Université Saâd DAHLAB de Blida

Faculté des Sciences

Département : Informatique

Mémoire en vue d'obtention du diplôme de Master

Spécialité : ingénierie du logiciel

Sujet : Conception Orientée Aspect et par composants d'une application d'E-Gouvernement

Présentée par :

Abdelhakim BAOUYA et Mohamed MAHDJOUB

Sous la direction du promoteur

Mr. Djamel BENNOUAR

Président du jury : Mme Boumahdi

Examinateur : Mme Azzouz

2010/2011

Résumé

L'architecture logicielle à base de composants et le développement d'application orienté aspects sont deux nouvelles disciplines du génie logiciel qui visent à boucler certaines limitations de l'orienté objet. L'objectif de cette étude est la détermination des composants d'une application d'E-gouvernement. Ces composants sont en général de deux catégories : Les composants métiers qui prennent en charge les fonctionnalités pures de l'application et les composants qui se chargent de la réalisation des fonctionnalités techniques telles que la sécurité, les états et le traçage.

En plus des composants il est nécessaire de déterminer les ports d'interactions des composants et la nature des connecteurs, ainsi que spécifier une technique d'assemblage (interconnexion) des composants. L'assemblage des composants devra se faire par la mise en œuvre des mécanismes de la conception orientée aspect.

La solution proposée utilise le modèle de composants IASA, elle utilise des fichiers XML pour décrire les composants et leur interconnexion, un outil développé utilise ces fichiers descripteurs pour charger et configurer les composants dynamiquement. La détermination des composants a suivi un processus par raffinement successif. La connexion des composants se base essentiellement sur les ports (interfaces) des composants. Pour le support des aspects, on a intégré le framework openSource SpringAOP.

Mots Clés :

Architecture Logicielle, Aspect, Composant, Connecteur, Port, IASA, SpringAOP, Langage de description d'architecture (ADL).

Keywords :

Software Architecture, Aspect, Component, Connector, Port, IASA, SpringAOP, Architecture Description Language (ADL).

Liste des Figures

Chapitre 1 : Introduction Générale

1.1. Généralités :

Le champ du génie logiciel est en évolution sans cesse croissante pour permettre la prise en charge efficace des nouveaux besoins complexes et très dynamiques dus au recours systématique aux solutions logicielles dans les divers secteurs de la vie humaine. La prise en charge efficace de ces besoins impose le passage à une situation où la production d'un logiciel respecte certaines contraintes comme le faible coût, la rapidité de la livraison et la maintenance.

Actuellement, la Programmation Orientée Objet est le paradigme de programmation le plus utilisé. Nul ne peut nier l'importance et l'apport de la POO dans le développement de systèmes complexes. Cependant, la Programmation Orientée Objet souffre d'une défaillance importante au niveau de la réutilisation. De plus, la dynamicité des besoins implique souvent des changements dans les fonctionnalités du système déjà existantes (i.e. nouvelles fonctionnalités). Ces changements sont généralement difficiles à maîtriser, et parfois nécessitent la refonte complète de la fonctionnalité concernée.

Dans l'approche orientée objet la réutilisation est employée typiquement à travers la notion d'héritage. L'héritage introduit malheureusement un fort couplage entre une classe et ses dérivées, ce qui cause le problème connu sous le nom « *the fragile base class problem* » [MS98], en d'autres termes une modification au niveau de la classe mère entrainera probablement un fonctionnement inattendu dans ses classes dérivées, d'où sa fragilité. Pour la prise en charge de ces préoccupation et accroitre les facteurs de qualité d'un logiciel (réutilisation, robustesse face aux changements, facilité de maintenance et d'évolution etc...), l'introduction d'autres paradigmes et approches est ainsi devenue nécessaire.

Actuellement, selon un consensus établi en génie logiciel, deux grandes approches sont cités comme les plus prometteuses pour une meilleure prise en charge des facteurs de qualité du logiciel : L'Architecture Logicielle et la Conception Orientée Aspect. Le mixage de ces deux approches est appelé l'Architecture Logicielle Orientée Aspect.

