MASTER EN SCIENCES INFORMATIQUES

Méthodologie de développement itérative orientée objet: étude de cas: vérificateur de liens

Dubus, Jean-François

Award date:
2002

FUNDP
Institut d'Informatique
Rue Grandgagnage, 21
B - 5000 NAMUR (Belgique)

méthodologie de développement itérative orientée objet
Etude de cas : vérificateur de liens

Jean-François DUBUS

Promoteur : Naji HABRA
(FUNDP - Institut d'Informatique, Namur)

Mémoire présenté pour l'obtention
du grade de licencié en informatique

Septembre 2002

Résumé

Le développement de logiciel regroupe un ensemble d'activités de gestion et d'ingéniérie. Ces dernières sont organisées en forme de cycle de vie. Il existe plusieurs types de cycle de vie disponibles ayant chacun leurs avantages et inconvénients. Le processus unifié (ou RUP pour Rational Unified Process) est un cycle de vie qui met l'accent sur l'aspect itératif du développement de logiciels. Dans ce travail nous avons adotpé un cycle de vie itératif, sorte de RUP allégé, et éprouvé certaines méthodologies et techniques de modélisation sur un exemple concret. Il s'agit d'un vérificateur de liens (URL) qui représente un sous-système d'une application plus large (un gestionnaire de ressources). Le présent travail a pour but de présenter, utiliser et évaluer ces différentes techniques comme UML et les Design Patterns afin de développer, en orienté-objet, un vérificateur de liens.

Abstract

The development of software gathers a whole of activities of management and ingeniery. These last are organized in the form of lifecycle. There are several types of lifecycle available having each one their advantages and disadvantages. The unified process (or RUP for Rational Unified Process) is a lifecycle which stresses the iterative aspect of the development of software. In this work we have adopted an iterative lifecycle, kind of RUP reduced, and tested certain methodologies and techniques of modeling on a concrete example. It is about a controller of links (URL) who represents a subsystem of a broader application (a resources manager). The purpose of this work is to present, use and evaluate these various techniques like UML and Design Patterns in order to develop, in oriented-object, a controller of links.

Remerciements

Je voudrais remercier toutes les personnes qui ont été touchés de près ou de loin par la réalisation ce mémoire, et plus particulièremnt le professeur Naji Habra et son équipe de recherche composé de Alain Renault, Miguel Lopez et Simon Alexandre.

J'aimerais encore remercier mes parents ainsi que tous mes proches sans qui l'accomplissement de mes études d'informaticien n'aurait certainement pas été possible.

Table des matières

Chapitre 1

Introduction

Le problème lors de la recherche d'une ressource (pages web, fichiers de documentation, images, etc) est de trouver les informations intéressantes et mises à jour et de ne pas perdre inutilement du temps sur les ressources n'apportant pas d'informations pertinentes. Le *Resources manager* [1] est un logiciel qui va permettre de stocker des liens vers des ressources, de les indexer, de faire des recherches sur ces ressources, de gérer leurs accès ainsi que leurs versions. L'accès aux ressources peut se faire par catégories ou par moteur de recherche. Chaque ressource est protégée par des droits d'accès (nom d'utilisateur ou groupe d'utilisateurs). Des ressources peuvent être proposées par des utilisateurs afin de partager leurs recherches ou leurs travaux. Ces ressources devront être validées afin d'assurer la qualité de l'outil. Pour garantir la disponibilité de ces ressources, l'outil devra vérifier les liens de temps en temps. Cette vérification se fera par le validateur de liens.

Le sous-système développé dans ce mémoire est un validateur de disponibilité d'une ressource. Cette validation consiste à vérifier si la ressource est accessible via les liens proposés. Cette vérification se fait tout au long de la vie du lien et ce à intervalles réguliers. Un administrateur peut voir ces vérifications en consultant les historiques de ces validations pour chacune des ressources afin de constater d'éventuelles anomalies : la ressource n'est plus disponible, le temps d'accès est trop long, etc.

Le mémoire comporte aussi comme objectif d'utiliser une méthode de développement itérative pour les projets à faibles ressources. Cette méthode itérative consiste à découper un grand projet en plusieurs sous-systèmes et à les développer à partir de l'analyse des exigences (cahier des charges) jus-

[1] http ://sourceforge.net/projects/resourcemanager

qu'à l'implémentation et ses validations. Chaque partie du projet se comporte comme un mini-projet. Dans chaque mini-projet, des itérations sont possibles entre les différentes phases du cycle de vie afin de rectifier les décisions prises en cas d'un non-respect du cahier des charges.

La première partie du mémoire décrit les avantages des méthodologies de développement de logiciel. Différents cycles de vie y sont aussi décrits et plus particulièrement le cycle de vie du processus unifié de Rational avec son principe d'itération. C'est ce cycle de vie qui est choisi pour développer le sous-sytème de validation de ressources. Le langage de modélisation UML ainsi que ses différents diagrammes y sont décrits. Une descripion des *Design Patterns* y figure aussi. Ces Design Patterns permettent de structurer certaines parties de l'architecture de façon sûre et stable du sous-système en se basant sur l'expérience de systèmes ayant des caractéristiques similaires.

La deuxième partie pose le problème de notre sous-système et reprend quelques solutions existantes se rapprochant de notre problème de validation. Cette partie décrit aussi certains choix concerant les donnéees à mémoriser afin de répondre aux exigences des utilisateurs sur la disponibilté des ressources.

La troisième partie décrit la conception du système grâce au langage UML. La conception donne la structure du système sous forme de classes, les interactions entre celles-ci, leurs rôles et leurs fonctionnements. Nous avons aussi la description de la base de données et des différentes méthodes construites pour le système. Enfin nous trouvons les tests de validations qui se basent sur les cas d'utilisations définis dans le cahier des charges et des exigences fonctionnelles et non fonctionnelles.

Chapitre 2

Méthodologie

2.1 Buts d'une méthodologie de développement de logiciels

Les exigences des systèmes informatisés demandées par la société croissent au fur et à mesure des avancées technologiques. Ces exigences augmentent en complexité, en taille et en responsabilité. En effet, l'économie actuelle dépend directement des systèmes informatisés et l'information en temps réel est devenue un besoin compétitif.

Régulièrement, de nombreux projets de développement logiciel échouent pour des raisons diverses. Jones et Yourdon identifient un certain nombre de symptômes qui annoncent ces échecs [11] [18] :

- une mauvaise interprétation des demandes des utilisateurs finaux ;
- une incapacité à tenir compte des changements du cahier des charges ;
- des modules qui ne fonctionnent pas bien ensemble ;
- des programmes difficiles à maintenir et à faire évoluer ;
- la découverte tardive de défauts sérieux dans le projet ;
- un manque de cohésion dans les équipes de développement, au point qu'il est impossible de savoir, a posteriori, qui a modifié quoi, quand, où et pourquoi ;
- un processus de construction et de livraison anarchique ;

Comme nous l'explique P. Kruchten [12], des causes plus profondes sont souvent à l'origine des échecs :

- une gestion du cahier des charges logiciel au coup par coup ;

- une communication ambiguë et imprécise ;
- des architectures fragiles ;
- des incohérences entre le cahier des charges logiciel, le modèle de conception et le modèle d'implémentation ;
- un nombre insuffisant de tests ;
- une évaluation subjective de l'avancement du projet ;
- une insouciance vis-à-vis des risques ;
- une absence de contrôle de la propagation des changements ;
- une trop faible automatisation des tâches ;

En traitant les causes profondes, on peut limiter les risques sources d'échec tout en améliorant le développement, la qualité et la maintenance des logiciels et cela d'une façon prévisible et reproductible.

P. Kruchten a étudié le fonctionnement des organisations de développement performantes pour analyser ces causes profondes et il constate qu'ils utilisent des pratiques communes :

- développer le logiciel de façon itérative ;
- gérer les exigences ;
- utiliser des architectures à base de composants ;
- modéliser graphiquement le logiciel ;
- vérifier la qualité du logiciel ;
- contrôler les changements apportés au logiciel ;

2.1.1 Développer le logiciel de façon itérative

Le développement itératif de logiciels se base sur le modèle en spirale de Barry Boehm et permet de développer, non pas le système entier, mais un sous-système exécutable en passant par les différentes étapes d'un cycle de vie. Les utilisateurs peuvent ainsi plus rapidement se manifester afin de corriger d'éventuelles erreurs ou soulever certaines omissions dans le cahier des charges. Cette manière de développer un logiciel permet aussi d'identifier des risques assez tôt permettant ainsi d'y remédier plus facilement.

Cette méthode élimine un certain nombre de problèmes :

- la mise en évidence, tôt dans le cycle de vie, de malentendus quand il est encore possible de les corriger ;
- la clarification du cahier des charges par les réactions des utilisateurs ;
- les tests effectués de façon répétée et continue permettent une évaluation de l'état de l'avancement du projet ;
- les incohérences entre le cahier des charges, les modèles de conception et d'implémentation sont découverts assez tôt ;
- la charge de travail des équipes est répartie uniformément sur tout le cycle de vie du produit ;
- les équipes exploitent le savoir acquis au cours du projet et améliorent ainsi le processus de développement ;
- les parties concernées reçoivent tout au long du cycle de vie du projet des preuves de son avancement ;

2.1.2 Gérer les exigences

La gestion des exigences consiste dans un premier temps à collecter toutes les exigences du client puis à les modifier dans le cas où les besoins des utilisateurs viennent à changer. En effet, il est quasiment impossible de définir de façon complète et exhaustive le cahier des charges. La gestion des exigences doit pouvoir mettre en évidence, organiser et décrire les fonctionnalités et les contraintes du système. Elle doit aussi permettre d'évaluer les changements à apporter au cahier des charges et d'estimer leur impact. Elle permet aussi de décrire les différents compromis, les décisions prises et d'en faire le suivi.

Cette méthode élimine un certain nombre de problèmes :

- les incohérences sont détectées plus facilement ;
- le dialogue entre les parties concernées se réfère à des besoins bien définis ;
- une adaptation est possible aux exigences changeantes ;
- une évaluation objective des fonctionnalités et des performances est possible ;
- une échelle de priorité des exigences est possible ;

2.1.3 Utiliser des architectures à base de composants

L'utilisation de composants permet de structurer le système en sous-système. Ces composants proviennent de l'expérience acquise au fur et à mesure des projets et sont renforcés dans leur stabilisation et dans leur généralité afin d'adopter une réutilisation plus fréquente.

Une architecture bien pensée permet un certain niveau de réutilisation, offre une division claire du travail entre les différents développeurs, isole les dépendances entre le logiciel et le matériel et facilite la maintenance.

Cette pratique élimine un certain nombre de problèmes :

- les composants facilitent la construction d'architectures souples et évolutives ;
- la modularité permet une séparation claire des problèmes répartis entre les éléments d'un système sujet aux changements ;
- une réutilisation facilitée par l'emploi d'architectures souples.

2.1.4 Modéliser le logiciel

Une modélisation est une simplification de la réalité afin de mieux comprendre le système. Plusieurs modèles sont construits suivant des points de vue précis pour la description du système. Ces modèles peuvent être des diagrammes de classe, des diagrammes d'états, des diagrammes des cas d'utilisation, diagrammes des composants, etc. Lorsqu'on utilise un langage de modélisation tel que l'UML (Unified Modelling Language), les membres des équipes de développement se communiquent les décisions de façon moins ambiguë. Les outils de modélisation permettent de voir différents niveaux d'abstraction, en cachant ou en montrant certains détails selon les besoins, facilitant ainsi la gestion des modèles. Ces outils aident aussi à maintenir la cohérence entre le cahier des charges avec les modèles de conception et d'implémentation.

Cette modélisation élimine un certain nombre de problèmes :

- les cas d'utilisation et les scénarios spécifient le comportement que doit avoir le système ;
- une meilleure visualisation de la conception du système ;

- les architectures non modulaires et rigides sont plus facilement reconnaissables ;
- le niveau de détails de conception peut être modifié à la demande ;
- les incohérences sont repérées plus facilement quand la conception est rigoureuse ;
- langage commun pour une meilleure communication entre les différents membres du projet.

2.1.5 Vérifier la qualité du logiciel

La vérification de la qualité d'un logiciel consiste à évaluer en continu les fonctionnalités, la fiabilité et les performances par rapport aux exigences définies préalablement dans le cahier des charges. Il est très important d'effectuer cette vérification en continu car la correction des problèmes opérée après la mise en service du logiciel engendre des coûts supplémentaires. Pour vérifier les fonctionnalités, des tests sont créés pour chaque scénario (ou comportement) du système. Cette phase de vérification est renforcée par le cycle de vie itératif qui permet de faire des tests à chaque itération.

Cette vérification élimine un certain nombre de problèmes :

- les résultats des tests permettent une évaluation du projet de façon objective ;
- cette évaluation met en évidence les incohérences entre le cahier des charges, le modèle de conception et le modèle d'implémentation ;
- les erreurs sont plus ciblées et donc plus faciles à corriger ;
- des tests plus ciblés peuvent être appliqués sur les zones à hauts risques.

2.1.6 Contrôler les changements apportés au logiciel

Le contrôle des changements apportés au logiciel permet une traçabilité des modifications apportées au fur et à mesure des versions. Cette traçabilité permet l'évaluation de l'impact des changements et de leurs gestions.

Cette méthode élimine un certain nombre de problèmes :

- l'impact des changements sur le logiciel peut être évalué et contrôlé ;

- le processus de gestion des changements est bien défini ;
- l'état d'avancement du projet est mieux évalué.

Voici, en guise de conclusion, les arguments fournis par P. Kruchten pour l'utilisation d'un tel processus de développement :

> *Le processus de développement sert à définir l'ordre des travaux, à spécifier ce qui doit être développé et à quel moment, à guider les développeurs dans leurs tâches et à offrir des critères pour le suivi et l'évaluation du projet. Ces processus de développement permettent de développer des systèmes complexes de façon reproductible et prévisible. Les équipes de développement utilisant un processus peuvent s'appuyer sur l'expérience de milliers de projets et ainsi d'augmenter leur efficacité et productivité.*

2.2 Cycles de vie

Le cycle de vie du développement d'un logiciel est un cadre de référence pour le processus de production d'un logiciel. L'utilisation d'un cycle de vie permet de faciliter la gestion du projet. En effet, le cycle de vie permet de découper le projet en phases qui seront transposées sur un calendrier. En outre, les différentes étapes d'un cycle de vie permettent d'identifier précisément tout ce qui est produit lors de chacune des étapes ; il sert aussi d'entrée à l'étape suivante. Enfin, chacune des phases et leurs produits seront placés sous la responsabilité d'une ou plusieurs personnes (selon la taille du projet).

Il existe plusieurs types de cycle de vie. Chaque type est adapté à certains projets. Les cycles de vie les plus connus et utilisés sont :

- Le cycle de vie en cascade (waterfall) ;
- Le modèle évolutif (evolutionary model) ;
- Le modèle transformationnel ;
- Le modèle par assemblage/réutilisation ;
- Le modèle en spirale ;
- Le cycle en V ;
- Unified Process ;

2.2.1 Le cycle de vie en cascade

Le cycle de vie en cascade est une succession linéaire de phases. La sortie d'une phase est l'entrée dans la phase suivante. Les différentes phases sont l'analyse et la définition des besoins, la conception, l'implémentation, les tests et la mise en service/maintenance. L'analyse et la définition des besoins ont pour but de décrire le logiciel futur (comportement, qualité, intégration,...). La conception a pour but de construire une architecture découpée en modules. Les modules ont des relations entre eux et sont détaillés dans leurs structures de données, algorithmes et interfaces. L'implémentation comprend le codage des modules ainsi que leurs tests unitaires et d'intégration. Les tests de vérification consistent à concevoir les jeux de tests globaux, à rechercher les failles du système et à les corriger s'il y a lieu. La mise en service et maintenance consistent à mettre le système dans son environnement, à mettre en place un apprentissage du système et à faire une maintenance corrective, évolutive et adaptative. Ce cycle de vie permet d'avoir une discipline dans le développement d'un système, avec des points de contrôle. Elle sert aussi de base pour les autres modèles. Les inconvénients de ce cycle de vie est dans la rigidité de ses phases : il est difficile de faire face aux changements. De plus, nous ne saurons que le système répond toujours aux exigences demandées par le client que lors de la mise en service.

2.2.2 Le modèle évolutif

Le but de ce cycle de vie est de permettre une réaction précoce de l'utilisateur afin de valider les spécifications par rapport à ses besoins réels. Ce cycle reprend le cycle de vie en cascade mais en intégrant une étape de prototypage. Le prototype évolue au fur et à mesure des validations par le client pour arriver au système final. Mais ce type de cycle de vie peut cacher une approche par essai-erreur et le logiciel peut être mal structuré. Ce cycle de vie est plus adapté dans un système dont les spécifications sont difficiles à établir ou dont la taille est très petite.

2.2.3 Le modèle transformationnel

Le but est d'avoir une continuité dans le développement du logiciel pour avoir un respect formel par rapport aux spécifications. La continuité du développement se fait par une suite de transformations justifiées par preuve formelle. La transformation va du plus abstrait en plus concret, du moins

efficace en plus efficace, du déclaratif en opérationnel.

2.2.4 Le modèle par assemblage/réutilisation

Le principe de ce modèle est de pouvoir réutiliser des composantes existantes et d'en faire un assemblage utile pour le développement du système. Les composantes sont utilisées telles quelles ou peuvent être adapter légèrement. Cela suppose d'avoir une bibliothèque de composantes documentées et spécifiées et que ces composantes soient prévues à la base pour la réutilisation. Ce modèle est principalement utilisé par des personnes produisant beaucoup de logiciels dans un domaine ciblé.

2.2.5 Le modèle en spirale

L'objectif de ce modèle est de mettre en évidence et de planifier l'analyse du niveau de risque à chaque nouveau produit. A chaque cycle du processus, il faut identifier les objectifs, les contraintes et les alternatives ; évaluer les alternatives, identifier les risques ; développer et vérifier le produit de ce niveau ; planifier le cycle suivant.

2.2.6 Le modèle de cycle de vie en V

Ce modèle met en évidence l'activité de validation et de vérification. A chaque étape du produit, des tests sont établis afin de valider le produit. Les validations débutent par des tests unitaires, ensuite les tests d'intégration et d'acceptation pour arriver à la mise en service et à la maintenance.

2.2.7 Unified Process

Ce modèle consiste à tenir compte des risques en appliquant un développement itératif et incrémental. L'itération permet de développer une partie d'un projet et de voir plus tôt les différents problèmes rencontrés et donc de mettre en évidence certains risques du projet global. Des itérations peuvent se faire aussi entre les différentes phases du cycle de vie.

Un cycle de vie est choisi en fonction des risques du projet et en fonction de la capacité de l'organisation à faire face aux contraintes que peut présenter un cycle de vie.

Sept risques majeurs ont été soulevés parmi tous les projets : la taille du projet, l'innovation technique, l'intégration avec l'existant, la complexité organisationnelle, l'impact organisationnel, l'instabilité de l'équipe de projet et la fiabilité des autres intervenants du projet.

- 1) Taille du projet
- 2) Innovation technique
- 3) Intégration avec l'existant
- 4) Complexité organisationnelle
- 5) Impacts organisationnels
- 6) Instabilité de l'équipe de projet
- 7) Fiabilité des autres intervenants du projet

Chaque projet aura des facteurs de risques prédominants en fonction de l'expérience de l'équipe projet, des outils utilisés, de l'organisation, etc.

Une des exigences du cahier des charges est d'utiliser une méthodologie itérative [OBJ003] pour les projets à faible effectif. Unified Process fait partie de ces méthodologies itératives mais elle est conçue pour les projets à grand effectif. Nous allégerons le RUP en ne développant que les artéfacts dont nous auront besoin.

2.3 Rational Unified Process

> Le Rational Unified Process (RUP) est un *processus de développement de logiciel*. Il permet d'affecter, dans une approche disciplinée, les tâches et les responsabilités au sein d'une organisation de développement. Son but est d'assurer la production d'un logiciel de grande qualité satisfaisant les demandes des utilisateurs finaux dans des délais et avec un budget prévisible [12].

Le processus de développement de logiciel RUP utilise des outils connus et éprouvés comme UML et les Design Patterns.

RUP reprend bien sûr les six pratiques développées dans les points précédents :

- développer le logiciel de façon itérative ;
- gérer les exigences ;
- utiliser des architectures à bases de composants ;
- modéliser graphiquement le logiciel ;
- vérifier la qualité du logiciel ;
- contrôler les changements apportés au logiciel.

Nous pouvons ajouter à RUP les caractéristiques suivantes[12] :

- RUP est piloté par les cas d'utilisation(*Use Cases*) ;
- RUP est un processus générique(*process framework*) que l'on peut adapter et étendre selon les organisations ;
- RUP intègre des outils qui facilitent l'exécution de certaines activités de développement logiciel.

2.3.1 Les éléments de modélisation du RUP

Un processus décrit *qui* fait *quoi, comment* et *quand.*

- *qui* : les travailleurs ;
- *quoi* : les artefacts ;
- *comment* :les activités et ses étapes ;
- *quand* : les enchaînements des activités.

Les travailleurs

Un travailleur est une personne ou une équipe ayant un rôle dans le projet comportant des responsabilités apportées par ce rôle. Une même personne ou équipe peut avoir plusieurs rôles dans un même projet. L'attribution des rôles est faite par le chef de projet lors de la planification et de la constitution des équipes.

Les artefacts

Un artefact est un élément d'information fabriqué, modifié ou utilisé par un processus. Les artefacts sont les résultats du projet, générés ou utilisés au fur et à mesure que l'on progresse vers le produit final [12].
Un artefact peut prendre différentes formes :

- un modèle (exemple : modèle de cas d'utilisation) ;
- un élément de modèle (exemple : un cas d'utilisation) ;
- un document (exemple : le cahier des charges) ;
- un code source ;
- un exécutable.

Les éléments livrables sont aussi des artefacts. Un artefact peut être composé de plusieurs artefacts, par exemple le cahier des charges contient les différents cas d'utilisation, les contraintes de temps, de budget, d'outils à employer, etc...

Les activités et leurs étapes

Comme nous le cite P. Kruchten [12], une activité est une unité de travail qu'un *travailleur* peut être amené à effectuer. Une activité a un but bien précis, il peut s'agir d'une création ou d'une mise à jour d'artefact, tels qu'un modèle, une classe ou un plan. Une même activité peut s'appliquer plusieurs fois sur un artefact comme dans le cas d'itération.
Planifier une itération ou trouver des cas d'utilisation sont deux exemples d'activités.

Une activité se divise en trois étapes :

- *Les étapes de réflexion*
  Le travailleur comprend la nature de la tâche, rassemble et examine les artefacts dont il dispose pour réaliser l'activité et formule un diagnostic (exemple : trouver des acteurs du système) ;
- *les étapes d'action*
  Le travailleur crée ou met à jour des artefacts (exemple : représenter le modèle de cas d'utilisation via des diagrammes de cas d'utilisation) ;

- *Les étapes d'inspection*
  Le travailleur examine les résultats et leur applique un certain nombre de critères pour décider de leur complétude et de leur qualité (exemple : évaluer les résultats).

Les enchaînements des activités

Un enchaînement d'activités est une suite d'activités qui produit un résultat observable. L'enchaînement peut être décrit, grâce à l'UML, par les diagrammes de séquence, de collaboration et d'activités.

P. Kruchten décrit dans son ouvrage qu'il existe neuf principaux enchaînements d'activités qui regroupent de façon logique tous les travailleurs et les activités du processus :

- 1 modélisation du métier (business modeling workflow) ;
- 2 gestion des exigences (requirements workflow) ;
- 3 analyse et conception (analysis and design workflow) ;
- 4 implémentation (implementation workflow) ;
- 5 test (test workflow) ;
- 6 déploiement (deployment workflow) ;
- 7 gestion de projet (project management workflow) ;
- 8 gestion de la configuration et des changements (configuration and change management workflow) ;
- 9 environnement (environment workflow).

Les six premiers sont des enchaînements d'activités d'ingénierie. Ils évoquent les phases séquentielles d'un processus traditionnel en cascade mais ils sont répétés de nombreuses fois tout au long du cycle de vie. Les trois derniers sont des enchaînements d'activités de soutien.

Phases

Le processus unifié comporte quatre phases de développement[10] : la création, l'élaboration, la construction et la transition.

La création Durant cette phase, il faut établir le "business case" et délimiter le projet. Il faut aussi identifier tous les intervenants ainsi que leurs

objectifs portant sur le système. Cela implique d'identifier tous les cas d'utilisation. Le business case inclut aussi les facteurs de succès, les facteurs de risque, l'évaluation des ressources nécessaires et un plan de phase avec des dates pour les étapes importantes.
A la fin de cette phase, nous devrions obtenir plusieurs artefacts :

- une vision générale sur les conditions du projet, sur les dispositifs et contraintes principaux ;
- un modèle général de cas d'utilisation ;
- un glossaire du projet ;
- les facteurs de succès et de risque ;
- une prévision financière ;
- un plan de projet montrant des phases et des itérations ;
- un ou plusieurs prototypes.

