Université Saad Dahleb de Blida

Faculté des sciences

Département d'Informatique

En vue d'obtenir le diplôme de Master

Domaine : Mathématique et informatique
Filière : Informatique
Spécialité : Ingénierie du logiciel

Sujet : Un Composant de Validation Pour les applications de l'eGouvernement

Mémoire Présenté par :
GUEDDOUN Messaouda

Sous la direction du promoteur
M^r^. BENNOUAR Djamal

Soutenu le 01/07/2012 devant le jury composé de :

Mr. CHERIF ZAHAR	Président
Mr. SIDOUMOU	Examinateur
Mr. BAOYA	Examinateur

2011/2012

Remerciements

Avant tout, nous remercions Dieu, le miséricordieux,
Le Tout Puissant
Pour nous avoir donné la force et le courage d'accomplir ce travail avec beaucoup de courage et d'abnégation.

Nos vifs et sincères remerciements
S'adressent spécialement à,

M^R. Djamal Bennouar,

La chance a voulu qu'il soit notre Professeur et notre Promoteur durant notre cursus universitaire à l'université de Blida,
Qu'il nous soit permis de lui réserver à notre tour, un chaleureux accueil,
pour son estime, pour son écoute attentive et son suivi au peigne fin de toutes les étapes et les démarches accomplies dans ce travail.
Ses talents ainsi que ses compétences et son sens du devoir nous ont subjugués à jamais.
Qu'il trouvera ici l'expression de notre grande estime et notre grande reconnaissance...

Aussi,
Nous adressons nos vifs remerciements à
M^r Baouya Abdelhakim, M^elle Guessoum Dalila pour leurs aides...

Nous remercions également,
Les membres du jury de nous avoir fait l'honneur d'expertiser ce mémoire.
Veuillez accepter l'expression de notre vive gratitude et notre parfaite déférence.

Nous rendons aussi un vibrant hommage à tout nos Professeurs,

Nous saisissons cette occasion pour vous exprimez notre profonde reconnaissance tout en vous témoignant notre respect et notre humble dévouement.

Enfin,
A toutes les personnes qui ont contribué de prés ou de loin, d'une manière directe ou indirecte à l'élaboration de ce travail de fin d'études.

....

Dédicaces

A ma mère,

L'école de la vie qui m'a enseigné mes premiers pas. Maman, je ne connais pas une personne aussi adorable, tendre que toi, tu as été toujours à mes côtés comme un ange guidant mes pas à chaque moment de ma vie, me couvrant de ta tendresse et de ton amour éternel.

A mon père,

Tous les mots du monde ne sauraient exprimer l'immense amour que je te porte, ni la profonde gratitude que je te témoigne pour tous les efforts et les sacrifices que tu n'as cessé de consentir pour mon instruction et mon bien-être.

J'espère

Avoir répondu aux espoirs que vous avez fondés en moi.
Je vous rends hommage ici par ce modeste travail en guise de ma reconnaissance éternelle à jamais...
Que Dieu, le Tout Puissant vous garde et vous procure santé, bonheur et longue vie.

A

La mémoire de deux êtres qui me sont très chers

Mon grand père paternel Mahiedine et ma grand-mère maternelle Fatima que malheureusement ne sont pas parmi nous et de ce monde.
Qu'ils reposent en paix, et le que le paradis soit leur dernière demeure (Inchallah).
Mon grand-père maternel Ali Graine et ma grand-mère paternelle, qui sont la richesse de notre famille,
Que Dieu leur accorde une longue vie.
Ma tante Kaissa, Nana Melha et Nana Ouiza, je n'oublierai jamais vos conseils.
Mon oncle Dada Rachid et toute sa famille. Merci de m'avoir soutenu et aidé.
Mon très cher frère, Hassen et mes deux anges, Belkacem et Samy,
Que Dieu vous garde auprès de nous.
A mes cousins dont je cite Tayeb, qui n'a cessé de m'orienter et de me prodiguer ses meilleures visions et ses meilleurs conseils dans la façon d'accomplir certaines taches de la vie, sans oublier pour autant le soutien incontesté de mes cousines.

Toute ma famille,

Grands et petits qui attendent ce travail avec impatience,
Trouvez ici l'expression de ma profonde affection et mon respect.

Mes professeurs,

Qui m'ont enseigné depuis ma première scolarité.
Trouvez ici l'expression de mes respects et mon éternelle reconnaissance.

Mes amies,

Que tous vos rêves soient exaucés et la réussite comble votre vie.

Mlle. Messaouda Gueddoun

ملخص

يستوجب على الهيئات الحكومية و التنظيمية إعطاء معلومات صحيحة و موثوق فيها لأنه لا حق لهم في ارتكاب خطأ نظرا لأهمية و طبيعة الوثائق المنقولة و كذلك لأن التصحيح قد يستغرق وقتا طويلا نوعا ما و يستدعي إجراءات إدارية و قانونية يمكن أن تكون جد معقدة. و بالتالي فإن هذه الهيئات مرغمة على توفير الوسائل و تنظيم الإجراءات اللازمة للتحقق من صحة المعلومة قبل أن تنشر في وثائق رسمية ورقية كانت أو رقمية .

و يتجلى هذا العمل في تهيئة مركب برنامج يتيح إدارة عملية التحقق من صحة أي نوع من المعلومات و ستكون كل معلومة مصادق عليها من خلال هذا المركب جد موثوق فيها و خالصة من الأخطاء. لن تزيد هذه الجودة في المعلومة المواطن إلا ثقة في البرامج المشغلة لمركب كهذا خاصة برامج eGouvernement.

Résumé

Les organismes gouvernementaux et réglementaires sont toujours menés à délivrer des informations valides et hautement fiable, car ils n'ont pas le droit a l'erreur vu l'importance et la nature des documents qu'ils délivrent, aussi le fait que la correction d'une information pourraient nécessité un temps assez long et des procédures administratives et juridiques pouvant être très complexe. Ces organismes sont obligés de mettre en place tout les outils et procédures nécessaire pour vérifier la véracité de l'information avant de la délivrer dans des documents officiels qu'ils soient sur papier ou sur support numérique.

Ce travail se concentre sur la mise en place d'un composant logiciel qui permet de gérer les processus de validation de tout type d'information. Toute information validée dans le contexte de ce composant sera une information hautement fiable exempt d'erreur. Une telle qualité d'information ne fera qu'augmenter la confiance du citoyen dans les logiciels exploitant d'un tel composant notamment les logiciels d'eGouvernement.

Mot clés : *architecture logicielle, approche par composant, composant logiciel, connecteur, interfaces, langage de description d'architecture, validation d'information*

Abstract

The government and regulatory bodies should always give valid and very reliable information. They do not have the right to be mistaken, due to the importance and the nature of the document they deliver. The correction of information could take a long time and require very complex administrative and legal procedures.

These bodies must set up everything needed to check the veracity of information before issuing it in official documents on paper or digital (media).

This work is concentrated on the establishment of a software component allowing managing the validation process of all types of information. Information validated using this component will be very liable and error free. Such quality of information will increase public confidence in software operating with this component especially "eGouvernement ".

Table des Matières

Chapitre I : Introduction Générale

Chapitre II : Architecture logicielle et approche par composant

Chapitre III : Analyse des besoins

Chapitre IV : Conception

Chapitre V : Implémentation et Testes

Conclusion et Perspectives

Bibliographie

Table des Figures

Chapitre II : Architecture logicielle et approche par composant

Chapitre III : Analyse des besoins

Chapitre IV : Conception

Chapitre IV : Implémentation et Testes

I

Introduction Générale

Introduction
Problématique
Objectifs

1. Introduction

Aujourd'hui, l'informatique est devenue omniprésente. Elle tend à couvrir tous les secteurs de la vie humaine et devient accessible de partout et à n'importe quel moment. Dans ce contexte, un bon nombre d'applications s'intéressent aux préoccupations du citoyens, notamment celles en relation avec les divers services des diverses institution gouvernementales « *locales régionales, central* » et même les institutions privées jouant un rôle important dans la vie de la citoyenne « santé, assurances, transport etc.... ». Ces applications sont souvent classifiées comme étant des applications du domaine de l'eGouvernement.

L'eGouvernement a pour objectif de rendre les services publics, et les services orientés citoyen gérés par des privé, plus accessibles aux citoyens et d'améliorer le fonctionnement interne de l'État pour éliminer les divers goulots d'étranglement du système administratif de l'état du notamment à la bureaucratie et parfois à l'incompétence et l'incompréhension des lois et des règlements. Il ne s'agit pas du gouvernement « traditionnel » auquel on aurait rajouté l'Internet mais d'un processus radical de changement de la manière dont l'État travaille et communique[1]. L'eGouvernement est un vecteur de nouveaux espoirs. Les dirigeants en attendent des bienfaits spectaculaires, notamment:

- L'amélioration de la qualité et de la disponibilité des services publics;
- Une plus grande transparence des institutions;
- Une participation démocratique accrue;
- Une gestion plus simple et efficace des ressources;
- Une réduction significative des coûts administratifs.

[1] **OCDE – mars 2002 : Confiance envers l'administration électronique - perspectives pour un cadre juridique des données personnelles et de la vie privée,** Georges CHATILLON

2. Problématique :

Malgré leurs promesses, les systèmes de l'eGouvernement font face à des défis très important qui réduisent la confiance des citoyens et augmentent leurs méfiances. Des exemples très simples en Algérie montrent le résultat de la méfiance et de l'absence de confiance. A titre d'exemple, quel est vraiment le taux de personnes qui paient leur facture d'électricités ou d'eau à travers les chèques postaux ou la banque ?

Dans ces exemples la méfiance vient de la défaillance des systèmes qui n'arrivent pas à transmettre à temps l'information indiquant que telle ou telle personne a payé sa facture d'eau ou d'électricité.

Une autre cause de méfiance ou d'absence de confiance, serait du à la nature de l'information elle-même :

- Est-elle réellement correcte ?
- A-t-elle été saisie correctement sans erreur ?

Si nous nous fions à ce qui se raconte au niveau des citoyens, il faut vraiment bien contrôler lors du retrait d'un document officiel par exemple lors du retrait d'un extrait de naissance, d'un certificat de scolarité, d'un permis de conduire, d'une carte grise, certificat médical, analyses médicales etc.…

Si l'information a été mal reportée. Une erreur ne pourrait parfois être corrigée que par décision de justice nécessitant un temps qui n'est nullement court. De plus, n'a-t-on pas entendu dire que telle ou telle note d'étudiants aurait été modifiée à l'insu de l'enseignant. Telle ou telle déclaration aurait été modifiée à l'insu du déclarant.

Pour recouvrir la confiance des citoyens et la maintenir, voir la renforcer, il est nécessaire que les systèmes de eGouvernement ne soient pas défaillants en ce qui concerne la véracité de l'information qu'il gère et aussi les processus qu'il supporte. Il faut que les systèmes d'eGouvernement soient dotés de capacités (mécanismes, méthode, technique etc..) qui réduisent à néant ces sources de problème. Ces capacités doivent veiller à ce que le système d'eGouvernement ne se trompe pas sur les informations qu'ils gèrent et sur les documents qu'il produit, notamment les informations personnelles. En plus de

cela, il faut que le système de eGouvernement soit capable de situer les responsabilités sur les diverses actions menées notamment les actions de transformation de l'information.

3. Objectifs

Le travail présenté dans ce mémoire s'inscrit dans l'optique de pourvoir les systèmes de eGouvernement de capacités leur permettant de gérer et produire une information correcte et hautement fiable. Ces mécanismes permettraient en plus d'aider à situer les responsabilités dans me cas de la dotation d'un système de eGouvernement de capacité de traçages d'actions importantes, notamment celles relatives à la transformation d'informations personnelles.

Mais que veut ton dire par information correcte et hautement fiable. Dans notre travail, une information correcte et hautement fiable est une information qui est déclaré vraie et correcte après avoir subi des opérations de validation / correction dans le contexte d'un processus de validation pouvant être très complexe. De plus, les concernés par ces informations (citoyens par exemple) peuvent intervenir avant la déclaration de la véracité de l'information. Une fois l'information déclarée vraie, elle ne devient corrigeable (cas très rare) que dans le contexte d'un processus d'invalidation dans lequel tous les acteurs ayant participé à la validation invalide l'information.

Ainsi, notre travail consiste à réaliser un sous système de validation de l'information. Cependant, une autre contrainte a été imposé pour le système a réaliser : Ce système doit être hautement réalisable, donc totalement indépendant de tout système de eGouvernement. Il doit être autonome. Parfois nous parlons même de système inconscient de son entourage.

Actuellement, il y a un consensus de plus en plus croissant, sur le fait que les approches à composant, notamment l'approche architecture logicielle, représentent actuellement l'approche la plus prometteuse permettant de produire des sous systèmes réutilisables. Ces sous systèmes sont appelés des composant. La réutilisation d'un composant est assurée par le modèle de composant lui-même. Un composant en architecture logicielle explicite les interfaces fournies

et les interfaces requises. Les composants implémentent les interfaces fournies et indique les types de services auxquels il doit être connecté pour bien fonctionner

C'est ainsi, que l'objectif final de notre travail est la conception et la réalisation d'un composant permettant la validation de l'information. Ce composant devra facilement s'insérer dans une conception orientée composant de système d'eGouvernement, ou tout autre système nécessitant la construction d'une base d'information correcte et hautement fiable. Le composant doit se focaliser fondamentalement sur la fonction de validation et ne doit en aucun cas se soucier des autres aspects, tel que la sécurité, la gestion des transactions, l'interfaçage homme machine etc. Il doit être inconscient. L'association des aspects techniques sera laissée aux applications qui utilisent le composant. La conception orientée aspect est une des approches permettant d'injecter dans tout composant les aspects techniques nécessité par l'application. Les conteneurs, tel les serveurs d'application de type J2EE, représentent une autre source à travers laquelle les aspects techniques sont associés aux composant

La suite de ce mémoire est organisée comme suit

Dans le chapitre 1, nous présenterons les éléments qui vont servir à la conception du composant. Ces éléments représentent une introduction aux approches à composant et à l'architecture logicielle. Par la suite, au chapitre 2, nous étudierons en détail la fonctionnalité de validation de l'information. Nous déterminerons en fin de ce chapitre la vue externe du composant de validation Cette vue externe sera plus précisée lors de la conception de ce composant qui sera présentée au chapitre 3. Le chapitre 4 sera consacré à l'implémentation et le test du composant. Nous terminerons ce mémoire par une conclusion

II

Architecture logicielle
Et Approche par Composant

Introduction
Architecture logicielle
Approche par composant
Conclusion

1. Introduction :

Les systèmes informatiques connaissent une croissance exponentielle aussi bien en termes de taille que de complexité. Pour concevoir des systèmes de plus en plus complexes, les langages de programmation ont gagné en pouvoir d'expression et en abstraction ainsi des approches de maintenance ont également été proposées afin de maintenir ces systèmes. Mais reste des défaillances importantes, en prenant le cas de la réutilisation, la robustesse face au changement, la difficulté de maintenance, ce qui a pousser les chercheurs à réfléchir a une nouvelle solution pour palier a ces problèmes.

Les architectures logicielles constituent « LA » solution, aussi bien pour la conception que la maintenance de ces systèmes informatiques complexes.

Ainsi l'approche composant est apparue pour pallier les défauts de l'approche objet, cette approche est fondée sur des techniques et des langages de construction des applications qui intègrent d'une manière homogène des entités logicielles provenant de diverses sources.

Ce chapitre sera consacre a l'architecture logicielle et aux approches par composant nous allons exposer une vue globale sur l'architecture logicielle, les approches a base de composant et ses concepts puis nous allons finir avec une conclusion.

2. Architecture logicielle :

Dans ce qui suit nous allons présenter les concepts d'architecture logicielle

2.1.Définition :

L'architecture logicielle d'un système est un élément essentiel de chaque phase du cycle de vie du logiciel [AA, 97]. Elle est utilisée pour concevoir le système, évaluer la qualité d'un système, le documenter ou encore pour orienter les phases de maintenance.

Les recherches ont permis de préciser la définition des architectures et des éléments architecturaux : *les composants, les connecteurs et la configuration.*

Elles ont également démontré les avantages et l'étendue des utilisations possibles de ces architectures logicielles [MD, 96].

Ces avantages ont été décrits, aussi bien dans le cadre de la conception [DD, 95], que dans celui de la maintenance [RK, 00]. Ainsi, l'architecture permet, par exemple, de faciliter la compréhension du système ou de l'analyser.

Les architectures logicielles sont représentées et analysées par un moyen de langage de description d'architecture (ADL, *Architecture Description Languages*).

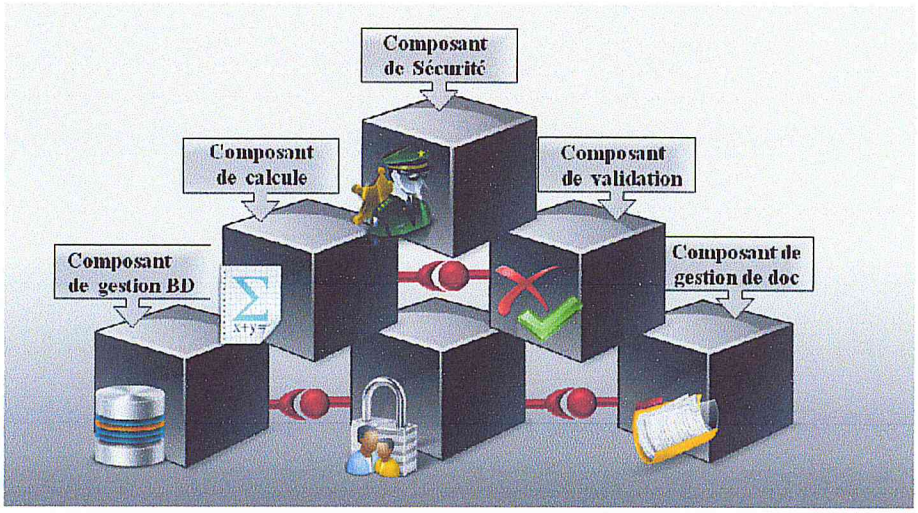

Figure 1: Vue générale sur l'architecture logicielle

Cette introduction donne une définition générale sur l'architecture logicielle, mais il n'y a pas une définition qui englobe tout les concepts. Pour obtenir une vue plus détaillé, la meilleure solution est de considérer plusieurs définition à la fois. Parmi ces dernières, nous citons :

« Une architecture logicielle définie la structure d'une application en précisant les parties (composants) qui la constituent, les connecteurs qui modélisent les interactions entre ces parties et les règles qui gouvernent ces interactions » [DG, 00].

« L'architecture logicielle implique la description des éléments à partir desquels les systèmes sont construits, les interactions entre ces éléments, les

patrons (patterns) qui guident leur composition et les contraintes sur ces patterns. » [SG, 96].

« L'architecture est une vue abstraite d'un système en terme d'éléments architecturaux. Ces éléments sont :

- Les composants qui décrivent les fonctionnalités métier de l'application ;
- Les connecteurs qui décrivent les communications et connexions entre les composants ;
- La configuration qui décrit la topologie des connexions entre composants et connecteurs. » [SH, 09].

2.2. Concepts d'Architecture Logicielle :

Les concepts fondamentaux d'une architecture logicielle sont les composants, les connecteurs, les ports et les configurations. Ces trois concepts sont brièvement décrits ci-dessous.

2.2.1. Les Composants :

Un composant logiciel est une unité d'encapsulation et de réutilisation. Il constitue la brique de base dans le processus d'élaboration de l'architecture d'une application. Ce concept, représenté par une boite noire, a toujours existé au niveau des modèles mentaux des architectes de logiciel et dans les spécifications informelles de logiciels [BEN, 09] *(Figure 2)*

Cette première définition donne une introduction simple et très significative d'un composant logiciel, et pour éclaircir la boite noire nous allons exposer de nouvelles définitions plus détaillé :

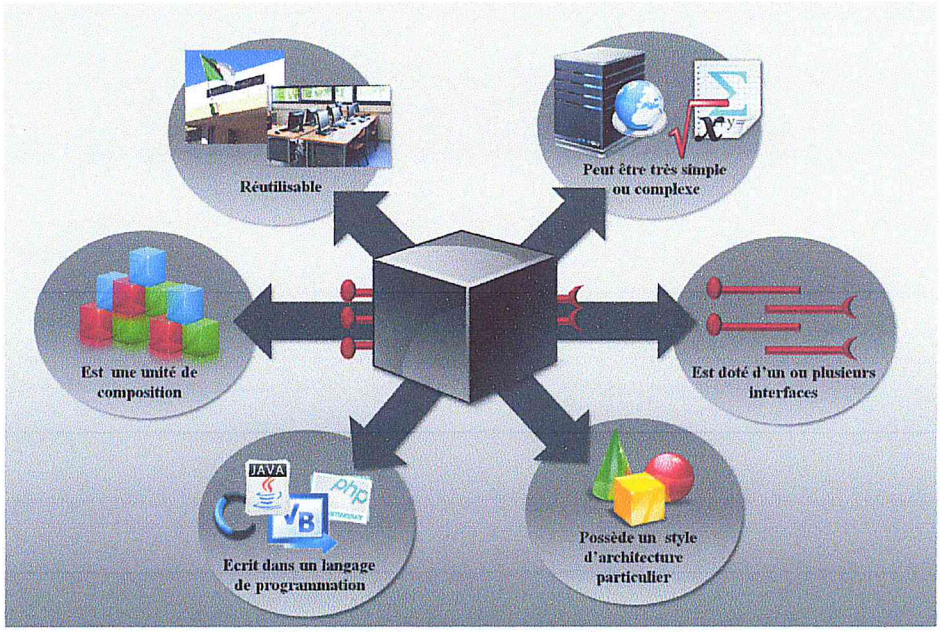

Figure2: Le composant logiciel

« Un composant logiciel est une unité de composition possédant des interfaces spécifiées par contrat et des dépendances contextuelles explicites. Un composant logiciel peut être déployé indépendamment et est sujet à la composition par un tiers.» [CS, 98]

« Un composant est avant tout une unité de composition, développé indépendamment du logiciel dans lequel il sera utilisé. Il peut cependant posséder des caractéristiques qui le rendent réutilisable dans un environnement particulier (i.e. un style d'architecture particulier [MED, 99], une technologie d'implémentation bien précise [OMG, 02]) »

« Un composant est une entité qui fournit des fonctionnalités de calcul et de stockage.»[KKO, 03].

« Un composant peut être très simple, comme une petite fonction C++, ou complexe comme un serveur HTTP ou un SGBD » [BEN, 07]

« Un composant est doté d'un ou plusieurs ports (ou interfaces) à travers lesquelles il exprime ses besoins (services requis) et exporte ses services (services fournis) et ses états » [RRN, 04]

« Un composant définit les conditions à remplir (sorte de contrat) pour une exploitation correcte » [SZY, 02].

« Un composant est une entité logicielle qui fournit un service particulier via une interface séparée de l'implantation mettant en œuvre ce service. » [CA, 02]

2.2.2. Les Connecteurs :

Les connecteurs constituent un élément architectural au même titre que les composants. Appelés selon notre terminologie connexion, les connecteurs correspondent à des éléments d'architecture qui modélisent les interactions entre deux ou plusieurs composants en définissant les règles qui gouvernent ces interactions [CC, 03]. A la différence des composants, les définitions des connecteurs sont peu nombreuses et relativement convergentes. Par exemple celle du [SG, 96] :

« Les connecteurs gèrent les interactions entre les composants ; c'est-à-dire, ils établissent les règles qui gouvernent les interactions entre composants et spécifient tous les mécanismes auxiliaires nécessaires. »

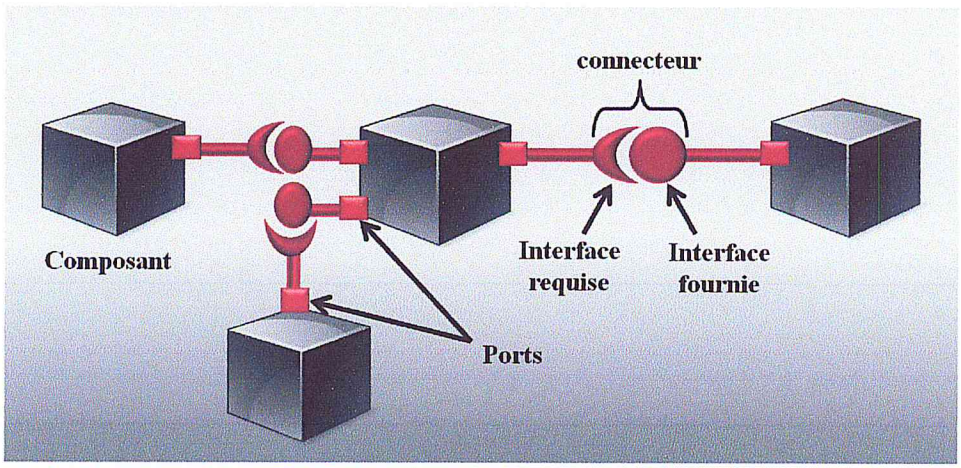

Figure3: Connecteurs, Interface

D'une manière générale, les connecteurs décrivent les communications et les connexions entre les composants en modélisant de manière explicite les interactions entre les composants. Un connecteur peut

décrire des interactions simples comme un appel de méthode ou des interactions plus élaborées telles que des accès à des bases de données.