L'architecture Logicielle ou la conception à base de composant logiciel est une nouvelle discipline dans le génie logiciel. Son objectif principal est de permettre la conception par assemblage de composant. La spécification d'Architecture Logicielle couvre aujourd'hui les phases abstraites et concrète d'un logiciel. Le développement de logiciel se fait souvent par raffinement successif, à partir d'une spécification très abstraite, pouvant être représentée par une spécification dans un langage formel, en aboutissant à une étape concrète représentée par une spécification à base de composants dits primitifs. Ces derniers sont réalisés soit à l'aide d'un langage de programmation tel Java ou à l'aide d'un langage de description d'architecture logiciel tel qu'ArchJava ou Fractal.

1.2. Problématique :

La problématique que nous adressons a été fortement ressentie au niveau du projet E-gouvernement lancé au CDTA depuis plus de 4 années. L'objectif initial de ce projet fut la réalisation d'un environnement de développement d'application rapide pour le e-gouvernement. L'environnement de développement devait reposer sur le modèle objet. Cependant le phénomène de changement (démission, utilisation des ressources humaines du projet pour réaliser d'autres tâches urgentes au CDTA, interférence de l'administration dans la conduite du projet, évolution des technologies de présentation et de persistance etc..), qui est un des plus grands problèmes auquel font face les projets de logiciel, n'as pas été pris en charge de manière efficace dans les débuts du projet. L'impact de ce phénomène qui est devenu important, a mené l'équipe du projet à revoir les objectifs du projet. Le nouvel objectif consistait à concevoir et réaliser selon l'orienté objet et l'approche MVC (Model View Controller) un système

dédié à mettre sur Internet les divers services d'une APC requis par le citoyen, notamment le service d'état civil et le service de vote. Le premier service pris en charge fut l'état civil. Même avec cette réduction des objectifs, le projet a été lourdement affecté par le phénomène changement, notamment les démissions du personnel de conception et de réalisation. Malgré cela le projet a avancé et une deuxième version du projet e-APC a été mise sur pied [**SW09**].

La version e-APC 2.0 étant réalisé selon une approche objet et selon le modèle MVC fait face à des défis énormes lorsqu'une opération d'ajout d'une nouvelle fonctionnalité ou l'ajustement d'une autre fonctionnalité est demandé. Premièrement ces opérations ne sont réalisable que par ceux qui ont conçu et réalisé la partie à améliorer ou changer. Ainsi le départ de ces personnes entrainerait une perturbation énorme, qui nécessiterait parfois la refonte complète de la partie. Très rares sont les ingénieurs qui sont satisfait d'un travail réalisé par un autre lorsque il leur est demandé de reprendre et achever un travail déjà réalisé.

Deuxièmement, dans le contexte du paradigme objet, où les classes sont parfois très fortement couplées, le changement au niveau d'une classe d'objet entrainerait le changement au niveau de plusieurs autres classes. Cette opération est souvent très coûteuse en temps.

Cette expérience a mené l'équipe du projet à revoir les principes et techniques sur lesquels le projet e-Gouvernement devra reposer. Ils sont arrivés à la conclusion que le projet devra se baser sur les éléments suivants :

- Adoption d'une approche basée sur les composants : Un composant est une unité indépendante, très faiblement couplée (pour ne pas dire non couplée)
- Les composants qui seront définis dans le projet doivent être très fins pour que le projet soit robuste face au phénomène de changement. Un composant fin pourrait être réalisé, testé et documenté dans des délais très courts, très inférieurs au temps d'existence d'une personne dans un projet. L'approche composant permettrait d'ajuster des fonctionnalités d'un composant ou de lui ajouter des fonctionnalités sans perturbation des autres éléments d'un système bâti par assemblage de composants.

1.3. Objectifs

L'objectif de cette étude est la refonte de la conception d'eAPC selon une approche à composant orientée aspect. Globalement nous devons atteindre trois grands objectifs :

- La détermination ou le choix d'un modèle de composants.
- La détermination ou le choix d'un modèle de connecteurs.
- La conception par assemblage de composant d'eAPC en se basant sur les modèles de composant et de connecteurs déterminés.

Dans le monde de la conception par assemblage de composant, deux tendances de conception sont possibles : Une conception du haut vers le bas (top down) ou du bas vers le haut. En général ces tendances sont souvent mixées. Selon ces deux tendances il s'agit de déterminer les composants et les assembler pour obtenir l'application.