Cette étape permet de voir la faisabilité du projet et les risques encourus.

L'élaboration La phase d'élaboration permet de préciser la plupart des cas d'utilisation et de concevoir l'architecture. L'architecture est exprimée sous forme de vues dite *architecture 4+1 vues*. Ces cinq vues sont la vue logique, la vue des processus, la vue d'implémentation, la vue des cas d'utilisation et la vue de déploiement[10]. La prévision des activités et l'estimation des ressources pour le projet doivent être réalisables.

Le modèle d'architecture 4+1 vues proposé par RUP, présentée ci-dessous, nous montre que la vue des cas d'utilisation est utilisée dans les quatre autres vues.

La vue logique s'intéresse aux aspects fonctionnels, donc à ce que le système doit faire pour les utilisateurs. Cette vue décrit les classes majeures du système et leur organisation en paquetages et sous-systèmes.

La vue d'implémentation décrit l'organisation des modules du logiciel. Cela comprend les codes sources, exécutables, les autres fichiers et la gestion de configuration et des versions.

La vue des processus décrit l'exécution des tâches en parallèle ainsi que des problèmes inhérents à la concurrence et au parallélisme : tolérances aux

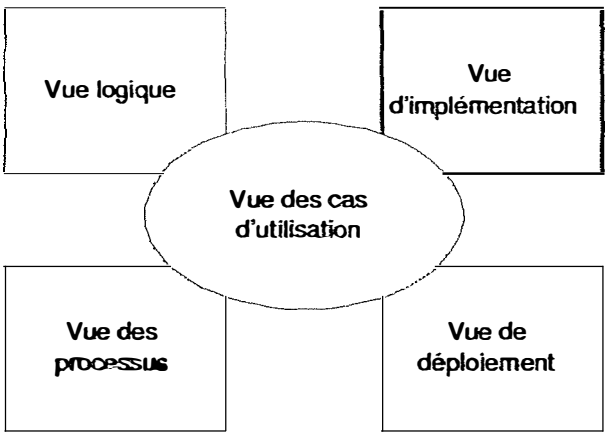


FIG. 2.1: Modèle d'architecture 4+1 vues

pannes, démarrage et arrêt du système, temps de réponse, capacité de traitement, etc.

La vue de déploiement s'occupe des problèmes de déploiement des modules dans un réseau d'ordinateurs et de ses exécutions, de l'installation du système, de performances, etc.

La vue des cas d'utilisation contient les scénarios et cas d'utilisation les plus importants. Ils sont utilisés pour guider la conception de l'architecture pendant les phases d'inception et d'élaboration. Ils servent aussi pour valider les différentes vues architecturales.

A la fin de cette étape, nous devrions obtenir comme artefacts :

- une liste de tous les cas d'utilisation et leurs descriptions développées ;
- une liste des acteurs définis ;
- les exigences supplémentaires non reprises dans les cas d'utilisation ;
- une description de l'architecture du système ;
- un prototype de l'architecture applicable ;
- une liste des risques ;
- un programme de développement pour le projet global, y compris un plan d'ensemble du projet montrant des itérations et des critères d'évaluation pour chaque itération.

La construction C'est pendant cette phase que le système se construit et que l'architecture et les plans évoluent pour obtenir en final un produit prêt à être livré aux utilisateurs. C'est ici que nous pouvons voir s'il y a des anomalies majeures ou mineures au point de vue de l'architecture. La fin de cette phase doit permettre d'obtenir un produit logiciel intégré à une plate-forme, un manuel d'utilisateur et une description de la version courante avec tous les cas d'utilisation intégrés dans cette version de produit.

Il faut vérifier si la version est stable et si elle est acceptable vis à vis des exigences établies et exploitable par les utilisateurs finaux.

La transition Cette phase consiste à transmettre le produit aux utilisateurs. Cela suppose aussi d'en assurer la fabrication, la livraison, la formation, le soutien et la maintenance jusqu'à ce que les utilisateurs soient satisfaits. C'est pendant cette phase que les anomalies et défauts sont constatés et corrigés par les développeurs ou remis pour la version suivante. Des améliorations peuvent être suggérés pour la version qui suivra éventuellement.
Le graphique de la figure 2.2 montre les efforts à fournir pendant les diffé-

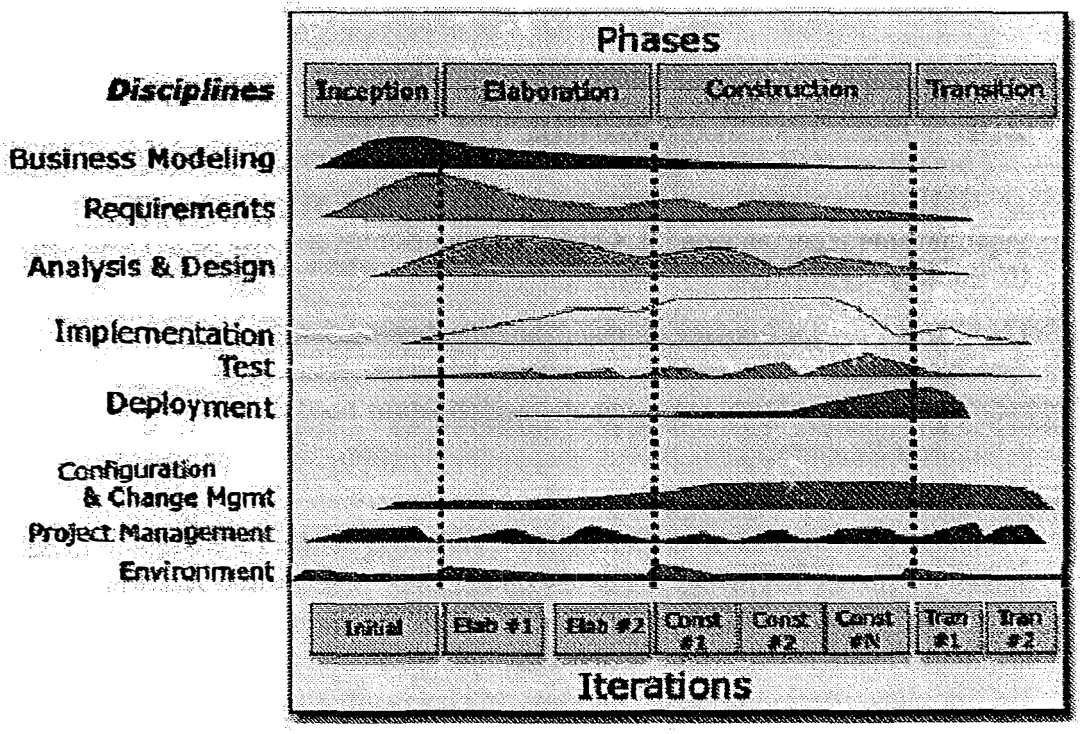


FIG. 2.2: Phases

rentes phases et disciplines utilisées. Nous pouvons aussi constater que plusieurs itérations successives d'une phase peuvent avoir lieu.

Nous pouvons remarquer que lors de la phase de création, les disciplines de modélisation métier et des exigences interviennent beaucoup. En effet, c'est pendant cette phase que le cahier des charges est établi, comprenant les objectifs du système, les intervenants et leurs buts.

Dans la phase d'élaboration, l'analyse et l'architecture prennent plus d'effort. C'est pendant cette phase qu'il faut construire toute l'architecture du système.

Lors de la phase de construction, l'implémentation et les tests sont prépondérants.

Ensuite lors de la transition, c'est le déploiement qui intervient.

Les activités de soutien (gestion de projet, environnement et gestion de la configuration et des changements) se font tout au long du cycle de vie.

2.4 Unified Modelling Language

UML est un outil de communication standard et fortement utilisé pour les différents intervenants [5]. Craig Larman donne une définition du langage de modélisation [14] :

> *UML est un langage de spécification, de représentation graphique, de construction et de documentation des artefacts des systèmes logiciels ainsi qu'un langage de modélisation métier et de systèmes non-logiciels.*

Ce langage commun va permettre d'échanger entre les différents intervenants du projet toutes les décisions de manière précise et non ambiguë.

Ce langage de modélisation consiste à créer une représentation informatique des éléments du monde réel auxquels on s'intéresse sans se préoccuper de l'implémentation donc du langage de programmation. Cette modélisation possède plusieurs caractéristiques[1] :

[1] www.uml.free.fr

- un langage sans ambiguïté ;
- un langage universel pouvant servir de support pour tout langage orienté objet ;
- un moyen de définir la structure d'un programme ;
- une représentation visuelle permettant la communication entre les acteurs ;
- une notation graphique simple, compréhensible même par des non informaticiens

UML offre plusieurs représentations afin de recouvrir les différents aspects du système. Ces représentations sont des diagrammes regroupés en vues. Nous distinguons les vues statiques et les vues dynamiques [5].

Les vues statiques représentent le système physiquement :

- diagrammes d'objets ;
- diagrammes de classes ;
- diagrammes de cas d'utilisation ;
- diagrammes de composants ;
- diagrammes de déploiement.

Les vues dynamiques montrent le fonctionnement du système :

- cas d'utilisation ;
- diagrammes de séquence ;
- diagrammes de collaboration ;
- diagrammes d'états transitions ;
- diagrammes d'activités.

Nous pouvons définir les principaux objectifs de ces différents diagrammes.

Les cas d'utilisation

Les cas d'utilisation définissent explicitement le comportement attendu du système [5] [15]. Ils fournissent aussi un lien important entre le cahier des charges et d'autres artefacts de développement tel que le modèle de conception et de tests. Les cas d'utilisation constituent donc un point de départ

pour l'activité de développement et jouent un rôle prépondérant essentiellement dans la gestion des exigences, la conception et les tests.

L'utilisation des cas d'utilisation (scénarios d'utilisation ou Use Cases) permet une meilleure compréhension de l'attente du client ; ces cas d'utilisation nous donnent les exigences fonctionnelles du système. Ces scénarios représentent une séquence d'actions que réalise le système et qui fournissent un résultat utile pour un acteur particulier. Cet acteur peut être quelqu'un ou quelque chose se situant à l'extérieur du système et qui interagit avec lui.

Les cas d'utilisation définissent les interactions des acteurs (personnes ou autres systèmes) avec le système.
Pour chaque acteur, il faut définir ce qu'il veut faire avec le système (les buts) et chacun de ces buts devient un Use Case. Nous obtenons ainsi les exigences fonctionnelles du système.
L'ensemble des Uses Case doit inclure tous les intervenants ainsi que leurs objectifs.

Diagrammes d'objets

Ce type de diagramme UML montre des objets (instances de classes dans un état particulier) et des liens (relations sémantiques) entre ces objets.

Les diagrammes d'objets s'utilisent pour montrer un contexte (avant ou après une interaction entre objets par exemple).

Ce type de diagramme sert essentiellement en phase exploratoire, car il possède un très haut niveau d'abstraction.

Diagrammes de classes

Une classe est un type abstrait caractérisé par des propriétés (attributs et méthodes) communes à un ensemble d'objets et permettant de créer des objets ayant ces propriétés.

Classe = attributs + méthodes + instanciation

Ne pas représenter les attributs ou les méthodes d'une classe sur un diagramme n'indique pas que cette classe n'en contient pas. Il s'agit juste d'un

filtre visuel, destiné à donner un certain niveau d'abstraction à son modèle.

Diagrammes de composants

Les diagrammes de composants permettent de décrire l'architecture physique et statique d'une application en terme de modules : fichiers sources, librairies, exécutables, etc. Ils montrent la mise en oeuvre physique des modèles de la vue logique avec l'environnement de développement.
Les dépendances entre composants permettent notamment d'identifier les contraintes de compilation et de mettre en évidence la réutilisation de composants.

Les composants peuvent être organisés en paquetages, qui définissent des sous-systèmes. Les sous-systèmes organisent la vue des composants (de réalisation) d'un système. Ils permettent de gérer la complexité, par encapsulation des détails d'implémentation.

Diagrammes de déploiement

Les diagrammes de déploiement montrent la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels.
Les ressources matérielles sont représentées sous forme de noeuds.
Les noeuds sont connectés entre eux, à l'aide d'un support de communication. La nature des lignes de communication et leurs caractéristiques peuvent être précisées.

Diagrammes de séquence

Les diagrammes de séquences permettent de représenter des collaborations entre objets selon un point de vue temporel ; on y met l'accent sur la chronologie des envois de messages.
Les diagrammes de séquences peuvent servir à illustrer un cas d'utilisation.
L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme ; le temps s'écoule "de haut en bas" de cet axe.
Les diagrammes de séquences et les diagrammes d'état transitions sont les vues dynamiques les plus importantes d'UML.

Diagrammes de collaboration

Les diagrammes de collaboration montrent les interactions entre objets (instances de classes et acteurs).
Ils permettent de représenter le contexte d'une interaction, car on peut y préciser les états des objets qui interagissent.

Diagrammes d'états transitions

Ce diagramme sert à représenter des automates d'états finis, sous forme de graphes d'états, reliés par des arcs orientés qui décrivent les transitions.
Les diagrammes d'états transitions permettent de décrire les changements d'états d'un objet ou d'un composant, en réponse aux interactions avec d'autres objets/composants ou avec des acteurs.

Un état se caractérise par sa durée et sa stabilité, il représente une conjonction instantanée des valeurs des attributs d'un objet. Une transition représente le passage instantané d'un état vers un autre.
Une transition est déclenchée par un événement. En d'autres termes : c'est l'arrivée d'un événement qui conditionne la transition.

Diagrammes d'activités

UML permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation avec l'aide de diagrammes d'activités (une variante des diagrammes d'états transitions).
Une activité représente une exécution d'un mécanisme, un déroulement d'étapes séquentielles.

Le passage d'une activité vers une autre est matérialisé par une transition.
Les transitions sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre (elles sont automatiques).
En théorie, tous les mécanismes dynamiques pourraient être décrits par un diagramme d'activités, mais seuls les mécanismes complexes ou intéressants méritent d'être représentés.

Quelques caractéristiques des diagrammes UML :

La structure des diagrammes UML et la notation graphique des éléments de modélisation sont normalisées.

Le recours à des outils appropriés est un gage de productivité pour la rédaction des diagrammes UML car :

- ils facilitent la navigation entre les différentes vues ;
- ils permettent de centraliser, organiser, partager et synchroniser les diagrammes (indispensable avec un processus itératif) ainsi que faire des versions ;
- ils facilitent l'abstraction par des filtres visuels ;
- ils simplifient la production de documents et autorisent (dans certaines limites) la génération de codes ;

2.5 Les Design Patterns

C. Alexander caractérise les Design Patterns de la manière suivante : ils décrivent un problème récurrent dans un environnement donné puis décrivent une solution à ce problème de manière telle que cette solution soit réutilisable à chaque fois qu'on rencontre le problème, sans cependant qu'elle soit deux fois exactement la même [6]. Le cycle de vie RUP utilise les design patterns pour structurer l'architecture du système.

Un Design Pattern doit toujours nommer, motiver et expliquer une solution d'architecture qui solutionne un problème récurrent dans un système orienté objet. Par conséquent, il doit également décrire précisément le problème, sa solution et ses conséquences. Il donnera en outre des conseils d'implémentations et des exemples. La solution obtenue sera donc un ensemble d'objets et de classes qui solutionne le problème donné [8].

Un pattern possède en règle générale quatre éléments essentiels :

Un nom pour le pattern qui exprime en un mot ou deux ce que le pattern décrit comme problème et donne comme solution.

Le problème situe le contexte dans lequel le pattern peut être appliqué. Le niveau de détail souhaité va jusqu'à décrire un problème d'architecture comme la représentation d'algorithmes en objets. On peut y joindre, si c'est possible, une liste des conditions qui doivent être rencontrées avant de choisir et d'appliquer un pattern.

La solution est constituée des différents composants, leurs liens, responsabilités et collaboration. Il ne s'agit pas de décrire une architecture particulière mais bien d'une architecture de plus haut niveau, d'un modèle à appliquer à un cas particulier.

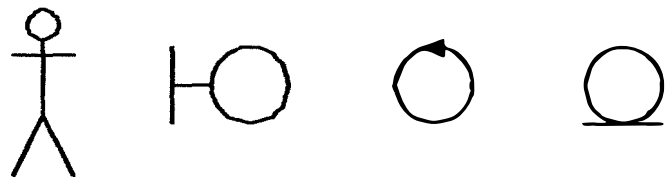
Les conséquences expliquent ce qu'occasionne l'utilisation du pattern. Les conséquences de ces choix d'architecture sont cruciales car elles peuvent avoir un impact plus ou moins important sur le comportement du système.

2.6 Les diagrammes de robustesse

Une première analyse par les schémas de robustesse, n'appartenant plus à UML, nous permet de mettre en évidence les différents objets du système à partir des Use Cases [17]. Cette étape intermédiaire permet de dégager les différents objets pour construire le diagramme de classe. Ivar Jacobson distingue quatre types d'objets [10] :

- *les acteurs* : objets qui correspondent à la définition des acteurs dans les Use Cases
- *les interfaces* : objets qui permettent l'interaction entre un acteur et le système
- *les contrôleurs* : objets dotées d'une certaine intelligence pour assurer l'objectif du Use Case, on peut également introduire une structuration de ces contrôleurs avec une relation de composition/décomposition
- *les entités* : objets qui se chargent de stocker l'information, une base de données, un fichier, etc.

La représentation de ces quatre types d'objet est :



Un diagramme de robustesse exprime l'interaction de ces objets au moyen de liens entre ceux-ci.

Les liens suivants sont valides :

- un acteur dialogue avec une interface
- une interface envoie l'information à un contrôleur
- un contrôleur communique avec un autre contrôleur
- un contrôleur utilise une entité
- un contrôleur initie une interface
- un contrôleur se décompose en n contrôleurs

2.7 La norme 9126

La norme 9126 proposée par l'ISO contient un modèle de qualité pouvant servir aux spécifications fonctionnelles ou non-fonctionnelles d'un système et évaluer un logiciel à acquérir.

Le modèle de qualité est un ensemble de caractéristiques et des relations qui produisent les bases de la qualité des spécifications des besoins et de la qualité d'évaluation.

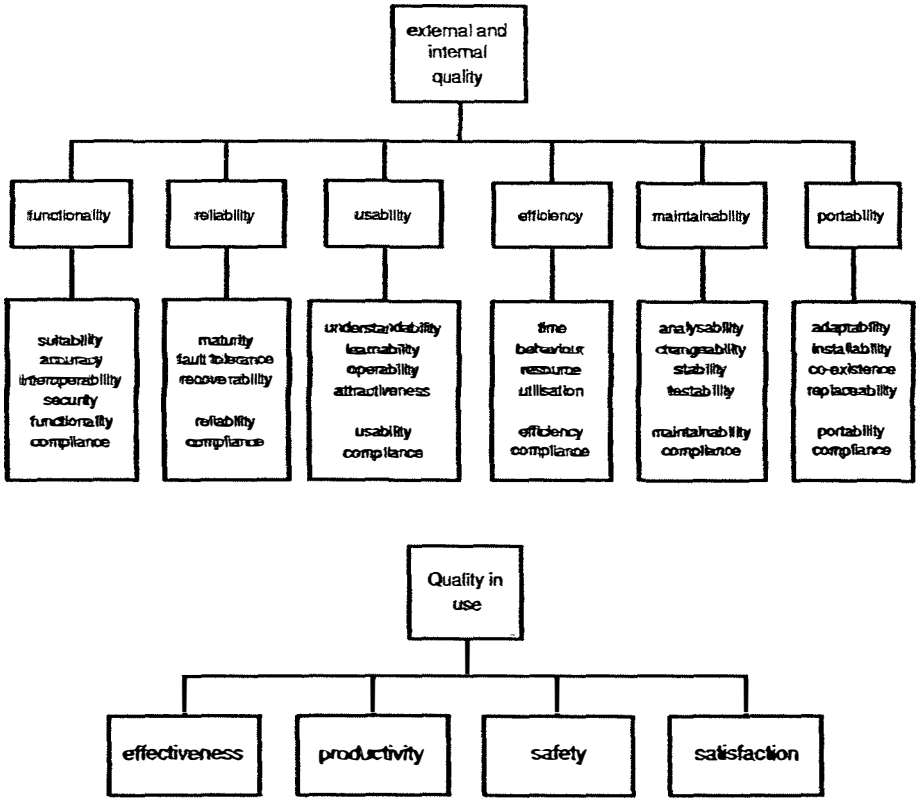
Le modèle est divisé en deux parties. Une première partie reprend la qualité interne et externe du logiciel. Une seconde partie reprend la qualité du logiciel en utilisation.

La qualité interne et externe comprend six caractéristiques (*functionality, reliability, usability, efficiency, maintainability, portability*)[2] qui peuvent encore être subdivisées.

La qualité interne est la totalité des caractéristiques du logiciel d'un point de vue intérieure : modèles statique et dynamique, code, etc.
La qualité externe est la totalité des caractéristiques du logiciel d'un point de vue extérieur (lors de l'utilisation du logiciel).

La qualité en utilisation se positionne en fonction de l'utilisateur. La qualité est la capacité du logiciel à produire le résultat demandé par l'utilisateur. Quatre caractéristiques sont proposées : *effectiveness, productivity, safety, satisfaction*[3].

[2](fonctionnalité, fiabilité, utilisabilité, efficacité, maintenabilité, portabilité)
[3]efficacité, productivité, sécurité, satisfaction



Chaque personne ayant un rapprochement avec le logiciel est dite utilisateur. Cet utilisateur a une influence sur le projet ou est influencé par le projet. Il existe plusieurs catégories d'utilisateur dites " les parties intéressées " [4] : les utilisateurs finaux, les développeurs, les testeurs, le manager de qualité, les opérateurs, les mainteneurs, les demandeurs, etc.

La norme 9126 propose des métriques pour les caractéristiques et sous-caractéristiques pour évaluer la qualité du produit en fonction des exigences.

[4] stakeholders

Chapitre 3

Description du problème étudié

3.1 Objectif

Le travail réalisé dans ce mémoire est de développer un validateur de liens. Ce travail représente aussi l'application d'une méthodologie itérative qui est une exigence soulignée dans le cahier des charges. Cette itération est l'application d'un cycle de vie portant sur un sous-système. Ce sous-système part du cahier des charges jusqu'à la mise en service en passant par la conception et l'implémentation. Ce vérificateur de lien validera des liens URL et permettra de consulter les données de ces validations.

Ce sous-système devra aussi satisfaire aux exigences non fonctionnelles exprimées dans le cahier des charges à l'aide de la norme ISO/IEC 9126. Ce sous-système sera utilisé par un administrateur pour lancer la validation des liens et la visualisation des données engendrées par les validations.

Certaines exigences non fonctionnelles sont requises comme la performance des temps de réponse du système pour répondre à une tâche déterminée, la capacité du système à être maintenu, sa portabilité ou encore son efficacité. Pour les temps de réponses, notre sous-sytème doit pouvoir afficher les visualisations dans un temps compris entre 0 et 7 secondes pour une connexion avec un modem et entre 0 et 3 secondes pour une connexion à haut débit[1]. La validation n'est pas concernée par ces temps car le temps de validation dépend du nombre de ressources à valider. Pour la capacité de maintenance, il faut pouvoir détecter et analyser les pannes assez facilement. Il faut pouvoir aussi modifier le programme afin de pouvoir l'adapter et/ou le corriger. Une adaptation possible peut passer par des paramètres du pro-

[1] Voir le point 3.5.3 dans le cahier des charges

gramme. Le langage de programmation PHP permet une grande portabilité sur différents systèmes d'exploitation. Du point de vue de l'efficacité et de la productivité, l'administrateur doit pouvoir facilement lancer une validation et lancer les visualisations des données. Ceci peut-être mesuré par le temps mis pour accomplir la tâche. Ce temps ne doit pas excéder les trois minutes comme indiqué dans le point 3.5.7.2 du cahier des charges.

3.2 Etat de l'art

Cette validation d'url a pour but de vérifier la disponibilité de toutes les ressources proposées par les fournisseurs de contenu[2]. Ces ressources peuvent être stockées sur des pages Web (html) ou sur des sites FTP.

La bibliothèque en ligne de l'université de Chicago a développé un programme (Bobcat) pour vérifier si les liens URL référencés dans leurs pages Web sont encore actifs.

Les liens peuvent référencer des ressources se trouvant sur des pages Web (liens http) ou sur des sites FTP.
Pour la validation des liens de ressources se trouvant sur des pages Web, ils utilisent la requête HEAD du protocole http. Cette requête permet de rapatrier seulement les informations sur le fichier demandé et ainsi que le code d'état de http (ex : 200, 304, 404).
Ils utilisent ensuite le code d'état pour reconnaître un éventuel problème.