2.2.3. Les Interfaces :

C'est le point de communication qui permet d'interagir avec l'environnement. On la retrouve donc associée aux composants et aux connecteurs *(figure 4)*

Pour un composant, il existe deux types d'interfaces [MHK, 09]:

- *Les interfaces fournies* décrivent les services proposés par le composant, elles sont représentés par un cercle *(figure 4)*.
- *Les interfaces requises* décrivent les services que les autres composants doivent fournir pour le bon fonctionnement du composant dans un environnement particulier, elles sont représenté par un demi-cercle *(figure 4)*.

2.2.4. La Configuration :

Les composants et les connecteurs peuvent être assemblés à partir de leurs interfaces pour former une configuration. Celle ci décrit l'ensemble des composants logiciels nécessaires pour le fonctionnement d'une application, ainsi que leurs connexions. On parle aussi de topologie.

Une configuration représente en fait une instance possible d'un style architectural. Plus précisément, une configuration décrit les instances de composants intervenants et les relations qu'elles entretiennent entre elles [JNJ, 94].

2.2.5. Langage de description d'architecture logicielle « ADL »:

Un ADL est utilisé pour décrire une architecture logicielle. Il peut être un langage textuel, graphique, ou une combinaison des deux. Il fournit une

syntaxe concrète *(notations et sémantiques)* explicitant les composants, les connecteurs et la configuration architecturale (structure) d'une application [MT, 00]

Au fil des années plusieurs ADLs ont été mis en œuvre de sorte que chaque ADL a ses propres caractéristiques, fonctions et objectifs.

Parmi ces ADLs, nous citons :

- UML 2.0
- Le modèle Wright [All, 97],
- Le modèle IASA (*Integrated Approach to Software Architecture*),
- Le modèle Fractal de France Telecom R&D et par l'INRIA [PS, 04],
- Le modèle C2 [MORT, 96],
- Le modèle AADL [SAE, 04], etc.

Le grand avantage d'utiliser un ADL réside dans la capacité de préciser rigoureusement l'architecture globale d'un système qui peut être ainsi analysée. Un ADL est destine à être a la fois humain et lisible par la machine et il offre un haut niveau d'abstraction. Ainsi il assure une meilleure communication entre le concepteur, les exécutants et les lecteurs, et ne laisse aucune place à l'ambiguïté. C'est un plan directeur pour la conception et peut prendre en charge la génération automatique de parties du logiciel [SH, 10].

2.3.Caractéristiques d'un composant logiciel :

Un composant logiciel possède plusieurs caractéristiques, il est important de noter que la description des caractéristiques est générique et qu'un modèle à composant particulier peut présenter des variations de ces caractéristiques.

2.3.1. L'interface dans un composant logiciel : [KM, 04]

L'interface d'un composant résume les propriétés qui sont visibles depuis son environnement extérieur. Elle va lister les différentes signatures des opérations fournies par le composant. Cette liste va permettre d'éviter les erreurs de typage au niveau des connexions du composant puisque les composants se connectent *via* leurs interfaces (*via* leurs ports).

Techniquement, une interface est un ensemble d'opérations nommées qui vont être invoquées par le client. La sémantique de chaque opération est spécifiée. Cette spécification joue un double rôle car elle permet :

- au fournisseur d'implanter le composant ;
- au client de pouvoir utiliser l'opération.

Le fournisseur et le client ne se connaissent pas, la spécification de l'interface devient donc le médiateur qui permet aux deux parties de travailler ensemble. Un composant peut :

- soit fournir directement des interfaces qui correspondent aux interfaces procédurales des librairies classiques;
- soit implanter des objets qui, s'ils sont accessibles par les clients, fournissent des interfaces. Ces interfaces implantées indirectement correspondent aux interfaces des objets contenus dans le composant.

2.3.2. L'implantation d'un composant logiciel: [KM, 04]

L'implantation d'un composant est la réalisation exécutable du composant, obéissant aux règles du modèle de composant.

Dépendant du modèle de composant, l'implantation du composant peut être fournie en code compilable C, en forme binaire, en code java, php, etc. Il n'est pas nécessaire qu'un composant contienne seulement des classes, il peut:

- contenir simplement des procédures classiques et des variables globales,
- être écrit entièrement dans une approche de programmation par fonctions,
- utiliser un langage assembleur.
- Il peu être un composant composite, c'est-à-dire a son tour il se compose d'un ensemble de composant

2.3.3. Composition ou assemblage de composant : [GS, 05]

La composition est réalisée comme une description qui résulte dans la création, la configuration et la connexion d'instances de connexions d'instances de composant.
Elle contient donc l'information concernant l'architecture d'une partie ou de l'ensemble d'une application.

Il est possible d'utiliser des composants constitué d'autres composants. Ces composants sont dites composites, permettant la construction incrémentales des applications en suivant le principe de décomposition et peuvent être réutilisé dans leur intégrité.
L'assemblage de composant englobe trois parties principales :

- *La conception d'un assemblage :* elle est réaliser en étudiant de manière de composer un ensemble de composant préexistants pour créer une composition représentant soit une partie, soit la totalité d'une application, c'est-à-dire une architecture.
- *La réalisation :* elle consiste à l'écriture du code permettant de réaliser la composition des instances de composants.

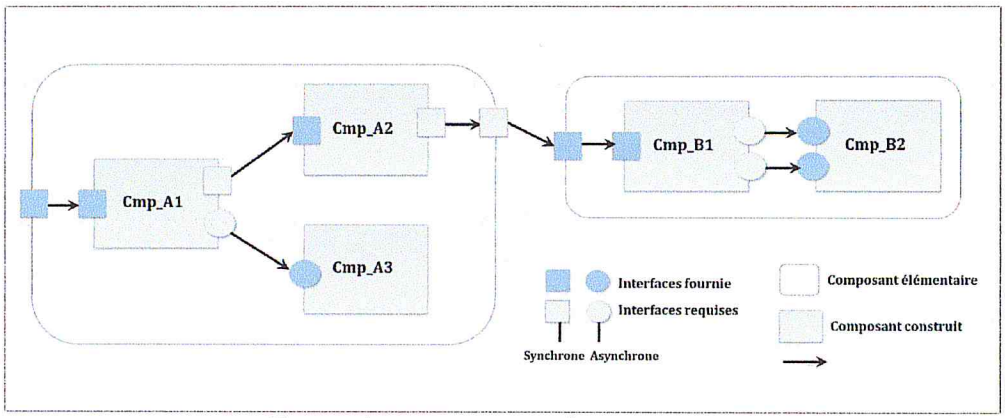

Figure4: composition de composant

2.3.4. Composant logiciel comme unité de déploiement :

Le composant est une unité de déploiement indépendante, d'une part cela veut dire que celui-ci doit être entièrement sépare de l'environnement et des autres composants.

D'autre part, celui-ci ne pourra jamais être déployé partiellement. Un composant pouvant être composable avec d'autres composants doit avoir une spécification précise de ses besoins mais aussi de ce qu'il peut fournir.

En d'autres termes, un composant encapsule son implantation et interagit avec l'environnement qui l'entoure par des interfaces clairement définies. [KM, 04]

Le déploiement du composant spécifie ses dépendances, ses interfaces, ses modes de déploiement et d'instanciation ainsi que le comportement de ces instances a travers les interfaces offertes. [KM, 04]

2.4.Le cycle de vie d'un composant : [KM, 04]

Le processus de développement d'un composant est, par plusieurs aspects, similaire au développement d'un système; les besoins doivent être *capturées, analysées et définis, le composant doit être conçu, implantée, vérifiée, validée et livrée (exécuté).*

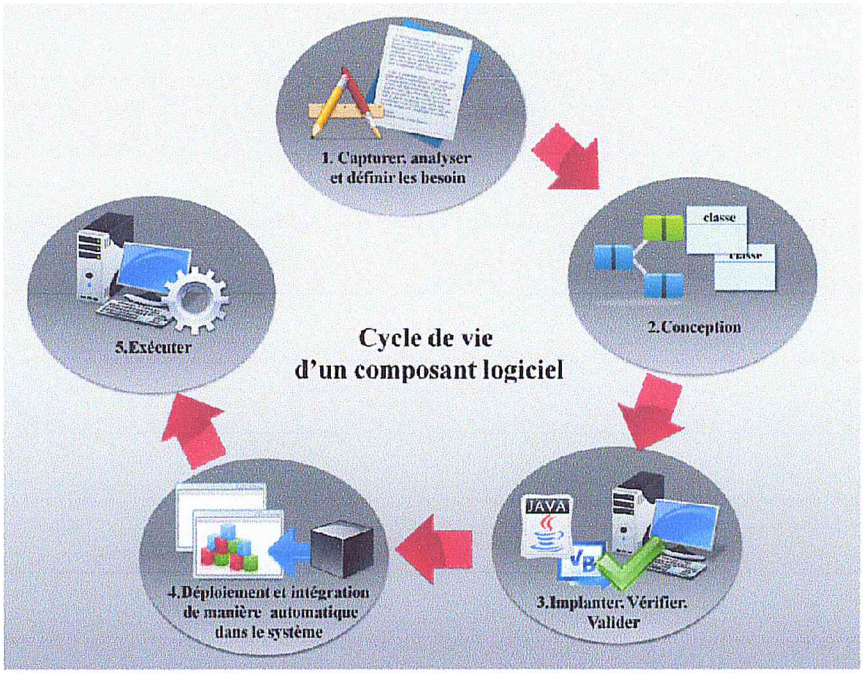

Figure 5: cycle de vie d'un composant logiciel

Lors de la construction d'un nouveau composant, les développeurs peuvent réutiliser d'autres composants et utiliser des procédures d'évaluation du composant semblables a celles utilisées pour le développement d'un système.

Cependant, il y a quelques différences significatives : les composants sont construits pour faire partie de quelque chose. Ils vont être réutilises dans différents produits. Ceci a pour conséquence :

- Plus de difficultés dans la gestion des besoins, causes par l'interaction entre le composant et le système.
- Plus de précisions dans la spécification du composant.
- De plus gros efforts sont nécessaires pour créer des unités réutilisables.
- Plus de rigueur et de documentation dans la vérification de la spécification des composants surtout lors des transferts de composants entre organisations.

Une fois que le composant a été testé, spécifié, stocké dans une librairie de composants, la prochaine étape dans le cycle de vie du composant est la phase de déploiement dans le système.

Le déploiement du composant consiste a son enregistrement dans le système ainsi qu'à établir la communication avec le reste du système.

Cette communication est obtenue par lien dynamique entre les interfaces du composant et le système.

Le déploiement doit se faire de manière automatique et sans provoquer de changement dans le reste du système.

2.5.Milieux des Composants Logiciels : [BEN, 09]

Un composant logiciel possède deux milieux, un milieu académique et un milieu industriel :

2.5.1. Le composant logiciel dans le milieu Industriel :

Dans le milieu industriel, le concept de composant a été mis en évidence dans les premiers environnements de développement rapide d'application (RAD) tels que VISUAL BASIC de Microsoft et DELPHI de Borland. Cette mise en pratique a illustré l'efficacité d'une approche basée sur les composants et a montré :

- Que le temps de réalisation des applications est fortement réduit.
- Qu'il est facile de réutiliser des composants de diverses sources.
- Qu'il est possible à des personnes n'ayant pas de grandes compétences en programmation de réaliser certaines parties d'une application.
- Qu'il est dorénavant possible de faire participer efficacement le client durant tout le cycle d'élaboration de son application.

2.5.2. Le composant logiciel dans le milieu académique :

Dans le milieu académique et au niveau des centres de recherches des grandes entreprises, l'importance d'une approche de conception basée sur la spécification explicite de l'architecture d'un logiciel a été ressentie depuis les années 70.

Cependant, ce n'est qu'à partir des années 90, et suite à la grande réussite du modèle objet, qu'un intérêt de plus en plus grandissant s'est porté sur cette approche.

Les objectifs consistent à mettre sur pied des modèles plus adaptés que le modèle objet pour la spécification d'architecture logicielle et à rendre réelle et efficace l'idée ambitieuse de construction de logiciels par assemblage de composants, sans écrire une seule ligne de code en langage de programmation.

Ce grand intérêt s'est soldé par un nombre important de travaux ayant abouti à un nombre important de langage de description d'architecture logicielle et de modèles de composants.

3. Conclusion :

Dans ce premier chapitre, nous avons présenté l'architecture logicielle et ses concepts puis le paradigme de l'orienté composant. Ainsi nous avons vu l'importance du développement par composants, de sorte qu'il est perçu comme un atout majeur pour le développement des systèmes informatiques et comme une avancée considérable dans le domaine de la méthodologie du développement logiciel.

Ainsi il introduit une méthodologie visant à faciliter la construction d'application a partir de l'assemblage de briques logicielles préfabriqué c'est-à-dire composant de logiciel.

Cette méthodologie insiste particulièrement sur une séparation entre l'étape du développement et celle de l'assemblage des composants. L'assemblage est réalisée à partir de composants qui peuvent entre obtenu ailleurs que dans le lieu ou ils sont assemblés.

III

Analyse des besoins

Introduction
Première partie « validation de l'information »
Deuxième partie « le composant de validation »
Conclusion

1. Introduction :

L'objectif de notre travail est de réaliser un composant dédié à la validation d'information. Dans ce chapitre nous allons présenter dans un premier temps le concept de validation, de processus de validation, de schéma de validation et d'autres concepts liés à la validation d'information.

Par la suite nous allons déterminer les diverses fonctionnalités d'un composant de validation et les divers acteurs qui interviennent dans un processus de validation de l'information.

Nous déterminerons aussi les diverses fonctionnalités requises par le processus de validation et qui ne peuvent pas être considérées comme fonctions métier d'un composant de validation.

Enfin nous devons proposer la vue externe du composant de validation à travers laquelle nous distinguerons les services fournis par le composant de validation et les services requis par ce composant de validation d'information.

2. Première Partie « Validation de l'information » :

Les organismes gouvernementaux et réglementaires sont toujours menés a délivrer des informations valides et hautement fiable, car ils n'ont pas le droit a l'erreur vu l'importance et la nature des documents qu'ils délivrent, et aussi le fait que la correction d'une information pourraient nécessité un temps assez long et des procédures administratives et juridiques pouvant être très complexes.

Ces organismes sont ainsi obligés de mettre en place tout les outils et procédures nécessaires pour vérifier l'information avant de la délivrer dans des documents officiels qu'ils soient sur papier ou sur support numérique.

La mise en place d'une infrastructure permettant de certifier qu'une information est correcte avant de la mettre sur un document officiel permettra d'une part de gagner la cofinance des citoyen et d'autre part à contribuer à la mise en place efficace du eGouvernement.

Un système d'eGouvernement doit ainsi validée entièrement ses informations avant de les délivrer.

Dans cette partie nous allons définir tout les concepts qui ont un rapport avec la validation de l'information. Nous entamerons celle-ci par la définition du processus de validation tous les concepts qui ont un rapport avec la validation,

2.1. Définition de la validation :

La *validation* d'information est l'attestation, la déclaration, la confirmation ou l'affirmation par examen et apport de preuves tangibles que l'information validée est correcte et est conforme à la réalité.

2.2. Processus de validation :

Un processus de validation est un ensemble d'étapes par lesquelles doit passer une information avant d'être déclarée valide. A chaque étape l'information est alors qualifiée de valide ou non par un ensemble de validateurs. Le processus de validation peut suivre un schéma simple ou très complexe.

La complexité dépendra de l'importance de l'information et de la politique adoptée par l'organisation détentrice de l'information. Dans cette dernière, l'organisation définit pour chaque type d'information, voir pour un ensemble d'information d'un type bien précis,

- Les diverses étapes du processus,
- Les validateurs qui interviennent à chaque étape
- Les conditions de passage de l'information d'une étape à une autre.
- Les conditions de retour d'une information d'une étape vers une étape précédente.

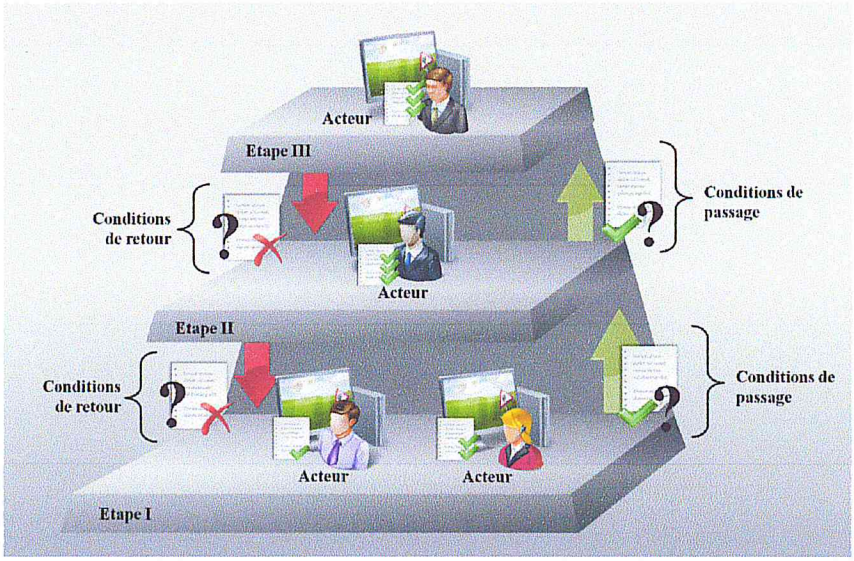

Figure 6 : processus de validation d'information

2.3. Reconnaissance des résultats de validation :

Une fois l'information validée, celle-ci pourrait passer par une ultime phase qui est la reconnaissance explicite de sa validité par l'acteur sujet de l'information ou ayant un droit sur l'information (par exemple une note dans un examen reconnue par l'étudiant concerné, une information personnelle reconnue par la personne elle-même ou ses parents etc.)

2.4. Verrouillage et déverrouillage d'une information dans un processus de validation

Malgré la mise en œuvre d'un nombre important d'outil et de méthodes d'aide à la validation d'information, la décision de valider une information reste est une décision humaine. L'erreur est ainsi possible, cependant que très réduite.

Dans une situation d'erreur il est nécessaire de procéder à la correction d'erreur. Dans le contexte d'un processus de validation, la correction d'erreur nécessite obligatoirement que l'information n'ait reçue aucune décision de validation.

Cependant, l'erreur pourrait être détectée en milieu du processus de validation ou tout simplement après que l'information ait été déclarée valide (l'information est passée par toutes les étapes du processus de

validation). La phase de reconnaissance de l'information pourrait être une phase dans laquelle l'erreur est détectée.

Pour pouvoir corriger une information totalement ou partiellement validée, il est nécessaire qu'elle revienne aux étapes précédentes. Le passage d'une étape notée E_i à une étape précédente notée E_{i-1} nécessite une opération explicite d'invalidation par tous les validateurs de l'étape E_i.

La décision de validation a pour effet de mettre un verrou sur l'information validée. Le verrou est un code secret propre au validateur. Chaque validateur dispose des ses propres codes de verrouillage.

Le déverrouillage nécessite la spécification du code représentant le verrou. Au niveau d'une étape E_i, si tous les verrous sont enlevés, l'information revient à l'étape E_{i-1}.

Et le même processus est réitéré jusqu'à ce que l'information revienne à la première étape E_0.

Une fois au niveau E_0, l'information pourrait alors être corrigée pour qu'elle soit ensuite réinjectée dans le processus de validation.

3. Deuxième partie « Le composant de validation » :

Apres une étude approfondit et une compréhension des concepts de validation, dans cette deuxième partie nous allons essayé de déterminer les fonctionnalités que doit supporter le composant de validations en présentant les services fournis par celui-ci, les fonctionnalités nécessaires au fonctionnement du composant de validation par la présentation des services requis, et nous présenterons les acteurs qui interagissent avec le composant de validation.

3.1.Aspects généraux du composant de validation :

Dans cette partie nous allons expliquer tout les aspects et la politique que doit remplir le composant de validation :

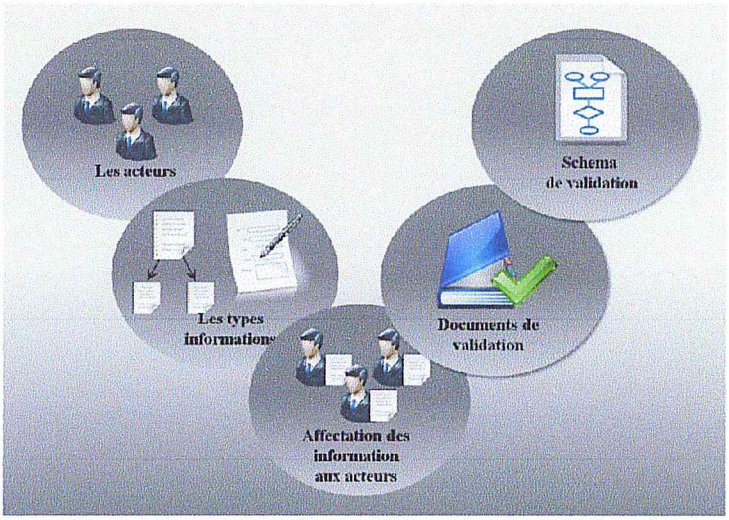

Figure 7 : aspect d'un composant de validation

3.1.1. Les acteurs :

Le composant possède Cinque types d'acteurs qui sont : « *Administrator*, *Validator*, *UValidator*, *InfoSupplier*, *InfoObserver*».

Chaque type d'acteur est associé à un rôle « *profil* » bien précis représenté par un ensemble d'actions spécifiques. Un acteur est associé à une clé qui permet de le distinguer des autres acteurs et qui permet de déterminer son champ d'action dans le contexte de son profil, notamment les droits

d'accès aux données. Nous allons détailler chaque type d'acteur qui interagit avec le composant de validation :

L'Administrator :

L'Administrator est un acteur singleton c'est-à-dire qu'il ne peut pas y avoir plus d'un seul acteur administrateur pour une instance d'un composant de validation.

Un acteur *« Administrator »* effectue toutes taches de manipulation de données qui peuvent servir dans la gestion du composant de validation, telle que la gestion des acteurs, la gestion des types d'informations, la gestion des schémas de validation…etc. Il possède le privilège de mettre en œuvre toutes les opérations des interfaces fournies du composant de validation sans exception.

Un acteur « *Administrator* » peut être à la fois un « *Validator*, *UValidator*, *InfoSupplier*, *InfoObserver*».

L'UValidator :

L'UValidator ou « *UpdaterValidator* » c'est-à-dire un validateur qui le droit de modifier l'information lors de la validation. C'est un acteur qui à des droits restreins par rapport a l'administrateur, il se charge de la validation ou l'invalidation de l'information qui lui aurait été affecté avec la possibilité d'effectuer des modifications sur les informations. Ainsi un UValidator peut corriger une information lors du processus de validation, si celle-ci est fausse. Par la suite il peut la valider ou l'invalider.

En plus de ses fonctionnalités spécifiques, un « UValidator » peut réaliser toutes les opérations que peut réaliser des acteurs « *Validator, InfoSupplier, et un inforObserver*»

Le Validator :

Le Validator est un acteur qui à des droits restreins par rapport à un acteur « *UValidator* », il se charge de la validation ou l'invalidation de l'information qui lui aurait été affecté. En plus de ses fonctionnalités, un acteur « Validator » hérite des fonctionnalités des acteurs « *InfoSupplier et InfoObserver*»

L'InfoSupplier :

L'InfoSupplier ou « *fournisseur d'information*», est un acteur qui a des droits restreins par rapport a un « *Validator*». Il se charge de soumettre au composant de validation les informations à valider. Les types de ces dernières doit correspondre au type information qui lui auraient été associés. En plus de ses fonctionnalités, un acteur « InfoSupplier » hérite des fonctionnalités d'un acteur « InfoObserver »

Un acteur «*InfoSupplier* » peut aussi modifier ou la retirer une information soumise tant que celle-ci n'est pas scellée par le processus de validation. Une information est dite scellée par le processus de validation si elle possède au moins une validation.

L'InfoObserver:

L'InfoObserver ou « *observateur d'informations*», un acteur qui les droit les plus retreins de touts les types acteurs. Il se charge d'accéder à l'information et suivre son état de validation.