Les composants qui seront déterminés sont en général de deux catégories. Les composants métiers qui prennent en charge les fonctionnalités pures d'eAPC et les composants qui se chargent de la réalisation des fonctionnalités techniques telles que la sécurité, les états, le traçage, et la persistance. L'application, qui est elle-même un gros composant composite, contiendra une partie qui se charge des fonctionnalités pures de l'application et une partie qui se charge des aspects techniques que l'on appellera la partie aspect. L'assemblage de ces deux grandes parties devra se faire par la mise en œuvre des mécanismes de la conception orientée aspect. La conception orientée composant et aspect devra aussi respecter rigoureusement le modèle MVC dans lequel les trois niveaux d'une application (La vue, le contrôle et le modèle) sont explicitement séparée.

Les composants qui seront déterminés doivent impérativement appartenir à un modèle de composant de référence qu'il faudrait soit choisir parmi les solutions existantes ou définir. Le modèle de composant joue le rôle de type pour toutes les instances de composant.

La détermination d'un modèle de composant consiste souvent à définir trois éléments fondamentaux :

- La vue externe d'un composant : C'est l'ensemble des ports d'interactions
- La technique d'assemblage de composant. Souvent nous parlons de connecteurs.
- La vue interne des composants. Celle-ci, si elle est présente, imposera à l'architecte une démarche pour la conception d'un composant.

1.4. Organisation du mémoire :

Le travail que nous reportons dans ce mémoire est organisé comme suit :

Dans le chapitre 2, nous commencerons par donner un aperçu de l'approche Architecture logicielle et de la conception orientée aspect. Nous présenterons quelques modèles de composant et nous terminerons ce chapitre par la présentation du modèle de composants et de connecteurs que nous utiliserons dans la conception d'eAPC selon une approche à composant orientée aspect.

Dans le chapitre 3 nous présenterons le fonctionnement à l'intérieur d'un service de l'APC, qui sera le domaine de l'application, à savoir le service d'état civil. Le chapitre 4 présentera la conception selon l'approche d'Architecture logicielle IASA. Le processus de conception est un processus qui opère par raffinement successif de composant. Selon les besoin nous opérons selon une approche du global vers le détail et parfois du détail vers le global. Le chapitre 5 se concentrera sur la transformation de la conception selon une approche architecture logicielle vers une vue d'implémentation. Nous présenterons dans ce contexte la technique que nous avions introduite pour transformer un composant IASA en un composant de niveau implémentation, directement déployable dans son environnement d'exécution. Nous terminerons ce mémoire par une présentation de la réalisation, des tests et une conclusion générale.

Chapitre 2 : Introduction aux Architectures Logicielles à base de composants

2.1. Introduction aux architectures logicielles à base de composants

En réalité, il n'existe pas de définition universelle de l'architecture logicielle. En effet, on peut trouver dans la littérature de nombreuses définitions qui, bien qu'elles présentent des similitudes, correspondent à autant de vues différentes du domaine. Parmi ces dernières, nous citons :

- ***L'architecture logicielle comprend la structure des composants d'un logiciel/système, les relations entre eux et les principes et directives régissant leur conception (design) et évolution au cours du temps.***

- ***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.*** [SG96]

L'architecture logicielle concerne plus particulièrement la conception de systèmes à forte composante logicielle et les familles de produits logiciels. Elle décrit l'ensemble de ses composants, donne la définition de leur protocole de communication, permet l'assemblage de ces composants et prend en compte les structures d'accueil nécessaires à leur déploiement et l'exploitation du système résultant.

2.1.1. Les concepts de base de l'architecture logicielle

Pour résoudre le problème de réutilisation, CBSD introduit une nouvelle solution qui à pour objectif un couplage minimal entre les artefacts logiciels et une cohésion maximale à l'intérieur du même artefact [Mey99]. Un système est développé en assemblant un ensemble de ***composants préfabriqués et testés***. Chaque composant est une entité de type « *boite noire* » qui peut être déployé indépendamment et qui peut fournir un ensemble de services bien spécifié [Szy98].

Le composant et le connecteur représentent les éléments de base dans la spécification d'architecture logicielle.

2.1.1.1. Le concept de composant

D'après Shaw et Garlan [SG96], Le terme de composant est probablement le plus répété dans les recherches en informatique. Ils définissent le composant comme une « *Unité de calcul* » (Computational Unit), ce qui implique que tout artefact logiciel est considéré comme un composant logiciel.

Brown [BW96] définie le composant comme « *une unité indépendante offrant un ensemble de services réutilisables* » ça signifie que le composant a pour rôle d'offrir un ensemble de services réutilisables, et que le composant doit être indépendant d'un contexte spécifique.

