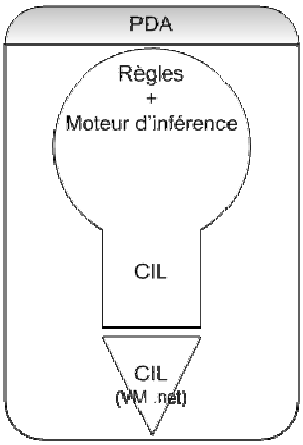
P# est un logiciel libre permettant de convertir du code prolog en code C# natif. Bien qu'a première vue cette solution semble idéale, plusieurs problèmes relatifs à cette implémentation nous sont apparus.

Le plus gênant d'entre eux est le code généré qui utilise des éléments du .net Framework non présent dans le Compact Framework. (La gestion du parallélisme dans le code généré.)

Après étude de ce problème il nous a paru préférable d'abandonner cet outil : l'adaptation d'un tel outil aurait été plus couteuse que de redévelopper un moteur d'inférence simple nous-mêmes.

Cependant, nous nous sommes inspiré de la conception de P# pour mettre au point un organigramme de compilation le plus efficace possible.





De cette manière, la compilation a lieu sur le serveur. Ceci présente plusieurs avantages :

Au niveau performance tout d'abord : La compilation des règles d'inférences est effectuées une fois pour toute, sur un pc puissant, ce qui évite de devoir traiter ce calcul (long) sur un PDA.

Au niveau Portabilité : Le code CIL est au .net ce que le bytecode est au java : un code intermédiaire, plus rapide qu'un code source, mais portable entre diverses plateformes.

Au niveau Sécurité : L'utilisateur ne voit jamais le code du moteur d'inférence ni ses règles. De ce fait il ne peut pas être tenté de les modifier. (Ce qui pourrait provoquer des bugs importants.)

2. Application de création de rapports

Notre application devra générer régulièrement des rapports concernant l'état du système. Ces rapports auront deux buts. Tout d'abord ils serviront à établir la base de fait pour le système expert, pour pouvoir ensuite appliquer les différentes règles programmées. Mais ces rapports serviront également pour l'apprentissage de nouvelles règles. En effet, ces rapports seront envoyés au serveur central qui pourra, une fois qu'un certain nombre de rapport aura été envoyé, commencer l'apprentissage sur les états entraînant un problème du système mobile.

Contenu des rapports

Ces rapports contiendront l'ensemble des données sur l'état du système qui nous paraissent nécessaire pour surveiller en temps réel le bon fonctionnement de l'appareil. Par exemple nous surveillerons l'état mémoire (physique et virtuelle), l'état de la batterie, la connexion réseau, les versions de programme utilisées etc.