Le diagramme de la « *figure 8* » montre la hiérarchie des acteurs dans le composant de validation :

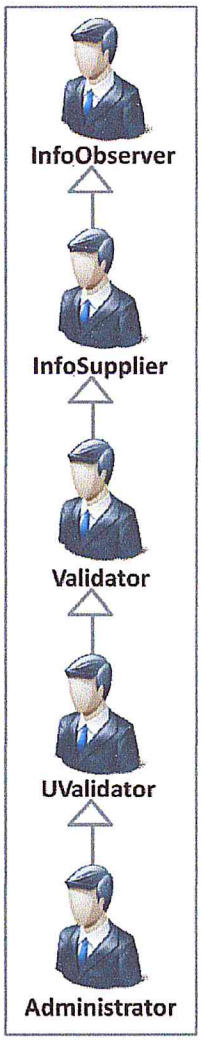

Figure 8 : Diagramme hiérarchique entre les utilisateurs

3.1.2. Les types informations :

Les types d'informations à soumettre pour validation ne sont pas connus au préalable par le composant de validation. Ces types sont spécifiés au composant de validation au fur et a mesure des besoins de l'application exploitant le composant de validation.

La spécification du type d'information à valider est une prérogative de l'administrateur du composant de validation. A titre d'exemple si le composant de validation est utilisé par une application de gestion de l'état civil d'une APC, les types d'informations pourront être les informations de naissances, les informations de décès, les informations de mariage etc. Et si

le composant est manipulé par une application de gestion de scolarité, les types d'informations à indiquer au composant de validation sont par exemple les informations relatives aux diverses évaluations des notes «d'examen, de TP, de suivi etc. dans les divers modules.

Un type d'information peut avoir une taille et une structure complexe. Pour permettre un traitement plus efficace de l'opération de validation, une information est alors considérée comme un ensemble d'unité d'information.

La structuration en unité d'information ouvre la voie vers la possibilité d'une validation partielle d'une information en permettant la validation de manière indépendante des unités d'information. Ainsi, le schéma de validation d'une information pourrait statuer qu'une unité d'information doit être validé par un acteur et une autre unité par un autre acteur. Ainsi, toute information est composée d'un ensemble d'unité d'information. La cardinalité minimale de cet ensemble est de 1 (toute l'information est mise dans une seule unité d'information).

3.1.3. Les documents utilisés dans le processus de validation :

Chaque type information peut avoir un ou plusieurs types de documents sur lesquels se base la validation. Les documents sont utilisés par les acteurs validateurs comme preuves pour confirmer la véracité d'une information appartenant à un type d'information.

Le composant de validation doit supporter deux types de documents utilisé dans la validation d'informations: des documents *obligatoires* et des documents *facultatifs*.

Un document obligatoire pour un type information bien précis doit être précisé lors de la validation, sinon la validation ne sera pas faite. En ce qui concerne les documents facultatifs ce ne sont pas des documents important à préciser lors de la validation des informations.

3.1.4. Assignation de l'information a un acteur :

Pour pouvoir manipuler les types informations, chaque type information sera assigné a un acteur bien précis « *Validator*, *UValidator*, *InfoSupplier*, *InfoObserver*». Apres l'assignation les acteurs peuvent manipuler les types information, mais uniquement les types information qui leur seront assigné. Sauf pour l'acteur administrateur, lui il possède le privilège de manipuler tout les types information.

L'assignation peut s'effectuer de plusieurs manière, on peux affecter directement toutes les valeurs d'un type information, comme on peux restreindre l'affectation a titre d'exemple on peut assigner une plage d'information bien précise a un acteur, ou on peut assigner l'information par rapport a un champ bien précis.

3.1.5. Les schémas de validations :

Le composant de validation manipule les schémas de validation. Un schéma de validation peut être créé pour un seul type information, et il peut contenir plusieurs unités informations.

Tout comme le type information, un schéma de validation n'est pas fixe il peut changer d'une application a une autre, et d'un type information a un autre, dans la même instance du composant de la validation, on peut trouver plusieurs schéma de validation.

Un schéma de validation doit obligatoirement avoir un nom et un type information et une date de validation. Il est peut contenir une ou plusieurs phases, chaque phase regroupe une ou plusieurs taches.

La spécification des unités d'informations « *dans les cas ou le type information est un ensemble d'unités d'information* » se fait au niveau des phases.

Chaque phase possède sa propre date de validation, une phase réfère en réalité a un service par exemple service secrétariat, ou un département, et une tache réfère en réalité aux validateur de ces services, ou du département.

Pour chaque tache, il sera affecté un acteur validateur bien précis. Une tache peut être obligatoire c'est-à-dire la validation de l'acteur affecter a chaque tache est impérative ou facultatif c'est-à-dire la validation de l'acteur n'est pas importante.

La validation ou l'invalidation de l'information sera effectuer par un acteur validateur, si c'est « *Validator* » il valide ou il invalide, si c'est un « *UValidator* » il peut effectuer la mise à jour de l'information lors de la validation.

3.2. Conditions de mise en place du composant de validation :

- Un composant de validation possède toujours un et un seul administrateur. Par défaut le nom de l'administrateur est **admin**, et mot de passe est **admin** et la clé est **admin**.
- Il est possible de changer chacune de ces valeurs après instanciation du composant de validation. La définition se fait par des interfaces telles que « *setAdmin(admin.xml, adminKey)* ».
- La définition de l'administrateur « *spécification des informations personnelles* » se fait soit à l'instanciation du composant de validation soit à l'aide d'un service « *setActor (actor)* ».
- La clé identifie de manière unique un acteur, et elle est attribue par le composant.

  La clé est réellement acquise à travers un composant spécialisé dabs la gestion de clé
- Tout acteur peut modifier sa clé grâce à un service de mise à jour de la clé de validation.

3.3. Interactions du composant de validation avec les autres applications :

Le composant de validation interagit avec les autres applications grâce aux services qu'il fournit. Il interagit aussi avec un composant externe spécialisé dans la gestion de clés.

3.4. Diagrammes de cas d'utilisation :

Ces diagrammes permettent d'identifier le comportement et les fonctionnalités que pourront effectuer les acteurs avec le composant de validation, ainsi, cela permet de voir comment le composant de validation gère ses fonctionnalités,

3.4.1. Administrator :

Les actions que peut réaliser l'administrateur du composant de validation sont :

- Gestion les acteurs
- Gestion des types d'information
- Affectation des Informations à valider aux acteurs
- Gérer des documents de base pour la validation d'un type d'information
- Assigner Document de validation a type information
- Gérer les schémas de validation
- Affectation des validateurs au schéma de validation

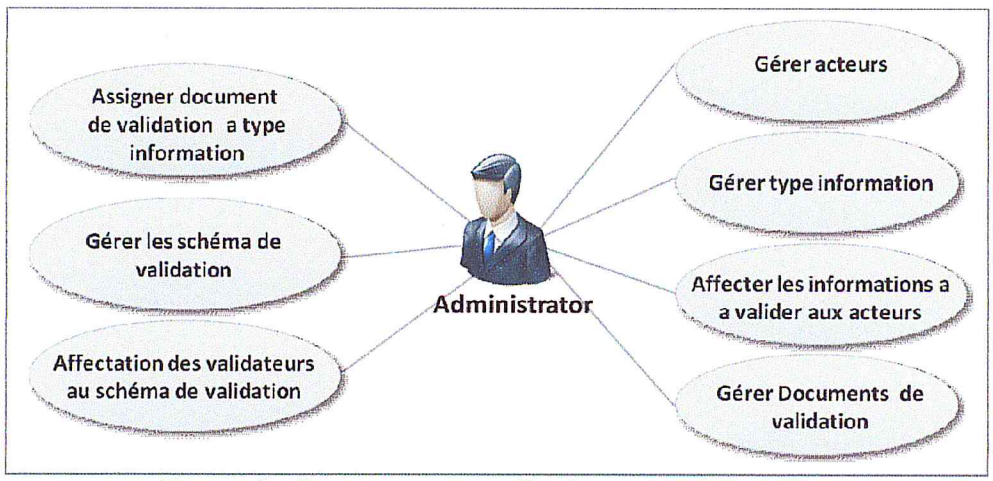

Figure 9 : Diagramme cas d'utilisation Administrator

Remarque :

Dans la suite de cette partie nous allons considérer les notations suivantes :

- « TI » pour «*Type Information* »
- « UI » veut dire « *Unité Information* »

Gérer acteurs :

La définition d'un Validator se fait de la même manier qu'un UValidator :

- Ajout d'un acteur
- Mise à jour d'un acteur
- Accès aux informations acteur
- Suppression d'un acteur

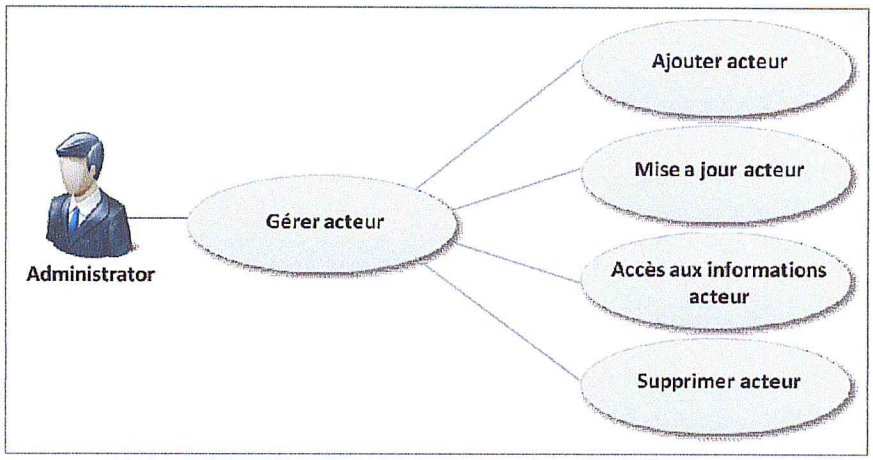

Figure 10 : Diagramme cas d'utilisation Gérer acteur

Gérer type d'information :

Dans cette partie nous allons détailler les différentes taches exécutées par l'administrateur pour gérer le type information :

- Ajout du type information et/ou les unités information
- Modification du type l'information et/ou les unités information
- Accès aux informations type information et/ou les unités information
- Suppression d'un type information et/ou les unités information

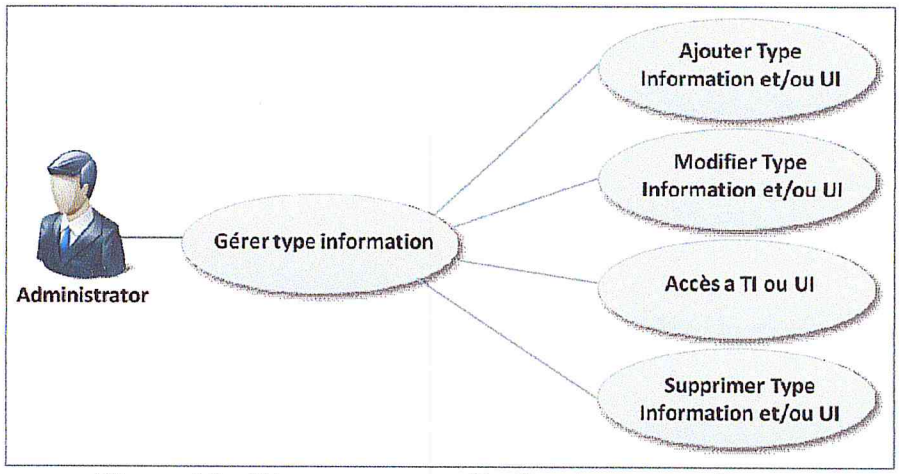

Figure 11 : Diagramme cas d'utilisation type information

Gérer documents de validation:

Dans cette partie nous allons détailler les différentes taches exécutées par l'administrateur peur gérer les documents sur lequel se base la validation

- Ajout document
- Modifier document
- Accès aux documents
- Suppression document

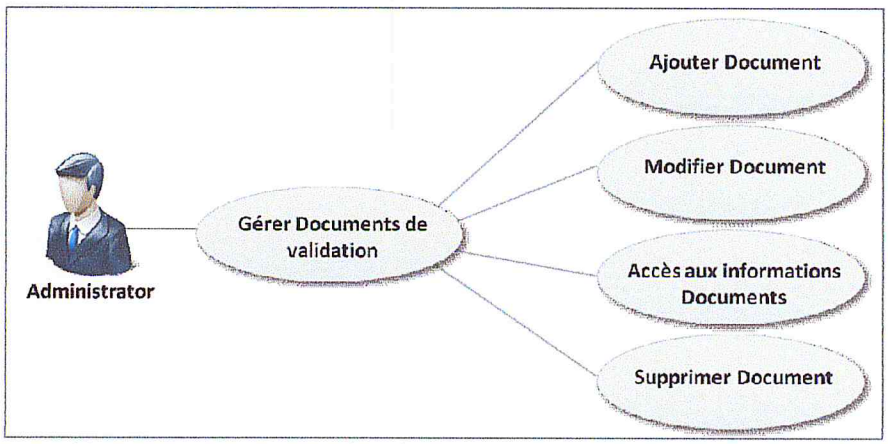

Figure 12 : Diagramme cas d'utilisation Gérer documents de validation

Gérer schéma de validation :

Pour Gérer un schéma de validation l'administrateur effectue les taches suivantes :

- Définir schéma validation,
- Modifier schéma validation,
- Accès aux informations du schéma validation,
- Suppression schéma validation

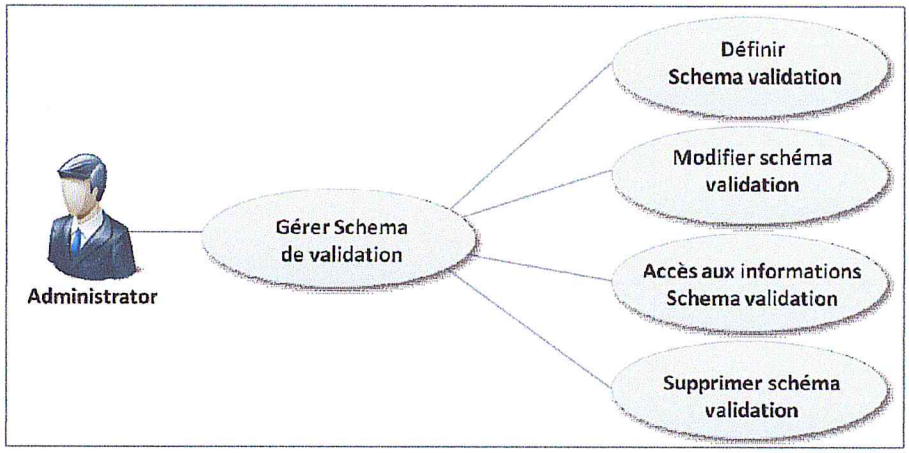

Figure 13 : Diagramme cas d'utilisation Gérer Schema de validation

Définir schéma de validation :

Pour définir un schéma de validation l'administrateur devra,

- Définir les types d'information(TI) et unités d'information (UI) pour le schéma de validation
- Définir les phases du schéma de validation
- Définir les taches du schéma de validation

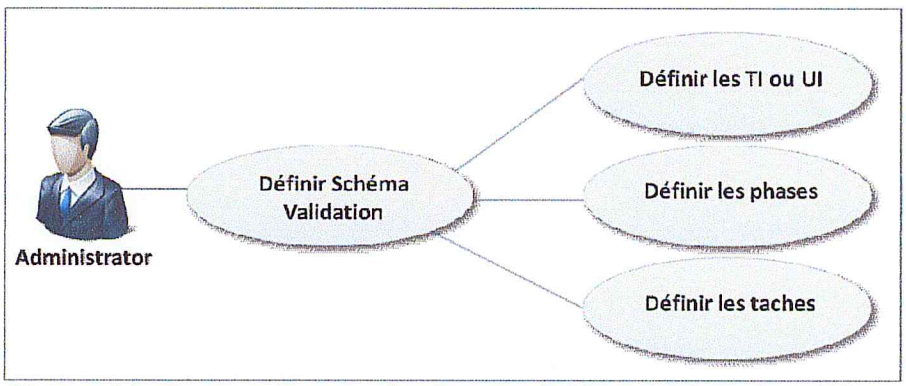

Figure 14 : Diagramme cas d'utilisation Définir schéma de validation

3.4.2. Validateurs « UValidator et Validator » :

Le composant de validation possède deux types de validateur, leur rôle principal est la consultation et la validation d'information mais la différence entre les rôles de ceux est que :

- Validator : valide/invalide l'information
- UValidator : valide/invalide et apporte des modifications a celle-ci, si nécessaire.

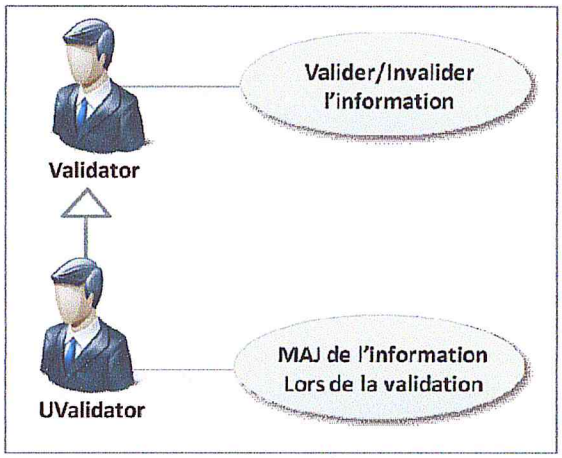

Figure 15 : Diagramme cas d'utilisation Validators

3.4.3. InfoSupplier :

Les opérations effectuées par un InfoSupplier sont :

- Soumission d'une Information/UI à valider
- Retirer l'information soumise si elle n'a pas été validée par un validateur
- Modifier l'Information soumise,
- Accès aux informations
- Suivre l'état de validation des informations soumises

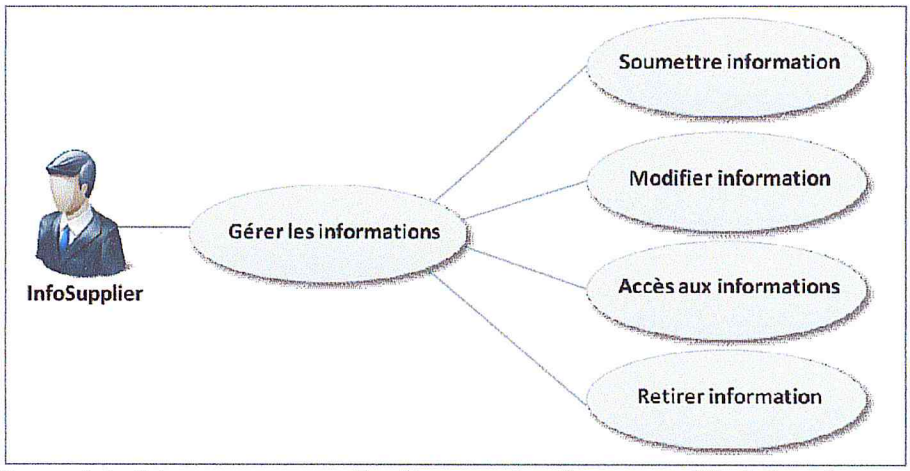

Figure 16: Diagramme cas d'utilisation des fonctions d'InfoSupplier

3.4.4. InfoObserver :

Cet acteur permet de consulter le statut de l'information, et voir si elle a été valide ou en attende de validation.

- Accès aux informations
- Suivre l'état de validation des informations auxquelles il a le droit d'accès

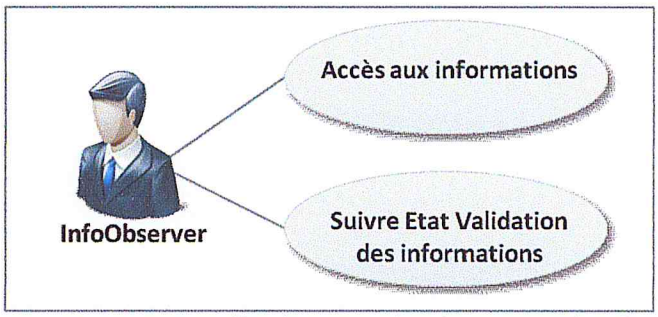

Figure 17: Diagramme cas d'utilisation des fonctions d'InfoObserver

4. Conclusion

Après cette étude approfondi et l'analyse complète de ce que doit avoir un composant de validation comme fonctionnalités et comme acteurs nous avons définit une première vue des interfaces du composant de validation :

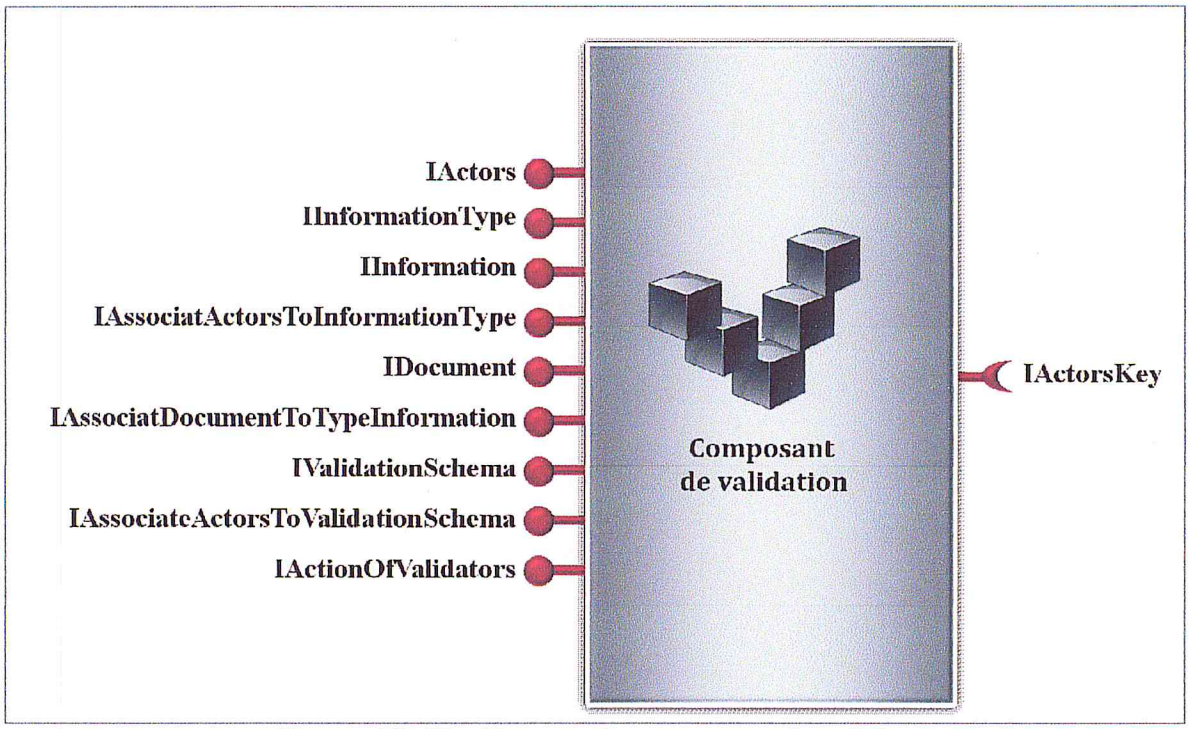

Figure 18: Vue Externe du composant de validation

IV

Conception

1. Introduction :

Nous présenterons dans ce chapitre les divers aspects liés à la conception de notre composant qui devra répondre aux objectifs définis dans le chapitre précédent.

Nous commencerons tout d'abord par introduire la forme générale du modèle de composant que nous utilisons dans notre conception. Ce modèle s'inspire du modèle de composant de l'approche IASA et de FRACTAL.

2. Modèle de composant :

Le modèle de composant que nous utilisons possède une organisation bien précise de la vue interne. La vue interne est composée de deux parties « *figure 19* » :

Une partie contrôle et une partie métier. La partie contrôle s'occupe des fonctionnalités liées à la gestion du cycle de vie du composant tel que la construction du composant, son initialisation et sa destruction.

L'initialisation du composant peut comprendre l'instanciation des divers composants formant la partie métier. La partie métier comporte toutes les fonctionnalités liées aux objectif fondamentaux du composant et accessibles une fois le composant mis en place « instancié, construit etc. ».

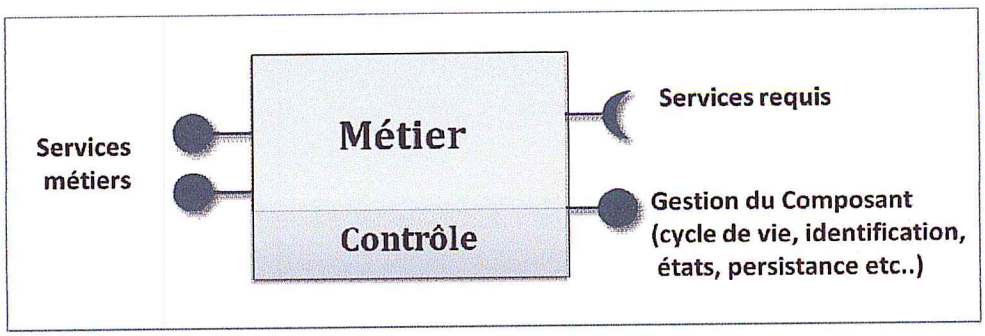

Figure19: Modèle de composant

3. Composant persistant et nécessité de l'identifier :

En général, un composant, possède un cycle de vie inférieure à l'application. Il est instancié puis détruit dans le contexte de l'exécution d'une même application. L'identification de ces composants possède une durée de vie limitée par l'application est souvent réalisée à l'aide de références (nom de variable) ou indirecte (pointeurs en C++, référence Java) mis en place par l'application durant l'exécution du programme. Une fois le programme terminé, ces références disparaissent. Si elles ont été établies dans une méthode, ces références n'existeront que durant la période ou la méthode est active.