Le code d'état reçu en réponse d'une requête HEAD[3] ou GET de HTTP est l'information la plus importante. Ce code nous renseigne directement sur la disponibilité de la ressource. Nous prenons certains code d'état provenant de la norme RFC1945[4] pour notre système :

- 200 : La requête a abouti ;
- 30X : La ressource ne se trouve plus à l'adresse indiquée et le serveur indique dans la réponse une liste des caractéristiques et localisations de la ressource demandée ;
- 40X : La ressource ne peut être lue par le client pour des raisons diverses : requête incorrecte, client non autorisé, interdit ou ressource non

[2]Un fournisseur de contenu est un utilisateur proposant une ressource
[3]La requête HEAD rapatrie seulement les informations concernant la ressource
[4]Voir en annexes

existante ;

- 50X : Le serveur n'a pu traiter la requête reçue.

Un code 200 nous indique que la ressource existe et est accessible.

Un code 300, 301 et 302 nous indique que l'URL n'est plus valide et qu'il faut avertir un administrateur capable de changer cet URL de la ressource. La réponse de la requête peut contenir le nouvel URL.

Un code 400, 401, 403 et 404 nous indique que la ressource n'est pas accessible ; soit la requête n'a pu être reconnue par le serveur (erreur 400), soit la requête du client n'est pas autorisée, ou l'authentification est mauvaise (erreur 401), soit le serveur interdit l'accès à la ressource (erreur 403) pour des raisons qu'il ne veut pas dire, soit encore le serveur n'a pas trouvé la ressource demandée (erreur 404).

Un code 500 et 503 nous indique que le serveur a reçu une bonne requête mais qu'il ne peut la satisfaire soit à cause d'un événement inattendu (erreur 500) soit par surcharge ou erreur de maintenance (erreur 503).

Pour les ressources se trouvant sur des serveurs FTP, la tâche est plus complexe car il faut d'abord s'identifier auprès du serveur (nom d'utilisateur, mot de passe). Ensuite, il faut pouvoir changer de répertoire avec la commande FTP CWD (Change Working Directory) pour avoir accès au répertoire énoncé dans le lien. Si le lien correspond au répertoire et pas à un fichier contenu dans ce répertoire, alors la vérification est un succès. Dans le cas contraire, il faut savoir si le fichier existe. Pour cela, il faut demander de le rapatrier par la commande FTP RETR en créant ainsi une nouvelle connexion. Immédiatement après la connaissance de l'existence ou non du fichier grâce au code d'état renvoyé au début du rapatriement, il faut arrêter son téléchargement par la commande FTP ABOR tout en espérant ne pas avoir pris trop de bande passante de la ligne. Pour finir, il faut fermer la connexion FTP par la commande QUIT.

D'autres ressources peuvent être envisagées comme un article venant de newsgroup, un lien 'Mailto', Telnet, Gopher.

Pour un article d'un newsgroup, grâce au protocole NNSP (Network News Transfert Protocol) la commande GROUP permet de sélectionner le groupe de discussion et la commande STAT permet vérifier l'existence d'un article. Des codes d'erreurs sont renvoyés si soit le groupe de discussion soit l'article

n'existe pas.

Un problème lié à leur bibliothèque en ligne était la redondance des liens. Sur les quelques 4900 pages Web comprenant environ 131 000 URL, 18 000 URL étaient différents. Une table de hashing permet de lister une seule fois les différents URL répertoriés dans leurs pages Web. Environ 9 000 URL sur les 18 000 n'ont pas été retenus pour des raisons internes. Leur programme 'bobcat' vérifie et valide le restant des liens URL entre deux et trois heures, ceci en prenant un lien à la fois.

Pour la vérificationd'un plus grand nombre d'URL (plusieurs dizaines de milliers), ils préconisent de faire du multi-tâches, c'est à dire de lancer plusieurs processus de vérification en même temps sans pour autant saturer la bande passante ni le processeur. University Florida utilise un programme pour vérifier les liens ; ils l'utilisent une fois par semaine.

La plupart des vérifications se font de semaine en semaine. Cela peut s'expliquer par le fait que d'une part les gens calculent en semaines (donc décident de travailler sur leur site un jour par semaine) et d'autre part, ils remettent à jour leur site Internet lorsqu'ils en ont le temps, en majorité le week-end. Pour les sites officiels, les vérifications se font le plus souvent pendant les heures d'ouverture des bureaux. Nous devons donc connaître le nombre de vérifications demandées par semaine ainsi que le jour de la semaine. La vérification devrait idéalement se faire la nuit afin de ne pas gêner le trafic.

3.3 Description de la disponibilité d'un URL

D'abord URL est l'abréviation du nom anglais de Uniform Resource Locator, c'est la syntaxe employée sur le WWW pour spécifier la localisation physique d'un fichier ou d'une ressource se trouvant sur l'Internet[5] sous la forme : protocole ://serveur/répertoire/document.extension.

Nous devons décrire les qualités de la disponibilité d'un URL pour savoir que mesurer lors de la phase de validation afin de correspondre le plus possible avec les attentes des utilisateurs.

[5]http ://www.culture.fr/culture/dglf/ressources/lexiques/internet.htm

Bien sûr, la première qualité que doit avoir cet URL est qu'il doit renseigner une ressource existante, et que celle-ci soit accessible à tout moment.

Le résultat d'études[6] menées sur le comportement des internautes lorsqu'ils se connectent à un site web montre qu'au bout de 7 secondes, les visiteurs se lassent et s'en vont découvrir d'autres horizons virtuels!

Des études ont essayé de fixer les différents seuils de vitesse et leurs conséquences sur la perception des internautes. Voici les principaux d'entre eux :

- jusqu'à 100 ou 200 millisecondes, la vitesse est si rapide que l'oeil humain perçoit les modifications comme instantanées
- jusqu'à 1 seconde, le visiteur perçoit les temps de chargement mais il peut naviguer très confortablement d'une page à une autre. Ce temps de 1 seconde est la limite maximale " autorisée " pour les systèmes Intranet, au sein d'un même réseau local
- entre 1 et 10 secondes, la sensation de lenteur augmente jusqu'à atteindre un niveau intolérable, qui dépend de chacun
- au-delà de 10 secondes : le niveau " tolérable " maximum s'établissant en moyenne à 7 secondes, vous pouvez être certain qu'au-delà de 10 secondes vous aurez perdu plus de 50% de vos visiteurs.

Pour la visualisation des ressources, il est donc préférable que les sites web ne dépassent pas la cinquantaine de kilo-octets (avec un modem 56K) afin de ne pas dépasser les sept secondes.

La conférence du World Wide Web décrit aussi qu'un maximum de deux secondes est souhaité pour l'affichage d'une ressource html car au delà de ce seuil, des utilisateurs commencent à trouver le temps long et quittent le site.[7]

Ceci s'applique aux internautes moyens cherchant des informations sans savoir si la ressource les contient. Dans notre cas, les utilisateurs du système sont des chercheurs et savent que la ressource en question contient des informations utiles grâce à la description mentionnée par le fournisseur de contenu et validé par un administrateur.

Donc cette étude n'est pas entièrement applicable pour notre système car le chercheur veut bien admettre d'attendre un peu plus longtemps pour des

[6]http ://www.w3ping.fr/news/archives/convaincre.html

[7]http ://www9.org/w9cdrom/ *Concepts for Improved Visualization of Web Link Attributes*, 4.3 Access Time

informations qu'il est sûr d'avoir. De plus, les ressources peuvent être des documents tels que des textes(pdf, ps, doc), tableau(xls), images (bmp, jpg) qui peuvent prendre plusieurs dizaines d'octets à quelques mégaoctets.

Nous regroupons les codes d'état de façon à mettre ensemble ceux qui produisent un même effet pour notre description de disponibilité : Nous avons le code 200 que nous souhaitons avoir pour tous les URL, les codes 400, 401, 403, 404, 301 et 302 nous disant que la ressource n'est pas accessible via l'URL indiqué. Nous avons aussi les codes 500 et 503 par lesquels nous ne pouvons être sûr de la disponibilité de la ressource; nous devrons donc essayer de nouveau la requête un peu plus tard.

Nous attribuons des points aux différents regroupements afin d'avoir par après une métrique se rapprochant le plus possible de l'idée que les utilisateurs ont sur le mot de disponibilité de la ressource :

- 1 point pour le code 200;
- 0.2 point pour les codes 500 et 503;
- 0 point pour les codes 400, 401, 403, 404, 301, 302.

Le temps de réponse de la requête HEAD ou GET est une donnée qui peut nous renseigner sur la qualité de la ligne (surcharge ou bande passante) pour atteindre la ressource. Plus le temps de réponse sera grand, plus l'accès à la ressource (visualisation ou téléchargement) sera long et plus l'utilisateur sera mécontent. Si après un certain temps nous avons pas encore reçu de réponse de la requête, alors nous devons associer à cet URL un code d'état 404 (pour lui attribuer zéro point).

Nous pouvons donc faire intervenir le temps de réponse dans l'attribution des points : en enlevant des points (ou pourcentage de points) à chaque laps de temps dépassé (type escalier) ou à partir d'un laps de temps, nous diminuons les points proportionnellement en fonction du temps (pente).

Nous mettons le premier seuil à deux secondes. Nous admettons que le temps entre l'envoi de la requête et la réception de la réponse a largement le temps de se faire en moins de deux secondes et ce avec une charge normale du Web.

Au delà de dix secondes, nous considérons que la ressource est inaccessible et par conséquent, nous devrions donner à l'URL zéro point.

Nous adoptons la diminution des points de façon linéaire. Le graphique que nous proposons nous donne le coefficient multiplicateur en fonction du temps de réponse de la requête. Ce coefficient sera appliqué au point obtenu par les codes d'état :

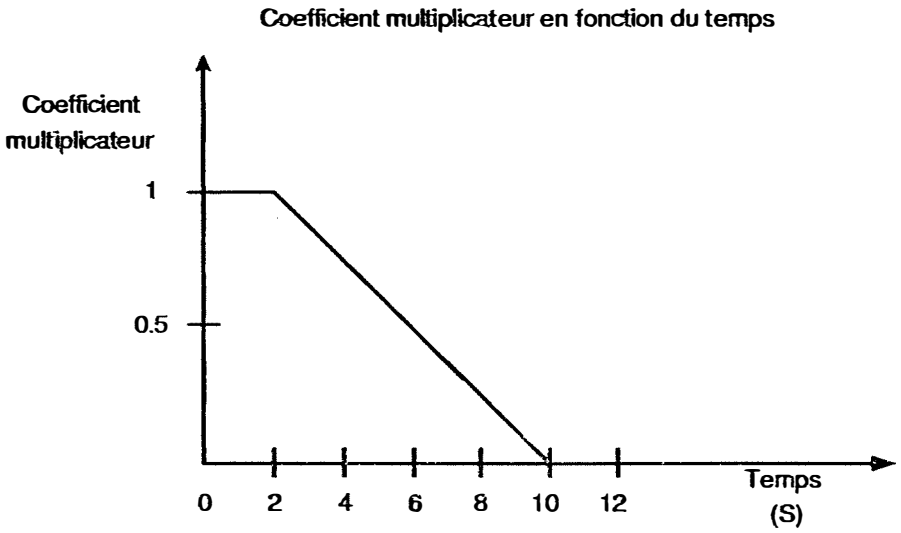


FIG. 3.1: Coefficient multiplicateur

L'équation du graphique est :

si 0 <= X <= 2 alors Y = 1
si 2 < X <= 10 alors $Y = \frac{-1}{8}X + \frac{5}{4}$
si X > 10 alors Y = 0

3.4 Stockage des données pour les statistiques

Les données que nous allons stocker doivent nous permettre de calculer les statistiques pour chaque ressource.

Plusieurs choix s'offrent à nous :

- nous stockons toutes les réponses des codes d'état ainsi que son temps de réponse pour chaque URL connu.

- nous stockons la moyenne des points ainsi que le nombre de réponses totales, nous pouvons aussi stocker la moyenne des temps de réponses.

La première solution a comme avantage qu'on peut faire des statistiques sur les X dernières validations ou sur les X derniers mois ou dernières semaines. Nous pouvons voir aussi le nombre de fois que chaque code d'erreur a été détecté, voir les moyennes des temps de réponses, ainsi que le nombre de fois que le temps maximum de temps de réponse a été dépassé, etc...

Mais elle a comme désavantage que le nombre d'éléments à stocker peut devenir énorme; en effet, après un an d'utilisation avec quelques milliers d'URL et pour chacun d'eux les couples de codes d'état et leurs temps de réponse, nous obtiendrions beaucoup trop de données à mémoriser pour notre application.

A chaque visualisation des statistiques, il faut parcourir tous les codes état avec leurs temps de réponse et calculer le pourcentage de disponibilité. Ceci prolongera le temps de la visualisation par l'administrateur.

La deuxième solution a comme avantage de stocker directement la moyenne car elle est calculée à chaque validation de l'URL : nous multiplions la moyenne des points par le nombre de validations déjà faites, nous ajoutons à ce produit le dernier point obtenu et nous divisons le tout par le nouveau nombre de validations (ancienne valeur plus un). La moyenne du temps de réponse peut être calculée de la même manière.

Mais le désavantage que présente cette solution est qu'elle ne permet pas de faire beaucoup de statistique en fonction du temps.

Un compromis peut se faire entre les deux solutions : avoir la moyenne des points et la moyenne du temps de réponse directement stockées et avoir aussi les données des X dernières validations opérées (X est un paramètre et nous le mettons à dix pour notre application). Ces données sont le code d'état et le temps de réponse. Grâce à ce compromis, nous pouvons avoir différentes statistiques sans pour autant mémoriser un grand nombre de données. Nous pouvons aussi stocker pour chaque URL le nombre des différents codes d'état rencontrés lors des validations.

Chapitre 4

Elaboration et construction

4.1 Elaboration

Pour l'élaboration, nous prendrons la notation UML, comme utilisé dans RUP, dont quelques diagrammes nous donnent des vues utiles à nos besoins.

4.1.1 Diagramme de robustesse

Nous construisons deux diagrammes de robustesse : un diagramme pour la validation et un diagramme pour la visualisation, afin de séparer les deux problèmes. Nous utiliserons deux contrôleurs, Validator et Visualisation, qui géreront chacun leur sous-système. Un contrôleur principal (Controller) gérera l'interaction avec le système et l'interface de l'utilisateur.

Pour la validation des liens URL, nous avons l'acteur Crontab, représentant une tâche automatique de l'administrateur, qui lance la validation via son interface. La validation a besoin de connaître les liens à valider, nous pouvons pour cela faire appel à deux contrôleurs (contrôleur QueryBuilder et DAL) qui vont se charger de prendre ces URL dans une entité (entité URL). QueryBuilder se charge de construire les requêtes et DAL s'occupe de la communication avec l'entité reprenant les données. DAL est une interface d'un sous-système permettant de dialoguer avec une base de données. Ensuite, il faut valider chaque lien, pour cela nous faisons appel à un autre contrôleur (contrôleur Network) qui va se charger d'ouvrir et fermer la connexion avec le serveur hébergeant la ressource et aussi de faire la requête sur cette ressource pour avoir le code d'état. Pour avoir le temps de réponse, nous faisons appel à un contrôleur (contrôleur Timer) qui nous donnera le temps du système en secondes avec quelques décimales. Pour notre application, une précision de l'ordre du dixième de seconde est suffisante (une précision plus grande

serait imperceptible par l'utilisateur). Pour connaître le temps de réponse, nous prendrons le temps du système juste avant et après la connexion au serveur, comprenant l'ouverte du socket, l'envoi de la requête et la fermeture du socket et de voir la différence entre ces deux temps. Nous comptons le temps d'ouverture et de fermeture pour imiter un utilisateur demandant à visualiser une ressource. Une fois le temps de réponse connu et le code d'état reçu, il nous faut calculer les nouvelles moyennes de temps et de points de l'URL. Ceci sera fait par le contrôleur Statistic. Afin d'avoir un système paramétrable, nous pouvons avoir un contrôleur (ApplicationParam) qui recherche dans une entité (en l'occurrence un fichier) les paramètres tels que par exemple le nombre d'URL à valider, le temps maximum pour une validation, etc.

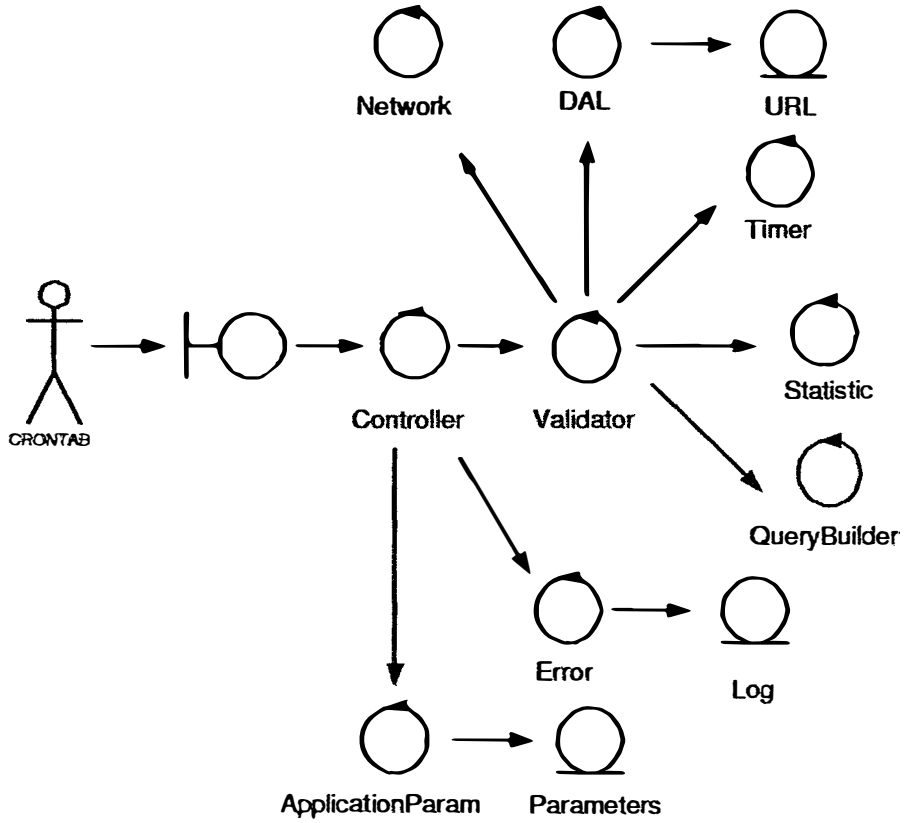


FIG. 4.1: Diagramme de robustesse de la validation

Pour la visualisation des statistiques d'un URL, c'est l'administrateur qui la lance via son interface. Le système doit afficher la liste des URL ainsi

que leurs moyennes et pour cela la visualisation s'aide du contrôleur QueryBuilder, DAL et de l'entité URL. L'administrateur pourra choisir via la liste affichée (nouvelle interface List) l'URL dont il veut voir les statistiques. Nous pouvons encore utiliser le contrôleur QueryBuilder pour faire la requête de rapatriement des données sur l'URL sélectionné. Pour afficher les historiques sous forme d'un graphique, Controller fait appel à un contrôleur (JpGraph) déjà existant qui est JpGraph 1.7[1] pour PHP 4. Les graphiques seront affichés dans une nouvelle interface (interface Graphics). Nous pouvons ajouter un contrôleur ApplicationParam afin de connaître les paramètres comme le nombre d'URL à afficher par pages lors du listing des URL.

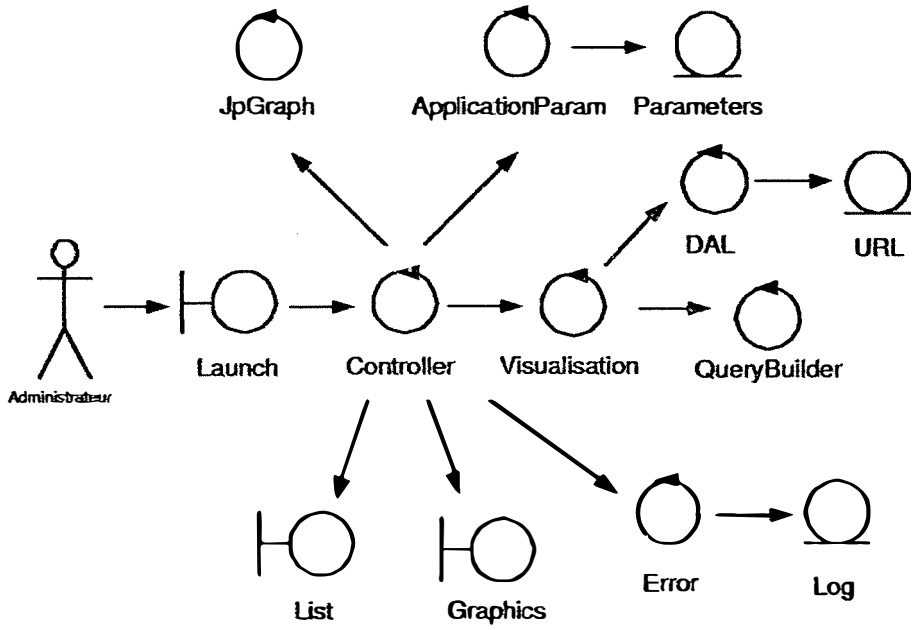


FIG. 4.2: Diagramme de robustesse de la visualisation

Dans les deux diagrammes, nous ajoutons un contrôleur Error et une entité Log afin de gérer les erreurs éventuelles et de les sauvegarder. Ceci répond aux exigences non fonctionnelles de stabilité du système et de la traçabilité des erreurs. Nous ajoutons aussi un contrôleur ApplicationParam et une entité Parameters afin de rendre paramétrable le système pour répondre aux exigences de réutilisabilité et de portabilité. Cet objet *Parameters* est représenté par un tableau (ArrayParam) dont les indices sont les mots clés, exemple :

[1] http ://www.aditus.nu/jpgraph/

arrayParam[languageSql].

Pour le fichier des paramètres de l'application, nous suivons le format établi par une majorité des fichiers ini :

- un titre reprenant un ensemble de variables est mis entre crochets ;
- une variable est suivie d'un signe égal puis suivi de la valeur ;
- il n'y a pas d'espace ;
- une variable ou un titre par ligne.

Exemple : [Validation]
numberUrlToValidate=100
tolerateResponseTime=10
[Database]
dbUsername=admin
dbPassword=admin
dbName=urldatabase

Pour le fichier contenant les erreurs survenues, le format est une ligne par erreur avec la description de celle-ci, la date et l'heure de l'apparition de l'erreur. Les erreurs possibles seront essentiellement celles d'ouverture et d'écriture des fichiers, les accès à la base de données et les accès aux serveurs distants pour la validation.

4.1.2 Les diagrammes de classes

Pour construire les diagrammes de classe, nous utilisons les design patterns et les objets dégagés par les diagrammes de robustesse. Une première découpe d'architecture se fait par le modèle MVC (Model View Controller). Ce modèle permet de diviser le système en trois catégories : le modèle, la vue et le contrôleur. La vue reprend la représentation des données aux utilisateurs, le modèle contient les données de l'application et le contrôleur défini la façon dont l'interface réagit à une entrée faite par l'utilisateur. Cette division du système diminue le nombre de points d'entrées des objets, donc diminue les dépendances, et augmente la réutilisabilité, la flexibilité et la compréhension du système.

Nous avons donc une classe Interface reprenant les différentes vues, une classe Controller pour gérer les interactions et le reste pour le modèle (voir annexe II). Comme nous pouvons déjà le voir dans les diagrammes de robustesse pour les contrôleurs Validator et Visualisation, le design pattern *Facade*

peut s'y appliquer car ces deux contrôleurs représentent un point d'entrée d'un sous-système. Validator va permettre de lancer le processus de validation en faisant appel à différents composants encapsulés dans le sous-système. Il en va de même avec la visualisation. Les composants ApplicationParam, Error, Querybuilder et DAL sont identiques pour le système de validation et le système de visualisation.

Le Design Pattern *facade* permet de réduire la complexité d'un système en le décomposant en sous-systèmes [8]. Le point d'entrée d'un sous-système est appelé une façade. Les points d'entrées diminuent la communication et les dépendances entre les sous-systèmes. Cette division en sous-systèmes peut permettre une réutilisation d'un sous-système et une modification plus facile.

Un objet URL reprenant ses attributs est représenté par une instanciation de DAL, exemple : *mydal = new dal;*. L'objet mydal contient entre autre le résultat de la dernière requête et donc peut représenter une agrégation d'objets URL lorsque plusieurs lignes de la table sont affectées.

Voici une description des classes avec leurs méthodes :

Validator : Cette classe s'occupe de la coordination de la validation des URL. La méthode *validate()* instancie les classes Timer, Network, Statistic, QueryBuilder et DAL et appellera les différentes méthodes afin gérer la validation. La méthode *loadDal* crée un objet DAL avec la configuration pour pouvoir l'utiliser.