Une autre définition formulée par Hans-Gerhard Gross dans Shaw et Garlan [SG96]:

«Un composant logiciel est une pièce indépendante qui fournit un ensemble de fonctions permettant l'accès aux services grâce à des interfaces »

Cette définition offre une vue plus claire sur le composant car il ne définie pas que les services qu'il peut offrir mais aussi les services requis afin de fonctionner correctement.

Un composant peut être primitif ou composite. Un composite est un réseau de composants interconnectés selon une topologie permettant d'atteindre les objectifs fonctionnels du composite.

2.1.1.2. Le concept de connecteur

Les connecteurs sont chargés d'assurer une communication correcte entre les composants. Un connecteur doit être indépendant de toute logique de composant. Il contient des *interfaces* qui indiquent les services fournis ou requis par les composants que le connecteur est chargé d'assembler. Une interface est associée à un ou plusieurs composants participants dans la connexion et a un *rôle* qui définit la nature de ces composants (serveur, fournit des services via cette interface ou bien client, qui utilise ces services).

Un connecteur peut être un connecteur d'assemblage ou de délégation. *Un connecteur d'assemblage* permet de connecter un composant qui fournit des services à un composant qui les utilise, on peut en déduire une relation *d'utilisation*. Tandis que *le connecteur de délégation* permet de déléguer la réalisation ou le requiert d'un service à un de ses sous-composants, on en déduit une relation de *composition* entre les participants.

2.1.2. Les modèles à base de composants :

2.1.2.1. Le modèle à composants *Fractal*

Fractal [OB05], réalisé par France Telecom R&D et par l'INRIA, vise à autoriser une définition, une configuration et une reconfiguration dynamique d'une architecture à base de composants. Un composant Fractal est formé d'une membrane et d'un contenu.

La membrane définit, par l'intermédiaire d'un ensemble d'interfaces, les services requis et fournis par le composant. Le contenu d'un composant Fractal permet de différencier deux sortes de composant : les primitifs et les composites.

Le contenu d'un composant primitif identifie un module logiciel (classe, ensemble de fonctions, *etc.*) permettant de réaliser les services du composant primitif. Le contenu d'un composite définit un assemblage de composants permettant de mettre en œuvre les services du composite.

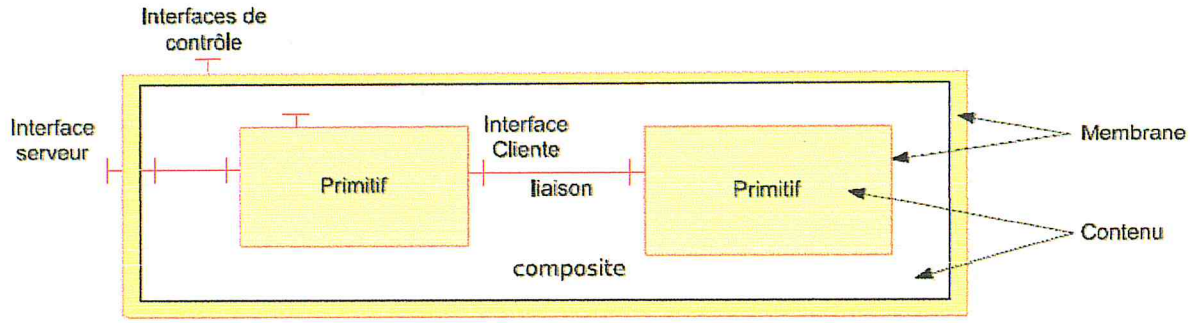


Figure 2.1 : Modèle de composant Fractal

Fractal met à disposition un ensemble d'interfaces de contrôle qui sont la gestion du cycle de vie (*life-cyclecontroler*), du nom (*name-controller*), des liaisons (*binding-controler*), des attributs (*attribute-controler*), des parents (*super-controller*) et du contenu (*content-controler*) pour les composites. Le modèle étant extensible, il est possible de définir ses propres interfaces de contrôle.

Une interface Fractal métier est du type client ou serveur. Une interface serveur identifie les services offerts par un composant alors qu'une interface client spécifie les fonctionnalités qu'un composant requiert pour son fonctionnement.