Dans certaines situations, les composants doivent avoir une durée de vie plus grande que l'application dans laquelle ils ont été construits pour la première fois et utilisés. Ces composants possèdent des états ou gèrent des données qu'il est nécessaire de retrouver lors du relancement de l'application ayant instancié ces composants.

La terminaison d'une application ne veut donc nullement dire qu'un composant de cette catégorie a disparu. Ce type de composant possède des états qui doivent être restauré une fois le composant instancié dans une application. Plus encore, un même composant pourrait être partagé entre plusieurs applications.

De ce fait, l'identité du composant ne doit pas être une référence du programme mais un nom à travers lequel une application le retrouve lors de son instanciation. Ainsi un composant doit posséder un nom qui l'identifie de manière unique dans le monde réel où il évolue.

4. Contrôle de l'instanciation :

Un composant ayant une identité particulière peut être instancié par diverses applications. Dans certains cas, pour protéger l'information que gère ce composant, il n'est pas recommandé que n'importe quelle application puisse l'utiliser. Ainsi il est nécessaire de faire un contrôle lors de l'instanciation du composant.

Ce contrôle permettra de retrouver les applications ayant le droit d'instancier ce type de composant. La technique la plus usuelle et efficace pour faire ce contrôle est la technique d'authentification de l'application désirant instancier le composant. Ainsi, au minimum, un nom d'utilisateur et un mot de passe serait requis pour instancier le composant.

En général le mot de passe et le nom utilisé coïncident souvent avec le nom d'utilisateur et le mot de passe de l'administrateur de l'application dans laquelle le composant a été instancié

5. Scénario d'instanciation du composant :

Ce scénario permet de déterminer l'interface de la partie contrôle et les divers techniques d'instanciation d'un composant persistant.

Nous distinguons deux grands scénarios menant à la mise en place d'un composant :

- Instanciation initiale du composant
- Instanciation ordinaire

L'instanciation initiale représente la première mise en place du composant. Durant cette opération il est nécessaire de spécifier le nom du composant qui devra être unique dans le monde ou il évolue. C'est durant cette instanciation initiale que les informations d'authentification de l'application sont définies soit de manière explicite ou implicite.

Dans la définition explicite, l'application doit indiquer le nom et le mot de passe qui seront utilisé pour instancier le composant par la suite.

Dans l'instanciation implicite, le composant utilise des valeurs par défaut qui sont dans notre cas « admin » et « admin ». Ces valeurs nécessitent une opération de modification de la part de l'application qui a instancié le composant

Dans une opération d'instanciation ordinaire, un nom de composant existant doit être indiqué ainsi qu'un nom d'utilisateur et un mot de passe. Le nom de l'utilisateur et le mot de passe sont bien sur requis dans le cas ou le composant

ne doit être instancié que par les ayant droits. L'interface de contrôle présente ainsi les opérations suivantes indiquées sur le schéma de la fig. L'écriture Java de cette interface est reportée par « *la figure 20* »

```
InterfaceIControlValCmp {
        /** Composant persistant **/
        build (String  cmpName)

        /** Composant persistant avec authentification**/
        build (String cmpName, String adminUser, Strin adminPass);

        /**Non persistant**/
        build()

        /** Initialisation du composant**/
        initialize();
        destroy();

        /**Une copie **/
        clone(String cmpName);

}
```

Figure20: partie contrôle du composant

6. Architecture globale du composant de validation :

Nous allons présenter l'architecture globale de notre composant de validation avec le diagramme de composant

La figure 21 présente l'architecture interne du composant de validation. Cette Architecture est réalisée en utilisant les composants suivant :

- **Gestion Acteur :** ce sous composant permet de gérer les interactions entre les acteurs du composant de validation.
- **Gestion type information :** ce sous composant permet la gestion des traitements effectué sur un type information
- **Assignation :** ce sous composant permet gérer les assignations entre les acteurs et les types information

- **Gestion documents :** ce sous composant permet de gérer les documents sur lequel se base la validation
- **Gestion schéma de validation :** ce sous composant permet de gérer le schéma de validation
- **Validation :** ce sous composant permet de gérer la validation
- **Gestion ID System :** ce sous composant permet de fournir des identificateurs system ou composants contenu dans le composant de validation.

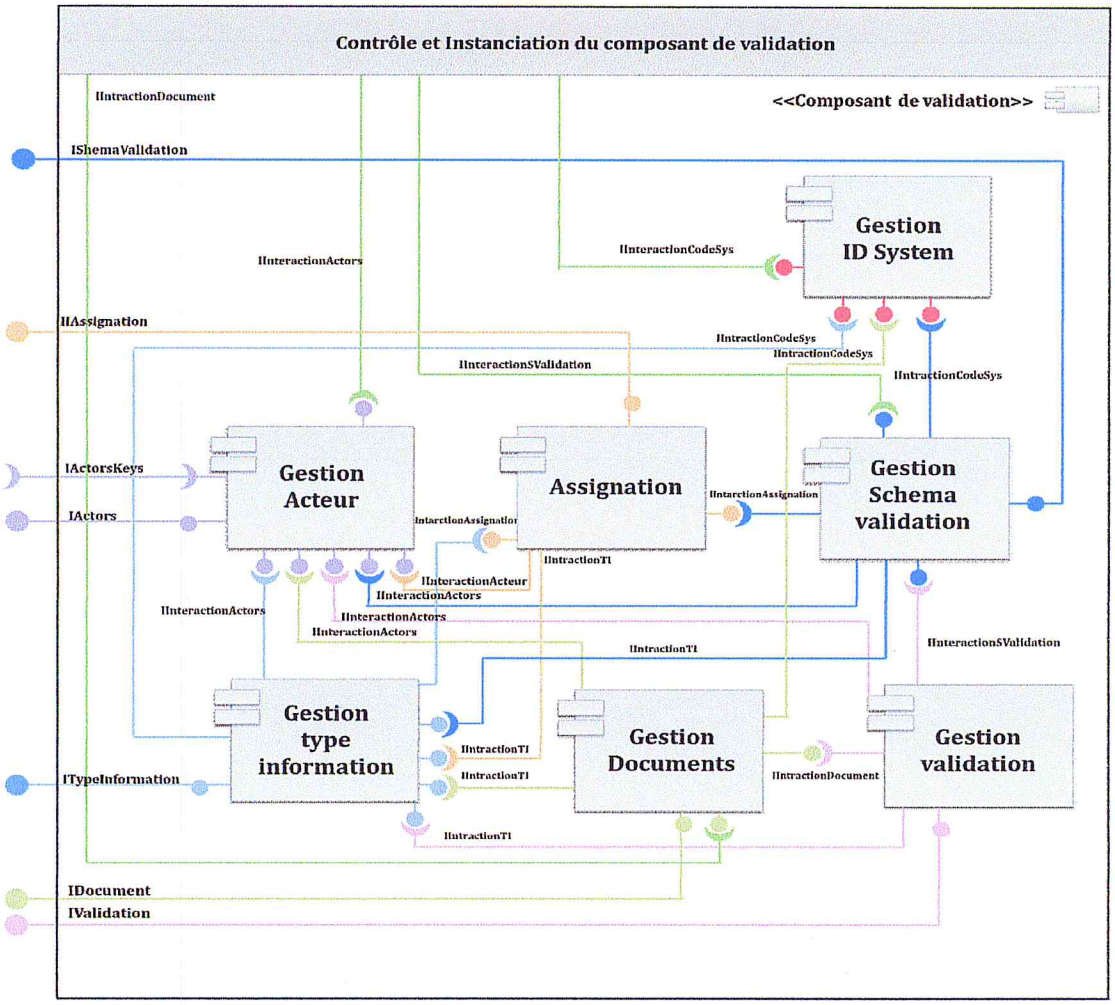

Figure21 : diagramme de composant d'architecture globale du composant de validation

7. Architecture détaillé du composant de validation :

Dans cette partie nous allons détailler chaque sous composant du composant de validation :

7.1.Gestion acteur :

Ce sous composant permet la gestion des acteurs dans le composant de validation, il permet la création, la suppression, la modification et tout les interactions spécifiques aux acteurs.

7.1.1. Diagramme de composant:

Le sous composant « *Gestions Acteur* » interagies avec les autres composants grâce aux interfaces fournies « *IActors, IInteractionActors* » et aux interfaces requises *« IActorsKey »*.

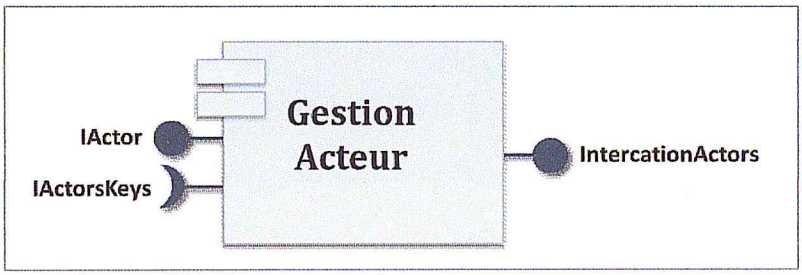

Figure 22: diagramme de composant gestion acteur

7.1.1.1. L'interface « IActors » :

Cette interface fournit les méthodes d'interaction avec les applications externes du composant, car lorsque le composant est instancié le seul accès vers le composant pour la gestion des acteurs c'est a travers les méthodes que fournit cette interface que nous allons expliquer :

- **addActors :**

  Cette méthode permet d'ajouter les acteurs qui interagissent avec le composant de validation, elle prend en paramètre la clé administrateur « *cleAdmin* », un objet «*Actors*» où sont décrites les informations personnelles des acteurs qu'il faudra spécifier au composant de validation. En cas d'erreur elle retourne une exception.

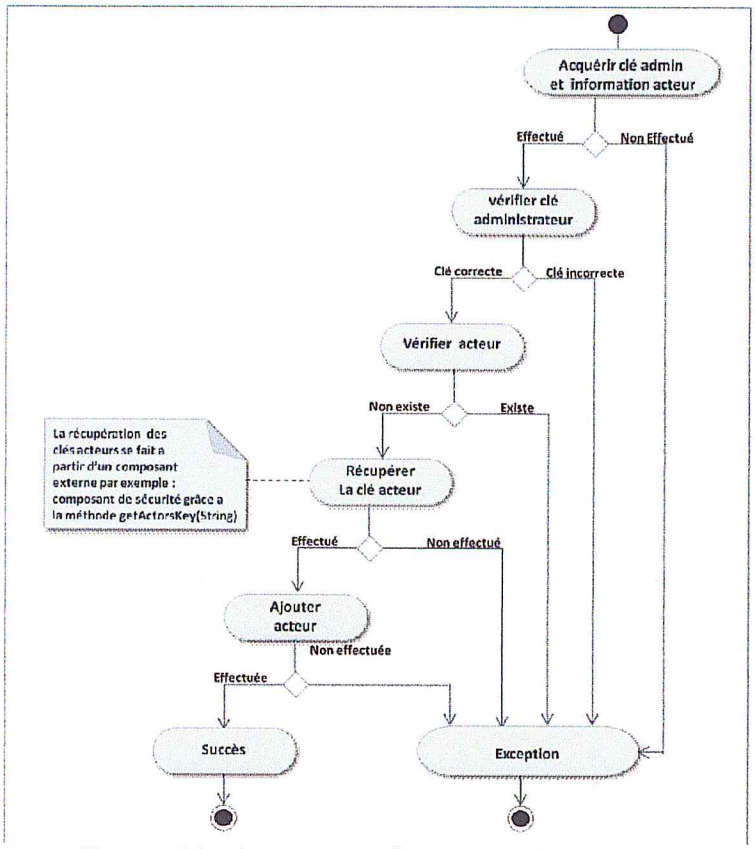

Figure 23: diagramme d'activité addActor

- **setActors :**

Cette méthode prend en paramètre un objet acteur et la clé administrateur. Elle permet de mettre à jour les acteurs à base de leurs clés acteurs et des nouvelles informations qui se trouvent dans l'objet « Actor », ou de mettre à jour le profil d'un acteur.

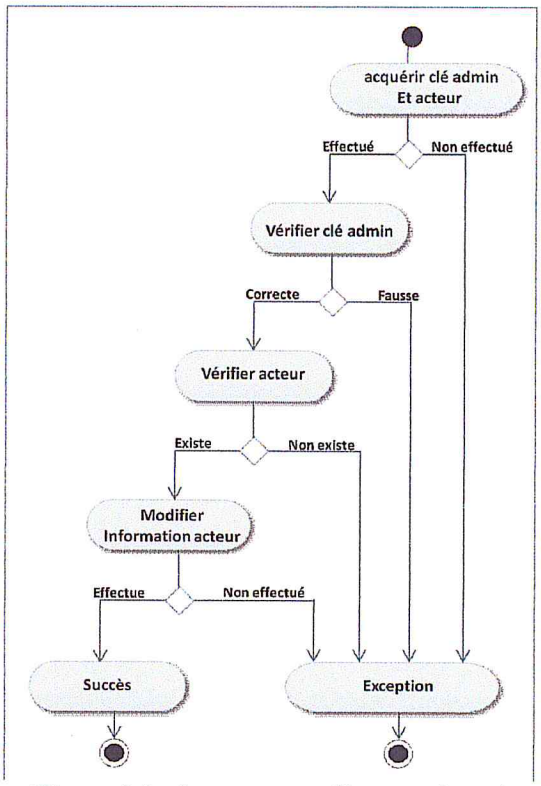

Figure24: diagramme d'activité setActors

- **delateActors :**

Cette méthode permet de supprimer un acteur possédant la clé « cleActeur », elle est exécuté a base de la clé administrateur, en cas d'erreur d'exécution une exception est levée.

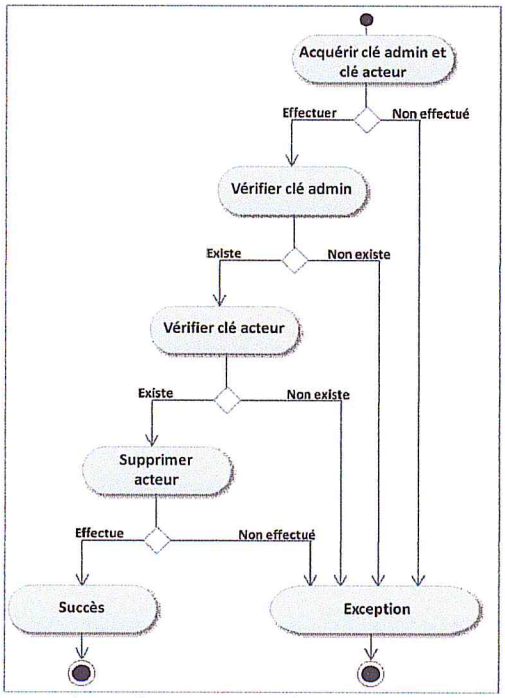

Figure25 : diagramme d'activité delateActors

- **getActor :**

Cette méthode permet retourner les acteurs portant les profile ou le role « profilActeur », elle s'exécute a base de la clé administrateur et le nom de profils pour lequel s'effectue la recherche des acteurs. En cas d'erreur une exception est retournée.

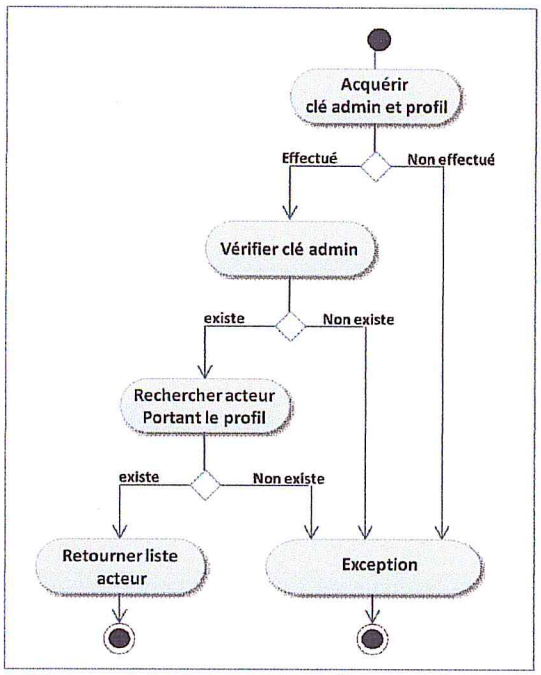

Figure 26: diagramme d'activité getActor

- **getAllActors :**

Cette méthode permet de retourner tout les acteurs qui interagissent avec le composant de validation, elle prend en paramètre la clé administrateur, et retourne une exception en cas d'erreur.

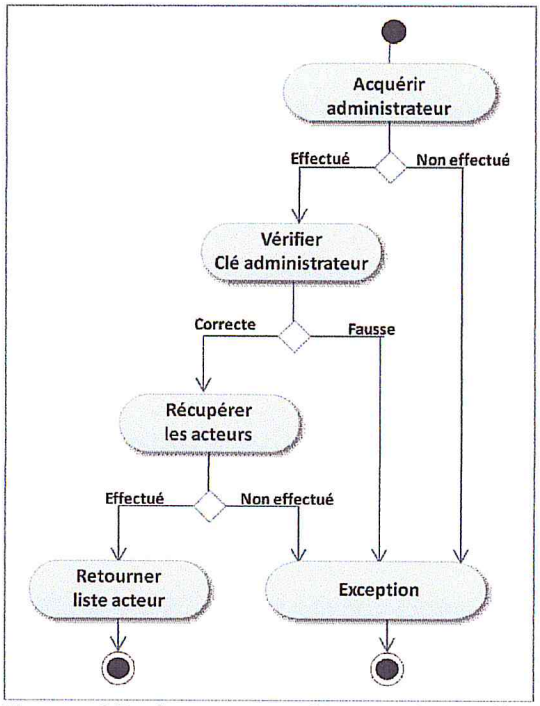

Figure 27: diagramme d'activité getAllActors

- **getKeyActor :**

Cette méthode permet de retourner la clé d'un utilisateur a base de son nom d'utilisateur et son mot de passe. En cas d'erreur elle retourne une exception sinon la clé qui correspond au nom d'utilisateur et mot de passe.

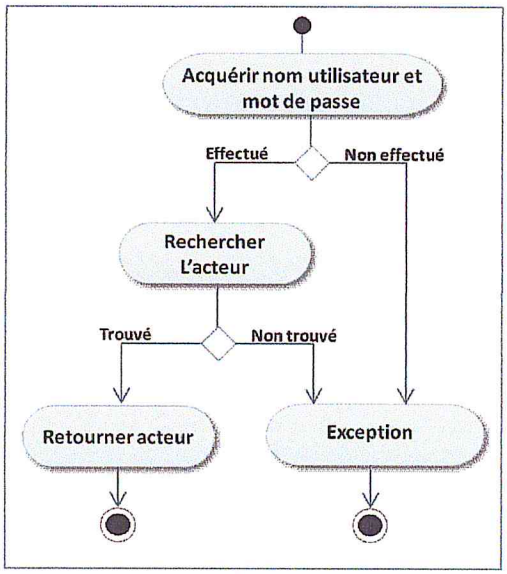

Figure 28: diagramme d'activité getKeyActor

7.1.1.2. L'interface « IInteractionActors » :

Cette interface permet de fournir les méthodes d'interaction interne du composant de validation par exemple : *createActors,* permet de créer la table acteur, elle est utiliser lors de l'instanciation du composant de validation. *InitActors* qui permet d'initialiser l'acteur administrateur, elle est utilisée par le sous composant « instanciation ».

7.1.1.3. L'interface « IActorsKey » :

Cette interface est l'interface requise du composant de validation, elle permet de fournir au composant de validation les méthodes suivantes :

- **getActorsKey :** cette méthode permet de fournir aux composant de validation les clés acteurs qui vont servir a la validation des informations.
- **getNewKey :** cette méthode permet de mettre a jour la clé d'un acteur.

7.1.2. Diagramme de classe :

Dans ce qui suit nous présenterons le digramme de classe du composant acteur :

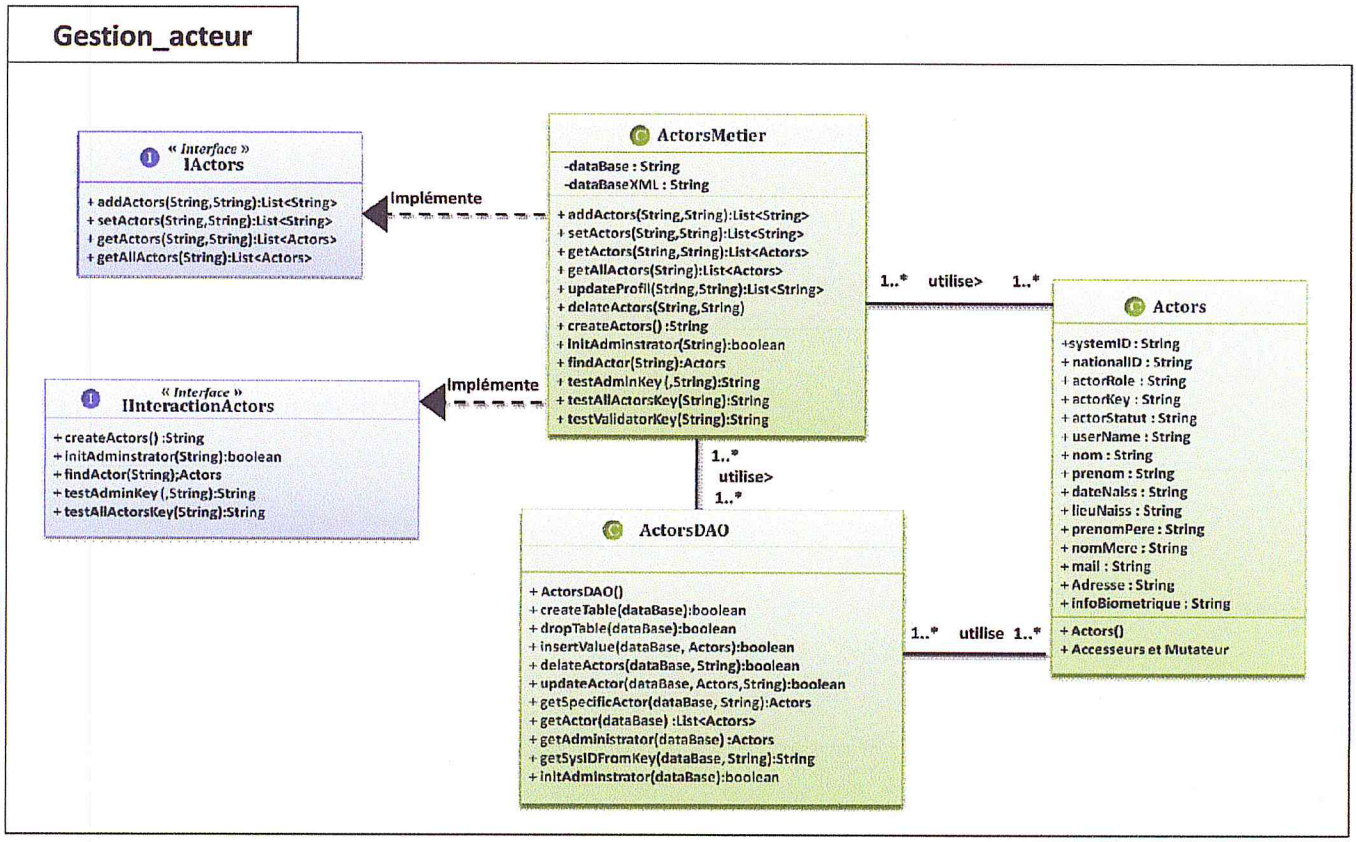

Figure 29: diagramme de classe gestion acteur

7.2. Gestion Type Information :

Le sous composant « *Gestions Type Information*» interagies avec les autres composants grâce aux interfaces fournies « *ITypeInformation, IInteractionTypeInformation* »

7.2.1. Diagramme de composant:

Ce sous composant effectue deux types de traitements concernant les types information et des traitements concernant les informations.