Timer : Cette classe s'occupe du temps du système pour permettre, dans notre cas, de mesurer le temps de réponse des requêtes GET ou HEAD. La méthode *readTime* prend le temps du système en secondes avec quelques décimales depuis le 1er janvier 1970. Nous pouvons laisser le temps mesuré dans ce format car nous ne nous occupons que de la différence entre deux temps. Ces deux-ci sont le temps pris juste avant l'ouverture de la connexion sur le serveur hébergeant la ressource et le temps juste après la fermeture de cette connexion avec entre-temps la requête GET ou HEAD.

Network : Cette classe s'occupe de la communication TCP/IP entre notre système et le serveur hébergeant la ressource sélectionnée afin d'obtenir son code d'état. La méthode *openSocket* ouvre une connexion avec le serveur et renvoie cette connexion (socket). La méthode *closeSocket* ferme la connexion avec le serveur. La méthode *getStatusCode* émet la

requête pour obtenir le code d'état de la ressource. Pour le protocole FTP, il faut monter jusqu'au bon répertoire puis de vérifier si le fichier existe. Le cahier des charges nous limite aux protocoles HTTP et FTP.

Statistic : Cette classe s'occupe de mettre à jour les valeurs d'un URL après sa validation. La méthode *calculateNewData* calcule le point obtenu en fonction du code d'état reçu et du temps de réponse en s'aidant dans la méthode *calculateCoeff*, met à jour le nombre de code d'état 200 ou 404 reçu, le nombre de validations effectuées, les moyennes des points obtenus et du temps réponse. La méthode *calculateCoeff* calcule le coefficient à appliquer en fonction du temps de réponse : plus temps de réponse est long, plus le coefficient diminue (Voir le graphique du coefficient multiplicateur : Fig 3.1).

ApplicationParam : Cette classe s'occupe de connaître quels sont les paramètres à appliquer dans le souci de faire un système paramétrable et à partir de quel URL il faut commencer la validation. La méthode *getParameters* ouvre le fichier contenant les paramètres, parcourt le fichier, met les paramètres dans un tableau et referme le fichier. La méthode *getLastIdUrlValidated* ouvre le fichier contenant les paramètres, indique quel est le dernier URL validé lors de cette validation et referme le fichier.

Error : Cette classe s'occupe de gérer les erreurs qui peuvent éventuellement survenir. La méthode *displayErrorMessage* fait une demande de notification d'une erreur en fonction du type d'erreur reçu et fait une demande de stockage de l'erreur grâce à la méthode *writeError*. La notification se fait par une demande d'affichage à l'écran.

QueryBuilder : Cette classe s'occupe de construire les requêtes SQL pour la base de données. Le cahier des charges nous limite à la base de données à MySQL.

DAL : Cette classe s'occupe de la communication avec la base de données et d'y soumettre les requêtes. Cette classe est déjà existante[2]. Avant de soumettre une requête, il faut sélectionner le langage SQL avec la méthode *selectDriver()*, DAL permet de choisir entre MySQL, PostGres et Msql. Ensuite il faut choisir le nom d'utilisateur et le mot de passe par

[2] http ://dal.sourceforge.net

l'intermédiaire des configurations : *selectConfiguration()*. Il nous reste à sélectionner la base de données concernées par les futures requêtes : *selectDatabase()*.

JpGraph : Cette classe[3] s'occupe de générer des graphiques pour la visualisation des statistiques. Nous donnons les différentes valeurs (ordonnées) et *JpGraph* construit le graphique et le sauvegarde sous forme de fichier png. Lors de l'affichage des statistiques d'un URL, nous demanderons d'afficher cette image.

[3] http ://www.aditus.nu/jpgraph/

4.1.3 Les diagrammes de séquences

Pour construire un diagramme de séquence, nous partons des scénarios principaux indiqués dans les Use Cases (voir annexe I). Les objets sont les différentes classes générées par le diagramme de classe.

Pour le diagramme de séquence de la validation et de la visualisation, nous n'avons pas mis la gestion des erreurs afin de ne pas surcharger le diagramme. Les erreurs peuvent arriver dans les classes Network, ApplicationParam et DAL. Nous voyons aussi que les boucles ne sont pas représentées. Une boucle possible est celle pour la validation de n URL; cette boucle débute, pour chaque URL par l'appel à la méthode *readTime* pour la capture du temps avant l'ouverture d'un socket jusqu'à la demande de la mise à jour de cet URL dans la base de données (voir le diagramme de séquence). Les flèches en trait plein sont des appels de méthodes et celles en trait discontinu sont les retours des méthodes rapportant les informations.

4.1.4 La base de données

Présentation

Comme nous avons vu dans le point *Stockage des données pour les statistiques*, il faut enregistrer certaines données appartenant aux URL :

- l'adresse de la ressource;
- la moyenne des points obtenus;
- la moyenne du temps de réponse;
- le nombre de fois que l'URL a été validé;
- le nombre de codes d'état 200 et 404 reçus;
- l'historique des dix dernières validations.

L'historique reprend le code d'état, le temps de réponse, la date de la validation et le point obtenu. Comme nous voulons que le système soit réutilisable et donc dans un certain sens paramétrable pour pouvoir s'adapter à des exigences autres, il nous faut pouvoir changer le nombre d'historiques sans pour autant modifier la table, c'est à dire que nous ne pouvons pas avoir les historiques sous forme d'attribut (sinon il faudrait ajouter ou supprimer des attributs à chaque modification du nombre d'historiques).

Une solution pour résoudre ce problème consiste à placer les historiques dans une autre table à six attributs : le premier contenant l'identifiant technique, le deuxième pour un nombre identifiant de la ressource dans l'autre

table pour savoir à quel URL l'historique appartient, le troisième pour le code d'état, le quatrième pour le temps de réponse, le cinquième pour la date de validation et le sixième pour le point obtenu lors de cette validation. Les historiques sont représentés en ligne et ainsi donc il y aura au maximum dix occurrences pour une référence d'un même URL. Une représentation de la base de données se trouve en annexe IV. Si nous voulons modifier le nombre d'historiques à mémoriser, il suffit de modifier le nombre acceptable d'occurrences dans cette table. Grâce à la date de la validation ou à l'identifiant, nous pouvons supprimer les validations les plus anciennes. Le nombre d'historiques à gérer se trouve dans le fichier des paramètres de l'application.

A chaque fin de validation d'un lien URL lors de l'insertion dans la table des historiques, nous devons vérifier si le nombre d'occurrences n'est pas supérieur au nombre voulu. Si c'est le cas, nous effaçons l'historique le plus ancien.

Détails des requêtes

Les requêtes suivantes permettent de créer les tables *url* contenant les données principales des URL et des historiques :

```
CREATE TABLE url (id_url INT(10) not null AUTO_INCREMENT
PRIMARY KEY,url CHAR(100) not null,averageUrlPoint FLOAT
not null, averageUrlResponseTime FLOAT not null, numberUrl-
Validation int(10) not null, numberOfCode200 INT(10) not null,
numberOfCode404 INT(10) not null)
```

```
CREATE TABLE historic (id_valid INT(10) not null
AUTO_INCREMENT PRIMARY KEY, id_url INT(10) not null,
statusCode INT(3) not null, responseTime FLOAT not null,dateValid
DATE not null,point FLOAT)
```

Pour la visualisation, il nous faut une requête qui puisse nous lister les URL ainsi que leurs données principales[4] par ordre croissant de la moyenne des points et ce en affichant par tranche de n URL par page[5]. Pour pouvoir afficher N URL par page, nous pouvons prendre un système de numéro de pages pour connaître la tranche des URL à afficher. Si par exemple nous

[4]Les données principales reprennent les valeurs contenues dans la table *url*.

[5]n est un paramètre contenu dans le fichier des paramètres de l'application.

voulons trente URL par page pour l'affichage et nous voulons afficher la troisième page, la requête

*SELECT * FROM url ORDER BY averageUrlPoint LIMIT 60,30*

donnera le résultat voulu. *url* est la table contenant les données, *ORDER BY averageUrlPoint* permet d'avoir les résultats par ordre croissant de la moyenne des points des URL et *LIMIT 60,30* permet de n'avoir que les trente premiers résultats mais à partir du soixantième de la liste ; soixante a été trouvé en faisant la page voulue moins un puis multipliée par le nombre de résultats voulus par page : (3-1)*30. Après que l'administrateur eût choisi la bonne page et sélectionné l'URL voulu, il faut prendre l'historique de l'URL contenu dans la table *historic*. Nous connaissons donc *id_url* de l'URL sélectionné, ainsi nous avons la requête suivante :

*SELECT * FROM historic WHERE id_url = idurl ORDER BY dateValid*

Le ORDER BY dateValid permettra d'obtenir les résultats dans un ordre croissant de dates afin de faciliter la mise en graphique. Nous affichons aussi les données principales de l'URL. Pour cela, nous préférons utiliser une requête pour reprendre les données principales plutôt que d'utiliser des variables de sessions entre les deux interfaces (cela augmente la complexité de l'implémentation.

Pour la validation, il faut pouvoir prendre les informations[6] des N URL à valider. Le fichier des paramètres de l'application contient le numéro du dernier url validé de la dernière validation : *idLastValidateUrl*. Prenons une variable idToStart =idLastValidateUrl + 1 pour la requête suivante avec n étant le nombre d'URL à valider pour cette validation :

*SELECT * FROM url WHERE id_url >= idToStart ORDER BY id_url LIMIT n*

Il ne faut pas oublier que si nous sommes arrivés à la fin des URL donc au maximum du numéro identifiant des URL (*id_url*), il faut revenir au début pour continuer la validation. Une requête pour connaître le numéro maximum est donc souhaitable pour savoir jusqu'où nous pouvons aller : *SELECT max(id_url) FROM url*

Si nous devons repartir au début de la table pour la validation, idToStart sera mis à zéro avant de lancer la requête pour prendre les N URL.

Une fois qu'un url a été validé, il faut mettre à jour la table *url* avec les nouvelles moyennes calculées, le nombre de validations effectuées et le

[6] Les informations contenues dans la table url suffisent.

nombre de code d'état reçus (et éventuellement le nombre de 200 ou 404 reçus). La requête suivante effectue la mise à jour :

UPDATE url SET averageUrlPoint = newAverageUrlPoint, averageUrlResponseTime = newAverageUrlResponseTime, numberUrlValidation = newNumberUrlValidation, numberOfCode200 = newNumberOfCode200, numberOfCode404 = newNumberOfCode404 WHERE id_url = idurl.

Ensuite, il faut ajouter dans la table *historic* les résultats de cette validation de l'URL. La requête suivante effectue cet ajout :

INSERT INTO historic ('id_valid','id_url','statusCode', 'responseTime','dateValid','point') VALUES ('','id_url','statusCode', 'responseTime','date','point').

Nous pouvons remarquer que nous n'avons pas mis de valeur pour le champ *id_valid* car ce champ est une incrémentation automatique induite par MySQL lors de la création de cette table[7].

Il faut vérifier après l'insertion dans la table *historic* que le nombre d'historiques pour cet URL ne soit pas plus grand que le nombre indiqué dans le fichier des paramètres de l'application. Le nombre d'historiques pour cet URL correspond au nombre d'occurrences dans la table ayant le même *id_url*. La requête suivante permet de recevoir la liste avec les données pour un même *id_url* :

*SELECT * FROM historic WHERE id_url = var1 ORDER BY dateValid .*

Le *ORDER BY date* permet de présenter les résultats par ordre croissant. En sachant le nombre d'occurrences grâce à la requête précédente et le nombre d'occurrences souhaité, il suffira s'il y a lieu de supprimer les lignes en trop en reprenant par le début. La requête suivante permet d'effacer les lignes dont la date de validation est inférieure à une date qui correspond à la Xième validation avec X étant le nombre d'historiques à mémoriser :

DELETE FROM historic WHERE id_url = var1 AND dateValid < var2.

Exemple : dans la validation précédente, un historique des dix dernières validations était indiqué. Dans cette validation, nous avons modifié ce nombre pour le mettre à cinq. Donc nous aurons à la fin d'une validation d'un URL, un ajout dans l'historique qui portera le nombre d'occurrences à onze. Nous faisons la différence entre le nombre d'occurrences avec le nombre souhaité :

[7]Voir la requête de la création de la table

$11 - 5 = 6$. Nous constatons que nous avons six occurrences en trop, donc à supprimer. Nous prenons la septième réponse de la requête précédente (réponse triée par date), nous prenons sa date et nous pouvons alors effectuer la suppression de toutes les validations dont la date est inférieure à celle-ci . Il restera donc cinq occurrences comme souhaités.

Une deuxième suppression peut être envisagée mais cette fois en fonction de l'identifiant de la table *historic*. Cette suppression permet d'effacer les historiques en trop non supprimés par la suppression précédente (au cas si un même URL a été validé plusieurs fois à la même date). En effet, la suppression par date supprime les historiques dont la date est strictement inférieure à la Xième et donc ne supprimera pas tous les historiques supplémentaires. Il faut donc prendre tous les historiques pour un même URL mais classés par id_valid, prendre le dernier id_valid correspondant au dernier historique à mémoriser puis supprimer les historiques de l'URL dont les identifiants sont inférieurs. Dans ce cas, nous faisons l'hypothèse que l'incrémentation lors des ajouts est monotone croissant. Cette hypothèse est vérifiée dans la plupart des cas mais en est-il ainsi lorsque l'identifiant arrive à son maximum et qu'il y a eu des suppressions laissant des identifiants non utilisés ?

Nous avons donc comme requête de sélection avec var1 pour l'URL sélectionné :

*SELECT * FROM historic WHERE id_url = var1 ORDER BY id_valid .*

Puis comme suppression avec var2 (identifiant du dernier historique à mémoriser) :

DELETE FROM historic WHERE id_url = var1 AND id_valid < var2.

4.1.5 DAL

Database Abstraction Layer[8] permet d'avoir accès à la base de données de façon transparente, il suffit d' indiquer la location de la base de données, quel est son type, sous quel nom nous voulons nous connecter ainsi que le mot de passe. Ces paramètres se trouvent dans le fichier de configuration de DAL. DAL s'occupe de créer les connexions avec la base de données ,il nous suffit simplement de donner les requêtes et DAL nous renvoie les résultats.

[8] http ://dal.sourceforge.net

Nous avons besoin des méthodes *selectDriver(string driverName)*, *selectDatabase(string database)* et *executeQuery(string query)* pour utiliser la base de données.

4.2 Implémentation des modules

4.2.1 ApplicationParam

Avec la méthode *getParameters*, nous parcourons le fichier pour mettre les paramètres de l'application sous forme de tableau. Le mot à gauche du signe égal représente le nom de la variable et est une clé du tableau. Le mot ou le nombre à droite du signe égal est la valeur de la variable. La méthode *setLastIdValidated* appelle la méthode *getParameters* pour mémoriser les paramètres, change la valeur du dernier URL validé et ensuite reconstruit entièrement le contenu du fichier.

4.2.2 Controller

La méthode *main* permet de sélectionner le type de service : une validation, un affichage de la liste de N URL ou un affichage des historiques d'un URL. Les méthodes *drawList* et *drawGraphic* permettent de construire les pages HTML pour les visualisations.

4.2.3 Error

La méthode *displayErrorMessage* permet de construire le message d'erreur en fonction du type d'erreur et gère le message. La méthode *writeError* permet d'écrire l'erreur dans un fichier en indiquant la date et l'heure de l'apparition de l'erreur.

4.2.4 JpGraph

La méthode *constructGraph* construit un graphique en partant des valeurs des points des historiques. Le graphique est sauvegardé dans le fichier image.png dans le répertoire 'jpgraph_cache' car retourner le graphique et l'afficher à partir de la classe Controller engendre des erreurs dans le flux de données (voir www.aditus.nu/jpgraph).

4.2.5 Network

La méthode *openSocket* ouvre des connexions sur les serveurs hébergeant les ressources. Il nous faut remplacer les espaces par le caractère %20 (méthode *removeSpace*). Pour le protocole FTP, il faut prendre le nom d'utilisateur et le mot de passe pour s'authentifier auprès du serveur ; les deux noms se trouvent dans le string host. La méthode *closeSocket* ferme la connexion en fonction du type de celle-ci : HTTP ou FTP. *getStatusCode* émet la requête GET pour une connexion HTTP et recueille le code d'état émanant du serveur. Nous utilisons la requête GET car certains serveurs n'acceptent pas de recevoir une requête HEAD et renvoient un code d'état de la famille des 300 ou 400. La requête GET télécharge la ressource entièrement mais dans notre implémentation, nous arrêtons ce téléchargement en fermant la connexion dès que nous avons obtenu le code d'état. Pour une connexion FTP, nous utilisons la méthode de PHP pour arriver dans le répertoire contenant la ressource.Ensuite, si la ressource est le répertoire lui-même ,il faut renvoyer le code d'état 200. Sinon il faut vérifier si la ressource est dans ce répertoire ; pour cela nous listons le répertoire. Si le fichier apparaît dans la liste ,nous mettons le code d'état à 200, sinon le fichier n'existe pas et le code d'état est mis à 404. Le serveur distant peut donner le chemin complet dans la liste des fichiers contenus dans le répertoire ou les noms des fichiers uniquement. La recherche portera donc sur le nom (simple) ou sur le nom avec le chemin complet.

4.2.6 QueryBuilder

Toutes les méthodes renvoient un string contenant la requête appropriée avec les valeurs souhaitées.

4.2.7 Statistic

La méthode *calculateNewData* calcule les nouvelles données de l'URL et renvoie le nouvel objet URL sous forme de tableau.

La méthode *calculateCoeff* calcule le coefficient à appliquer au point en fonction du temps de réponse et des tolérances renseignées dans le tableau des paramètres de l'application.

4.2.8 Timer

La méthode *readTime* lit l'horloge du système en seconde à partir du premier janvier 1970 avec quelques décimales. Nous laissons le temps dans

ce format car nous faisons une différence de temps par après.

4.2.9 Validator

La méthode *validate* gère la validation des URL. Elle reçoit le tableau des paramètres de l'application pour effectuer la validation. Nous prenons le temps du système au début de la validation afin de s'arrêter après le temps maximum de validation indiqué. Nous commençons d'abord à charger le système DAL afin de pouvoir dialoguer avec la base de données. Pour les liens à valider, nous commençons à la suite du dernier URL validé en prenant les n URL suivants. Si nous sommes arrivés à la fin de la table des URL, il faut recommencer à partir du début, la méthode *getMaxId* sert à cela.

Pour chaque URL à valider, nous remettons en forme le lien en changeant les backslash en slash, nous séparons le lien de la forme http ://www.aaa.com/bbb/ccc.pdf pour avoir le protocole (http), l'hôte (www.yyy.com) et le chemin (/bbb/ccc.pdf). Puis nous ouvrons une connexion sur le serveur hébergeant la ressource, nous soumettons la requête pour avoir le code d'état et nous refermons la connexion. Un deuxième test pour recueillir le code d'état est lancé dans le cas où le code d'état du premier test est différent de 200 ou 404. Dans ce deuxième test, nous ajoutons au chemin le caractère slash (/) car si la ressource renseigne un dossier et non un fichier et qu'il manque le slash à la fin, le premier test nous indiquera un code d'état de la famille des 300. Ce deuxième test sert donc à corriger la syntaxe du lien dans le cas ou celui-ci est un répertoire.

Une première possibilité a été soulevée en analysant la dernière partie du chemin : si cette partie contient un caractère point (.), celle-ci renseigne alors un fichier avec son extension. Malheureusement, un répertoire peut contenir des points dans son nom, comme par exemple /home/fichier.pdf/. Les valeurs du deuxième test, s'il a lieu, seront les valeurs mémorisées. Une fois le code d'état et le temps de réponse reçus, nous pouvons calculer les nouvelles valeurs de l'URL avec l'appel de la méthode *calculateNewData*.

Nous mettons ensuite à jour ces nouvelles valeurs dans la table et nous ajoutons dans l'historique les valeurs de cette validation. Par après, nous devrons effacer les historiques excédants. Cette technique est expliquée dans le point "La base de données". A la fin d'une validation d'un URL, nous prenons le temps-système et nous le comparons avec le temps de départ incrémenté du temps de validation. Si le temps est dépassé alors il faut arrêter la validation, sinon nous pouvons continuer la validation avec le prochain lien.

4.2.10 Visualisation

La méthode *visualize* lance en fonction du type de service soit l'affichage de la liste des URL, soit l'affichage des historiques. Pour l'affichage de la liste, la méthode renvoie un objet DAL contenant la liste des URL à afficher. Pour l'affichage des historiques, la méthode renvoie un tableau contenant trois objets DAL. Le premier présente les données principales de l'URL sélectionné, le deuxième et le troisième sont identiques et contiennent les historiques de l'URL. Le deuxième objet sera utilisé pour acquérir les points destinés à construire le graphique et le troisième pour afficher les données des historiques.

4.3 Intégration et tests : choix des tests

Pour produire les jeux de tests d'acceptation, nous avons utilisé les cas d'utilisation, le système devant répondre aux attentes des utilisateurs. Les exigences non fonctionnelles font bien sûr partie des exigences.

Afin de faciliter les tests, nous avons développé une interface graphique permettant d'une part de lancer une validation et d'autre part une visualisation. Cette interface ne fait pas partie des exigences car le système que nous avons développé est censé être intégré dans un système plus large fournissant les interfaces nécessaires. Une première fenêtre nous donne le choix entre le lancement d'une validation et le lancement de la visualisation.

Pour le choix de la visualisation des historiques d'une ressource, nous avons une liste des liens avec le nombre voulu de liens à afficher mentionné dans le fichier des paramètres de l'application. Cette liste est triée dans l'ordre croissant de la moyenne des points obtenus. Lorsque nous cliquons sur l'URL choisi, une nouvelle fenêtre apparaît avec les historiques des X dernières validations ainsi et le graphique associé. Si nous modifions le nombre d'historiques à mémoriser et que nous reprenons une visualisation d'historiques, cette visualisation porte bien sur le nombre d'historiques demandé.

Pour valider le test de validation de liens, nous lançons une validation et nous affichons chaque URL pris avec leurs données principales après la mise à jour ainsi que le code d'état et le temps de réponse. Nous constatons que les liens existant reçoivent un code d'état 200 dans un temps très court (quelques dixièmes de secondes). Différents types de liens ont été insérés dans la base de données : lien vers un fichier et un répertoire avec le protocole http ou FTP, lien vers un site HTTP ou FTP sans chemin, lien vers des gros fichiers.

Plusieurs exigences non-fonctionnelles du cahier des charges interviennent dans notre sous-système de validation : *Efficiency*, *Maintainability*, *Portability* et *Reusability*.

4.3.1 Efficiency

Pour cette exigence non-fonctionnelle, la métrique du temps de réponse d'une tâche (temps entre la soumission de l'action et la réponse, exemple : lancer la visualisation des historiques et l'apparition de la page correspondante), ne doit pas dépasser le temps précisé dans le point 3.5.3.1 du cahier

des charges. Ce point reprend les temps de réponse moyens et maximum pour que le système réponde à une tâche déterminée. Le temps d'affichage de la liste ou des historiques doit avoir une moyenne de 0.8 seconde pour une connexion à haut débit et 2.4 secondes pour une connexion par modem. Le temps d'affichage maximum ne doit pas dépasser 3.3 secondes pour une connexion à haut débit et 7.7 secondes pour le modem.

Première constatation, la taille du code affiché pour la visualisation des historiques des dix dernières validations ne dépasse pas les six kilo-octets avec le graphique. Donc le téléchargement pose peu de problème de temps dans le cas d'une connexion par modem et encore moins avec une connexion à haut débit. Il faut ajouter à ce temps de téléchargement le temps de création de cette page : les différents accès à la base de données, création du graphique, temps de calcul, etc. Le plus simple pour mesurer le temps global d'une visualisation des historiques est de se placer comme un administrateur à distance avec une connexion à haut débit (une demi seconde à une seconde dans le temps total peut différer par rapport à une connexion par modem puisque la taille est inférieure à six kilo-octets) ;à distance car une des exigences est qu'un administrateur peut faire cette visualisation de chez lui s'il le désire. Nous chronométrons le temps entre le lancement de la visualisation des historiques et l'affichage de la page. Le test de la visualisation des historiques se fera en lançant dix fois à la suite une visualisation quelconque d'un URL et en répétant ceci à des moments différents de la journée.

Les tests ont été effectués le jeudi 29 août 2002 à 10H00, 14H00 et 18H00. Le tableau reprend les données avec les moyennes calculées.

Nous constatons que les trois moyennes sont équivalentes et sont légèrement supérieures à la moyenne préconisée (1 seconde pour une connexion à haut débit) dans l'exigence non-fonctionnelle de *Efficiency* (voir annexe V). Mais les moyennes sont bien en-dessous du maximum acceptable (3 secondes). Pour la visualisation de la liste des URL, nous avons pris le temps d'affichage à chaque début des tests et nous avons une moyenne de 1.11 seconde. Nous pouvons alors admettre que l'exigence est satisfaite.