Fractal ne possède pas de notion de connecteur explicite avec une sémantique de processus. Cependant, il utilise la notion de *liaison* pour spécifier les interactions entre composants. Une liaison Fractal est définie comme un lien orienté entre une interface client et une interface serveur. Ce lien permet aux composants d'interagir.

2.1.2.2. Le modèle à composants IASA

Un composant IASA (Integrated Approach to Software Architecture) est vu de l'extérieur comme une boite noire qui communique avec le monde externe grâce à des ***Ports***, qui définissent les services qu'il peut offrir ou requérir. La vue interne d'un composant primitif est inaccessible, tandis que celle d'un composant composite est bien définie, elle consiste en trois parties : *Partie Opérationnelle,* Partie *Aspect et Partie Contrôle.*

- **Partie Contrôle** : a pour rôle de charger et configurer les composants internes et satisfaire la demande des composants externes, en délégant ces demandes à un composant interne.

- **Partie Aspect** : C'est les fonctionnalités techniques, qui ne font par partie de la logique du métier de l'application.
- **Partie Opérationnelle** : Regroupe les composants métier (Opérations principales pour le fonctionnement de l'application)

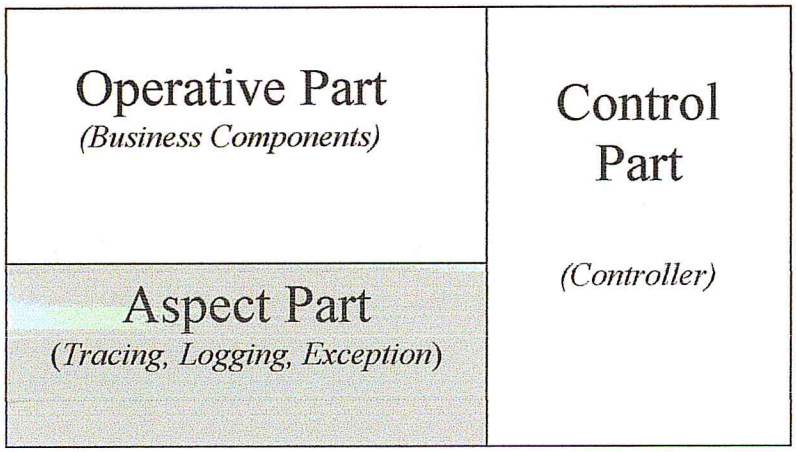


Figure 2.2 : les différentes parties de la vue interne d'un composant selon IASA

L'instanciation d'un composant se fait dans le contexte du concept d'***enveloppe*** qui sert principalement à isoler la vue interne d'un composant de son environnement d'exploitation, l'enveloppe détient toutes les ressources dont il a besoin pour le support de la communication et de la spécification de la connexion inter-composants. Un ***point d'accès*** est le plus petit élément dans la spécification d'une application, il définit les ressources requises ou fournies qui peuvent être des données ou des opérations. On peut regrouper un ensemble de points d'accès dans le concept de ***Port***. [**BHS09**]

Dans la suite on utilisera le modèle de composants IASA pour représenter nos composants car ce modèle permet de prendre en considération la partie aspect d'une application dans le niveau conceptuel. Les deux modèles qu'on vient de voir nous permettent de représenter l'architecture de manière abstraite (conceptuelle), pour pouvoir implémenter ces concepts il nous faut un modèle concret pour décrire l'architecture et un ADL.

2.1.3. Les langages de description d'architecture (ADL)

Un ADL (Architecture Description Language) est une manière formelle pour la spécification de l'architecture d'un système, en fournissant les moyens pour décrire les entités (composants) logicielles qui le constituent et les relations entre ces entités, il présente l'avantage d'être compris par l'humain et la machine et facilite la communication, la réalisation et la validation de l'architecture.

2.1.3.1. l'ADL Darwin

Darwin a été développé au Distributed Software Engineering Group de l'Imperial Collège de Londres. Il propose un modèle de composant pour la construction d'applications distribuées.

Comme pour la plupart des ADLs, un composant est une instance définie par une interface qui décrit les services qu'il fournit et qu'il requiert, cette dernière représente le *port* de communication, et *une signature* qui décrit la signature de la fonction du composant pour lui permettre de fonctionner et de s'intégrer dans une architecture complexe.