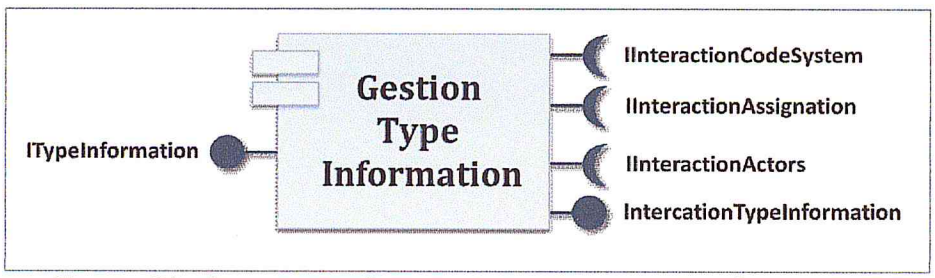

Figure 30: diagramme de composant gestion type information

7.2.1.1. L'interface ITypeInformation :

Les interfaces suivantes permettent de créer et gérer les types information et les unités information ainsi les informations :

- **addInformationType:**

Cette interface permet d'ajouter un nouveau type information qui est décrit dans le fichier « *typeInformation.xml* » fournit en paramètre de cette méthode mais avant un test sur la clé administrateur est fait, si la clé correspond à celle de l'administrateur alors la création sera faite et l'identifiant du type information est retourné, dans le cas contraire une exception est levée.

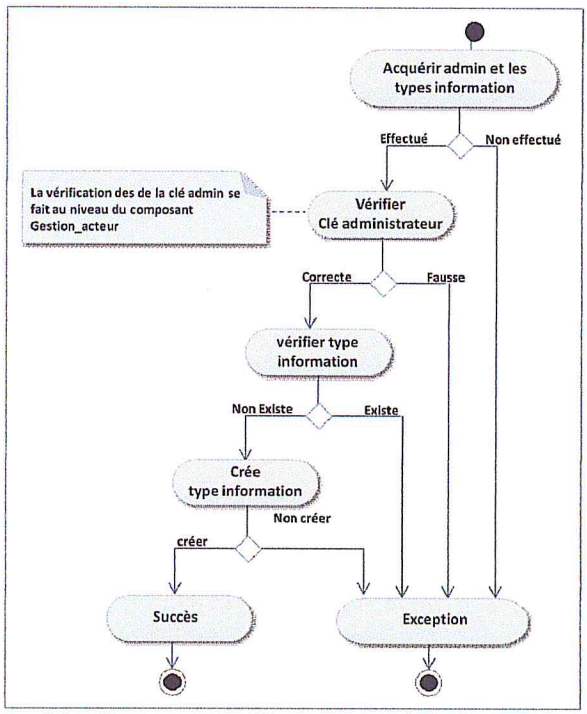

Figure 31: diagramme d'activité addInformationType

- **getInformationTypes :**

Cette méthode retourne le type information qui est gérée par le composant de validation. Elle prend en paramètre la clé administrateur. En cas d'erreur elle retourne une exception.

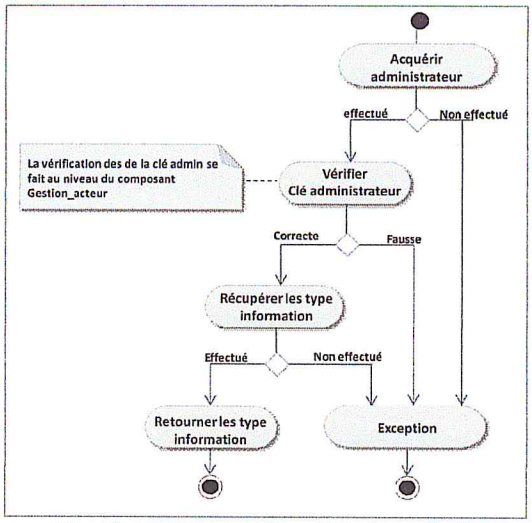

Figure 32: diagramme d'activité getInformationTypes

- **deleteInformationType :**

Cette méthode permet de supprimer le type information fournit en paramètres et ses unités, elle s'exécute à base de la clé administrateur. En cas d'erreur elle retourne une exception.

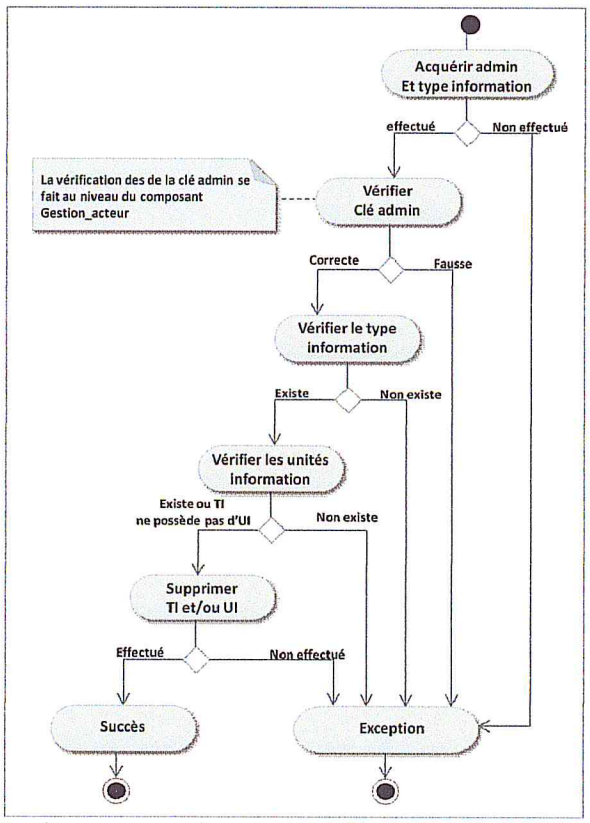

Figure 33: diagramme d'activité deleteInformationType

getUnitInformation :

Cette méthode permet de retourner toutes les unités informations spécifique au nom information fourni en paramètre de la méthode.

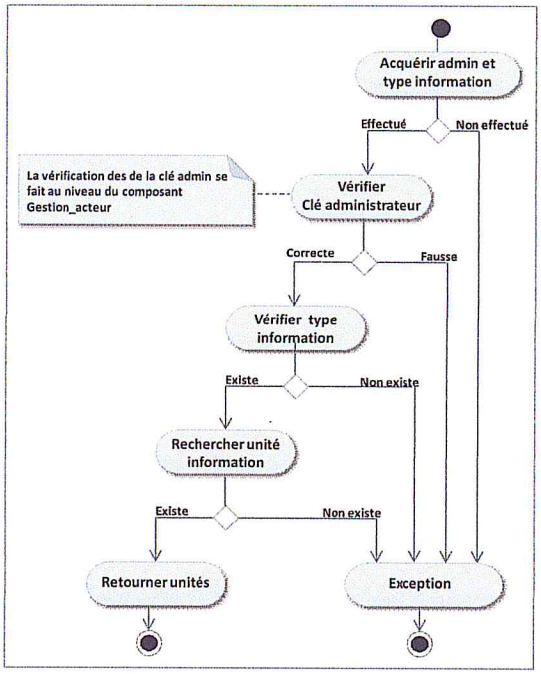

Figure 34: diagramme d'activité getUniteInformation

addInformation :

Cette interface permet d'ajouter des informations à un type information existant spécifier dans le fichier « information.xml » fournit dans les paramètres de la méthode. Ce service est exécuté par tous les acteurs sauf pour un acteur InfoObserver. En cas d'erreur une exception est levée.

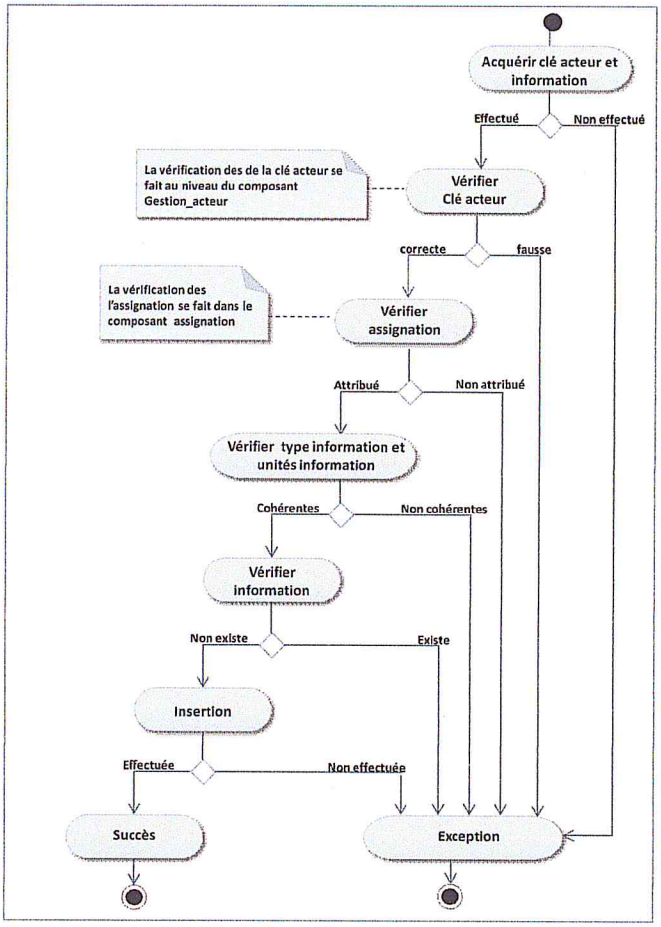

Figure 35: diagramme d'activité addInformation

- **updateInformation :**

Cette méthode permet de faire la mise à jour d'un type information dont la description se trouve dans « *information.xml* » fournit en paramètres de la méthode. Ce service est exécuté par tous les acteurs sauf l'acteur InfoObserver.

La mise à jour d'une information est basée sur l'identifiant unique. En cas d'erreur le service retourne une exception.

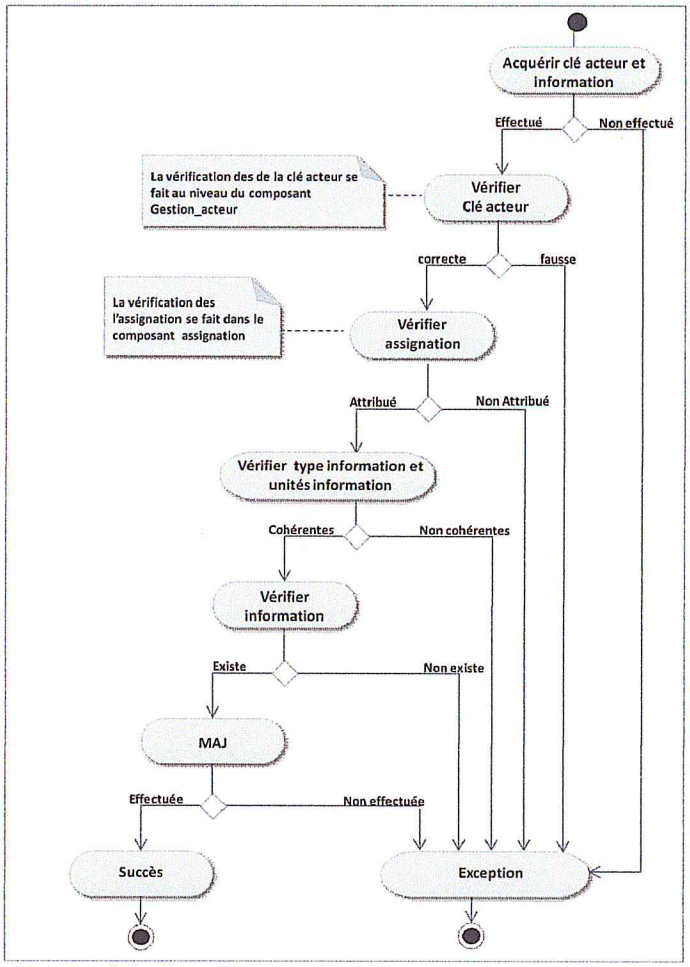

Figure 36: diagramme d'activité updateInformation

- **getInformation :**

Cette méthode peut être utilisée par tous les acteurs. En se basant sur la clé acteur fournit en paramètre elle permet de récupérer les types informations qui sont assigné à l'acteur, et par la suite elle retourne les informations. Ce service retourne une liste des informations du type information recherché.

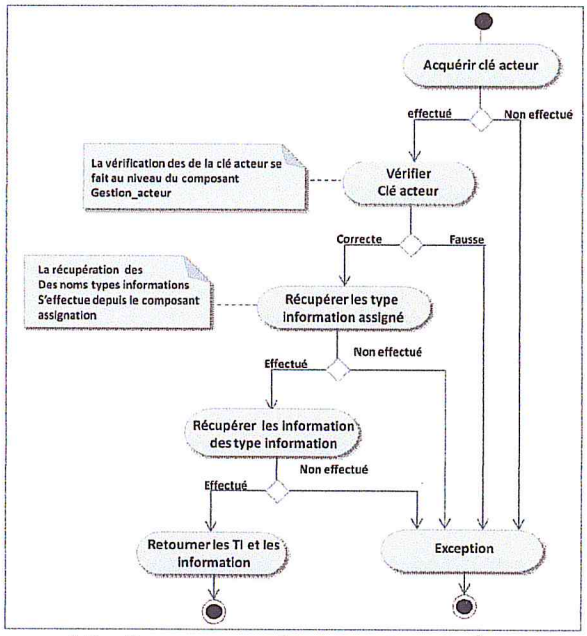

Figure 37: diagramme d'activité getInformation

- **importInformationFromDataBase :**

Cette méthode permet d'importer des données depuis une autre table dans autre base de données en se basant sur les informations décrites dans le fichier « *importerInformation.xml* » fournit en paramètres de la méthode et la clé acteur. Pour chaque information importée une clé unique est attribuée au type information.

Ce service facilite grandement l'importation des données lorsque les tables de l'application existent, l'utilisateur n'aura pas besoin d'insérer les informations mais juste préciser ou sont les informations a importées et cette méthode va se charger du reste des traitements.

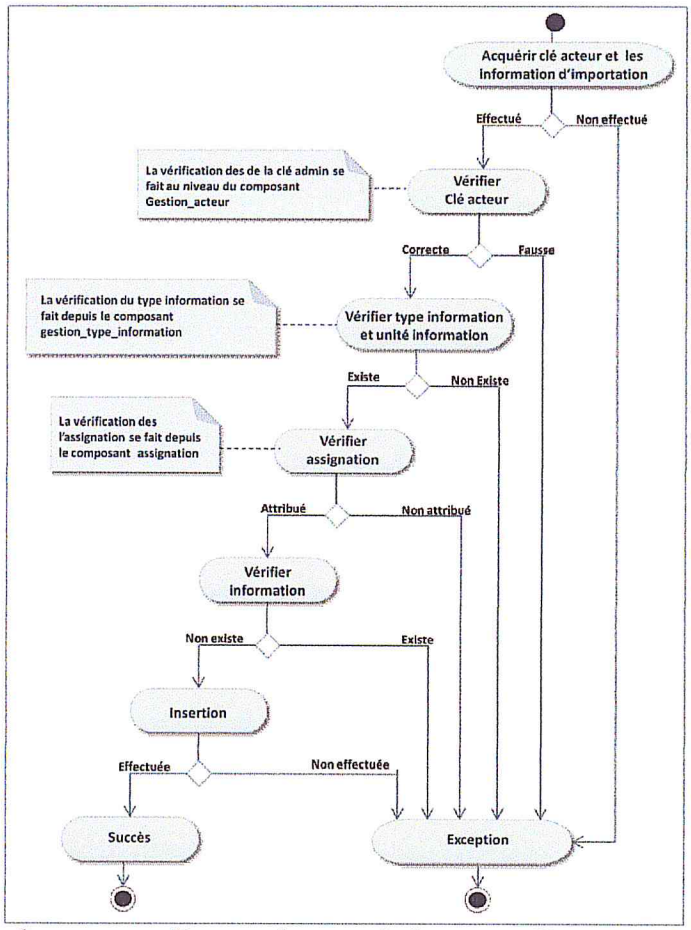

Figure 38: diagramme d'activité importInformationFromDataBase

7.2.1.2. L'interface IInteractionTypeInformation :

Cette interface fournit les interactions et les testes internes comme, *TestTypeInformation* permet de fournir aux composants les types informations, *getIDTypeInformation* Cette interface permet de fournir aux composants les identificateurs des types informations *updateStatutTypeInformation,* permet de mettre a jour les types informations après validation, ou invalidation de l'information. *TestPlageAdresse* cette interface permet de tester si une plage d'adresse est correcte ou non.

7.2.2. Diagramme de classe

Ci-dessous le diagramme de classe de gestion de type information

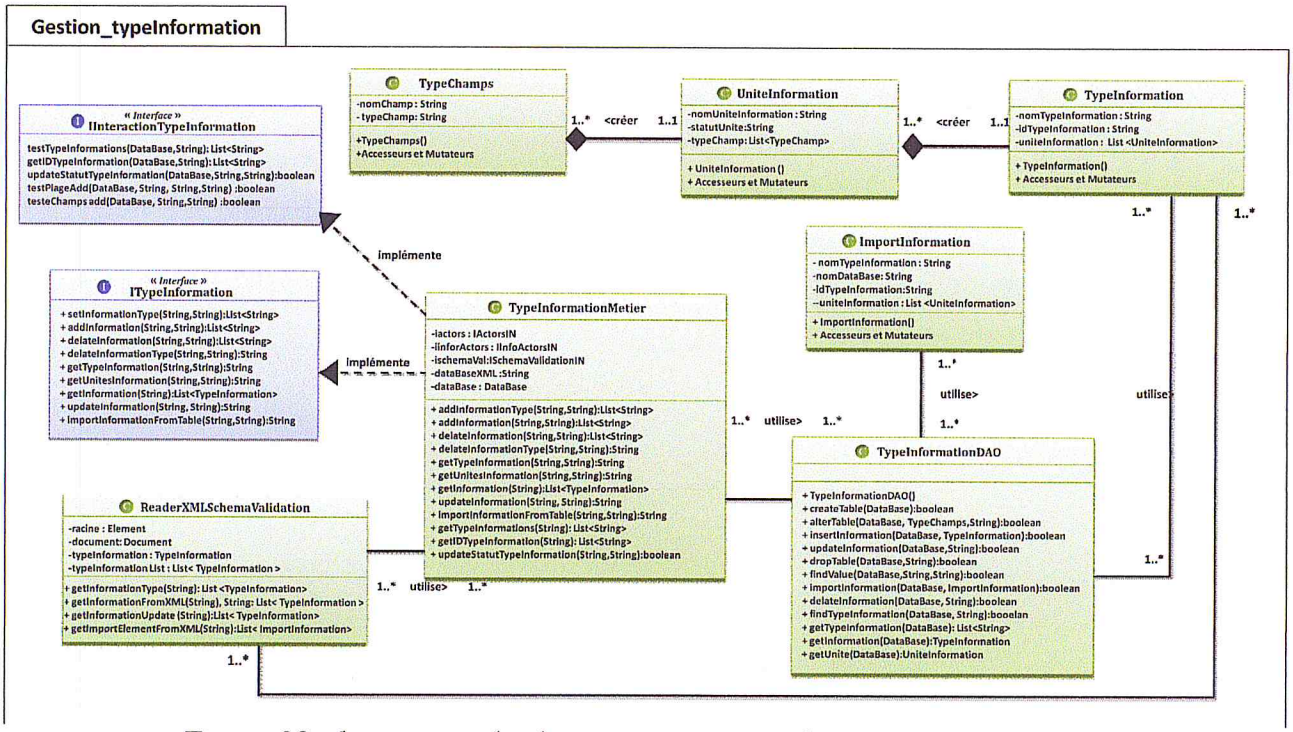

Figure 39: diagramme de classe gestion type information

7.3.Gestion Documents de validation :

Ce sous composant permet la gestion des documents sur laquelle se base la validation, il permet la création, la suppression, la modification et tout les interactions spécifiques aux documents de validation.

7.3.1. Diagramme de composant :

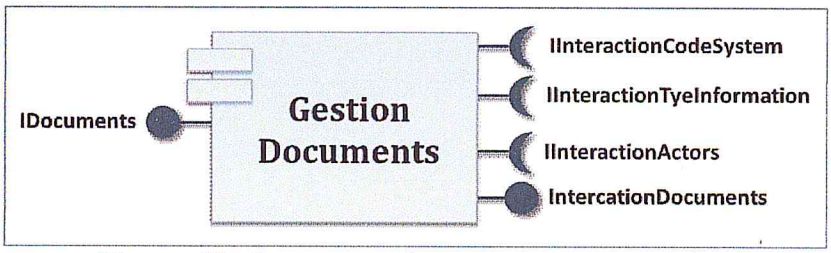

Figure 40: diagramme de composant gestion document

7.3.1.1. L'interface IDocument :

Tous les services de gestions de documents sont exécutés que par un acteur administrateur sauf la consultation des documents.

- **addDocument :**

Cette méthode permet d'ajouter les documents sur lequel se base la validation et de les assigner aux types information à partir de la description du document qui se trouve dans le fichier « *document.xml* » fournit au paramètres de la méthode. Cette interface est exécuté a base de la clé administrateur fournit au paramètres de la méthode en cas d'erreur elle retourne une exception

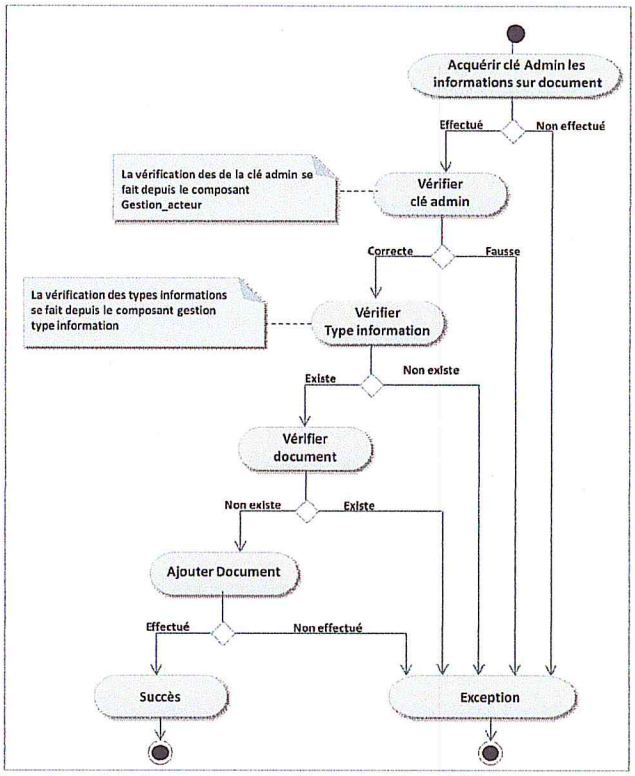

Figure 41: diagramme d'activité addDocument

- **updateDocument :**

Cette méthode permet de mettre à jour un document en se basant sur les informations se trouvant dans *« document.xml »* fournit en paramètres de la méthode en cas d'erreur une exception en cas d'erreur est levé.

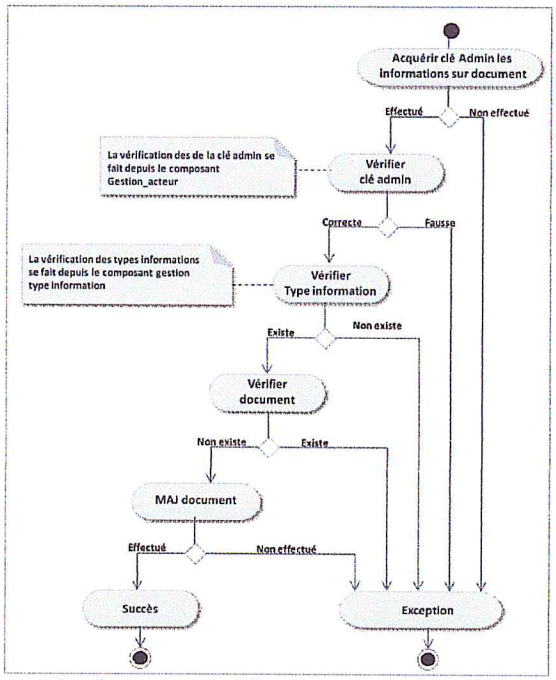

Figure 42: diagramme d'activité updateDocument

- **delateDocument :**

Cette méthode permet de supprimer le nom du document fournit en paramètre. Cette interface s'exécute à base de la clé administrateur. En cas d'erreur elle retourne une exception.