4.3.2 Maintainability

Nous avons pour cette exigence deux métriques (s'appliquant à notre système) à respecter : *Analysability* et *Modifiability* (voir annexe V).

Mesures à 10H	Mesures à 14H	Mesures à 18H
2.08	1.27	1.28
1.10	1.80	1.73
1.25	1.25	1.38
1.17	1.18	1.33
1.08	1.12	1.24
1.14	1.17	1.27
1.04	1.88	1.15
1.11	1.34	1.24
1.06	1.16	1.36
1.06	1.13	1.22
1.209	1.33	1.32

FIG. 4.3: Tableau de mesures

Analysability

Pour le critère de capacité d'analyse d'une panne, la classe Error permet d'archiver les erreurs survenues dans le fichier Log.txt indiquant le moment de l'erreur et une description pour remédier à la panne. Ce fichier nous a servi à identifier les erreurs lors des tests de compilation.

Modifiability

Pour le critère de modifiabilité du logiciel, la découpe en classe et son architecture avec les Design Patterns permettent de dissocier les services différents du système et de comprendre les relations existant entre ces classes. Chaque classe reprend une spécificité et permet donc une plus grande facilité de modifications aux bons endroits ; grâce aux fonctions et leurs spécifications, il est aisé de comprendre le raisonnement des algorithmes et donc de les modifier ou de les compléter.

4.3.3 Portability

Adaptability

Le système est adaptable sur différents systèmes d'exploitation à condition que ceux-ci disposent d'un serveur Apache et de l'interpréteur PHP ainsi qu'un serveur SQL pour gérer la base de données. Le système a été développé et testé sous Windows 98 deuxième édition, Windows NT4 et Linux. Le système accessible par le WEB : www.cetic.be/jdubus/lancement.html.
Il n'y a aucune contrainte sur le type de browser utilisé pour une visualisation des historiques.

4.3.4 Reusability

Le mainteneur peut facilement modifier les paramètres de l'application grâce au fichier *ApplicationParam.ini* afin de permettre une réutilisation du logiciel.
Plusieurs tests ont été effectués sur les changements de paramètres et leurs impacts sur le logiciel. Le nombre de liens validés est exactement le nombre mentionné dans le fichier des paramètres ; le nombre d'historiques ne dépasse jamais le nombre mentionné, le nombre de liens affichés dans le listing correspond bien au nombre mentionné sauf bien sûr la dernière page dans laquelle le reste des liens est affiché,enfin le coefficient à attribuer au point obtenu est bien fonction des seuils de temps de réponse (urlTimeout et tolerateResponseTime).

De plus, des nouveaux paramètres peuvent être ajoutés dans ce fichier (extension future du logiciel). Si ces nouveaux paramètres suivent la syntaxe du fichier, ceux-ci pourront être accessibles dans le programme par le tableau des paramètres généré lors de l'appel de la fonction *getParameters* et ceci sans aucun changement dans le programme. De plus, comme le code PHP est interprété, après une modification d'un ou plusieurs paramètres, le code est directement exécutable augmentant ainsi l'exploitabilité du système.

Chapitre 5

Conclusion

Le sous-système de validation de liens URL et de visualisation des historiques répond bien aux différentes exigences demandées dans le cahier des charges. Ces exigences sont développées dans les Use Cases et ceux-ci servent par après de tests d'acceptation à la fin du développement du produit afin de le valider.
Le sous-système peut se manipuler comme un module pouvant s'adapter à d'autres exigences grâce d'une part à son caractère paramétrable représenté par le fichier *ApplicationParam.ini*. D'autre part sa découpe en composants permet à des modifications ou à des extensions de se faire sans grande difficulté.

La méthode de développement du Unified Process est itérative et incrémentale. L'itération consiste à décomposer le système en plusieurs sous-systèmes et l'incrémentation consiste à ajouter les différents sous-systèmes afin de former le système global.

Le modèle itératif permet de développer un sous-système et de voir les problèmes plus tôt et que nous pourrions rencontrer dans les autres sous-systèmes. Le cahier des charges peut être modifié pour remédier à ces problèmes, toujours en accord avec les exigences fonctionnelles et non-fonctionnelles du logiciel. Ces itérations permettent de diminuer les risques liés à la taille du projet ou encore à sa complexité.

La méthode de développement du Unified Process utilise UML et les Design Patterns. UML est langage de modélisation assez répandu et utilisé par un grand nombre de développeur. Les Design Patterns permettent de structurer l'architecture de certaines parties du système en se basant sur l'expérience de nombreux projets antérieurs.

Dans notre cas, c'est le Design Pattern *facade* que nous avons utilisé pour rassembler les sous-systèmes et d'avoir une interface pour chaque sous-système. Ainsi nous avons un point d'entrée aux différents sous-systèmes tels que celui de la validation, celui de la visualisation et celui du DAL. Cette décompostion permet une vue plus claire et une meilleure compréhension du logiciel, ce qui sous-entend des facilités en cas de modifications ultérieures. Chaque Design Pattern a sa spéficité d'utilisation donnant une architecture fréquemment utilisée dans le cas recherché.

La méthode de développement du Unified Process s'applique normalement sur des gros projets. Cette méthode gère aussi bien le développement du logiciel que le planning des différents processus et des ressources ainsi que leurs budgets. Il est certain que pour notre sous-système de validation de liens URL, nous n'avons pas utilisé la gestion des budgets et peu utilisé la gestion des ressources.

Nous avons utilisé dans le langage UML quelques diagrammes nous apparaissant utilent pour concevoir le logiciel : le diagramme des cas d'utilisation, le diagramme de classes et le diagramme de séquence. Ces trois diagrammes suffisent pour développer un sous-système car ils reprennent les exigences du logiciel, la découpe en classe et la chronologie des interactions entre ces classes par les appels de méthodes. Les autres diagrammes ne produisaient pas d'informations inconnues primordiales pour le développement de notre sous-système.

Le diagramme de robustesse nous aide à dégager les différents objets que nous aurons besoin et à modéliser le système par un haut niveau d'abstraction pour mieux le comprendre grâce aux quatre types d'objet : les acteurs, les interfaces, les contrôleurs et les entités. Il est déjà possible à ce stade d'intégrer les Design Patterns afin de structurer le diagramme de classe qui en découle.

Un des points où il faudra apporter de l'attention lors de l'intégration du système de validation au Resources Manager ou à un autre système est celui de la base de données. En effet, la base de données contient les liens à valider qui doivent venir, dans notre cas, de la validation des ressources dans le système global par un administrateur. Ces ressources sont proposées par les fournisseurs de contenu.

Futures works.

Des fonctionnalités peuvent être apportées à notre système de validation

telle que dans la notification, dans les protocoles utilisés ou un notificateur lorsque une moyenne des points d'un URL passent en-dessous d'un seuil. Dans l'état actuel, la notification d'un problème se fait par un fichier Log.txt reprenant le type d'erreur et le moment à laquelle l'erreur s'est produite. Nous pouvons envisager d'autres types de notification telles que la notification par mail à l'administrateur ou encore par SMS (Short Message Service) en indiquant l'erreur afin d'avertir rapidement le ou les administrateurs.

Nous pouvons aussi ajouter des protocoles pouvant accueillir des ressources telles que des articles venant de newsgroup, un lien 'Mailto', Telnet. Pour cela, les modifications se feront dans la classe *Network* tout en respectant les spécifications des méthodes afin de rester compatible avec le reste du système.

L'insertion d'un notificateur qui avertit un administrateur lorsque qu'un seuil franchi. Des seuils comme la moyenne des points, sur un écart-type de la moyenne des points, sur une fréquence de mauvais points.

GLOSSAIRE

Architecture [12] : structuration du système à développer en un ensemble de composantes ayant entre elles des relations bien définies possédant certaines propriétés requises et respectant certaines contraintes.

Artefact [10] : Partie d'information utilisée ou produite lors du processus de développement d'un logiciel. L'artefact peut prendre la forme d'un modèle d'architecture, d'une description, d'une documentation, etc.

Design Pattern [10] : pattern architectural définissant une structure ou un comprtement spécifique. Solution courante à un problème courant dans un contexte donné.

Exigence fonctionnelle [10] : exigence spécifiant une action qu'un système doit être capable d'effectuer. Elle dérive directement des demandes des utilisateurs ou d'une norme, d'une spécification ou tout autre document.

Exigence non fonctionnelle [10] : exigence spécifiant des propriétés du systèmes, telles que les contraintes liées à l'environnement et à l'implémentation, les exigences en matière de performance, de dépendances de plate-forme, de facilité de mainetance, d'extensibilité et de fiabilité.

Itération [12] : ensemble d'activités menées en fonction d'un plan de l'itération. Ceci peut être un mini-projet ou l'application d'un cycle de phases à l'intérieur d'un projet ou d'un mini-projet.

mini-projet complet [12] : cela commence par la planification et le définition du cahier des charges et se termine par la livraison (interne ou externe) d'une version du produit.

Pattern [14] : description nommée d'un problème et d'une solution qui indique quand et comment appliquer la solution à de nouveaux contextes.

Phase [10] : espace de temps séparant deux jalons majeurs d'un processus de développement, durant lequel on réalise un ensemble d'objectifs.

Qualité [1] : aptitude à être en conformité avec les exigences établies dans le cahier des charges.

UML [14] : (Unified Modeling Language) langage de modélisation standard pour le logiciel : langage de visualisation, de spécification, de construc-

tion et de documentation des artefacts d'un système à forte composante logicielle.

Bibliographie

[1] Software engineering-product quality - part 1 : Quality model. ISO/IEC TR 9126-1.

[2] Software engineering-product quality - part 2 : External metrics. ISO/IEC TR 9126-2.

[3] Software engineering-product quality - part 3 : Internal metrics. ISO/IEC TR 9126-3.

[4] Software engineering-product quality - part 4 : Quality in use metrics. ISO/IEC TR 9126-4.

[5] Omg unified modeling language specification. version 1.3. Rational Software, June 1999.

[6] C. ALEXANDER, S. I. A Pattern Language. Oxford University Press, New York, 1977.

[7] COCKBURN, A. Writing Effective Use Cases. Addison-Wesley, 2000.

[8] E. GAMMA, R. HELM, R. J., AND VLISSIDES, J. Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston, 1994.

[9] FOWLER, M. Analysus Patterns : Reusable Object Models. Addison-Wesley, 1997.

[10] I. JACOBSON, G. B., AND RUMBAUGH, J. Introduction au Rational Unified Process. Eyrolles, 2000.

[11] JONES, C. Patterns of software Systems Failure and Success. International Thomson Computer Press, London, 1996.

[12] KRUCHTEN, P. Introduction au Rational Unified Process. Eyrolles, 2000.

[13] LARMAN, C. Tutorial mastering design patterns, 2002.

[14] LARMAN, C. UML et les Design Patterns. CampusPress, 2002.

[15] LILLY, S. How to avoid use-case pitfalls. http ://www.sdmagazine.com, 2000.

[16] ROSENBERG, D., AND SCOTT, K. Driving design with use cases. http ://www.sdmagazine.com, 2000.

[17] SCOTT, K., AND ROSENBERG, D. Successful robustness analysis. http ://www.sdmagazine.com, 2002.

[18] YOURDON, E. Death March : Managing 'Mission Impossible' Projects. Prentice Hall, 1997.

Table des annexes

Spécification de Cas d'Utilisation : Valider les disponibilités des URL.

1. Nom du Cas d'Utilisation

Valider les disponibilités des URL.

1.1 Brève Description

Le système lance le validateur d'URL (Unified Resource Locator) portant sur un nombre de liens (paramètre). Ce lancement se fait à une certaine heure (TBD) afin de ne pas encombrer la bande passante pendant les heures d'ouverture des bureaux. Pour chaque URL, il faut enregistrer l'état de la réponse (code d'état du HTTP) et le temps de réponse afin de faire des statistiques sur les liens.

2. Flux d'événements

2.1 Flux de base

1. La crontab lance le processus de validation de liens.
2. Le processus de validation entre en communication avec la base de données et prend un certain nombre (paramètre) de liens à valider (données principales des URL).
3. Le processus de validation vérifie un lien et recalcule la moyenne des points, met à jour les données principales et enregistre comme historique les valeurs de cette validation (code d'état, temps de réponse, la date et le point obtenu)
4. Retour au point 2 pour prendre le lien suivant.

2.2 Flux alternatifs

2.2.1 Accès à une ressource non disponible.

2.b Si le système ne peut accéder à la base de données ou à Internet, le processus s'arrête et notifie l'administrateur.

2.2.2 Plus de lien à valider

2.b Il n'y a plus de lien à valider, alors il faut arrêter le processus de validation.

3. Exigences non fonctionnelles

3.1 Fiabilité

Les résultats recueillis par le validateur sont importants mais une erreur sur vingt résultats peut être tolérée. Le système doit pouvoir redémarrer normalement après un arrêt dû à un accès non disponible d'une ressource système (BD, Internet).

3.2 Performance

Nous devons donner un temps de réponse maximum pour chaque validation d'URL afin de faire un maximum de validation dans le temps qui nous est imparti.

3.3 Supportabilité

Des outils OpenSource pour faciliter la maintenance et l'amélioration du système (PHP & MySQL).

3.4 Utilisabilité

Pour les utilisateurs.

Des outils OpenSource (PHP & MySQL) pour l'indépendance du système.

3.5 Fonctionnalité

Paramètres à changer facilement (timer, nombre URL, heure de lancement, etc...).

4. Préconditions

1.1 Accès à Internet.

1.2 Accès à la DB des URL.

1.3 La DB contient des valeurs cohérentes.

5. Postconditions

5.1 Stockage des résultats recueillis dans la base de données.

5.2 Le système a averti l'administrateur en cas de problème.

5.3 La DB contient des valeurs cohérentes.

Spécification de Cas d'Utilisation : Visualisation des statistiques sur les disponibilités des URL.

1. Nom du Cas d'Utilisation

Visualisation des statistiques sur les disponibilités des URL.

1.1 Brève Description

L'administrateur veut visualiser les statistiques de disponibilité concernant une ressource. Ces statistiques sont calculées suivant la liste des codes d'état pour cette ressource ainsi que les temps de réponse et le nombre de chaque code d'état rencontré.

2. Flux d'événements

2.1 Flux de base

1. L'administrateur lance le processus de visualisation.
2. Le système entre en communication avec la base de données et affiche les URL avec leurs moyennes. La liste est triée par ordre croissant de la moyenne des points. N URL sont affichés par page.
3. L'administrateur choisit une ressource dans la liste triée.
4. Le système accède à la base de données et lit les données concernant la ressource choisie.
5. Le système affiche les données et les historiques sous forme graphique.

2.2 Flux alternatifs

2.2.1 Accès à une ressource non disponible.

2.a Le système ne peut accéder à la base de données, le processus s'arrête et notifie l'administrateur.

2.2.2 Accès à la page précédente ou suivante.

3.a L'administrateur choisit d'afficher la page suivante ou précédente.

4.a Le système accède à la base de données, lit les données des N URL suivantes ou précédentes et les affiches sous forme de liste triée par ordre croissant de la moyenne des points.

3. Exigences non fonctionnelles

3.1 Fiabilité

Le système doit pouvoir redémarrer normalement après un arrêt dû à un accès non disponible à la base de données.

3.2 Rendement

La visualisation des statistiques concernant la ressource demandée ne doit pas prendre plus d'un certain temps (quelques secondes tout au plus) afin de satisfaire l'administrateur.

La lecture de la liste des codes d'état dans la base de données doit se faire en quelques secondes.

3.3 Supportabilité

Le système doit pouvoir accueillir et supporter de plus en plus de codes d'état ainsi que leurs lectures.

Des outils OpenSource pour faciliter la maintenance et l'amélioration du système (PHP & MySQL).

3.4 Utilisabilité

Le système doit pouvoir afficher les statistiques à l'administrateur dans un temps raisonnable (quelques secondes).

3.5 Fonctionnalité

L'administrateur doit pouvoir choisir une ressource aisément.

Les statistiques doivent apporter une idée précise concernant la disponibilité de la ressource.

4. Préconditions

4.1 Accès à la DB.

4.2 L'administrateur est authentifié.

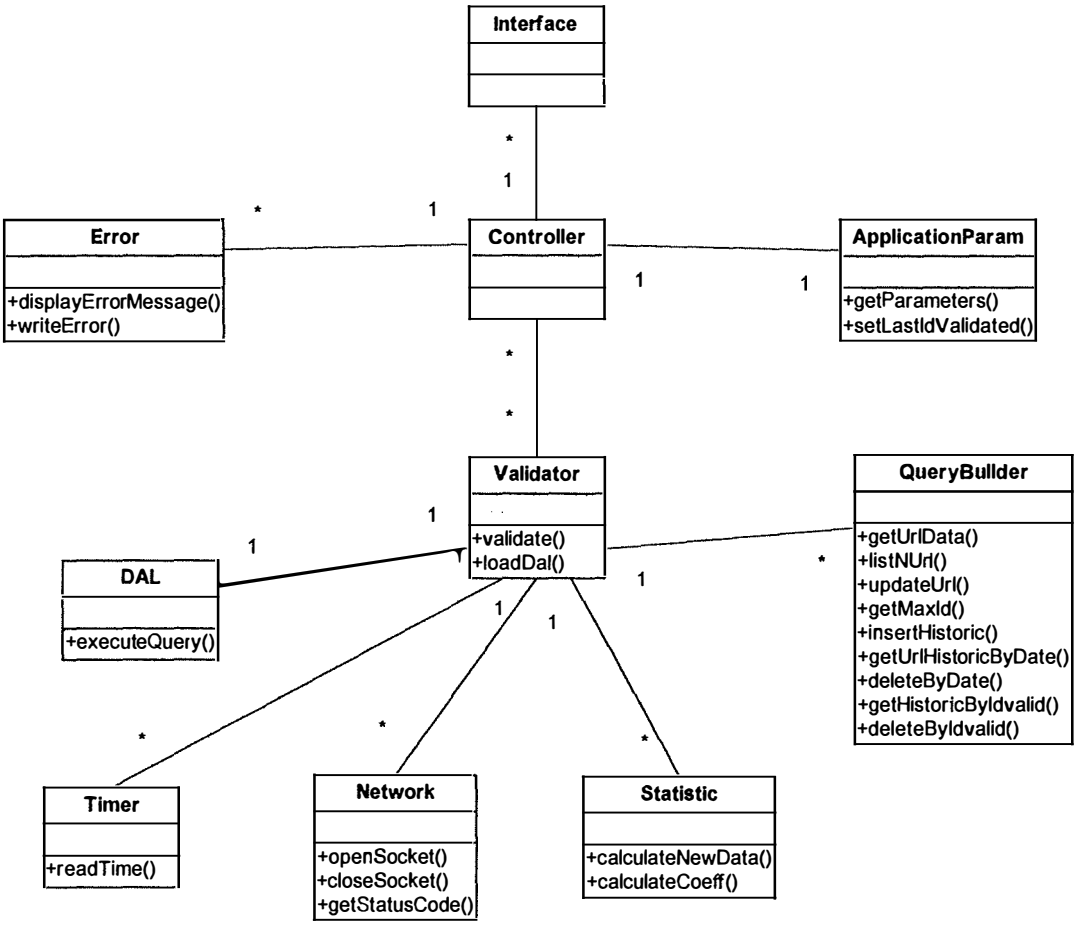
4.3 La DB contient des valeurs cohérentes.

5. Postconditions

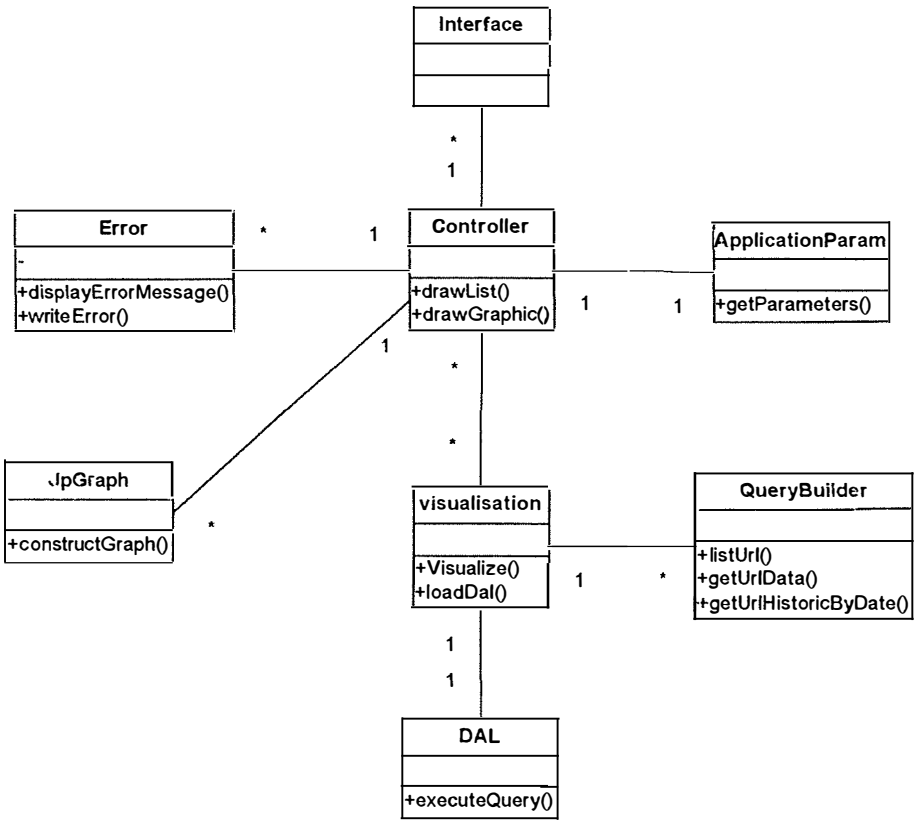
5.1 Visualisation des statistiques par l'administrateur.