2.1.3.2. ArchJava

ArchJava se situe à la frontière entre le langage de description d'architectures et les modèles de construction d'applications à composants. Il a été créé à l'université de Washington par Jonathan Aldrich et Craig Chambers. ArchJava a pour objectif d'améliorer la compréhension des programmes, de garantir l'architecture de l'application, de permettre une meilleure évolutivité des applications.

ArchJava étend le langage Java afin d'incorporer les éléments d'architectures à l'intérieur du code. Un travail important a été réalisé sur le compilateur ArchJava afin de respecter l'intégrité des communications entre les composants. La communication entre les composants se fait à l'aide d'un appel de méthode.

2.1.3.3. l'ADL FRACTAL

En amont du modèle de composant Fractal, il est possible de décrire l'architecture d'une application à base de composants à l'aide de l'ADL Fractal. Basé sur XML, ce langage définit une syntaxe abstraite indépendante de tout langage de programmation pour la description de l'architecture en termes de composants, d'interfaces, de liaisons et d'attributs.

L'ADL Fractal, dans la lignée d'ADL est un ADL extensible. Il permet d'intégrer facilement de nouveaux concepts au niveau de la description de l'architecture en modifiant juste le descripteur XML ce dernier est chargée par *l'Usine Fractal* pour faire la liaison des différents composants et sous composants

L'Usine Fractal est un composant présent dans chaque implémentation dont le rôle est de lire le descripteur et faire la liaison des différents composants.

En résumé, on peut dire que l'approche d'ArchJava et Darwin présente l'inconvénient de devoir recompiler à chaque fois qu'une modification est apportée à l'architecture ou à la configuration des composants. Par contre, avec Fractal le fichier de spécification n'a pas besoin d'être recompilé. En cas de modification de l'architecture ou des composants, un simple éditeur de texte fera l'affaire, puis les modifications apportées seront prises en considération par l'usine Fractal.

2.2. Introduction à la Conception Orientée Aspect

Le développement Orienté Aspect (Aspect-Oriented Software Development) est un plus récent paradigme de développement qui vise à améliorer la séparation des préoccupations. C'est une propriété cruciale pour développer des systèmes plus compréhensibles et plus facile à la gestion et la maintenance. Les technologies orientées objet échouent dans la modularisation de certaines préoccupations car le paradigme orienté objet ne permet qu'une simple dimension de décomposition, en l'occurrence *l'hiérarchie de classes* [TOHS99]. Comme conséquence, certaines préoccupations dites « *transverses* » sont dispersées et dupliquées à travers plusieurs modules dans le système, cette duplication (*enchevêtrement*) de code rend l'ajout, la modification ou la suppression d'une telle préoccupation très difficile à effectuer. Des exemples typiques des préoccupations transverses sont les préoccupations de débogage telle que la journalisation, les préoccupations de sécurité, vérification d'intégrité… etc.

AOSD résout ce problème en introduisant une ou plusieurs dimensions de décomposition qui permettent la modularisation des préoccupations transverses. Elle

permet d'intégrer ces solutions en introduisant les nouveaux concepts d'aspect, point de jonction, greffon (advice code), et de coupe.

Une application Orientée Aspect consiste en deux parties [Pawlak2005] :

- Métier : elle est constituée des classes qui font la base de l'application et implémentent la logique du métier de l'application.
- Aspects : elle intègre les éléments supplémentaires (classes, méthodes et données) qui correspondent aux besoins non-métiers.

Le développement d'une application orientée aspect, tout comme les autres approches, passe par des étapes, Ramnivas Laddad cite les étapes suivantes [WEB01]:

- ***Décomposition aspectuelle*** : les besoins sont ici décomposés pour identifier les préoccupations fonctionnelles et transversales.
- ***Implémentation des préoccupations*** : Chaque préoccupation est implémentée indépendamment des autres.
- ***Recomposition Aspectuelle*** : Des règles de recomposition sont spécifiés en créant des unités appelées aspects. Le processus de recomposition, aussi connu sous le nom de tissage ou d'intégration, utilise ces informations pour composer le système final.

2.2.1 Les concepts de base de la Conception Orientée Aspects :

2.2.1-A – Aspect :

Un aspect est une entité logicielle qui capture une fonctionnalité transversale à une application.

2.2.1-B – Point de Jonction (join Point):

Un point de jonction est un point dans le flot de contrôle d'un programme dans lequel un ou plusieurs aspects peuvent être appliqués. Parmi les types de points de jonctions communément rencontrés on cite : les méthodes, les constructeurs, les opérations de lecture/écriture sur attributs et les exceptions.