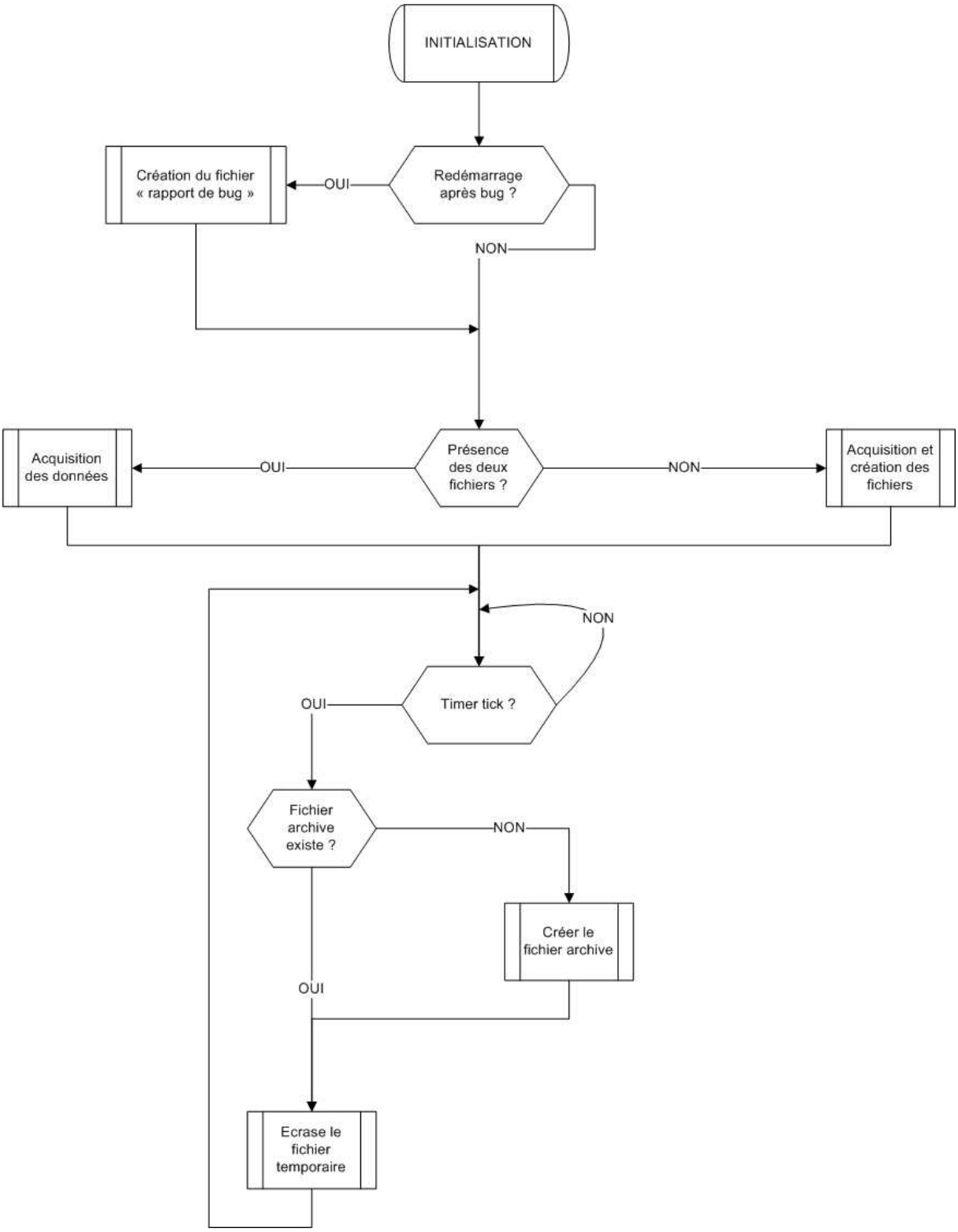
Structure des rapports

Chaque rapport sera stocké dans une structure C# qui contiendra tous les champs surveillés par notre application. Cette structure sera ensuite retranscrite dans un fichier binaire utilisable par le système expert et le serveur. Pour ensuite récupérer la structure depuis le fichier binaire il suffira de remplir les champs les un après les autres en lisant depuis ce fichier.

Déroulement de la génération

Notre application générera en réalité deux rapports. L'un des rapports servira à être récupéré par le serveur pour créer la base d'apprentissage. Nous appellerons ce fichier l'archive. L'autre sera utilisé pour alimenter le système expert avec des données « temps réel » du système, nous l'appellerons fichier temporaire. Celui-ci sera écrasé de manière périodique dans le but de limiter le nombre de fichier présent sur la machine. L'archive quant à elle sera rapatriée sur le serveur grâce à MediaContact qui la supprimera de la machine. Notre application, détectant l'absence du fichier archive la régénérera en copiant le fichier temporaire. En cas de problème critique entraînant l'arrêt du mobile, un nouveau fichier sera créé au moment du redémarrage. Ce fichier sera la copie du fichier temporaire le plus récent (juste avant le problème). Ce fichier sera récupéré également par MediaContact puis étiqueté comme « rapport de bug ».

Tout cela peut être résumé par le graphe suivant



Exemple d'utilisation d'une méthode système

La plupart des méthodes système sont dans coreDLL.dll, une dll qui est présente sur toutes les machines équipées de Windows Mobile. Pour utiliser ces méthodes la procédure est systématique. On commence par définir une structure qui contiendra les informations que nous souhaitons obtenir.

```
public struct STORE_INFORMATION
        {
            public UInt32 dwStoreSize;
            public UInt32 dwFreeSize;
        }
```

Ensuite nous importons la méthode depuis coreDLL.dll

```
[DllImport("CoreDll.dll")]
        public static extern bool GetStoreInformation
        (

            ref STORE_INFORMATION lpsi
        );
```

Puis nous instancions notre structure et appelons la méthode en passant cette structure en paramètre.

```
STORE_INFORMATION stinfo = new STORE_INFORMATION();
            GetStoreInformation(ref stinfo);
```

La structure est maintenant remplie et nous pouvons accéder à ses champs pour obtenir les informations souhaitées.

C. Communication

Notre projet se déroulant en partenariat avec l'entreprise Telelogos, la partie communication client/serveur de notre application sera gérée via l'outil MediaContact développé par cette entreprise. MediaContact est une suite logicielle utilisée dans la gestion de terminaux distants et proposants les services d'administration des postes, la possibilité de synchronisation des applications et de sauvegarde des données et aussi la possibilité de lancer des processus à distance.

MediaContact étant orienté processus, toutes les actions de communication entre le client et le serveur seront fait via des processus. Ces processus sont composés par une séquence hiérarchique de tâches, une tâche étant un traitement élémentaire composé d'une séquence ordonnée et contrôlée d'étapes Afin d'exécuter des processus sur des stations clientes, il faut leur associer une population. Ainsi chaque processus est défini pour une population de clients.

Lors de la première installation du système expert, la station cliente sera placée dans une population contenant tous les nouveaux utilisateurs de l'application. On pourra alors, via MediaContact transférer l'exécutable du système expert. Une fois le transfert achevé, le processus sera exécuté à distance avec MediaContact, ce dernier autorisant l'exécution d'un programme côté client. Il faudra spécifier que le serveur n'attende pas de signal de fin de processus, ce dernier tournant en tache de fond.