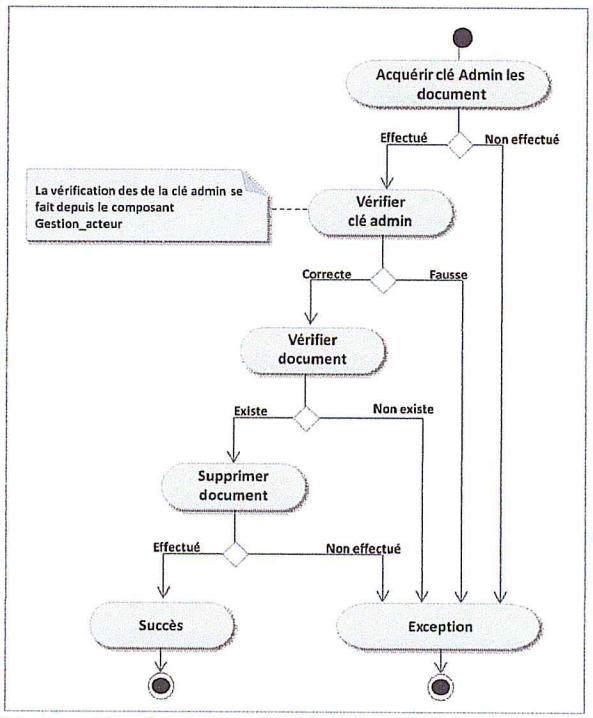

Figure 43: diagramme d'activité delateDocument

- **getDocument :**

Cette méthode permet de retourner tout les documents. Elle peut être exécute part l'administrateur du composant.

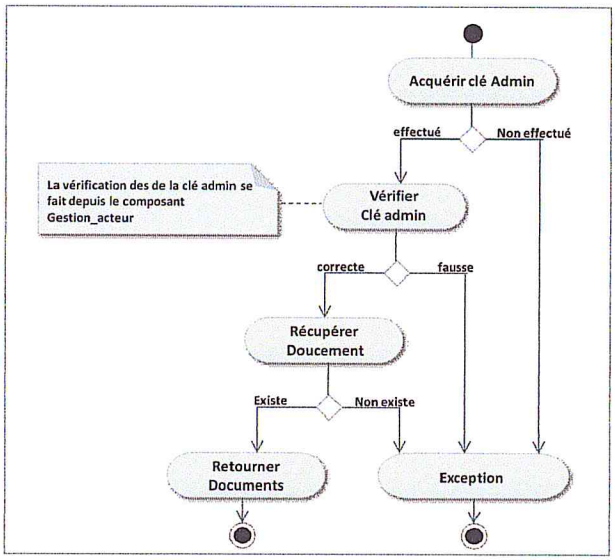

Figure 44: diagramme d'activité getDocument

- **getDocumentOfInformationType :**

Cette méthode permet de retourner l'ensemble des documents relatif au type information spécifié dans les paramètres de cette méthode.
Si le type information existe, elle retourne l'ensemble des documents qui le concerne sinon elle retourne une exception.

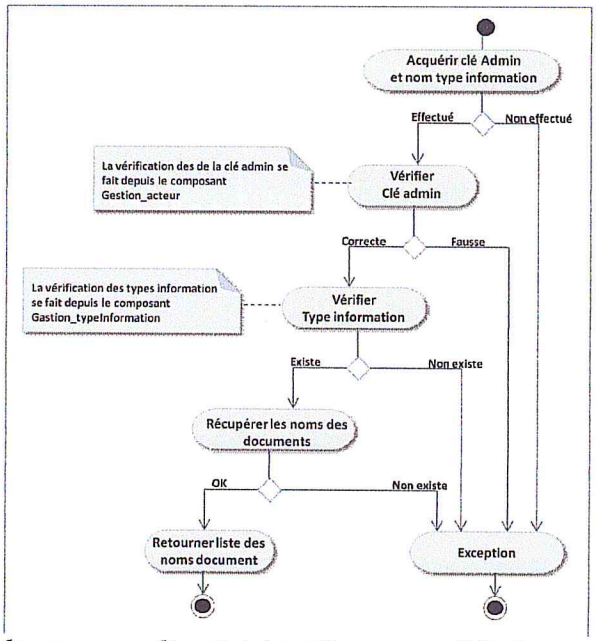

Figure 45: diagramme d'activité getDocumentOfInformationType

7.3.1.2. L'interface IInteractionDocument :

Cette interface fournit les interactions et les testes internes comme *createDocument* permet de créer la table document et *getStatutDocument,* qui retourne le statut du document.

7.3.2. Diagramme de classe :

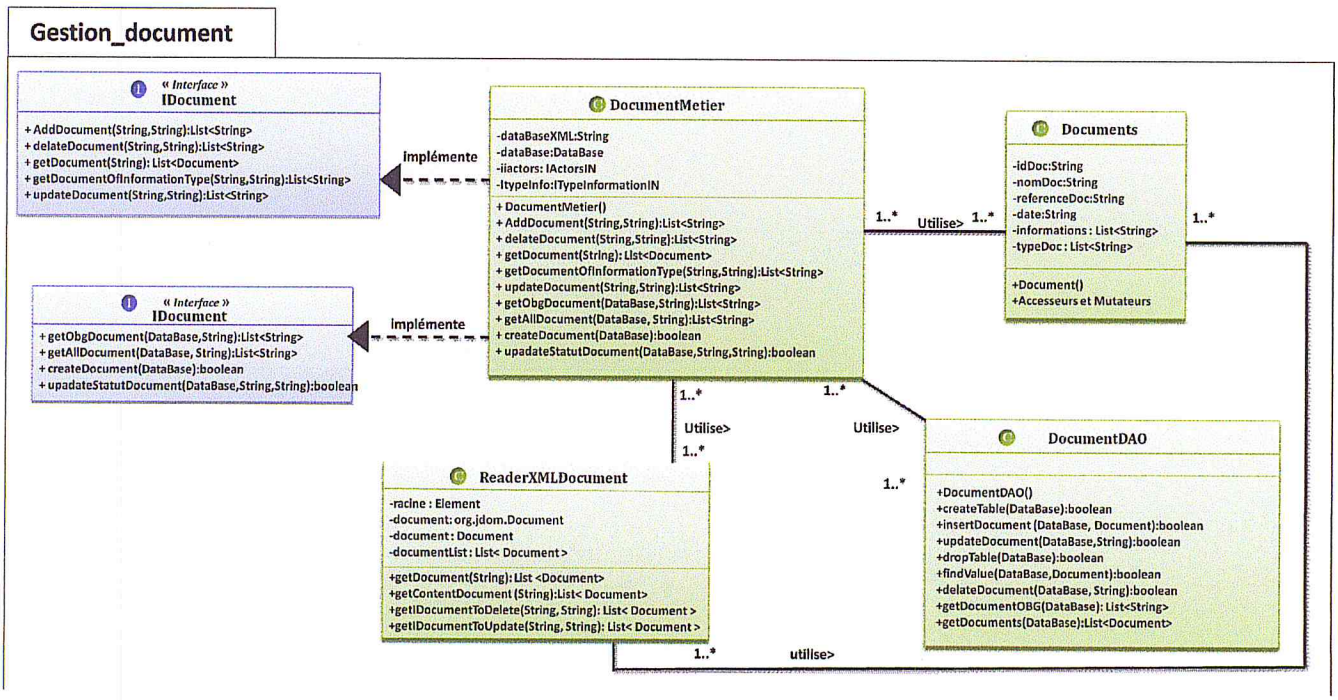

Figure 46: diagramme de classe gestion document

7.4. Gestion de l'assignation d'un acteur a un type information :

Ce sous composant permet l'assignation des types information aux acteurs. L'assignation peut être attribuée par plage, ou par champ ou bien directement par type information.

7.4.1. Diagramme de composant :

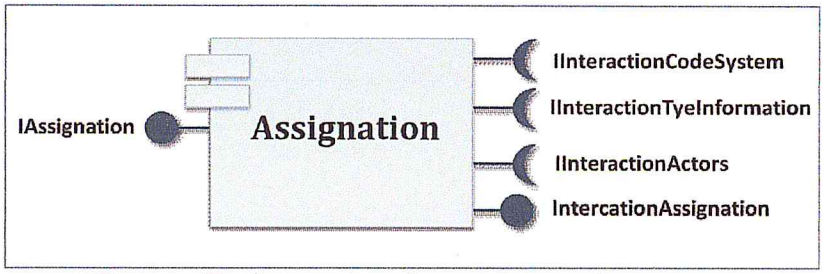

Figure 47: diagramme de composant Assignation

7.4.1.1. L'interface IAssignation :

Ces interfaces sont utilisées pour affecter aux acteurs des types informations.

Une fois qu'un acteur possède le type information, il peut effectuer les traitements dont il est autorisé à faire selon son rôle « *SValidateur, UValidateur, InfoSupplier InfoObserver* »,

- **assignTypeInformation :**

  Cette méthode permet d'assigner un type information à un acteur bien précise, en cas d'erreur elle retourne une exception. Elle prend en paramètre le nom du type information, la clé de l'acteur pour lequel sera attribué le type information, et la clé administrateur.

- **assignInformationRange :**

Cette interface permet d'assigner une plage bien précise du type information spécifier en paramètre de l'interface, par exemple on attribue un intervalle de [0-100].

Elle est exécutée par un acteur administrateur et elle prend en paramètre deux entier qui précise le début et la fin de la plage d'adresse, le nom du type information, la clé de l'acteur pour lequel sera attribué le type information, et la clé administrateur.

- **assignInformationFieldName :**

Cette méthode permet d'assigner un type information celons un champ bien précis. Elle est exécutée par un acteur administrateur et retourne une exception en cas d'erreur.

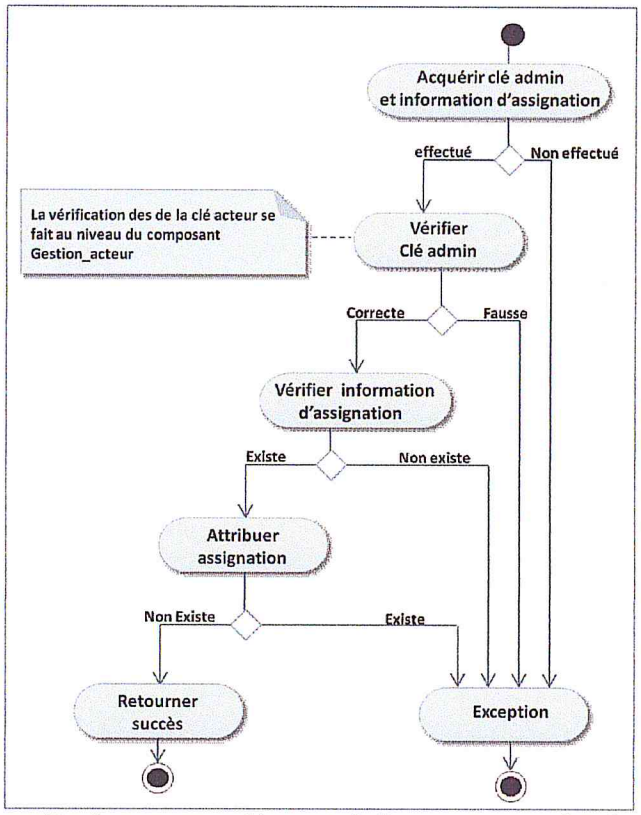

Figure 48: diagramme d'activité d'assignation d'information

Remarque :

Le diagramme d'activité est le même pour les trois interfaces « *figure 48* ».

7.4.1.2. L'interface IInteractionAssignation :

Cette interface fournit les interactions et les testes internes comme *createAssignation* permet de créer la table d'assignation ou seront stocker les

informations d'assignation des acteurs et des types information et comme *testAssignation* qui permet de tester les assignations des acteurs.

7.4.2. Diagramme de classe :

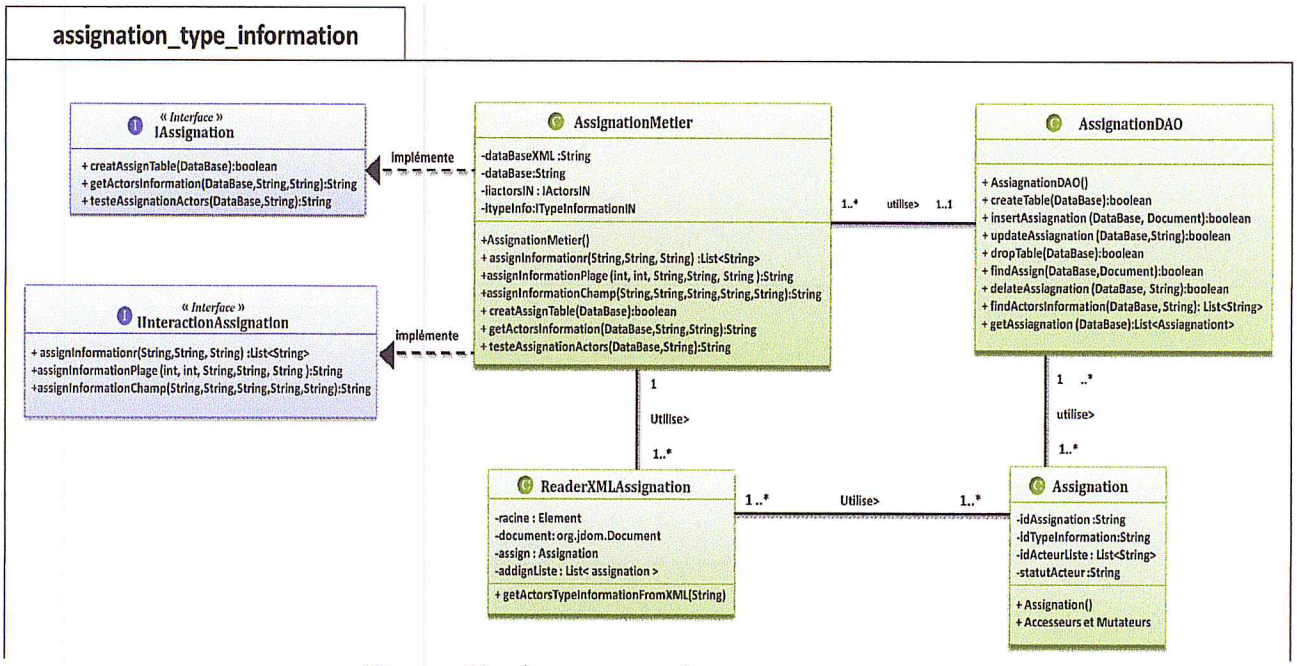

Figure 50: diagramme classe assignation

7.5. Gestion du schéma de validation :

Ce sous composant permet la gestion des schémas de validation dans le composant de validation, il permet la création, la suppression, la modification et tout les interactions spécifiques aux schémas de validation.

7.5.1. Diagramme de composant :

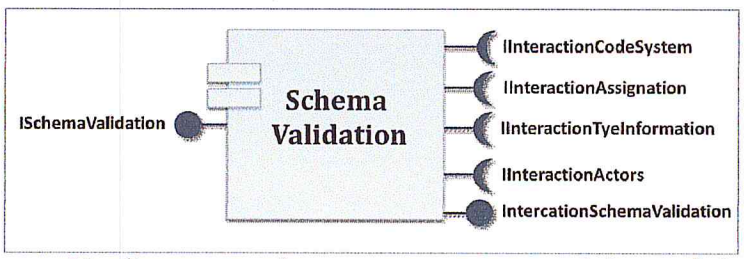

Figure 51: diagramme de composant gestion schéma validation

7.5.1.1. L'interface ISchemaValidation :

Le schéma de validation est mis en place en se basant sur le processus de validation dans l'application ou sera instancier le composant, celui-ci est différent d'une application a une autre et il varie d'un très simple a un état très complexe.

- **addValidationShema :**

Cette méthode permet d'ajouter les schémas de validation qui se trouvent dans le fichier *«schemaDeValidation.xml»* fournit en paramètre de la méthode. Ce service est autorisé juste pour un administrateur. En cas d'erreur le service retourne une exception.

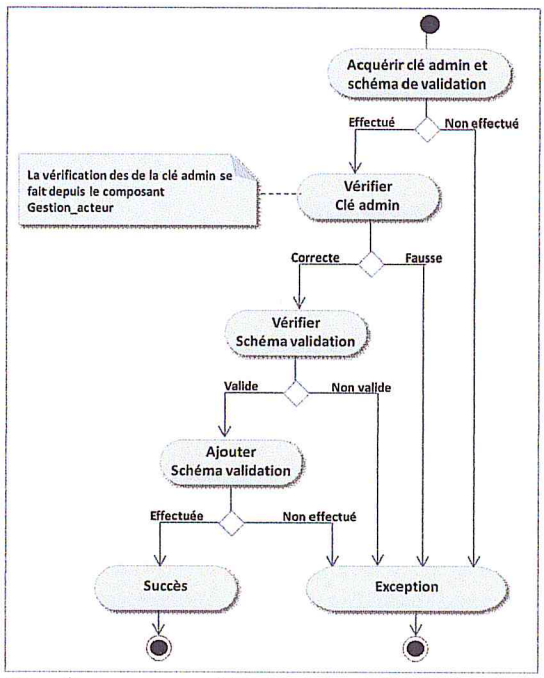

Figure 52: diagramme d'activité addValidationShema

- **delateValidationShema:**

Cette méthode permet de supprimer un schéma de validation fournit en paramètre de la méthode. Ce service ne peut être exécuté que par un administrateur, en cas d'erreur une exception est levée.

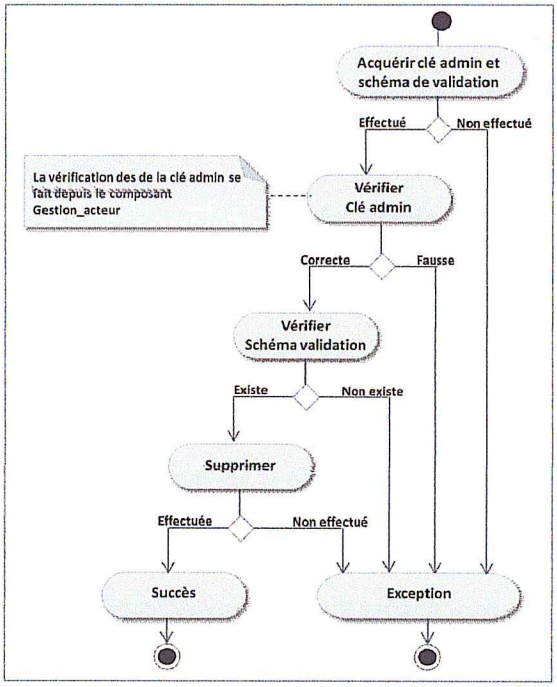

Figure 53: diagramme d'activité delateValidationShema

- **updateValidationShema :**

Cette méthode permet de mettre a jour un schéma de validation décrit dans le fichier *«schemaValidation.xml »* dans le cas contraire une erreur est retourné. Ce service ne peux être exécute que par une clé administrateur.

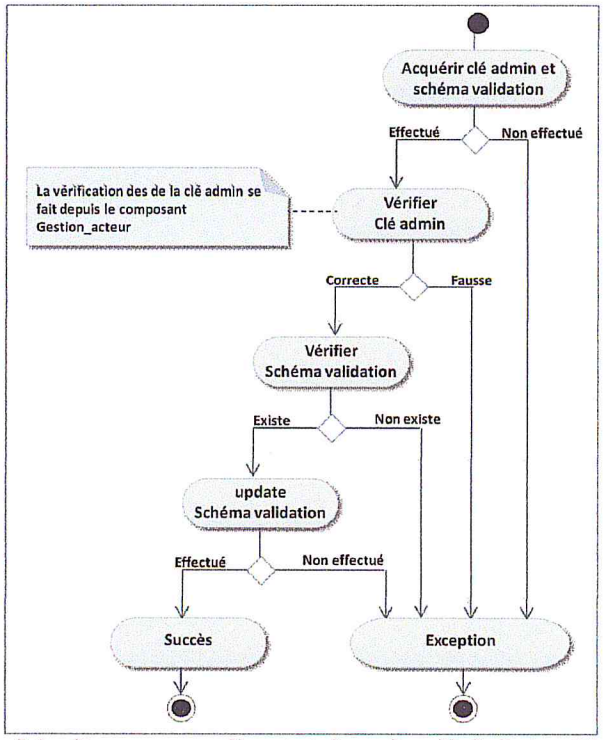

Figure 54: diagramme d'activité updateValidationShema

- **getShemaToValidate :**

Cette méthode permet de retourner le code de validation attribué à un acteur pour valider une information, le nom du type information, et la date limite de validation, sinon elle retourne un code d'erreur. Cette méthode est exécuté que par les validateurs « *Administrateur, Validateur, UValidateur*» qui participe a un processus de validation. En cas d'erreur une exception est levée.

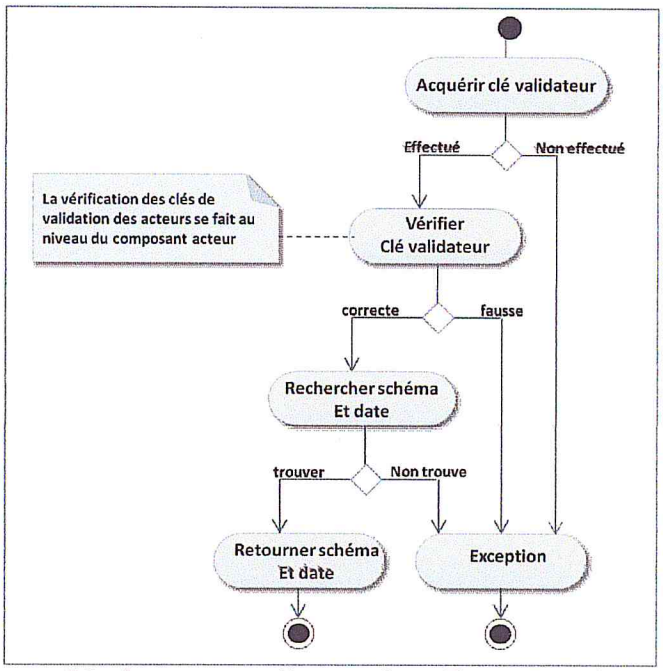

Figure 55: diagramme d'activité getShemaToValidate

- **addActorsToValidationShema:**

Cette méthode permet d'affecter un ou plusieurs acteurs à des différentes phases et taches d'un schéma de validation. Elle est effectue par un acteur administrateur.

Elle en paramètre le nom du schéma de validation, le nom du type information, le schéma de validation, la phase, la tache ainsi la clé de l'acteur pour lequel on désir ajouter au schéma de validation. En cas d'erreur une exception est levée.

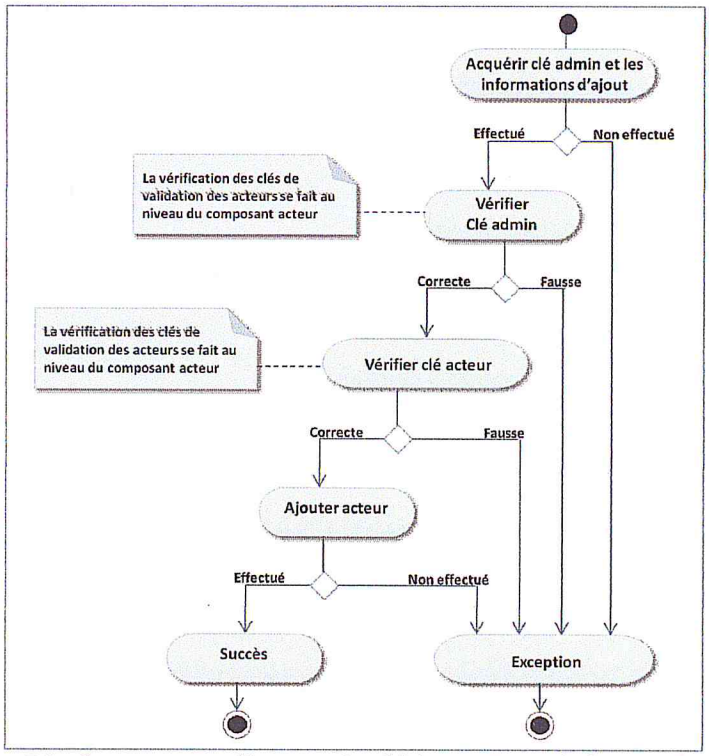

Figure 56: diagramme d'activité addActorsToValidationShema

- **delateActorFromSpecficPhase :**

Cette méthode permet de retirer un acteur depuis une phase bien précise d'un schéma de validation. Cette action est faite que par un acteur administrateur.

Elle prend en paramètre le nom du schéma de validation, le nom du type information, le schéma de validation, la phase, la tache ainsi la clé de l'acteur pour lequel on désir ajouter au schéma de validation. En cas d'erreur une exception est levée.