2.2.1-C – Coupe (Pointcut):

Une coupe sélectionne un ensemble de points de jonction. La définition d'une coupe utilise une syntaxe qui peut inclure des expressions génériques et des opérateurs logiques, elle est propre à chaque implémentation. La coupe exprime « ***où*** » l'aspect doit être inséré.

2.2.1-D – Code Greffon (Advice Code):

Un code greffon est un bloc de code définissant le ***comportement*** d'un aspect. Il est tjrs associé à une coupe (plus exactement aux points de jonctions sélectionnés par cette coupe). Un code advice n'est jamais appelé depuis le programme, il est invoqué à chaque fois qu'un point de jonction sélectionné par la coupe associée survient. Il peut être exécuté selon plusieurs modes, les plus connus sont : Avant, après ou autour du point de jonction.

2.2.1-E – Mécanisme d'introduction :

Le mécanisme d'introduction est un mécanisme d'extension permettant d'introduire de ***nouveaux*** éléments structuraux au code d'une application.

Tout comme l'héritage dans la POO, il permet d'étendre la partie statique d'une classe en introduisant de nouvelles méthodes, de nouveaux attributs – les gestionnaires d'exceptions peuvent être introduits dans certaines implémentations (**J**ava **A**spect **C**omponents) —cependant, contrairement à l'héritage, il ne permet pas de redéfinir les éléments existants, cette limitation à pour cause de préserver l'intégrité de l'application spécialement lorsque plusieurs aspect sont appliqués sur une même coupe.

2.2.1-F – Tissage (Weaving):

Le tissage (weaving) est le processus qui prend en entrée un ensemble d'aspects et une application de base et fournit en sortie une application dont le comportement et la structure sont étendus par les aspects.

Une application orientée aspect contient des classes et des aspects. L'opération qui prends en entrée les classes et les aspects et produit une application qui intègre les fonctionnalités des classes et des aspects est connu sous le nom de tissage d'aspect (aspect weaving). Le programme qui réalise cette opération est appelé tisseur d'aspects (aspect weaver) ou bien tisseur (weaver) tout court.

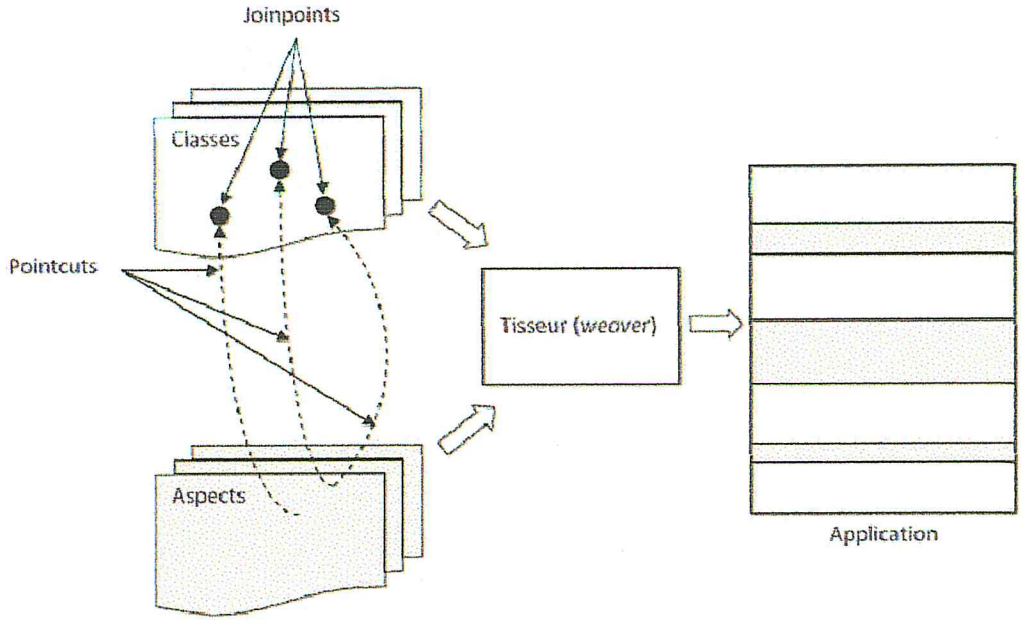


Figure 2.3: Tissage des aspects dans une application

2.2.2 Exemples d'implémentations de la POA

De nombreux efforts ont été menés a fin de construire des outils permettant d'adopter les principes de la POA. AspectJ étant le plus connu, Il fut conçu au Xerox PARC par l'équipe de Gregor Kiczales, Il s'agit d'une extension du langage Java, qui prend en charge les concepts cités précédemment.