Pour la mise à jour du système expert, il suffira de placer l'exécutable de la nouvelle version dans le dossier d'envoi du serveur. Avec un processus de mise à jour, il suffira de transférer le nouvel exécutable de façon standard en remplaçant l'ancien, dans une première tache, puis grâce à une seconde tâche, le système expert sera relancé.

MediaContact permet l'envoi du contenu d'un répertoire à un intervalle de temps régulier, celui-ci sera défini en fonction des besoins de l'algorithme d'apprentissage choisi. L'exécutable

placera dans ce répertoire d'envoi les rapports à envoyer, et MediaContact se chargera de faire parvenir ces rapports au serveur selon l'intervalle de temps défini. Si le PDA est en veille au moment où il faut envoyer le rapport, il sera automatiquement envoyé lors de la prochaine utilisation de ce dernier.

IV. Chronologie

Avec ce rapport, nous concluons la première phase du déroulement de notre projet : la phase d'analyse. Nous allons maintenant étudier les prochaines phases à venir.

La première de ces phases correspond à la planification. Elle est essentielle pour le bon déroulement du projet, et permet d'établir la feuille de route du projet permettant de décrire : l'organisation, la répartition et la synchronisation des tâches, les indicateurs de délais, les contraintes organisationnelles, etc. Notre planification s'appuiera sur l'outil MS Project, ce qui nous permettra de rendre un rapport de planification initiale. Ce dossier sera constitué, pour l'essentiel, de documents de travail auxquels s'ajoutent une brève présentation du projet, ainsi que les explications nécessaires sur la méthode de planification utilisée. D'environ dix pages, il devra être retourné au responsable des projets pour le 15 décembre 2009. Cette phase donnera finalement lieu à une soutenance le 18 décembre 2009.

La phase suivante est la phase de conception. La première partie de cette phase correspond aux spécifications fonctionnelles. Elle permet de décrire précisément le fonctionnement externe de l'application. A la fin de cette étape devra apparaître une première idée de l'architecture logicielle générale de notre application : description des grands modules et de leurs interactions. La deuxième partie correspond, quant à elle, à la conception logicielle ; c'est la conception détaillée de l'application. Elle s'appuiera sur une définition précise de l'architecture logicielle interne de l'application (classe, package, module, communication de l'information, etc.). Le détail de la conception logicielle permettra d'affiner la planification initialement prévue en révisant le découpage des tâches et les contraintes associées. Cette phase de conception sera accompagnée d'un rapport de conception logicielle décrivant l'architecture du logiciel, sa modélisation et l'interfaçage des différents modules à rendre pour le 19 février 2010.

Les deux dernières phases sont la phase de construction et celle de déploiement/livraison. Durant la construction, la première étape consiste au développement, c'est-à-dire au codage de l'application. Cette partie est fortement imbriquée avec la suivante : la phase de test. En effet les tests doivent s'effectuer tout au long du développement et avoir été intégrés dans la planification du projet. Les tests sont de deux types : les tests structurels de type « boîte blanche » et ceux de type « boîte noire ». Le développement est également couplé à la documentation du logiciel. Dans cette optique, deux pages HTML décrivant de manière synthétique le projet devront être envoyées par email pour le 9 avril 2010. Et une documentation « en ligne » du logiciel devra être livrée en fin de projet (28 mai 2009).

Finalement, la phase de déploiement/livraison consiste à livrer l'application, suivre sa mise en production, assurer le support, la formation et valider la qualité du produit. Pour notre projet,

nous devrons donc assurer la livraison du projet sur CD-Rom structuré contenant les rapports (dont le rapport final), les transparents des soutenances, les sources, la documentation, les exécutables, leur procédure d'installation et les jeux de tests pour le 28 mai 2010. Puis nous devrons effectuer une démonstration externe de l'application le 27 ou 28 mai 2010 et enfin une démonstration interne au projet pour vérifier la conformité avec le cahier des charges durant la dernière semaine bloquée.

V. Conclusion

Au cours de cette phase de spécifications, nous nous sommes employés à identifier et décrire plus formellement les besoins de la société Télélogos pour le logiciel Manage Yourself. Dans cette optique, nous avons tout d'abord définis plus précisément les objectifs du projet dans le contexte actuel (collaboration entre l'entreprise Télélogos et l'équipe de recherche DREAM), les besoins qu'il devra satisfaire (diagnostic et apprentissage sur PDA), et l'environnement dans lequel il pourra être utilisé (Windows Mobile). Dans un second temps, nous avons décrit les fonctionnalités proposées par le produit final: interface administrateur, format des règles et apprentissage côté serveur; et système expert, application de création de rapports côté embarqué.

Cette étape de spécifications n'a pas été simple du fait du nombre de spécifications. En effet, les solutions possibles pour traiter les différentes parties du projet étaient souvent multiples et leur choix n'était pas forcément évident (pour exemple, l'apprentissage reste une voix exploratoire du projet). Cependant après réflexion, nous avons établi les choix qui nous semblaient les plus adaptés.

Nous devons donc maintenant nous projeter dans la prochaine phase du projet : la planification concrète. Elle est essentielle pour le bon déroulement du projet et permet de décrire la suite des événements. Et pour cela, nous allons donc devoir étudier l'outil MS Project.