5.2 Le système a averti l'administrateur en cas de problème.

Découpe en classe Validation

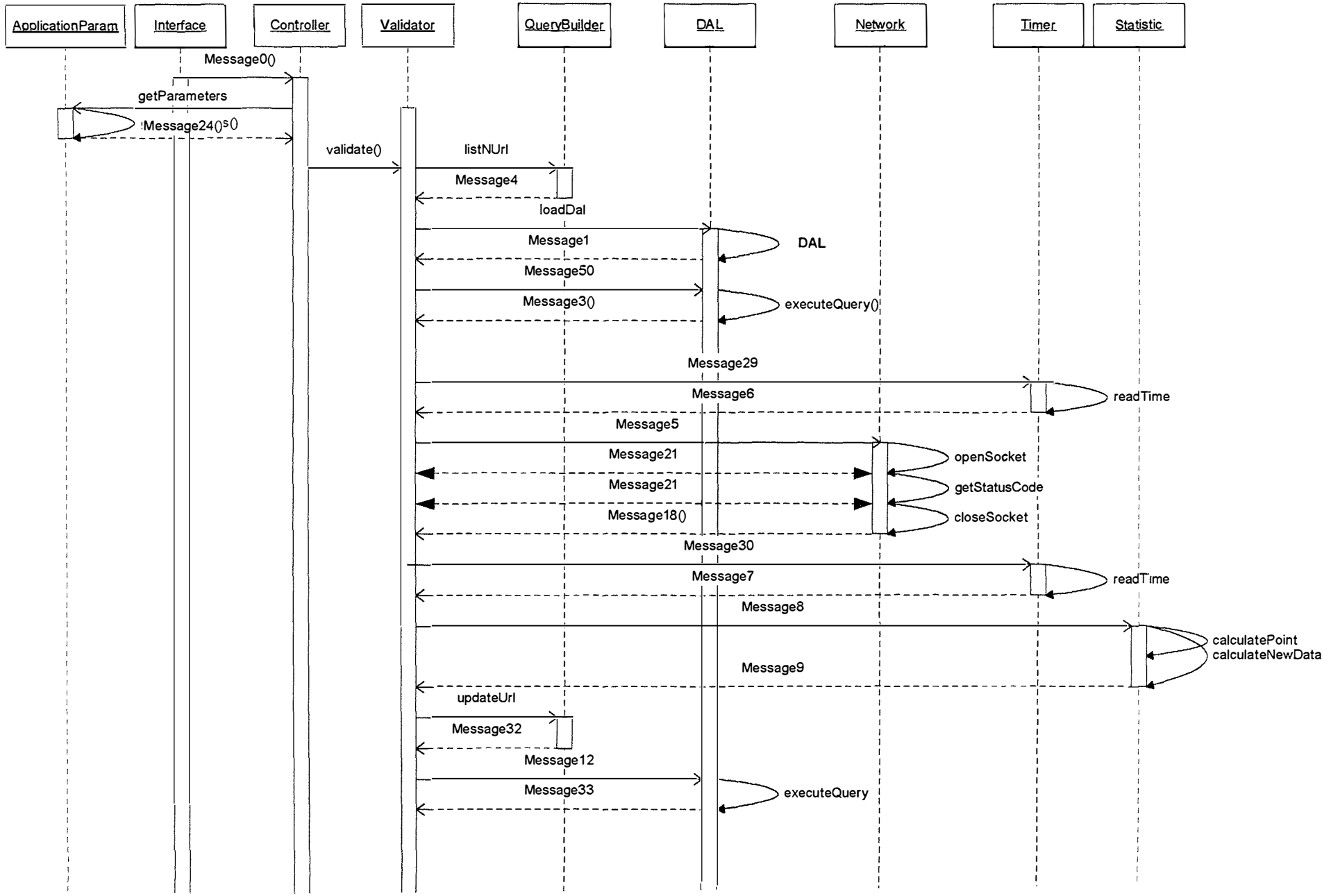


Découpe en classe
Visualisation



ANNEXE III

Diagramme de séquence : Validation



ANNEXE III

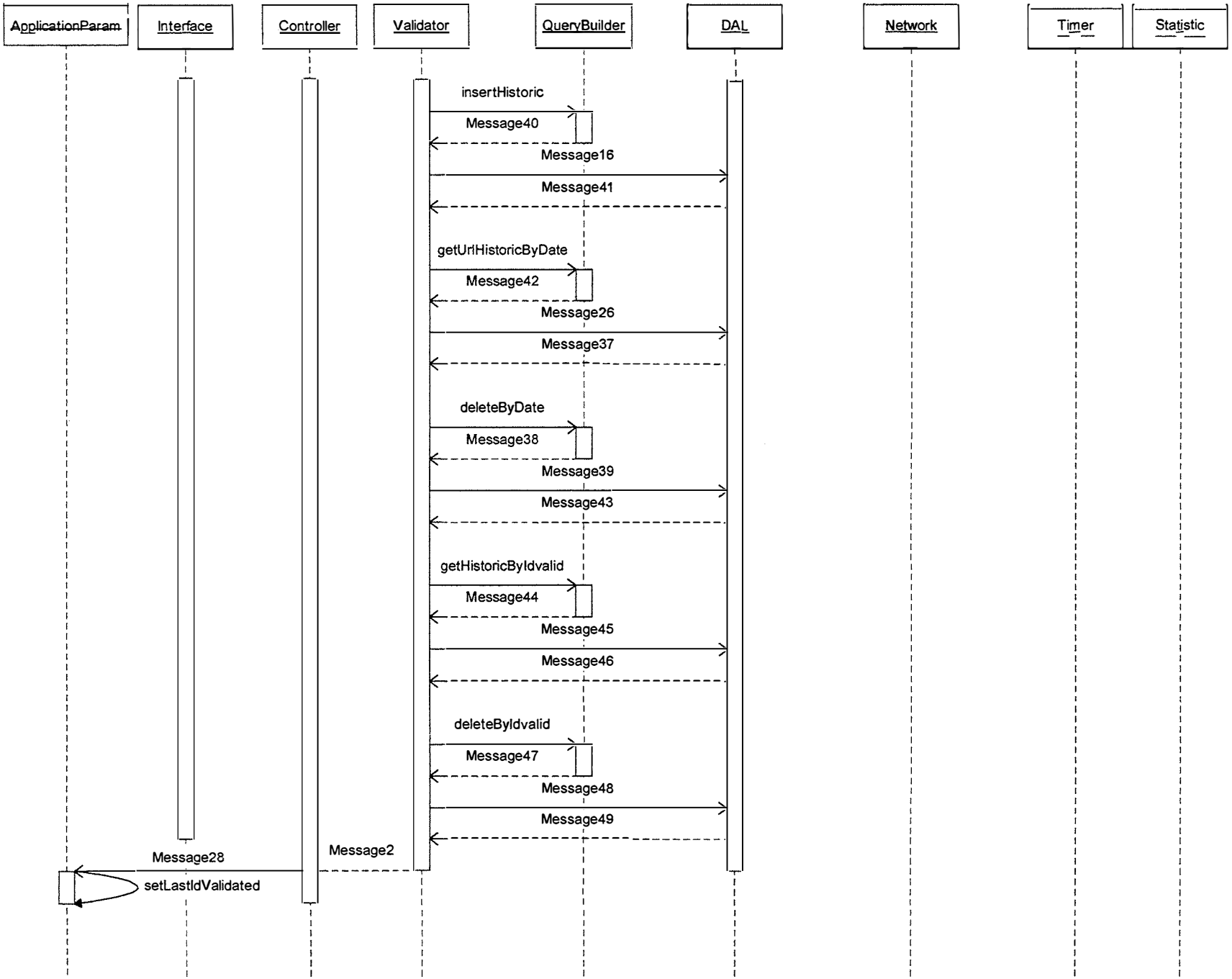
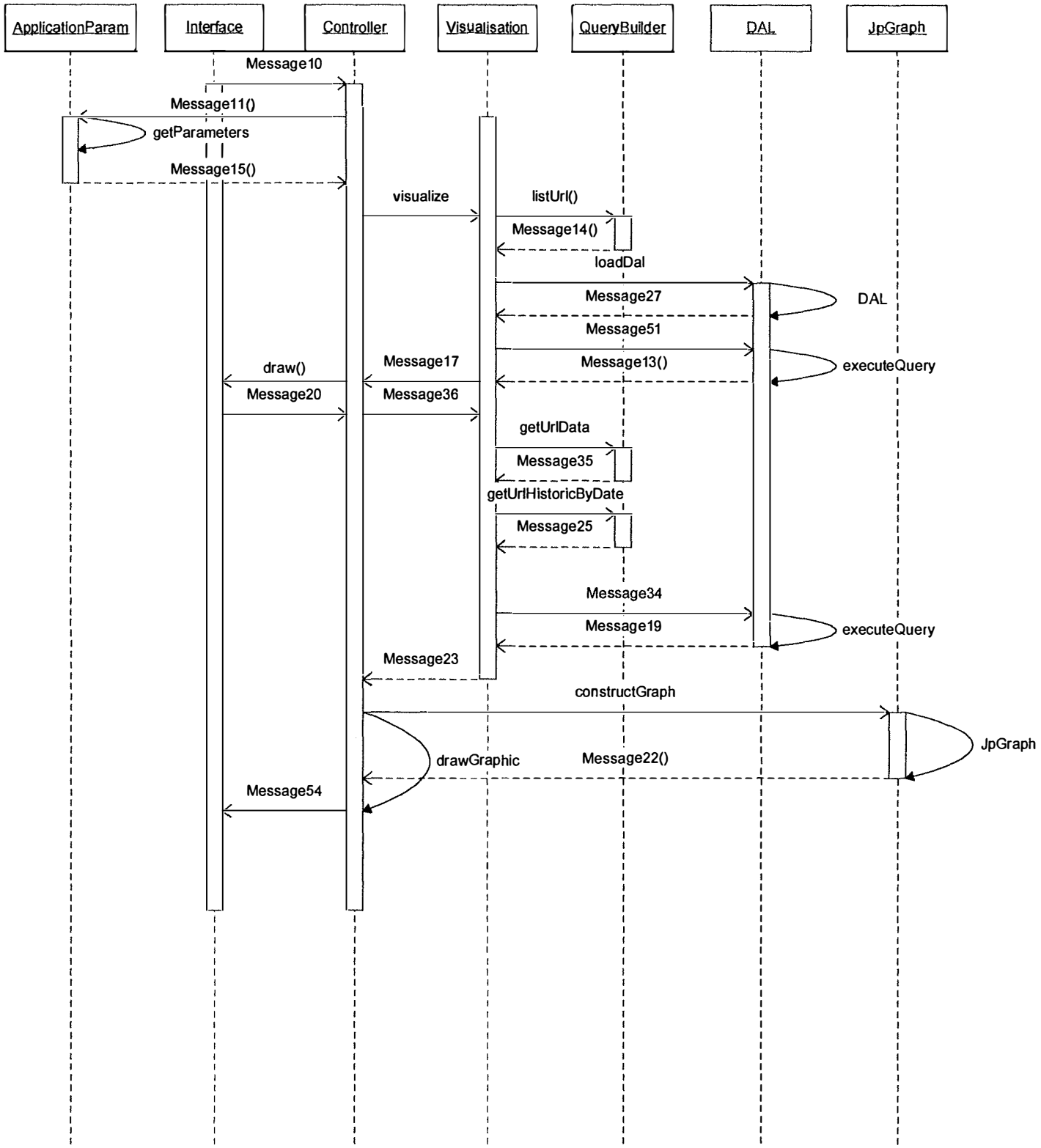
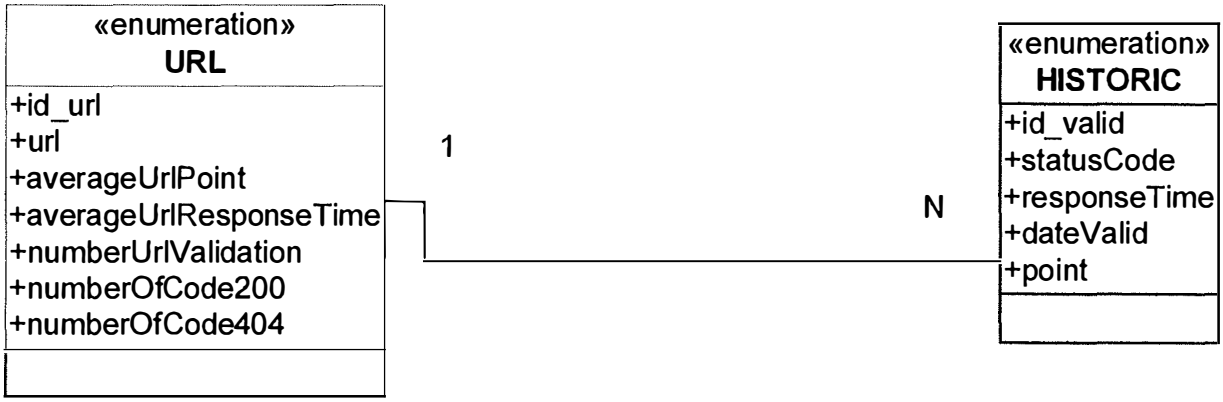

ApplicationParam
Interface
Controller
Validator
QueryBuilder
DAL
Network
Timer
Statistic
insertHistoric
Message40
Message16
Message41
getUrlHistoricByDate
Message42
Message26
Message37
deleteByDate
Message38
Message39
Message43
getHistoricByIdvalid
Message44
Message45
Message46
deleteByIdvalid
Message47
Message48
Message49
Message28
Message2
setLastIdValidated

Diagramme de séquence : Visualisation



ANNEXE III

Schéma de la base de données



Représentation de la base de données

URL

ID_URL	url	averageUrlPoint	averageUrlResponseTime	numberUrlValidation	numberOfCode200	numberOfCode404
1	http://www.rational.com/	0,846	1,87	54	52	2
2	http://www.info.fundp.ac.be/~software-quality/fr/index.html	0,968	1,23	87	87	0
3	ftp://ftp.fundp.ac.be/pub/	1	1,78	1	1	0
4	http://www.linux.com/	0	0	0	0	0

ANNEXE IV

VALIDATION

ID_VALID	ID_URL	statusCode	ResponseTime	dateValid	point
1	1	200	1,41	01/09/2002	1
2	2	200	2,65	01/09/2002	0,85
3	1	200	0,74	03/09/2002	1
4	2	404	5,68	03/09/2002	0
5	1	200	0,47	05/09/2002	1
6	2	200	1,21	05/09/2002	1
7	1	200	0,57	07/09/2002	1
8	2	304	0,97	07/09/2002	0
9	3	200	1,78	07/09/2002	1

Exigences non-fonctionnelles

3.5.3.1. Time Behaviour

Le Time Behaviour se définit par la consommation de temps durant les tests ou en production.

Les métriques utilisées dans ce cas-ci sont :

- Temps de réponse moyen

- Temps de réponse maximum

Pour les temps de réponses, toute requête du système doit mettre un temps précisé ci-dessus. Si une fonctionnalité demande un temps différent, les valeurs se trouvent dans le UC correspondant.

Nom	Description	Formule	Valeur	Input	Acteur-cible
Temps de réponse moyen	Combien de temps prend le système en moyenne pour répondre à une tâche déterminée	X= Tmean / Txmean Tmean = $\sum(T_i)/N$, (pour i =1 à N) TXmean = Temps moyen requis T_i = temps de réponse pour la mesure i N = nombre de tests effectués	0.80 TXmean (modem)= 3 sec TXmean (haut-débit)= 1 sec	Rapport de travail et de test.	FC, Admin, Client, Mainteneur.
Temps Maximum de réponse	Quelle est la limite absolue pour recevoir une réponse du syst ème queque soit l'opération.	X = Tmax / Rmax Rmax= temps de réponse maximum requis Tmax = MAX(T_i) (pour i = 1 à N) N= nombre de tests effectués T_i= temps de réponse pour la mesure i	1.1 Rmax (modem)= 7 sec Rmax(aut-débit) = 3 sec	Rapport de tests.	FC, Admin, Client, Mainteneur.

3.5.4. Maintenabilité

3.5.4.1. Analysabilité

Cette métrique permet de mesurer l'effort réalisé par le mainteneur pour diagnostiquer la cause de la panne.

Nom	Description	Formule	Valeur	Input	Acteur-cible
Capacité d'analyse de la panne	Le mainteneur peut-il identifier des opérations spécifiques qui ont causées la panne ?	X=1-A/B A = nombre de pannes dont les causes restent introuvées. B=Nombre total de pannes enregistrées.	0.80	Bug report(TBD)	FC, Admin, Client, Mainteneur.

3.5.4.2. Modifiabilité

La modifiabilité mesure l'effort du mainteneur pour implémenter une modification.

Nom	Description	Formule	Valeur	Input	Acteur-cible
Modifiabilité paramétrisable	Le mainteneur peut-il facilement changer les paramètres de l'application pour modifier le logiciel ?	X = 1-A/B A = Nombre de cas pour lesquels le mainteneur échoue la modification du logiciel par modification des paramètres. B = Nombre total de cas où le mainteneur tente de modifier le logiciel par modification des paramètres	0.30 (TBD valeur minimum)	Rapport de résolution de bugs(TBD)	FC, Admin, Client, Mainteneur.
Complexité des modifications	Le mainteneur peut-il facilement modifier le logiciel pour résoudre des problèmes ?	T = $\sum(A/B)/N$ A = temps en minutes mis pour effectuer un changement.	TBD : sachant que plus T tend vers 0, mieux c'est.	Rapport de résolution de bugs	FC, Admin, Client, Mainteneur.

	(TBD : définir problèmes : bugs ? non-conformité aux exigences ?)	changement. B = taille du changement du logiciel (TBD : choisir la métrique de la taille) N = nombre de changements.			
Temps d'implémentation du changement	Le mainteneur peut-il facilement modifier le logiciel pour résoudre des problèmes ?	Tmoy = $\sum (T_m) / N$ T_m = T_{out}-T_{in} T_{out} = Temps (HH :MM) auquel le problème est découvert T_{in} = Temps (HH :MM) auquel la cause du problème est découverte. N = le nombre de problèmes enregistrés.	180 min	Rapport de résolution de problèmes	FC, Admin, Client, Mainteneur.
Enregistrement des changements	Les modifications sont-elles introduites dans le code sous forme de commentaires ?	X=A/B A = nombre de modifications par classes ayant un commentaire expliquant la modification. B = Nombre total de classes modifiées sur base du code originale. TBD : définir code original.	0.80	Logs des versions	Developpeurs Mainteneurs Requirer(TBD : traduire).

ANNEXE V

Captures d'écran de la visualisation

Liste des URL par ordre croissant

N°	URL	Moyenne des points
1	ftp://ftp.info.fundp.ac.be/pub/publications/RP/Front.RP-00-001.ps.Z	0.185278
4	ftp://ftp.rational.com/public/java/Rotation.zip	0.244815
5	ftp://ftp.belnet.be/pub/doc/rfc/rfc/rfc.txt.gz	0.27505
6	ftp://ftp.inria.fr/doc/initinfo.zip	0.313411
3	ftp://ftp.columbia.edu/pub/mw.tar.Z	0.407643
10	http://www.rational.com/products/index.jsp	0.857143
15	http://linux.com/	0.880219
11	http://www.cse.dcu.ie/essiscope/sm2/9126ref.html	0.934733
2	ftp://ftp.sei.cmu.edu/public/documents/02.reports/pdf/02sr001.pdf	0.959119
12	http://lists.w3.org/Archives/Public/w3c-wai-eo/2001AprJun/0018.html	1
14	http://www.info.fundp.ac.be/~software-quality/	1

Previous Next

Visualisation de l'URL choisi : http://www.cse.dcu.ie/essiscope/sm2/9126ref.html

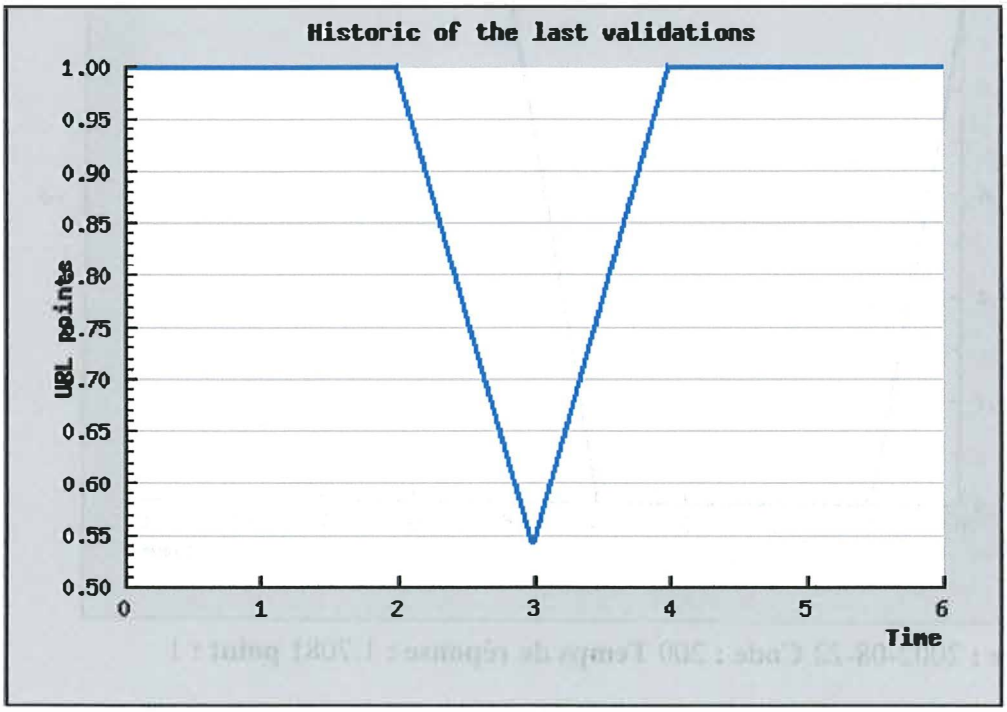
Moyenne des points : 0.934733

Moyenne du temps de réponse (sec) : 0.965852

Nombre de code 200 reçus : 7

Nombre de code 404 reçus : 0

Nombre de validation : 7



0 Date : 2002-08-24 **Code :** 200 **Temps de réponse :** 0.070722 **point :** 1

1 Date : 2002-08-24 **Code :** 200 **Temps de réponse :** 0.104848 **point :** 1

2 Date : 2002-08-24 **Code :** 200 **Temps de réponse :** 0.523312 **point :** 1

3 Date : 2002-08-28 **Code :** 200 **Temps de réponse :** 5.65493 **point :** 0.54

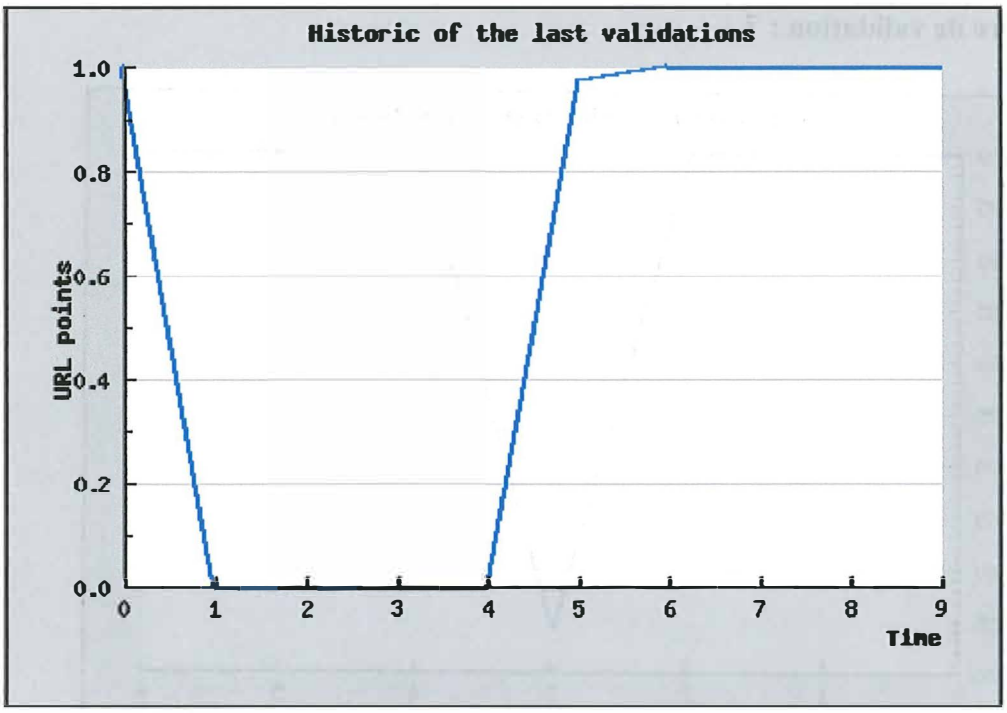
4 Date : 2002-08-29 **Code :** 200 **Temps de réponse :** 0.201682 **point :** 1

5 Date : 2002-08-30 **Code :** 200 **Temps de réponse :** 0.0670609 **point :** 1

6 Date : 2002-08-30 **Code :** 200 **Temps de réponse :** 0.138412 **point :** 1

Visualisation de l'URL choisi : ftp://ftp.columbia.edu/pub/mw.tar.Z

Moyenne des points : 0.314283
Moyenne du temps de réponse (sec) : 1.60997
Nombre de code 200 reçus : 6
Nombre de code 404 reçus : 13
Nombre de validation : 19