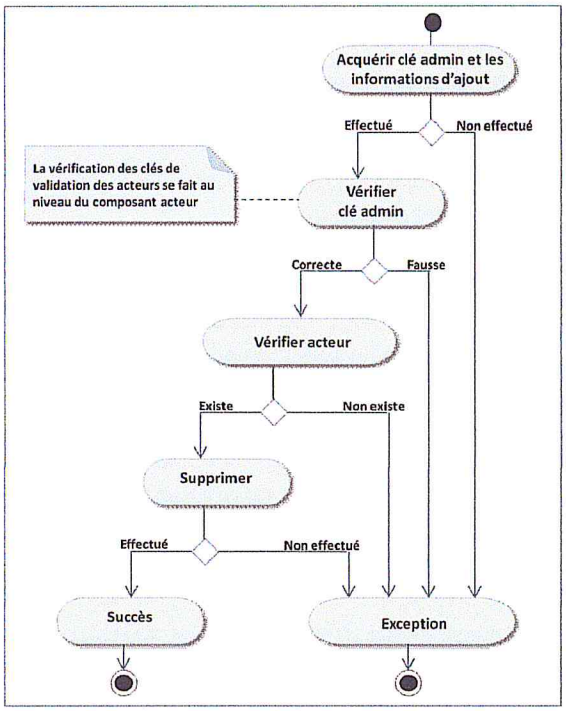

Figure 57: diagramme d'activité delateActorFromSpecficPhase

- **updateDateOfValidationShema :**

Cette méthode permet de modifier la date d'un schéma de validation. Elle est exécutée que par un acteur administrateur. Elle prend en paramètre le nom du schéma de validation, le type information, la date. En cas d'erreur une exception est levé.

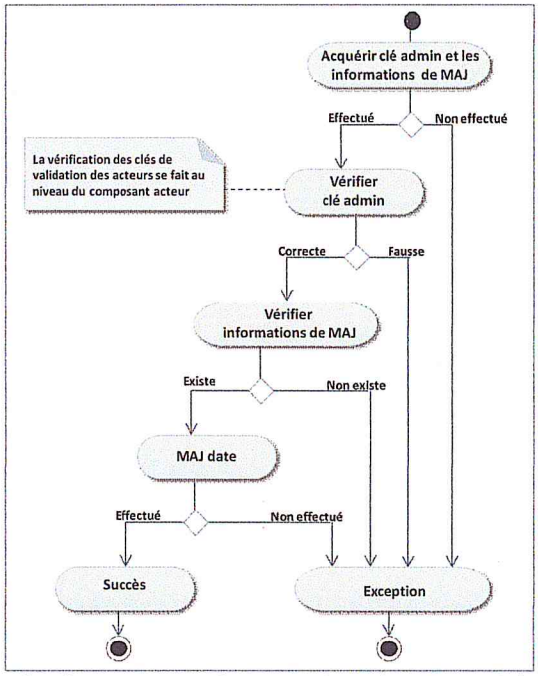

Figure 58: diagramme d'activité updateDateOfValidationShema

- **verifyDateOfValidationShema:**

Permet de vérifier si la date d'un schéma de validation a été expirer ou non, elle retourne la date du schéma de validation. Elle prend en paramètre le nom du schéma de validation.

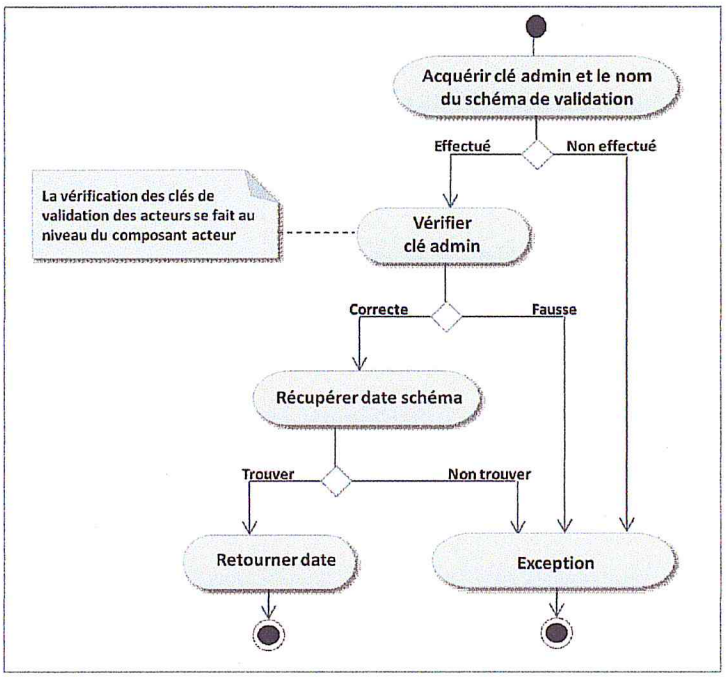

Figure 59: diagramme d'activité verifyDateOfValidationShema

7.5.1.2. L'interface IInteractionSchemaValidation :

Cette interface fournit les interactions et les testes internes comme *createSchema* permet de créer la table du schéma de validation, et *findSysInfoInShema*, permet de tester les identificateurs des types informations dans un schéma de validation

7.5.2. Diagramme de classe :

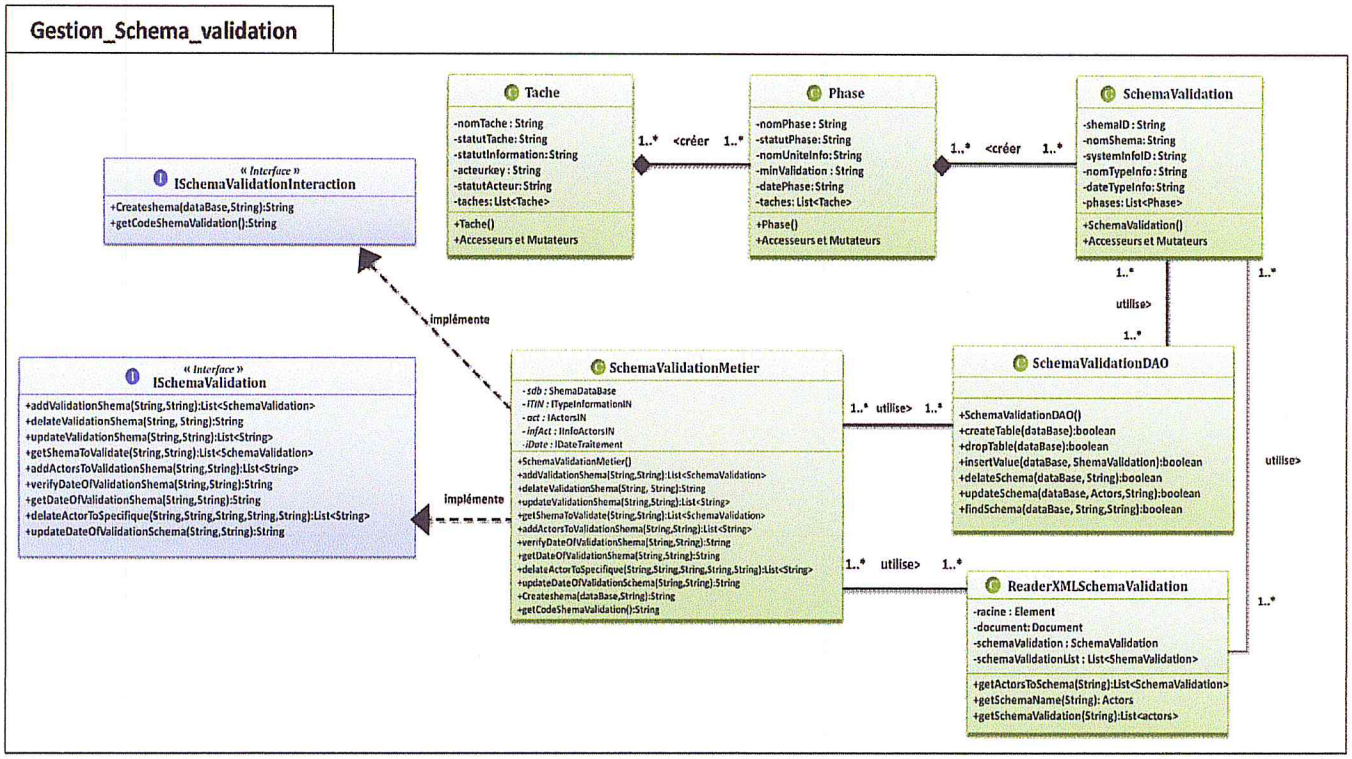

Figure 60: diagramme de classe gestion schéma validation

7.6.La Validation :

Ce sous composant permet la gestion de la validation d'information.

7.6.1. Diagramme de composant :

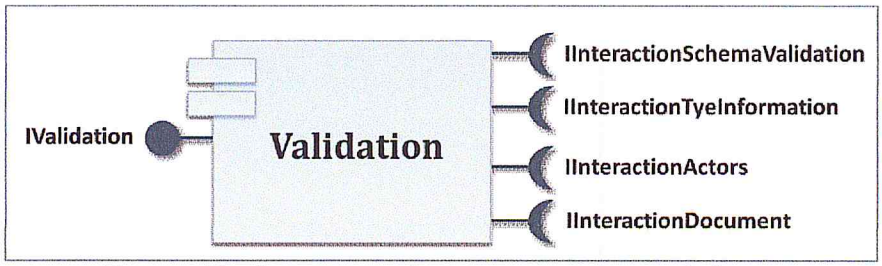

Figure 61: diagramme de sous composant validation

7.6.1.1. L'interface IValidation:

- **validateInformation:**

Cette méthode permet de validé, ou d'invalider le type information en se basant sur les documents sur lesquels de base la validation, si les documents ne sont pas préciser lors de la validation, l'information sera automatiquement invalidé.

Elle est exécuté avec les clés : « *administrateur, UValidateur et SValidateurs* ». Si c'est un acteur validateur, elle autorise la mise à jour de l'information si elle n'a pas reçu une validation au préalable sinon elle retourne une exception.

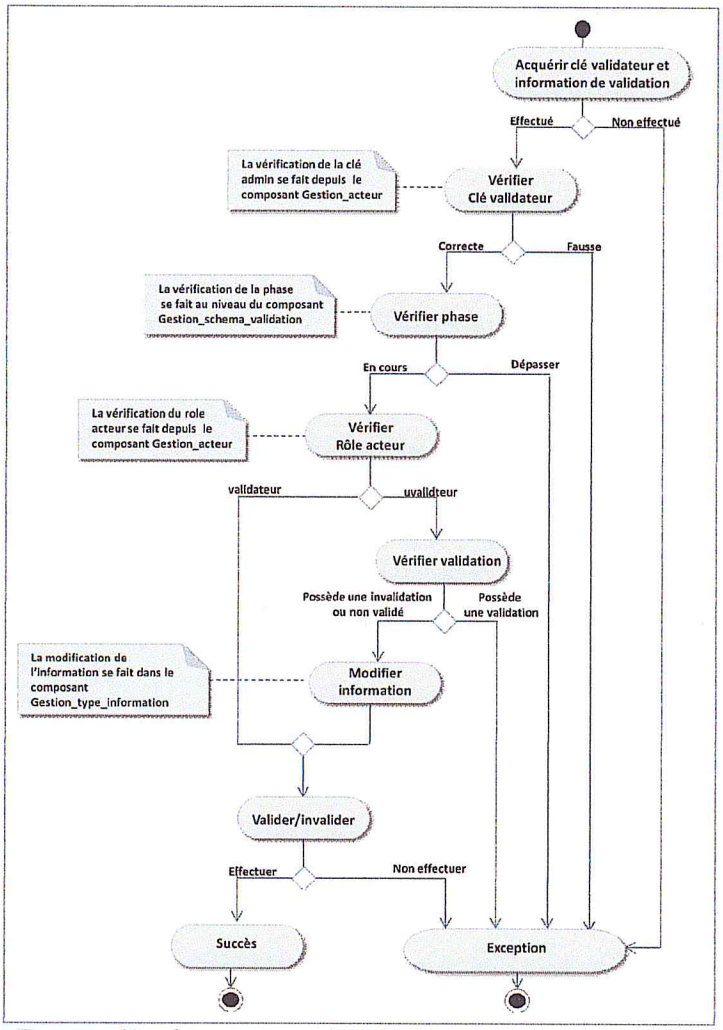

Figure 62: diagramme d'activité validateInformation

7.6.2. Diagramme de classe :

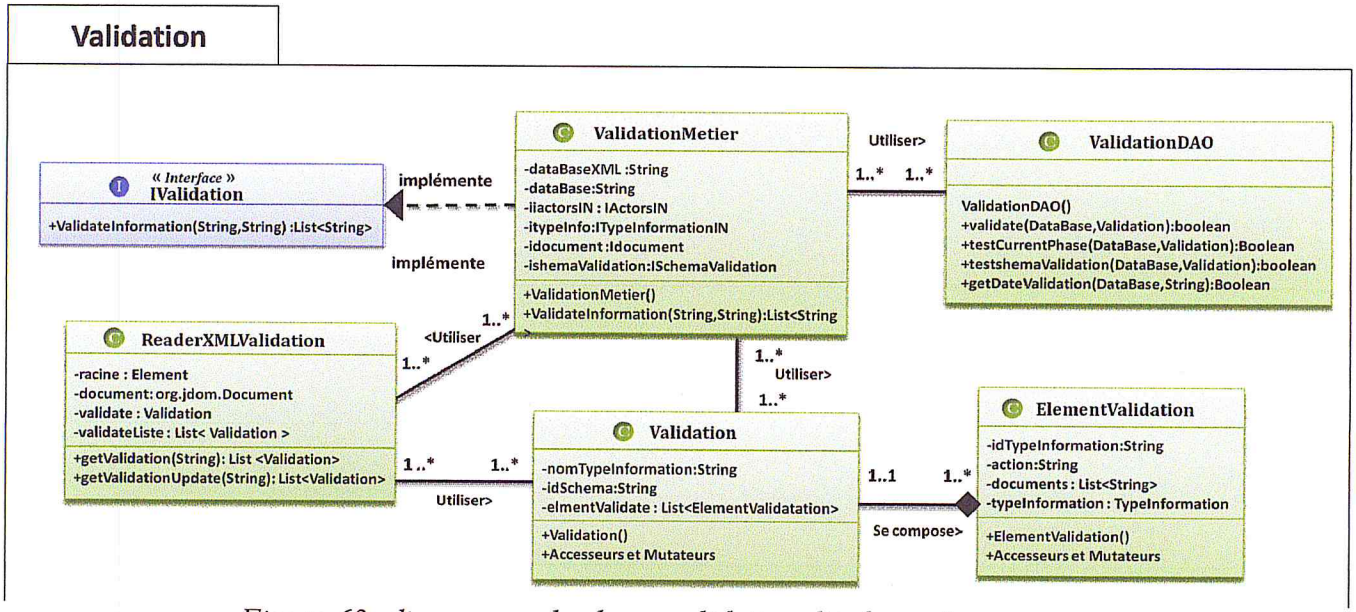

Figure 63: diagramme de classe validation d'information

8. Conclusion :

Dans cette partie nous avons présenté en premier lieu l'architecture générale de notre composant de validation, pour cela nous nous sommes basé sur un modèle de composant. Ce modèle s'inspire du modèle de composant de l'approche IASA et de FRACTAL.

Puis nous avons présenté la conception détaillé de chaque sous composant a l'aide des diagrammes d'activités et des classe.

Dans le chapitre suivant nous présenterons l'implémentation et les teste de notre composant de validation.

V

Implémentation et Testes

Introduction
Environnement de développement
Implémentation dans une application
Implémentation dans un ADL
Conclusion

1. Introduction :

Dans ce chapitre nous présenterons nos outils, nous parlerons de nos choix de technologies utilisées pour le développement du composant de validation, telles que le langage de programmation, le SGBD, les serveurs…etc., ensuite nous présenterons deux manières d'implémentation de notre composant, la première dans une application web de gestion de scolarité et une autre dans un ADL.

2. Environnement de développement :

Dans cette partie nous allons présenter les outils et technologies de développement que nous avons utilisées :

2.1.Le langage java :

Nous avons choisi le langage de développement java car java est un langage orienté objet simple et portable « Il peut être utilisé sous différentes plates formes sans aucune modification », java possède aussi une riche bibliothèque de classes.

2.2.Eclipse :

Eclipse est un environnement de développement intégré (Integrated Development Environment) dont le but est de fournir une plate-forme modulaire pour permettre de réaliser des développements informatiques.

2.3.MySQL 5.0 :

MySQL est un serveur de bases de données relationnelles SQL développé et diffusé sous une licence libre, la GNU General Public License. MySQL fonctionne sur un très grand nombre de plates-formes et de systèmes d'exploitation. Le SQL dans "MySQL" signifie "Structured Query Language" : le langage standard pour les traitements de bases de données.

3. Implémentation dans une application :

Pour intégrer le composant logiciel nous avons mis en place une application de teste, et nous avons choisit comme thématique la gestion de scolarité.
Et pour intégrer le composant de validation dans l'application, nous l'avons intgerer en « .jar »

3.1. Définition de l'application :

Cette application permet de gérer la scolarité, c'est application web développé en utilisant les jsp/servlet. Pour ce faire nous avons utilisé comme serveur d'application Tomcat 7.0 et éclipse JEE.

Le but de cette application est de montrer comment le composant de validation s'intègre dans les applications, d'une autre manière l'objectif de son développement est le teste du composant de validation

Dans la suite de cette partie nous allons présenter des captures d'écran de l'application mis en place et en parallèle le code sources d'intégration du composant de validation.

3.2. Les étapes d'intégration :

Nous allons prendre un exemple d'ajout d'une carte d'étudiant

3.2.1. Intégration du composant :

Puis que le composant de validation est un fichier « .jar » nous allons l'intégrer dans la librairie de l'application de gestion de scolarité.

Nous avons précisé dans le chapitre précédent que le composant de validation requis les services d'un composant de sécurité, c'est pour cela que nous avion ajouter le composant de sécurité dans l'application.

Il est à remarque qu'au niveau du « WEB-INF/src » il n'y a aucun code qui permet d'interagir avec la validation.

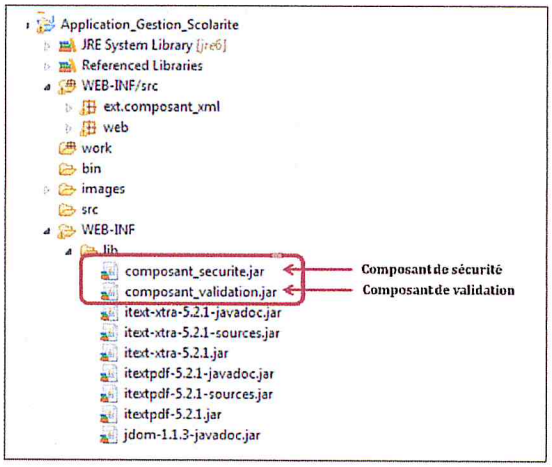

Figure 64: intégration du composant de validation dans l'application de scolarité

3.2.2. Instanciation du composant :

Lors de l'installation de l'application l'administrateur va configurer le fichier xml « shemaDataBase.xml » qui contient des informations sur la base de données et sur le composant de validation,

```
<?xml version="1.0" encoding="UTF-8"?>
<shemaDataBase>
    <connexion>
      <userName>root</userName>
      <pasword>123</pasword>
    </connexion>
    <shema>
      <nomComposant>composant_validation</nomComposant>
      <userAdmin>admin</userAdmin>
      <pswAdmin>admin</pswAdmin>
    </shema>
</shemaDataBase>
```

Information sur la connexion avec la base de donnée

Information sur la connexion avec le composant de validation

Figure 65: fichier « schemaDataBase.xml »

Ce fichier sera instancier grâce a la méthode build « *figure 66* » (2)

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ page import="controle_instanciation.** %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
 <link rel="stylesheet" href="images/style.css" />
<title>Composant de validation</title>
</head>
<body style ="background-image:url(images/i
<!-- -------------- JSP---------------------- -->
        <%

           Controle_Instantiation.build("d:/xml/shemaDataBase.xml");
        %>
<!--  -------------- fin---------------------- -->
        <!-- <img src="images/enTete.png" > -->
```

Importer le composant de validation (1)

Contrôle et instanciation du composant (2)

Figure 66: code JSP d'intégration

Si les informations présenter dans le fichier xml sont correctes, l'accès au composant de validation sera autoriser dans le cas contraire l'accès ne sera pas autoriser.

Figure 67: Résultat d'intégration

3.2.3. Ajout d'un acteur :

L'ajout d'un acteur fait appel à l'interface du composant addActors,

- Interface graphique correspondant à la gestion de l'acteur :

Figure 68: ajout acteur

- Code JSP correspondant a l'interface graphique et a l'interface du composant

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ page import="cmp.gestion_actor.*,ext.composant_securite.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
 <link rel="stylesheet" href="images/styleResultat.css" />
<title>Composant de validation</title>
</head>
<body style ="background-image:url(images/img.jpg); " >

        <%
        ActorsBEAN  acteur = new ActorsBEAN();
        acteur.setActorRole(request.getParameter("role"));
        acteur.setNationalID(request.getParameter("nationalID"));
        acteur.setSystemIDEx(request.getParameter("systemID"));
        acteur.setActorStatut("initial");
        acteur.setPrenom(request.getParameter("prenom"));
        acteur.setNom(request.getParameter("nom"));
        acteur.setLieuNaiss(request.getParameter("lieuN"));
        acteur.setUserName(request.getParameter("user"));
        acteur.setPsw(request.getParameter("psw"));
        acteur.setFonction(request.getParameter("fonction"));
        acteur.setMail(request.getParameter("mail"));
        acteur.setAdresse(request.getParameter("add"));
        acteur.setNomMere(request.getParameter("prM"));

        IActors act = new ActorsMetier();
        String key =request.getParameter("cle");

        out.println("<h3 style=\"margin-top :230px; margin-left : 300px;  font-family:camberia,serif;\">"
          +act.addActor(acteur, key)+"</h3>");
        %>
```

Importer le composant de validation
Et le composant de sécurité

Invocation de l'interface addActor

Figure 69: code JSP ajout acteur

- Résultat de l'ajout d'un acteur :

Figure 70: résultat de l'ajout, la clé acteur

3.2.4. Ajout type information :

- L'ajout d'un type information se fait par l'interface addInformationType de la même manière que l'ajout d'un acteur

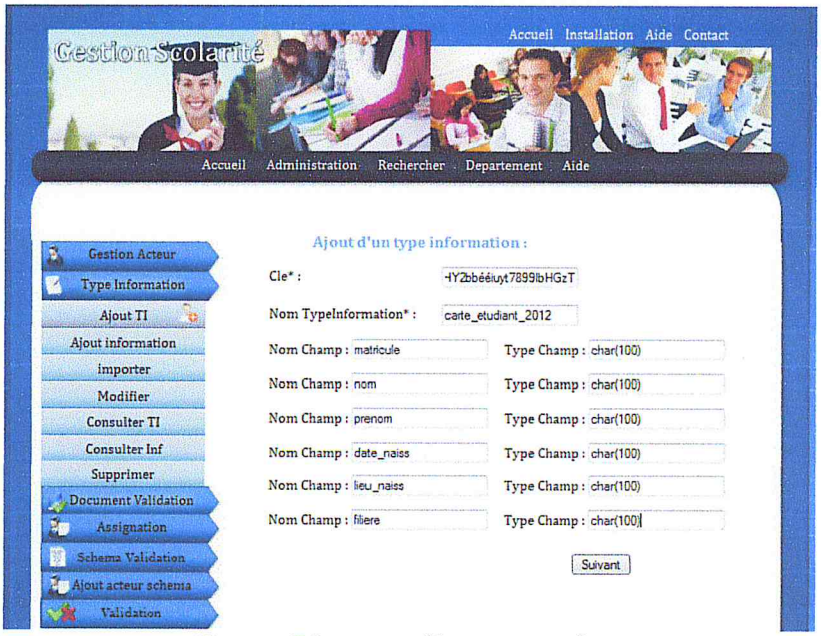

Figure 71: ajout d'un type information

- le code JSP correspondant a l'ajout du type information

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ page import="cmp.gestion_type_information.*,ext.composant_xml.*" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" href="Images/styleResultat.css" />
<title>Composant de validation</title>
</head>
<body style ="background-image:url(images/img.jpg); " >
<%
        String key =request.getParameter("cle");
       TypeInformationBEAN  ti = new TypeInformationBEAN();
        ti.setNomTable(request.getParameter("typeInf"));