0 Date : 2002-08-22 **Code :** 200 **Temps de réponse :** 1.7081 **point :** 1

1 Date : 2002-08-22 **Code :** 404 **Temps de réponse :** 1.90153 **point :** 0

2 Date : 2002-08-22 **Code :** 404 **Temps de réponse :** 1.4487 **point :** 0

3 Date : 2002-08-22 **Code :** 404 **Temps de réponse :** 1.53382 **point :** 0

4 Date : 2002-08-22 **Code :** 404 **Temps de réponse :** 1.43484 **point :** 0

5 Date : 2002-08-23 **Code :** 200 **Temps de réponse :** 2.22888 **point :** 0.97

6 Date : 2002-08-24 **Code :** 200 **Temps de réponse :** 1.73279 **point :** 1

7 Date : 2002-08-24 **Code :** 200 **Temps de réponse :** 1.80698 **point :** 1

8 Date : 2002-08-24 **Code :** 200 **Temps de réponse :** 1.48057 **point :** 1

9 Date : 2002-08-28 **Code :** 200 **Temps de réponse :** 1.83794 **point :** 1

Visualisation de l'URL choisi : ftp://ftp.belnet.be/pub/doc/rfc/rfc/rfc.txt.gz

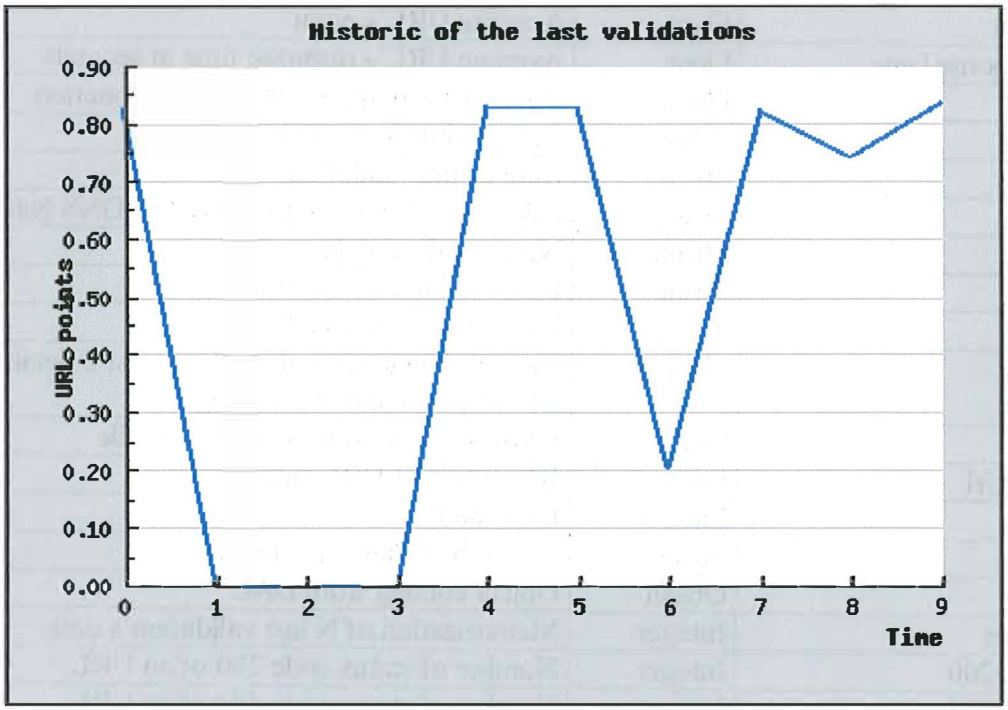
Moyenne des points : 0.267537

Moyenne du temps de réponse (sec) : 4.62661

Nombre de code 200 reçus : 7

Nombre de code 404 reçus : 12

Nombre de validation : 19



0 Date : 2002-08-22 **Code :** 200 **Temps de réponse :** 3.36781 **point :** 0.83

1 Date : 2002-08-22 **Code :** 404 **Temps de réponse :** 3.35124 **point :** 0

2 Date : 2002-08-22 **Code :** 404 **Temps de réponse :** 3.62886 **point :** 0

3 Date : 2002-08-22 **Code :** 404 **Temps de réponse :** 3.64926 **point :** 0

4 Date : 2002-08-23 **Code :** 200 **Temps de réponse :** 3.36876 **point :** 0.83

5 Date : 2002-08-24 **Code :** 200 **Temps de réponse :** 3.40042 **point :** 0.82

6 Date : 2002-08-24 **Code :** 200 **Temps de réponse :** 8.36663 **point :** 0.20

7 Date : 2002-08-24 **Code :** 200 **Temps de réponse :** 3.424 **point :** 0.82

8 Date : 2002-08-28 **Code :** 200 **Temps de réponse :** 4.06753 **point :** 0.74

9 Date : 2002-08-29 **Code :** 200 **Temps de réponse :** 3.33907 **point :** 0.83

Codes

Le type des variables

Data	Type	Description
arrayNUrl	Array	A request type containing the n URL to validate
arrayParam	Array	Array containing all parameters coming from initialsiation's file
arrayUrl	Array	Array containing mains data of a URL(id_url, url, averageUrlPoint, averageUrlResponseTime, numberUrlValidation, numberOfCode200, numberOfCode404)
arrayUrlStats	Array	Array : key URL, values : averageUrlPoint
averageUrlPoint	Float	Average URL's point
averageUrlResponseTime	Float	Average URL's response time in seconds
coefficient	Float	Coefficient to apply based on the fonction
currentTime	Float	System time in seconds
dateValid	String	Date of the validation
dbAddress	String	Address of Database to use (IP or DNS Name)
dbName	String	Name of the database
dbPassword	String	Password to access db
dbUsername	String	Username to access db
endTime	Float	System time in seconds with at least 2 decimals when the validation is ending
errorMessage	String	Error message to write in the log file
idLastValidateUrl	Integer	ID of the last URL validate
idUrl	Integer	ID of an URL
languageSql	String	Which SQL language to use
mydal	Object	Object coming from DAL
numberHistories	Integer	Memorization of N last validation's data
numberOfCode200	Integer	Number of status code 200 of an URL
numberOfCode404	Integer	Number of status code 404 of an URL
numberUrlToValidate	Integer	Number of URL to validate
numberUrlValidation	Integer	Number of validations of an URL
pageToDisplay	Integer	Page number to display for visualisation
path	String	Contains address without protocol and host
point	Float	Point based on status code
port	Integer	Port number of the connection
protocol	String	Network Protocol
rangeUrl	Integer	Number of URL to list in a page
responseTime	Float	Response time in seconds
selectedUrl	URL	URL selected by user for graphic's visualisation
request	String	Correct SQL request
statusCode	Integer	Status code specified in http protocol
startTime	Float	System time in seconds with at least 2 decimals when the validation is starting
tolerateResponseTime	Float	Response time before losing points in seconds
url	URL	URL to validate
urlCoeff	Float	URL's coefficent based on the function
urlPoint	Float	URL's point based on the function and status code
urlTimeout	Float	Timeout in seconds
validationTime	Integer	Length of validation (in minutes)

```php
<?php
/*=========================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-François
|        Emails:  jfdubus@hotmail.com
|
|     Language:   PHP 4.0
|     Platform:   NT 4/ Intel
|   To Compile:   --
|
|         File:   ApplicationParam.php4
|     Creation:   10/07/2002
|     Modified:
+-------------------------------------------------------------------------
|
|  Description:  allow to load parameters contained in initialisation's file
|
|   Known Bugs:  None
|
 +=======================================================================*/

class ApplicationParam {

    /*-------------------getParameters-------------------------------------
    |   Function: getParameters
    |
    |    Purpose: The getParameters function loads parameters contained
    |             in initialisation's file in an array
    |
    |    Arguments: $fileName : file's name containing the parameters
    |
    |    Returns:  array
    |
    *-------------------------------------------------------------------------*/
  function getParameters($fileName){

    if($file=fopen($fileName,"r")){ // r for read only
        //temporary array containing all line of the file
        $arrayLine = file($fileName);
        $numberElement = count($arrayLine);

        for ($compt = 0; $compt < $numberElement; ++$compt){
            // if it is a line containing a parameter
            if (strpos($arrayLine[$compt],"=") != 0){

                $line = explode('=',trim($arrayLine[$compt]));

                $key=trim($line[0]);
                $value = trim($line[1]);

                $arrayParam[$key]=$value;
            }
        }
        fclose($file);
        return ($arrayParam);
    }
    else{
        $error = new Error();
        $error->displayErrorMessage("errorOpenFile","");
    }

    }

    /*-------------------setLastIdValidated--------------------------------
    |   Function: setLastIdValidated
    |
    |    Purpose: writes in the paramters's file the next URL's id for
    |             the next validation
    |
    |
    |    Arguments: $idUrl : gives the next URL's id for the next
    |                validation
    |
    |    Returns: boolean
    |
```

```php
77      *-------------------------------------------------------------------------*/
78    function setLastIdValidated($idUrl){
79      $arrayParam = $this->getParameters("Parameters.ini");
80      if($file=fopen("Parameters.ini","w+")){
81
82        $line1  = "[Database]\n";
83        $line2  = "languageSql=$arrayParam[languageSql]\n";
84        $line3  = "dbAddress=$arrayParam[dbAddress]\n";
85        $line4  = "dbUsername=$arrayParam[dbUsername]\n";
86        $line5  = "dbPassword=$arrayParam[dbPassword]\n";
87        $line6  = "dbName=$arrayParam[dbName]\n";
88        $line7  = "numberHistories=$arrayParam[numberHistories]\n";
89
90        $line8  = "\n[Validation]\n";
91        $line9  = "numberUrlToValidate=$arrayParam[numberUrlToValidate]\n";
92        $line10 = "validationTime=$arrayParam[validationTime]\n";
93        $line11 = "urlTimeout=$arrayParam[urlTimeout]\n";
94        $line12 = "tolerateResponseTime=$arrayParam[tolerateResponseTime]\n";
95        $line13 = "idLastValidateUrl=$idUrl\n";
96
97        $line14 = "\n[Visualisation]\n";
98        $line15 = "rangeUrl=$arrayParam[rangeUrl]\n";
99
100       fputs($file,$line1.
101             $line2.
102             $line3.
103             $line4.
104             $line5.
105             $line6.
106             $line7.
107             $line8.
108             $line9.
109             $line10.
110             $line11.
111             $line12.
112             $line13.
113             $line14.
114             $line15);
115
116       fclose($file);
117       return (true);
118     }
119     else{
120         $error = new Error();
121         $error->displayErrorMessage("errorOpenFile","");
122         return(false);
123     }
124   }
125 }
126 ?>
```

```php
<?php
/*==========================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-François
|        Emails:  jfdubus@hotmail.com
|
|     Language:   PHP 4.0
|     Platform:   NT 4/ Intel
|   To Compile:   --
|
|         File:   Controller.php4
|     Creation:   10/07/2002
|     Modified:
+-------------------------------------------------------------------------
|
|  Description:  allow to load parameters contained in initialisation's file
|
|   Known Bugs:  None
|
+=========================================================================*/

/*----------------INCLUDE---------*/

include ("ApplicationParam.php4");
include ("Validator.php4");
include ("Visualisation.php4");
include ("JpGraph.php4");
include ("Error.php4");
include ("QueryBuilder.php4");
include ("/home/jdubus/public_html/dal-0.4.0/src/class_dal.phps");

class Controller {

    /*-----------------controller-----------------------------------
    |   Function: controller
    |
    |    Purpose: The controller function
    |
    |    Arguments: $mode : to select the service type (validation,
    |                listing or drawing)
    |           $param :  array containing the application parameters
    |
    |    Returns:
    |
    *-------------------------------------------------------------------*/

    function main($mode,$param){

        $fileName = "Parameters.ini";
        $error = new Error();
        $ini = new ApplicationParam();
        if ($arrayParam = $ini->getParameters($fileName)){

            switch ($mode){
                case"1" :$valid = new Validator();
                    $rep = $valid->validate($arrayParam);
                    $ini->setLastIdValidated($rep);

                    break;

                case"2" ://listing
                    $visual = new Visualisation();
                    $respp = $visual->visualize
                        ($mode,$param,$arrayParam);

                    $this->drawList($respp,$param);
                    break;

                case"3" ://graphics
                    $JpG = new JpGraph();
                    $visual = new Visualisation();

                    //param contains id_url selected
                    $resp = $visual->visualize
                        ($mode,$param,$arrayParam);
```

```
77
78                      $resp1 = $resp[0];// main url data
79                      $resp2 = $resp[1];// historic url data
80                      $resp3 = $resp[2];
81
82                      $i=0;
83                      while($resultrow= $resp2->fetchArray())
84                          {
85                          $dataPoint[$i] =
86                              $resultrow[point];
87                          $i = $i + 1;
88                          }
89
90                      $JpG->constructGraph($dataPoint);
91
92                      $this->drawGraphic($resp1,$resp3);
93                      break;
94              }
95          }
96          else{$error->displayErrorMessage("errorOpenFile","");}
97      }
98
99      /*-------------drawList-----------------------------------------
100     |   Function: drawList
101     |
102     |    Purpose: The drawList function creates a HTML page listing N URL
103     |
104     |    Arguments: $resp : resultset containing a list of url data
105     |                 (without historics)
106     |             $page : indicates which page to display
107     |
108     |    Returns:
109     |
110     *-------------------------------------------------------------------*/
111
112     function drawList($resp,$page){
113         $prev = $page-1;
114         $next = $page+1;
115         echo("<html>
116             <head>
117             <title>Visualisation des statistiques : listing</title>
118             <meta http-equiv='Content-Type' content='text/html;
119                         charset=iso-8859-1'>
120             </head>
121
122             <body bgcolor='#FFFFFF' text='#000000'>
123             <p><font size='+2'>Liste des URL par ordre
124                 croissant </font></p>
125             <table width='77%' border='0'>
126               <tr>
127                 <td width='7%'>
128                   <div align='center'><b>N&deg;</b></div>
129                 </td>
130                 <td width='68%'>
131                   <div align='center'><b>URL</b></div>
132                 </td>
133                 <td width='25%'>
134                   <div align='center'><b>Moyenne des points
135                     </b></div>
136                 </td>
137               </tr>");
138         while($res = $resp->fetchArray())// Fetch as array
139         {
140             echo(" <tr>
141                 <td width='7%'>
142                   <div align='center'>$res[id_url]</div>
143                 </td>
144                 <td width='68%'>
145                   <div align='left'><a href='
146                      test_controller.php4?a=3&b=$res[id_url]'
147                      target='_blank'>$res[url]</a></div>
148                 </td>
149                 <td width='25%'>
150                 <div align='center'>$res[averageUrlPoint]</div>
151                 </td>
152               </tr>");
```

```
153         }
154         $nextId = $res[id_url]+1;
155 
156         echo("</table>
157         <p><a href='test_controller.php4?a=2&b=$prev'>Previous</a>
158         <a href='test_controller.php4?a=2&b=$next'>Next</a></p>
159         </body>
160         </html>");
161     }
162 
163     /*-------------------drawGraphic-----------------------------------
164     |   Function: drawGraphic
165     |
166     |   Purpose: The drawGraphic function creates HTML page to see details
167     |
168     |    Arguments: $array1 : resultset containing main url data
169     |               $array2 : resultset containing historic url data
170     |               $picture : graphic of the statistics
171     |
172     |    Returns:
173     |
174     *-----------------------------------------------------------------*/
175 
176     function drawGraphic($array1,$array2){
177 
178         $res = $array1->fetchArray();// Fetch as array
179         echo("<html>
180             <head>
181             <title>Historiques</title>
182             <meta http-equiv='Content-Type' content='text/html;
183                 charset=iso-8859-1'>
184             <meta HTTP-EQUIV='Pragma' CONTENT='no-cache'>
185             <body bgcolor='#FFFFFF' text='#000000'>
186             <h2><font face='Georgia, Times New Roman,Times, serif'
187                 color='#3366CC'>Visualisation de l'URL choisi :
188                     $res[url] </font></h2>
189             <table width='75%' border='0'>
190               <tr>
191             <td width='10%'> </td>
192             <td width='90%'><b>Moyenne des points :</b>
193                     $res[averageUrlPoint]</td>
194               </tr>
195               <tr>
196                 <td width='10%'> </td>
197                 <td width='90%'><b>Moyenne du temps de
198                     r&eacute;ponse (sec) :</b>
199                     $res[averageUrlResponseTime]</td>
200               </tr>
201               <tr>
202                 <td width='10%'> </td>
203                 <td width='90%'><b>Nombre de code 200
204                     re&ccedil;us :
205                     </b> $res[numberOfCode200]</td>
206               </tr>
207               <tr>
208                 <td width='10%'> </td>
209                 <td width='90%'><b>Nombre de code 404
210                     re&ccedil;us :
211                     </b> $res[numberOfCode404]</td>
212               </tr>
213               <tr>
214                 <td width='10%' height='2'> </td>
215                 <td width='90%' height='2'><b>Nombre de
216                     validation :
217                     </b> $res[numberUrlValidation]</td>
218               </tr>
219                <tr>
220                 <td width='10%' height='2'> </td>
221                 <td width='90%' height='17'><img
222                     src='jpgraph_cache/image.png'></td>
223               </tr>
224                <tr>
225                 <td width='10%' height='2'> </td>
226                 <td width='90%' height='2'>");
227 
228 
```

```
229                 while($res2 = $array2->fetchArray())
230                 {echo("<p>
231                     <b>$loop</b>  <b>Date :</b>
232                         $res2[dateValid]
233                       <b>Code :</b> $res2[statusCode]
234                       <b>Temps de réponse :</b>
235                         $res2[responseTime]
236                       <b>point :</b> $res2[point]
237                       </p>");
238                 $loop = $loop+1;
239                 }
240             echo("</td>
241               </tr>
242             </table> <p> </p>  </body> </html>");
243     }
244 }
245 ?>
```

```php
<?php
/*=============================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-Francois
|        Emails:  jfdubus@hotmail.com
|
|     Language:   PHP 4.0
|     Platform:   NT 4/ Intel
|   To Compile:   --
|
|         File:   Error.php4
|     Creation:   10/07/2002
|     Modified:
+-----------------------------------------------------------------------------
|
|  Description:  File for all error treatments
|
|   Known Bugs:  None
|
 +============================================================================*/


class Error {

    /*-------------------displayErrorMessage-------------------------------
    |   Function: displayErrorMessage
    |
    |    Purpose: The displayErrorMessage function manages the differents
    |             errors
    |
    |    Arguments: $code : error code
    |             $suit : supplementary information on the error
    |
    |    Returns:
    |
    *-------------------------------------------------------------------*/

  function displayErrorMessage($code,$suit){

    $logFile="Log.txt";

    switch ($code){

        case "errorOpenFile" :
           $errorMessage = "Error in opening the initialisation file";
            $this->writeError($logFile,$errorMessage);
            break;

        case "errorOpenSocket" :
            $errorMessage = "Error in opening a socket for $suit";
            $this->writeError($logFile,$errorMessage);
            break;

        case "errorSelectConfiguration" :
            $errorMessage =
            "Error in selection of the configuration in DAL";
            $this->writeError($logFile,$errorMessage);
            break;

        case "errorSelectDatabase" :
            $errorMessage =
            "Error in selection of the database in DAL";
            $this->writeError($logFile,$errorMessage);
            break;
    }

   }
    /*-------------------writeError-----------------------------------------
    |   Function: writeError
    |
    |   Purpose: The writeError function writes in the end of the log file
    |            the errors message
    |
    |    Arguments: $error : String containing the error message
    |
```

```
77      |
78      |    Returns:
79      |
80      *-------------------------------------------------------------------------*/
81    function writeError($logFile,$errorMessage){
82
83      if($file=fopen($logFile,"a")){
84
85          $date = getdate();
86          fputs($file,"$errorMessage at $date[mday]-$date[mon]-$date[year]
87          $date[hours]:$date[minutes]:$date[seconds]\n");
88          fclose($file);
89      }
90    }
91
92  }
93  ?>
```

```php
<?php
/*=============================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-Francois
|        Emails:  jfdubus@hotmail.com
|
|      Language:  PHP 4.0
|      Platform:  NT 4/ Intel
|   To Compile:  --
|
|          File:  JpGraph.php4
|      Creation:  25/07/2002
|      Modified:
+-----------------------------------------------------------------------------
|
|  Description:  File to construct graphics for the visualisation
|
|   Known Bugs:  None
|
+============================================================================*/

/*-----------------INCLUDE----------*/
include ("/home/jdubus/public_html/jpgraph-1.7/src/jpgraph.php");
include ("/home/jdubus/public_html/jpgraph-1.7/src/jpgraph_line.php");

class JpGraph {

    /*-----------------constructGraph-------------------------------------
    |   Function: constructGraph
    |
    |    Purpose: The constructGraph function
    |
    |    Arguments: $arrayData : array containing data to construct the
    |           graphic and saves it in a file
    |
    |    Returns:  none
    |
    *-----------------------------------------------------------------------*/

  function constructGraph($arrayData){
    // Create the graph. These two calls are always required

    $graph = new Graph(500,350);
    $graph->SetScale("textlin");

    // Create the linear plot
    $lineplot=new LinePlot($arrayData);

    $graph->Add($lineplot);

    // Increase the margins (left,right,top,bottom)
    $graph->img->SetMargin(60,30,30,60);

    // Add graph and axis title
    $graph->title->Set("Historic of the last validations");

    $graph->xaxis->title->Set("Time");

    $graph->yaxis->title->Set("URL points");

    //Change apparences
    $graph->title->SetFont(FF_FONT1,FS_BOLD);

    $graph->yaxis->title->SetFont(FF_FONT1,FS_BOLD);

    $graph->xaxis->title->SetFont(FF_FONT1,FS_BOLD);

    $lineplot->SetColor("blue");
    $lineplot->SetWeight(2);  // Two pixel wide

    $graph->Stroke("/home/jdubus/public_html/jpgraph_cache/image.png");
  }
}
?>
```

```php
<?php
/*==========================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-François
|        Emails:  jfdubus@hotmail.com
|
|     Language:   PHP 4.0
|     Platform:   NT 4/ Intel
|   To Compile:   --
|
|         File:   Network.php4
|     Creation:   10/07/2002
|     Modified:
+-------------------------------------------------------------------------
|
|  Description:  allow to connect to the web : open, close socket and get
|                the status code
|
|   Known Bugs:  None
|
 +=========================================================================*/


class Network{

    /*-------------------openSocket-----------------------------------------
    |   Function: openSocket
    |
    |   Purpose: The openSocket function returns a socket on the server
    |            containing the resource
    |
    |   Arguments: $host : only host url address : IP or DNS name
    |       $protocol : string containing the protocole of the
    |                   url (http, ftp)
    |       $port : port number on which the socket must be opened
    |
    |   Returns:  socket
    |
    *-------------------------------------------------------------------------*/
    function openSocket ($protocol, $host, $port){
        $protocol=strtoupper($protocol);
        if ($protocol=="HTTP"){
            $host = $this->removeSpace($host);
            if($sock=fsockopen($host,$port)){
                return($sock);
            }
            else{
                $error = new Error();
                $error->displayErrorMessage
                        ("errorOpenSocket",$host);
                return(false);
            }
        }
        else{
            if($protocol=="FTP"){
                $ftp_user_name = "anonymous";
                $ftp_user_pass = "wsftp605@";
                $serv = $host;

                $pos = strrpos($host,"@");
                if($pos !=0){
                    $server2 = explode("@",$host);
                    $log = $server2[0];
                    $serv = $server2[1];
                    $authentification = explode(":",$log);
                    $ftp_user_name = $authentification[0];
                    $ftp_user_pass = $authentification[1];
                }

                $conn_id = ftp_connect($serv, 21);
                if($login_result = ftp_login($conn_id,
                            $ftp_user_name,$ftp_user_pass)){
                    return($conn_id);
                }
                else{
```

```
                    $error = new Error();
                    $error->displayErrorMessage
                            ("errorOpenSocket",$host);
                    return(false);
                }
            }
        }
    }


    /*-----------------closeSocket-----------------------------------
    |   Function: closeSocket
    |
    |  Purpose: The closeSocket function closes the socket given in
    |           parameter
    |
    |  Arguments: $sock : socket to close
    |
    |   Returns:  none
    |
    *---------------------------------------------------------------*/
    function closeSocket($sock,$protocol){
        $protocol=strtoupper($protocol);
        switch ($protocol){
            case "HTPP" : fclose($sock);break;
            case "FTP" :  ftp_quit($sock);break;
        }
    }

    /*-----------------getStatusCode---------------------------------
    |   Function: getStatusCode
    |
    |    Purpose: The getStatusCode function returns status code for the
    |           GET request of the URL
    |
    |    Arguments: $protocol : string containing the protocole of the
    |                url (htpp, ftp)
    |       $socket : socket opens the server having the resource
    |       $path : URL without protocol and host (ex : \home\file.txt)
    |
    |
    |    Returns:   statusCode
    |
    *---------------------------------------------------------------*/
function getStatusCode($protocol,$sock,$path){

  $path = $this->removeSpace($path);
  $protocol=strtoupper($protocol);
  $statusCode = 404;
  if ($protocol=="HTTP"){

      fputs($sock,"GET $path HTTP/1.0\r\n\r\n");

          $buffer=fgets($sock,1024);
          $split=explode(' ',$buffer);
          $http = $split[0];
          $statusCode=$split[1];
  }
  else{
      if($protocol=="FTP"){
          //delete the last segment separated by '/' to have
          //the directory
          $pos = strrpos($path,"/");
          //get all path without the last / and after
          $directory = substr($path,1,$pos);
          //go into directory
          if (ftp_chdir($sock,$directory)){

              $length = strlen($path);
              $myFile = substr($path,$pos+1,$length);
              if ($myFile ==""){//path is a directory
                  $statusCode = 200;
              }
              else{
                  $dir=ftp_pwd($sock);
                  $list=Array();
```

```
153
154                     $list=ftp_nlist($sock, "$dir");
155                     $i=0;
156                     while($i<count($list)){
157                         if ($list[$i]==$myFile ||
158                           $list[$i]==$dir."/".$myFile){
159                             $statusCode = 200;
160                         }
161                         $i++;
162                     }
163                 }
164             }
165             else{
166                 $statusCode = 404;
167             }
168         }
169     }
170     return($statusCode);
171 }
172
173
174     /*--------------------removeSpace--------------------------------------
175     |   Function: removeSpace
176     |
177     |    Purpose: The removeSpace function returns the string remplacing
178     |       space by '%20'
179     |
180     |    Arguments: $string : String to modify
181     |
182     |
183     |    Returns:   path(string)
184     |
185     *-------------------------------------------------------------------*/
186     function removeSpace($string){
187
188         if(strpos($string," ") != false){
189
190             $temp = explode(' ',$string);
191             $number = sizeof($temp);
192             $string ="$temp[0]";
193             for($i=1;$i<$number;++$i){
194
195                 $temp2 = "%20".$temp[$i];
196                 $string = $path2.$temp2;
197             }
198         }
199         return($string);
200     }
201 }
202
203 ?>
```

```php
<?php
/*=============================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-François
|        Emails:  jfdubus@hotmail.com
|
|     Language:   PHP 4.0
|     Platform:   NT 4/ Intel
|   To Compile:   --
|
|         File:
|     Creation:   10/07/2002
|     Modified:
+-------------------------------------------------------------------------------
|
|  Description:  allow to construct the queries according to SQL language
|
|   Known Bugs:  None
|
 +============================================================================*/

class QueryBuilder {


    /*-------------------getUrlHistoricByDate-------------------------------
    |   Function: getUrlHistoricByDate
    |
    |    Purpose: The getUrlHistoricByDate function returns the request to
    |         get all data from historic table for a given id_url and
    |         order by date
    |
    |   Arguments : $idUrl : URL id to search
    |               $arrayParam : array containing parameters for database
    |
    |    Returns:   String
    |
    *-------------------------------------------------------------------------*/
  function getUrlHistoricByDate($idUrl,$arrayParam){

        $languageSql = strtoupper($arrayParam[languageSql]);
        if ($languageSql == "MYSQL"){
            $request = "SELECT * FROM historic WHERE id_url =
                    $idUrl ORDER BY dateValid";
            return($request);
        }
    }

    /*-------------------getUrlData-----------------------------------------
    |   Function: getUrlData
    |
    |    Purpose: The getUrlData function returns the request to get all
    |           data from url table for the given id_url
    |
    |    Arguments: $idUrl : URL id to search
    |               $arrayParam : array containing parameters for database
    |
    |    Returns:   String
    |
    *-------------------------------------------------------------------------*/

  function getUrlData($idUrl,$arrayParam){

        $languageSql = strtoupper($arrayParam[languageSql]);
        if ($languageSql == "MYSQL"){
            $request = "SELECT * FROM url WHERE id_url = $idUrl";
            return($request);

        }
    }
```

```
77      /*-------------------listNUrl-----------------------------------------
78      |    Function: listNUrl
79      |
80      |     Purpose: The listNUrl function returns the request to get N url
81      |        and their data from url table where firstid is the
82      |        first URl selected
83      |
84      |     Arguments: $firstId : first id_url of the N URL
85      |                $arrayParam : array containing parameters for database
86      |
87      |     Returns:     String
88      |
89      *-------------------------------------------------------------------*/
90    function listNUrl($firstId,$arrayParam){
91          $languageSql = strtoupper($arrayParam[languageSql]);
92          $numberUrl = $arrayParam[numberUrlToValidate];
93
94          if ($languageSql == "MYSQL"){
95              $request = "SELECT * FROM url WHERE id_url >= $firstId
96                      ORDER BY id_url LIMIT $numberUrl";
97              return($request);
98          }
99      }
100
101     /*-------------------listUrl-----------------------------------------
102     |    Function: listUrl
103     |
104     |     Purpose: The listUrl function returns the request to get a range
105     |        of data from url table order by averageUrlPoint
106     |
107     |     Arguments : $page : page number to get a range for the visualisation
108     |                 $arrayParam : array containing parameters
109     |
110     |
111     |     Returns:    String
112     |
113     *-------------------------------------------------------------------*/
114   function listUrl($page,$arrayParam){
115
116         $languageSql = strtoupper($arrayParam[languageSql]);
117         $rangeUrl = $arrayParam[rangeUrl];
118         $start = $page*$rangeUrl;
119
120         if ($languageSql == "MYSQL"){
121             $request = "SELECT * FROM url ORDER BY averageUrlPoint
122                     LIMIT $start,$rangeUrl";
123             return($request);
124
125         }
126     }
127
128     /*-------------------getMaxId-----------------------------------------
129     |    Function: getMaxId
130     |
131     |     Purpose: The getMaxId function returns the request to get the id
132     |            maximum in the url table
133     |
134     |     Arguments: $arrayParam : array containing parameters
135     |
136     |
137     |     Returns:     String
138     |
139     *-------------------------------------------------------------------*/
140   function getMaxId($arrayParam){
141
142         $languageSql = strtoupper($arrayParam[languageSql]);
143
144         if ($languageSql == "MYSQL"){
145             $request = "SELECT max(id_url) FROM url";
146             return($request);
147         }
148     }
149
150
151
152
```

```
153     /*-------------------updateUrl-----------------------------------------
154     |   Function: updateUrl
155     |
156     |    Purpose: The updateUrl function returns the request to update the
157     |           url table after an URL validation
158     |
159     |    Arguments: $arrayParam : array containing parameters
160     |               $row : array containing the new URL data
161     |               $statusCode : the status code of current validation
162     |               $responseTime : the response time of the current
163     |               validation
164     |
165     |
166     |    Returns:    String
167     |
168     *-------------------------------------------------------------------*/
169   function updateUrl($row,$statusCode,$responseTime,$arrayParam){
170
171         $languageSql = strtoupper($arrayParam[languageSql]);
172
173         if ($languageSql == "MYSQL"){
174             $var1 = $row[id_url];
175             $var2 = $row[averageUrlPoint];
176             $var3 = $row[averageUrlResponseTime];
177             $var4 = $row[numberUrlValidation];
178             $var5 = $row[numberOfCode200];
179             $var6 = $row[numberOfCode404];
180             $var7 = $statusCode;
181             $var8 = $responseTime;
182             $numberHistories  = $arrayParam[numberHistories];
183
184             $request = "UPDATE url SET averageUrlPoint =
185                 $var2,averageUrlResponseTime = $var3,
186                 numberUrlValidation = $var4,
187                 numberOfCode200 = $var5,numberOfCode404 = $var6
188                 WHERE id_url = $var1";
189             return($request);
190         }
191     }
192
193     /*-------------------insertHistoric-------------------------------------
194     |   Function: insertHistoric
195     |
196     |    Purpose: The insertHistoric function returns the request to insert
197     |           in the historic table the new historic
198     |
199     |    Arguments: $arrayParam : array containing parameters
200     |               $row : array containing the new URL data
201     |               $statusCode : the status code of current validation
202     |               $responseTime : the response time of the current
203     |                   validation
204     |
205     |    Returns:    String
206     |
207     *-------------------------------------------------------------------*/
208   function insertHistoric($row,$statusCode,$responseTime,$arrayParam){
209
210         $languageSql = strtoupper($arrayParam[languageSql]);
211         $var1 = $row[id_url];
212         $var7 = $statusCode;
213         $var8 = $responseTime;
214         $var9 = date("Y-m-d");
215         $var10 = $row[point];
216
217         if ($languageSql == "MYSQL"){
218                 $request = "INSERT INTO historic
                (id_valid,id_url,statusCode,responseTime,
219                     dateValid,point) VALUES
                    ('','$var1','$var7','$var8','$var9','$var10')";
220             return($request);
221         }
222     }
223
224
225
226
```

```
227     /*-------------------deleteByDate-----------------------------------------
228     |   Function: deleteByDate
229     |
230     |    Purpose: The deleteByDate function returns the request to delete
231     |           in the historic table the oldest historics
232     |
233     |    Arguments: $arrayParam : array containing parameters
234     |               $id : id_url selected
235     |               $date : the minimum date to memorize
236     |
237     |
238     |    Returns:    String
239     |
240     *-----------------------------------------------------------------------*/
241   function deleteByDate($id,$date,$arrayParam){
242
243         $languageSql = strtoupper($arrayParam[languageSql]);
244
245         if ($languageSql == "MYSQL"){
246             $request = "DELETE FROM historic WHERE id_url = $id AND
247                         dateValid < $date";
248             return($request);
249         }
250     }
251
252     /*-------------------deleteById-------------------------------------------
253     |   Function: deleteById
254     |
255     |    Purpose: The deleteById function returns the request to delete in
256     |           the historic table the oldest historics
257     |
258     |    Arguments: $arrayParam : array containing parameters
259     |               $id : id_url selected
260     |               $id_valid : the minimum id_valid to memorize
261     |
262     |    Returns:    String
263     |
264     *-----------------------------------------------------------------------*/
265   function deleteByIdvalid($id,$id_valid,$arrayParam){
266
267         $languageSql = strtoupper($arrayParam[languageSql]);
268
269         if ($languageSql == "MYSQL"){
270             $request = "DELETE FROM historic WHERE id_url = $id AND
271                         id_valid < $id_valid";
272             return($request);
273         }
274     }
275
276     /*-----------------getHistoricByIdvalid-------------------------------
277     |   Function: getHistoricByIdvalid
278     |
279     |    Purpose: The getHistoricByIdvalid function returns the request to
280     |           get historic from an URL order by id_valid
281     |
282     |    Arguments: $arrayParam : array containing parameters
283     |               $id : id_url selected
284     |
285     |    Returns:    String
286     |
287     *-----------------------------------------------------------------------*/
288   function getHistoricByIdvalid($id,$arrayParam){
289
290         $languageSql = strtoupper($arrayParam[languageSql]);
291
292         if ($languageSql == "MYSQL"){
293             $request = "SELECT * FROM historic WHERE id_url = $id
294                     ORDER BY id_valid";
295             return($request);
296         }
297     }
298 }
299 ?>
300
```

```php
<?php
/*==========================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-Francois
|        Emails:  jfdubus@hotmail.com
|
|     Language:   PHP 4.0
|     Platform:   NT 4/ Intel
|   To Compile:   --
|
|         File:   Statistic.php4
|     Creation:   10/07/2002
|     Modified:
+-------------------------------------------------------------------------
|
|  Description:  allow to calculate new data of an URL
|
|   Known Bugs:  None
|
 +=========================================================================*/

class Statistic{

    /*-------------------calculateNewData-----------------------------
    |   Function: calculateNewData
    |
    |   Purpose: The calculateNewData function calculates the new URL's
    |        points average, the new URL's response time, update the
    |         number of code 200 or 404 and returns the modified array
    |
    |
    |   Arguments:  $arrayUrl : array containing all data of the URL
    |               $statusCode : status code given by HEAD http request
    |               $responseTime : response time between the send and the
    |               receive of the request in seconds
    |               $arrayParam : array containing all parameters of
    |                   initialisation's file
    |
    |   Returns:  array
    |
    *-------------------------------------------------------------------*/
       function calculateNewData($array,$statusCode,$responseTime,$arrayParam){

        switch ($statusCode) {
                case "200": $point = 1;break;
                case "500" : $point = 0.2;break;
                case "503" : $point = 0.2;break;
                default : $point = 0;
        };

        $points = $array[averageUrlPoint];
        $nbr = $array[numberUrlValidation];
        $rep = $array[averageUrlResponseTime];

        $urlCoeff = $this->calculateCoeff($responseTime,$arrayParam);
        $urlPoint = $point*$urlCoeff;

        $points = (($points*$nbr)+$urlPoint)/($nbr+1);
        $rep = (($rep*$nbr)+$responseTime)/($nbr+1);

        $nbr = $nbr + 1;

        if ($statusCode == "200"){
            $array[numberOfCode200] = $array[numberOfCode200] + 1;
        }

        if ($statusCode == "404"){
            $array[numberOfCode404] = $array[numberOfCode404] + 1;
        }


        $array[averageUrlPoint] = $points;
        $array[numberUrlValidation] = $nbr;
        $array[averageUrlResponseTime] = $rep;
        $array[point] = $urlPoint;
```

```php

        return($array);
    }

    /*-------------------calculateCoeff-------------------------------------
    |   Function: calculateCoeff
    |
    |    Purpose: The calculateCoeff function returns the coefficient to
    |       apply on the URL's point. The coefficient is based on
    |       response time, tolerate response time and timeout.
    |
    |    Arguments: $responseTime : response time in seconds
    |               $arrayParam : array containing all parameters of
    |                   initialisation's file
    |
    |    Returns:  urlCoeff : float
    |
    *-----------------------------------------------------------------*/
    function calculateCoeff($responseTime, $arrayParam){

        $tolerateResponseTime = $arrayParam["tolerateResponseTime"];
        $urlTimeout = $arrayParam["urlTimeout"];

        if ($responseTime < $tolerateResponseTime){

            $urlCoeff = 1;
        }
        else{
            if ($responseTime < $urlTimeout){
                //Y-Ya = (Yb-Ya)/(Xb-Xa)*(X-Xa)
                $urlCoeff = -1/($urlTimeout-$tolerateResponseTime)
                *($responseTime-$tolerateResponseTime)+1;
            }
            else{//$responseTime > $urlTimeout
                $urlCoeff = 0;
            }
        }

    return($urlCoeff);
    }

}
?>
```

```php
<?php
/*=============================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-François
|        Emails:  jfdubus@hotmail.com
|
|     Language:   PHP 4.0
|     Platform:   NT 4/ Intel
|   To Compile:   --
|
|         File:   test_controller.php4
|     Creation:   10/07/2002
|     Modified:
+-----------------------------------------------------------------------------
|
|  Description:  allow to launch the controller
|
|   Known Bugs:  None
|
 +============================================================================*/

include("Controller.php4");
$control = new Controller();
$control->main($a,$b);

?>
```

```php
<?php
/*=============================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-François
|        Emails:  jfdubus@hotmail.com
|
|     Language:   PHP 4.0
|     Platform:   NT 4/ Intel
|   To Compile:   --
|
|         File:   Timer.php4
|     Creation:   10/07/2002
|     Modified:
+-----------------------------------------------------------------------------
|
|  Description:  allow to get the system time
|
|   Known Bugs:  None
|
 +============================================================================*/


class Timer {

    /*--------------------timer--------------------------------------------
    |   Function: readTime
    |
    |    Purpose: The readTime function returns the system time in seconds
    |       with several decimals since the Unix Epoch (0:00:00
    |       January 1, 1970 GMT)
    |
    |    Arguments: none
    |
    |    Returns:  float
    |
    *----------------------------------------------------------------------*/

  function readTime(){
     list($usec, $sec) = explode(" ",microtime());
     return ((float)$usec + (float)$sec);
    }
  }
?>
```

```php
<?php
/*=============================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-Francois
|        Emails:  jfdubus@hotmail.com
|
|     Language:   PHP 4.0
|     Platform:   NT 4/ Intel
|   To Compile:   --
|
|         File:   Validator.php4
|     Creation:   10/07/2002
|     Modified:
+-----------------------------------------------------------------------------
|
|  Description:  allow to launch the validatation
|
|   Known Bugs:  None
|
 +============================================================================*/

/*-----------------INCLUDE----------*/

include ("Timer.php4");
include ("Network.php4");
include ("Statistic.php4");


class Validator {

    /*-----------------validate-----------------------------------------
    |   Function: validate
    |
    |    Purpose: The validate function launchs the validation
    |
    |    Arguments: $arrayParam : array containing all parameters of
    |                initialisation's file
    |
    |
    |    Returns:   id_url : the last id validated
    |
    *-------------------------------------------------------------------*/
  function validate($arrayParam){

    $timer = new Timer();
    $startTime = $timer->readTime();

    $numberUrl = $arrayParam[numberUrlToValidate];
    $validationTime = $arrayParam[validationTime];

    $error = new Error();


    $query = new QueryBuilder();


    $mydal = $this->loadDal($arrayParam);
    $mydal2 = $this->loadDal($arrayParam);

    ////////////////////////////////////////////////////////////////
    //get the next id_url to validate
    ////////////////////////////////////////////////////////////////

    $idStart = $arrayParam[idLastValidateUrl] + 1;

    $request = $query->getMaxId($arrayParam);

    if (!$mydal->executeQuery($request)) {
        echo $mydal->lastError;
    }

    while ($bar=$mydal->fetchArray()){
        while (list ($cle,$val) = each($bar)){
            $max = $val;
            }
```

```
77      }
78      if ($max < $idStart){
79          $idStart = 0;
80      }
81
82      ////////////////////////////////////////////////////////////////
83      //get N URL to validate
84      ////////////////////////////////////////////////////////////////
85
86      $request = $query->listNUrl($idStart,$arrayParam);
87
88      if (!$mydal->executeQuery($request)) {
89          echo $mydal->lastError;
90      }
91
92      $net = new Network();
93
94      $stat = new Statistic();
95
96      while($rowUrl = $mydal->fetchArray()){
97
98              $url = $rowUrl[url];
99              $urlId = $rowUrl[id_url];
100
101
102             ////////////////////////////////////////////////////////////
103             //get protocol, host and path of the complete url
104             ////////////////////////////////////////////////////////////
105             $url=str_replace("\\", "/", $url);
106
107             $array=explode("/",$url);
108             $protocol = $array[0];
109
110             //delete ':' in protocol
111             $protocolTemp = explode(":",$protocol);
112             $protocol = $protocolTemp[0];
113             $host = $array[2];
114             $nbr=sizeof($array);
115             $nbr=$nbr-3;
116
117             $path="";
118             for($i=0;$i<$nbr;++$i){
119
120                 $tel = 3+$i;
121                 $path=$path."/$array[$tel]";
122             }
123             $time1 = $timer->readTime();
124
125             ////////////////////////////////////////////////////////////
126             //open a socket
127             ////////////////////////////////////////////////////////////
128             if (strtoupper($protocol)=="HTTP"){$port=80;}
129
130             $socket = $net->openSocket($protocol,$host,$port);
131
132             if ($socket != false){
133                 //get status code
134                 $statusCode = $net->getStatusCode
135                         ($protocol,$socket,$path);
136                 //close socket
137                 $net->closeSocket($socket,$protocol);
138
139                 $time2 = $timer->readTime();
140                 $responseTime = $time2-$time1;
141
142                 //if path is a directory then terminate it by
143                 //'/' and do validation again
144                 if($statusCode!="200" && $statusCode!="404"){
145                     $path = $path."/";
146
147                     $time3 = $timer->readTime();
148                     //open a socket
149                     $socket = $net->openSocket($protocol,
150                                 $host,$port);
151
152                     //get status code
```

```
153                     $statusCode = $net->getStatusCode
154                         ($protocol,$socket,$path);
155                     //close socket
156                     $net->closeSocket($socket,$protocol);
157
158                     $time4 = $timer->readTime();
159                     $responseTime = $time4-$time3;
160                 }
161                 $rowUrl2 = $stat->calculateNewData($rowUrl,
162                     $statusCode,$responseTime,$arrayParam);
163
164                 echo("<BR>id_url : $rowUrl2[id_url] , url :
165                      $rowUrl2[url] , averageUrlPoint :
166                      $rowUrl2[averageUrlPoint] ,
167                      averageUrlResponseTime :
168                      $rowUrl2[averageUrlResponseTime] ,
169                      numberUrlValidation :
170                      $rowUrl2[numberUrlValidation],
171                      numberOfCode200 :$rowUrl2[numberOfCode200]
172                      , numberOfCode404 :
173                      $rowUrl2[numberOfCode404] , statusCode :
174                      $statusCode , responseTime :
175                      $responseTime<BR>");
176
177                 $var1 = $rowUrl2[id_url];
178                 $numberHistories= $arrayParam[numberHistories];
179
180                 //update the url table
181                 $request1 = $query->updateUrl($rowUrl2,
182                     $statusCode,$responseTime,$arrayParam);
183                 if (!$mydal2->executeQuery($request1)) {
184                     echo $mydal2->lastError;
185                 }
186
187                 //insert into historic table the present
188                 //validation data
189                 $request2 = $query->insertHistoric($rowUrl2,
190                     $statusCode,$responseTime,$arrayParam);
191                 if (!$mydal2->executeQuery($request2)) {
192                     echo $mydal2->lastError;
193                 }
194
195                 //Check if not too much historics then delete by date
196                 $request3 = $query->getUrlHistoricByDate($var1,
197                             $arrayParam);
198                 if (!$mydal2->executeQuery($request3)) {
199                     echo $mydal2->lastError;
200                 }
201
202                 $count = $mydal2->rowcount;
203                 $difference = $count - $numberHistories;
204
205                 if ($difference > 0){
206
207                     for ($i=0;$i<=$difference;++$i){
208                         $resultrow = $mydal2->fetchArray();
209                     }
210                     $idvalid = $resultrow[id_valid];
211
212                     $datevalid = $resultrow[dateValid];
213                     $request4 = $query->deleteByDate($var1,
214                                 $datevalid,$arrayParam);
215                     if (!$mydal2->executeQuery($request4)){
216                         echo $mydal2->lastError;
217                     }
218                 }
219
220                 //Check if not too much historics then delete by id_valid
221                 $request33 = $query->getHistoricByIdvalid
222                             ($var1,$arrayParam);
223                 if (!$mydal2->executeQuery($request33)) {
224                     echo $mydal2->lastError;
225                 }
226
227                 $count = $mydal2->rowcount;
228                 $difference = $count - $numberHistories ;
```

```
                if ($difference > 0){

                    for ($i=0;$i<=$difference;++$i){
                        $resultrow = $mydal2->fetchArray();
                    }
                    $idvalid = $resultrow[id_valid];

                    $datevalid = $resultrow[dateValid];

                    $request4 = $query->deleteByIdvalid(
                        $var1,$idvalid,$arrayParam);
                    if (!$mydal2->executeQuery($request4)){
                        echo $mydal2->lastError;
                    }
                }

            }
            else{
                $error->displayErrorMessage("errorOpenSocket","");
            }

            $currentTime = $timer->readTime();
            if (($startTime + ($validationTime*60) ) < $currentTime){
                break;
            }

    }
        $endTime = $timer->readTime();
        $currentValidationTime = $endTime-$startTime;
        echo("<BR>Temps de la validation : $currentValidationTime<BR>");

    return($rowUrl2[id_url]);

}


    /*-------------------loadDal-----------------------------------------------
    |   Function: loadDal
    |
    |    Purpose: The loadDal function creates a dal with parameters
    |
    |    Arguments: $arrayParam :array containing all parameters of
    |               initialisation's file
    |
    |
    |    Returns: a DAL object
    |
    *-------------------------------------------------------------------------*/

  function loadDal ($arrayParam){
        $langue = strtolower($arrayParam[languageSql]);

        $foo = new dal;
        $foo->selectDriver($langue);

        if (!$foo->addConfiguration('conf3', $arrayParam[dbUsername],
                        $arrayParam[dbPassword])) {
            echo $foo->lastError;
          }

        if (!$foo->selectConfiguration("conf3")) {
            echo $foo->lastError;
            }

        if (!$foo->selectDatabase($arrayParam[dbName])) {
             echo $foo->lastError;
            }

        $foo->enableFakeTransactions();
        return($foo);
    }
}
?>
```

```php
<?php
/*=============================================================================
|   Assignment:  PHP
|
|       Authors:  Dubus Jean-François
|        Emails:  jfdubus@hotmail.com
|
|     Language:   PHP 4.0
|     Platform:   NT 4/ Intel
|   To Compile:   --
|
|         File:   Visualisation.php4
|     Creation:   10/07/2002
|     Modified:
+-----------------------------------------------------------------------------
|
|  Description:  allow the visualisation of the URL's statistic
|
|   Known Bugs:  None
|
 +============================================================================*/

class Visualisation {

    /*-------------------visualize-----------------------------------------
    |   Function: visualize
    |
    |    Purpose: The visualize function displays to interface a choice of
    |              the URL with their points averages and displays the
    |              statistics of the chosen URL
    |
    |    Arguments: $mode : choice between listing or historic
    |        $param : if mode is the listing then param is the page to display
    |                 if mode is historic then param is the id_url
    |        $arrayParam : array containing the application parameters
    |
    |    Returns: resultset coming from Database or an array containing
    |              two resultsets
    |
    *-------------------------------------------------------------------------*/
  function visualize($mode,$param,$arrayParam){

    $query = new QueryBuilder();

        $mydal = $this->loadDal ($arrayParam);
        $mydal2 = $this->loadDal ($arrayParam);
        $mydal3 = $this->loadDal ($arrayParam);

    switch ($mode){

        case"2" :   //listing param = first id_url to display
                    $request = $query->listUrl($param,$arrayParam);

                    if (!$mydal->executeQuery($request)) {
                            echo ("<BR>$mydal->lastError<BR>");
                        }

                    return($mydal);
                    break;

        case"3" :   //graphics param = id_url
                    $request1 = $query->getUrlData($param,$arrayParam);
                    $mydal->executeQuery($request1);

                    $request2 = $query->getUrlHistoricByDate($param,$arrayParam);
                    $mydal2->executeQuery($request2);
                    $mydal3->executeQuery($request2);

                    $data[0] = $mydal;
                    $data[1] = $mydal2;
                    $data[2] = $mydal3;

                    return($data);
                    break;
      }
    }
```

```
77
78      /*--------------------loadDal-----------------------------------------
79      |   Function: loadDal
80      |
81      |   Purpose: loadDal function loads the differents drivers to use DAL
82      |
83      |   Arguments: $arrayParam :array containing the application parameters
84      |
85      |
86      |   Returns:  a DAL object
87      |
88      *-----------------------------------------------------------------------*/
89    function loadDal ($arrayParam){
90          $langue = strtolower($arrayParam[languageSql]);
91          $foo = new dal;
92          $foo->selectDriver($langue);
93
94          if (!$foo->addConfiguration('conf3', $arrayParam[dbUsername],
95                          $arrayParam[dbPassword])) {
96                echo $foo->lastError;
97            }
98
99          $foo->selectConfiguration("conf3");
100
101         if (!$foo->selectDatabase($arrayParam[dbName])) {
102              echo $foo->lastError;
103             }
104
105         $foo->enableFakeTransactions();
106         return($foo);
107     }
108
109 }
110 ?>
```