        UniteInformationBEAN unite = new UniteInformationBEAN();
        List <UniteInformationBEAN> liste = new LinkedList <UniteInformationBEAN>();
        unite.setStatutInfo("initial");
        unite.setNomTable(request.getParameter("typeInf"));
        TypeChamp typeChamp = new TypeChamp();
        List <TypeChamp> tpeCliste = new LinkedList <TypeChamp>();
        typeChamp = new TypeChamp();
        typeChamp.setNomChamp(request.getParameter("nom1")); typeChamp.setTypeChamp(request.getParameter("type1")); tpeCliste.add(typeChamp); typeChamp = new TypeChamp();
        typeChamp.setNomChamp(request.getParameter("nom2")); typeChamp.setTypeChamp(request.getParameter("type2"));tpeCliste.add(typeChamp);typeChamp = new TypeChamp();
        typeChamp.setNomChamp(request.getParameter("nom3")); typeChamp.setTypeChamp(request.getParameter("type3"));tpeCliste.add(typeChamp);typeChamp = new TypeChamp();
        typeChamp.setNomChamp(request.getParameter("nom4")); typeChamp.setTypeChamp(request.getParameter("type4"));tpeCliste.add(typeChamp);typeChamp = new TypeChamp();
        typeChamp.setNomChamp(request.getParameter("nom5")); typeChamp.setTypeChamp(request.getParameter("type5"));tpeCliste.add(typeChamp);typeChamp = new TypeChamp();
        typeChamp.setNomChamp(request.getParameter("nom6")); typeChamp.setTypeChamp(request.getParameter("type6"));tpeCliste.add(typeChamp);typeChamp = new TypeChamp();
         unite.setTypeChampList(tpeCliste);
         liste.add(unite);
        ti.setUnitesInfo(liste);
        TypeInformation ti2 = new TypeInformation();
        GenererXML_IHM.creerTypeInformation("d:/xml/typeInformation.xml",ti);
        ComposantXML_IHM.creerCmpInformation("d:/xml/contenuInformation.xml",ti2);
        out.println("<h3 style = \"margin-top :230px; margin-left : 300px;  font-family:cambria,serif;\">"+rtiI.addInformationType("d:/xml/typeInformation.xml", key)+"</h3>");
```

Importer le composant de validation

Générer le fichier xml correspondant

Invoquer l'interface de création

Figure 72: code JSP, ajout d'un type information

- Le résultat de l'ajout d'un type information

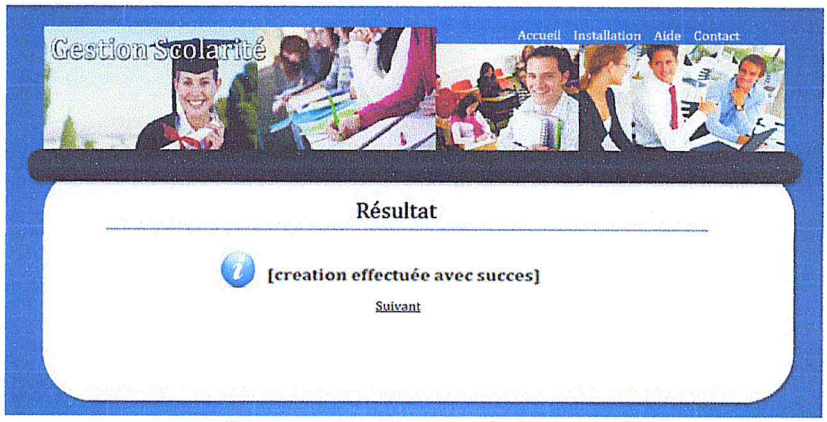

Figure 73: résultat de l'ajout du type information

- Ajout de l'information d'un type information

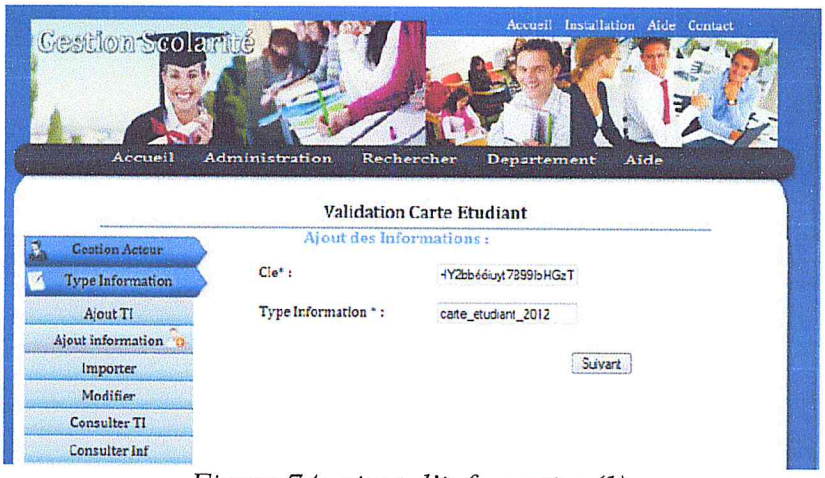

Figure 74: ajout d'information(1)

Figure 75: ajout d'informations(2)

- Code de l'information retourné

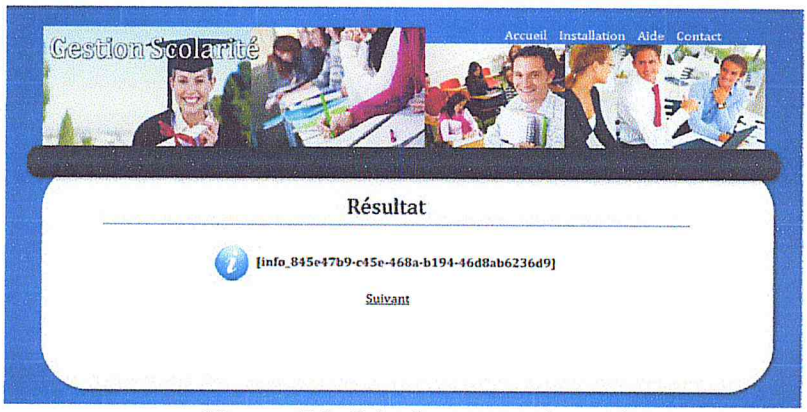

Figure 76: Résultat ajout des informations

3.2.5. Création schéma de validation :

Figure 77: création d'un schéma de validation

3.2.6. validation

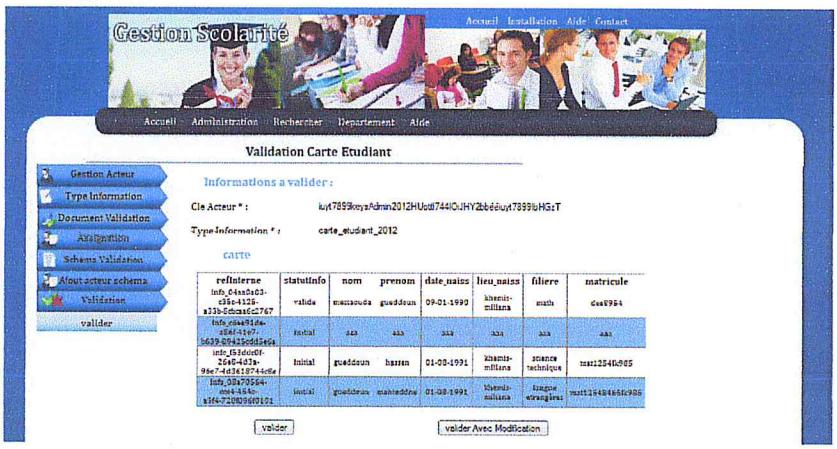

Figure 78: Accès a la validation d'information

- validation sans modification :

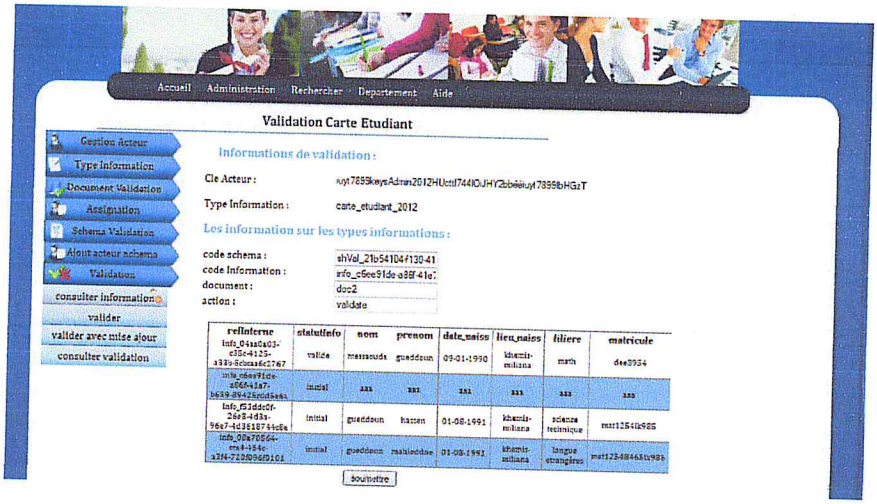

Figure 79: validation sans modification

4. Implémentation dans un ADL :

Nous avons choisit d'implémenter notre composant de validation dans un ADL « Assembler4J » qui a été réaliser dans le cadre du projet Master [AM, 11].

4.1. Définition de l'ADL :

Cet ADL permet d'assembler les composants. Il facilite la mise à jour et il offre une meilleure réutilisation des composants qu'on a conçus. Il supporte l'orienté composant et l'orienté aspect.

4.2.Les étapes d'implémentation :

Pour l'implémentation dans l'ADL « Assembler4J » nous avons choisit d'intégrer deux sous composant de notre composant de validation :

- Le sous composant gestion acteur
- Les sous composants gestion type information

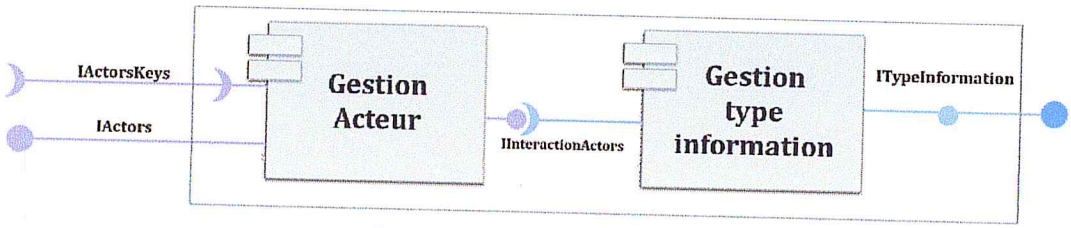

Figure 80: les sous composants implémenté dans l'ADL

5.2.1. Configuration des sous composant :

Pour faciliter la tache, nous avons pris en compte que l'interface fournit de gestion acteur au sous composant gestion type information.

- Les deux sous composants vont implémenter l'interface « Binding » de l'ADL « assembleur4J »
- Puis il faut implémenter les deux méthodes « bind() et lookup() »
- Puisque le composant acteur fournit aux types information, on n'aura pas besoin de l'implémenter

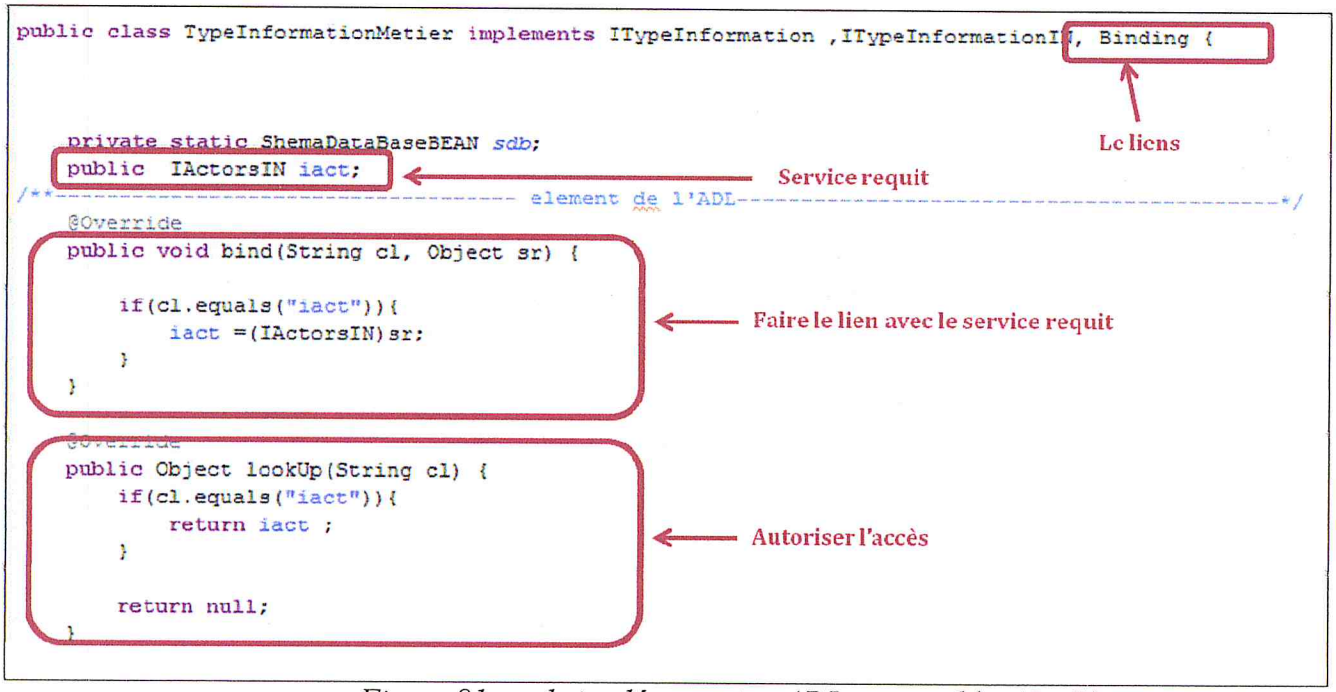

Figure 81: code implémentation ADL « assembler4J »(1)

- Le contrôleur :

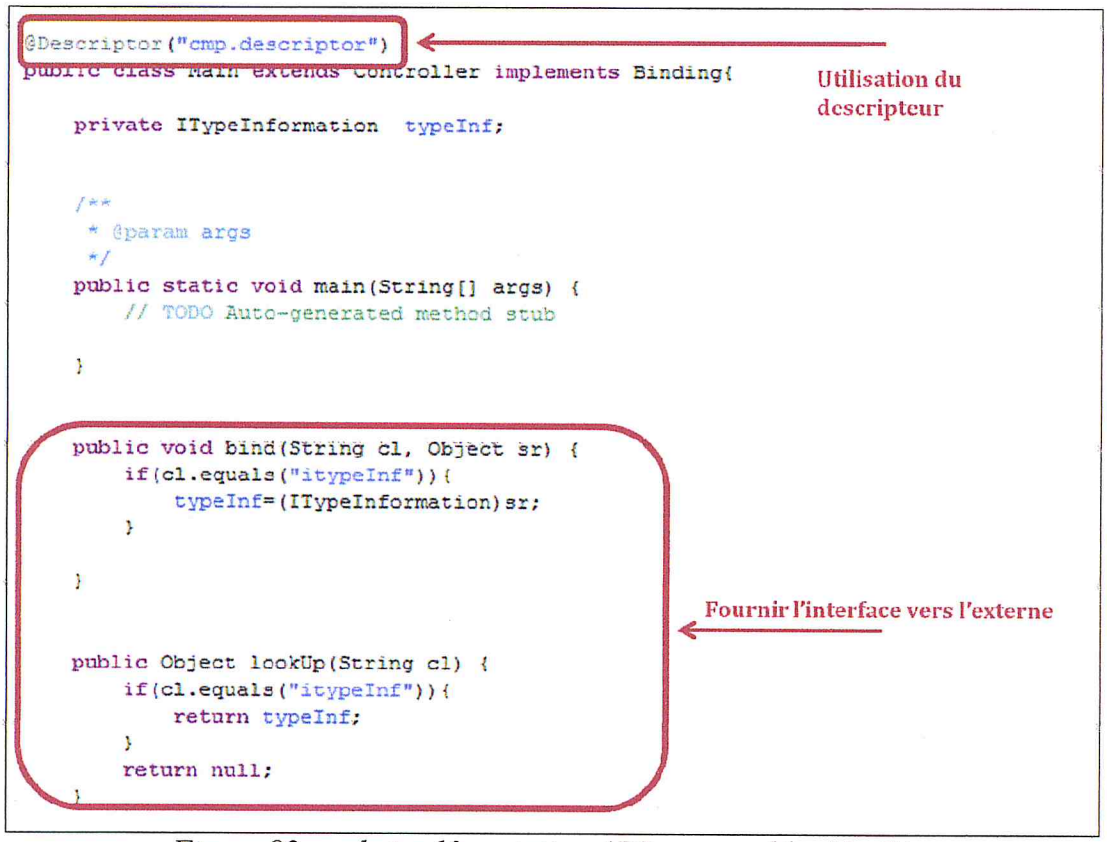

Figure 82: code implémentation ADL « assembler4J »(2)

- Le descripteur

Figure 83: code implémentation ADL « assembler4J »(3)

- Teste

```
 */
public static void main(String[] args) {

    ITypeInformation i=  (ITypeInformation) ((Main) new Main().ReadDescription()).lookUp("itypeInf");

    try {
        System.out.println(i.getTypesInformation("iuyt7899keysAdmin2012HUott1744IOiJHY2bbééiuyt7899IbHGzT"));
    } catch (JDOMException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
```

Figure 84: code implémentation ADL « assembler4J »(4)

5. Conclusion :

Dans cette partie nous avons présenté l'implémentation et le teste de notre composant de validation, ainsi tout les outils que nous avons utilisé pour le mettre en place.

Conclusion et Perspectives

Nous avons présenté dans ce travail, la conception et la réalisation d'un composant de validation pour l'application d'eGouvernement, nous avons utilisé un modèle qui s'inspire du modèle de composant de l'approche IASA et de FRACTAL pour la conception de notre composant de validation. Puis nous l'avons implémenté dans un langage de programmation.

Le composant qu'on a mis en place s'exprime a travers ses services fournies et requis et supporte tout le schéma de validation et les scenarios de validation possible ainsi il peut être intégrer dans d'autre applications grâces a ses interfaces et grâce a la diversité des scenarios des schémas de validation qu'il supporte.

Et pour monter son fonctionnement et son intégration dans l'application, nous avons mis en place dans un premier temps une application de gestion de scolarité où il a été intégrer. Après la réussite de son intégration nous l'avons intégrer dans un ADL.

Nous avons choisit l'ADL « assembler4J » qui a été réaliser dans le cadre du projet Master [AM, 11] sous la direction de Mr BENNOUAR Djamal.

L'intégration du composant a été faite avec succès dans l'ADL, mais a défaut de temps nous avons intégrer seulement deux sous composant, ainsi pour améliorer ce travail il est nécessaire d'intégrer tout les composants composites du composant de validation,

On peut aussi ajouter au composant de validation de nouvelles interfaces, comme la génération d'information validé ou invalide dans des documents PDF, définir d'autre manière d'assignation d'information a un utilisateur, et imaginer de nouvelle interfaces celons un nouvel contexte pour augmenter sa réutilisation.

Pour notre part, ce projet nous a permis d'améliorer nos connaissances en matière d'architecture logicielle et de développement d'applications web avec Java, il nous a permis aussi d'acquérir une certaine expérience dans le domaine des composant logiciel et des ADL.

Bibliographie

[AA, 97] Antonia BERTOLINO, Antonio BUCCHIARONE, Stefania GNESI et Henry MUCCINI. An architecture-centric approach for producing quality systems. In *QoSA/SOQUA*, pages 21–37, *2005*.

[ALL, 97] Robert Allen. A Formal Approach to Software Architecture. PhDthesis, Carnegie Mellon, School of Computer Science, January 1997.

Issued as CMU Technical Report CMU-CS-97-144.

[AM, 11] "Conception orientée aspect et par composant d'une application E-Gouvernement, Baouya abdelhakim,Mahdjoub Mouhamed ,2011 ,Promoteur Bennouar Djamel" USDB, Blida, 2011.

[BCK, 03] Len BASS, Paul CLEMENTS et Rick KAZMAN. Software Architecture in Practice. Addison-Wesley Professional, 2003.

[BEN, 07] D. Bennouar, T. Khammaci, A. Henni : *A New Approach To Component's Port Modeling In Software Architecture,* ACIT'07, Lattekia, Syrie, décembre 2007

[BEN, 09] D.Bennouar : *Une approche intégrer pour une architecture logicielle.* L'école nationale supérieure d'informatique d'Alger, Algerie, 2009.

[CA, 02] Djalel CHEFROUR et Françoise ANDRÉ. Aceel : modèle de composants auto-adaptatifs. In *Systèmes à composants adaptables et extensibles, Grenoble, France*, Grenoble, France, *2002*.

[CC, 03] Cyril Carrez. *Contrats Comportementaux pour Composants.* Thèses de PHD, l'école Nationale Supérieure des Télécommunications, Spécialité : Informatique et Réseaux, Décembre 2003.

[CH, 05] Djalel CHEFROUR : Plate-forme de composants logiciels pour la coordination des adaptations multiples en environnement dynamique, université de Renne, France, 2005

[CS, 98] Clemens SZYPERSKI. *Component Software*. ISBN: 0-201-17888-5. Addison-Wesley, *1998*.

[DG, 00] David GARLAN. Software architecture: a roadmap. In ICSE '00: Proceedings of the Conference on The Future of Software Engineering, pages 91–101, New York, NY, USA, 2000. ACM.

[FECA, 05] Robert E. Filman, Tzilla Elrad, Siobh_an Clarke, and Mehmet Ak_sit, editors. Aspect-Oriented Software Development. Addison-Wesley, Boston, 2005.

[GO, 05] Goaer Olivier. De l'adaptation des composants logiciels vers leur évolution. Mastersthesis, Laboratoire d'Informatique de Nantes Atlantique, 7 septembre 2005.

[GS, 05] Ghorab Wafa, Sandjakedine nassima. Conception et implémentation d'un modèle de sécurité pour la pose d'interaction entre composant spécifique aux SI. Mémoire d'ingenier, université Saad Dahleb, Blida.

[JNJ, 94] Jeff Magee, Naranker Dulay, and Jeff Kramer. A constructive development environment for parallel and distributed programs. In *IWCCS'94 : Proceedings of the IEEE Workshop on Configurable Distributed Systems*, North Falmouth, Massachusetts, USA, Mars 1994.

[JP, 06] Jacques PRINTZ. Architecture logicielle : Concevoir des applications simples, sûres et adaptables. Dunod, Paris, 2006, ISBN 2 10 049910 6

[KKO, 03] N. Kanaoui, T. Khammaci, M. Oussalah, Les ADLs : une voie prometteuse pour les architectures logicielles, Rapport interne IRIN, Université de Nantes, 2003.

[KM, 04] Karine Macedo De Amorim. Modélisation d'aspects qualité de service en UML : *application aux composants logiciels*, thèse de doctorat, université de Rennes 1, Mention Informatique.

[MD, 96] Mary SHAW et David GARLAN. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, USA, *1996*.

[MED, 99] Nenad Medvidovic, Architecture-Based Specification-Time Software Evolution, Phd Thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 1999

[MORT, 96] Nenad Medvidovic, Peyman Oreizy, Jason E. Robbins, and Richard N. Taylor. Using object-oriented typing to support architectural design in the c2 style. In In Proceedings of ACM SIG- SOFT'96 : Fourth Symposium on the Foundations of Software En-gineering. ACM Press, 1996.

[MT, 00] Nenad Medvidovic & Richard N. Taylor. A Classi_cation and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering, vol. 26, no. 1, pages 70_93, 2000.

[OB, 05] Olivier Barais Construire et Maitriser l'évolution d'une Architecture Logicielle à base de Composants, Doctorat de l'université des Sciences et Technologies de Lille, septembre 2005.

[OK, 05] Modèles formels et outils génériques pour la gestion et la recherche de composants, These de Doctorat. Institut National Polytechnique de Grenoble.

[OMG, 02] Object Management Group, CORBA Components, Technical Report 02-06-65, June, 2002.

[OU, 05] M. OUSSALAH. *Ingénierie des composants : concepts, techniques et outils.* Editions Vuibert, *2005*.

[PS, 04] Nicolas Pessemier and Lionel Seinturier. Components, adl and aop : *Towards a common approach.* In In Workshop ECOOP Reection, AOP and Meta-Data for Software Evolution, 2004.

[PW,92] Dewayne E. PERRY et Alexander L. WOLF. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4) :40–52, *1992*.

[RK, 00] Rainer KOSCHKE. *Atomic Architectural Component Recovery for Program Understanding and Evolution.* Thèse de Doctorat, University of Stuttgart, *2000*.

[RRN, 04] R. Roshandel and N. Medvidovic, Multi-View Software Component Modeling for Dependability, in *Architecting Dependable Systems II, Lecture Notes in Computer Science*, vol. 3069, pp. 286-304, 2004 (ISSN 0302-9743)

[SAE, 04] SAE. Architecture Analysis & Design Language (AS5506), September 2004. available at http://www.sae.org.

[SG, 96] Shaw M. and Garlan D. Software Architecture - Perspectives on an Emerging Discipline, Prentice Hall, ISBN: 0-13-182957-2, 1996.

[SH, 09] Sylvain CHARDIGNY. Extraction d'une architecture logicielle à base de composants depuis un système orienté objet. Une approche par exploration, 2009.

[SL, 10] Sihem LOUKIL : *Extension d'un langage de description d'architecture pour la programmation orientée aspect.* Thèse de master. Université de Sfax, Tunisie, 2010.

[SZY, 02] SZYPERSKI C., Component software beyond object-oriented programming, Addison Wesley, États-Unis, 2002.