Actuellement il existe une panoplie d'outils de développement d'applications orientées Aspect, on peut citer à titre d'exemple Java Aspect Components, JBOSS et Spring AOP.

Toutes ces implémentations permettent d'introduire les concepts de base de la POA, avec une différence mineure dans la terminologie utilisée, dans la conception, et dans l'approche utilisée pour tisser les aspects, l'approche ***Langage*** consiste à étendre un langage orienté objet pour prendre en charge la POA, comme c'est le cas pour AspectJ qui nécessite un compilateur spécial, cette approche agit lors de la compilation pour tisser les aspects. L'approche ***Framework*** n'a pas besoin d'un compilateur vu qu'elle est utilisée avec un langage orientée objet comme Java par exemple, elle fournit un framework avec une API qui permet de représenter les aspects et de les tisser au moment du chargement ou lors de l'exécution, en se référant à des fichiers de configurations souvent écrits en XML, cette technique

permet une meilleure flexibilité , on peut modifier la configuration des aspects même lorsque l'application est en exécution.

2.3. Modèle de composants adopté :

Dans la solution que nous proposons, nous utiliserons le modèle IASA pour décrire nos composants, chaque composant comportera une partie qui s'occupe de ses fonctionnalités métier, une partie qui s'occupe de ses fonctionnalités non métier (aspects) et une partie de contrôle qui se chargera de l'instanciation des composants et d'établir les connexions dont un composant a besoin pour fonctionner correctement.

Le modèle de connecteurs repose sur le principe d'interface. En effet, un composant communique avec son environnement d'exploitation grâce à ses interfaces (fournies et requises), il n'est pas sensé tenir compte des détails d'implémentation de l'extérieur, s'il a besoin des services d'un autre composant, ce dernier se verra implémenter (fournir) une interface I_1 qui lui fournira ces services. Il pourra utiliser ces services en déclarant une interface requise I_2 ayant la même signature qu' I_1. Un outil qu'on a développé (l'Assembleur) se charge de relier ces interfaces en tenant compte d'une configuration détaillée dans un fichier descripteur écrit en XML. Chaque composant composite détient son propre fichier descripteur, ce fichier décrit la vue interne du composite c'est-à-dire :

- les interfaces fournies et requises.
- les sous-composants et leurs interfaces.
- les aspects.
- les connecteurs entre les interfaces des différents composants (que ça soit par délégation ou assemblage)

Le fichier de description est inspiré de l'ADL Fractal auquel on a introduit le concept d'aspect et de connecteur de délégation.

Lors de l'instanciation d'un composant, sa partie contrôle fait appel à l'assembleur pour charger et configurer les composants dont il a besoin en se référant à son fichier descripteur, c'est dans cette étape que les connexions sont

établies et les dépendances injectées. Nous parlerons de l'assembleur en détail dans le chapitre 5.

A fin d'introduire le support d'aspects dans notre solution, nous avons choisi d'intégrer la partie AOP du framework Spring. Nous avons choisi L'API Spring AOP, pour des raisons multiples parmi lesquels on cite le tissage d'aspects au moment de l'exécution, le fait qu'elle soit basée sur le principe d'inversion de contrôle (injection de dépendance) et sa configurabilité.

Chapitre 3 : Etude de l'existant

Avant de passer à la conception de notre application, il vaudrait mieux expliquer le fonctionnement au sein du système dans la réalité. Cela nous donnera une meilleure compréhension du domaine et facilitera la conception.

La commune est la collectivité territoriale de base dotée de la personnalité morale et de l'autonomie financière. Elle est créée par la loi. Elle est administrée par une assemblée élue, l'assemblée populaire communale (APC) et un exécutif présidé par un Maire élu par le peuple qui dirige aussi la commune.

3.1. domaine d'étude

Notre domaine d'étude se limite au service d'état civil, l'APC est représentée par l'hiérarchie suivante :

- Le président de l'APC (PAPC).
- Le secrétaire général (SG).
- Les membres de l'APC.

L'hiérarchie du service d'état civil est comme suit :

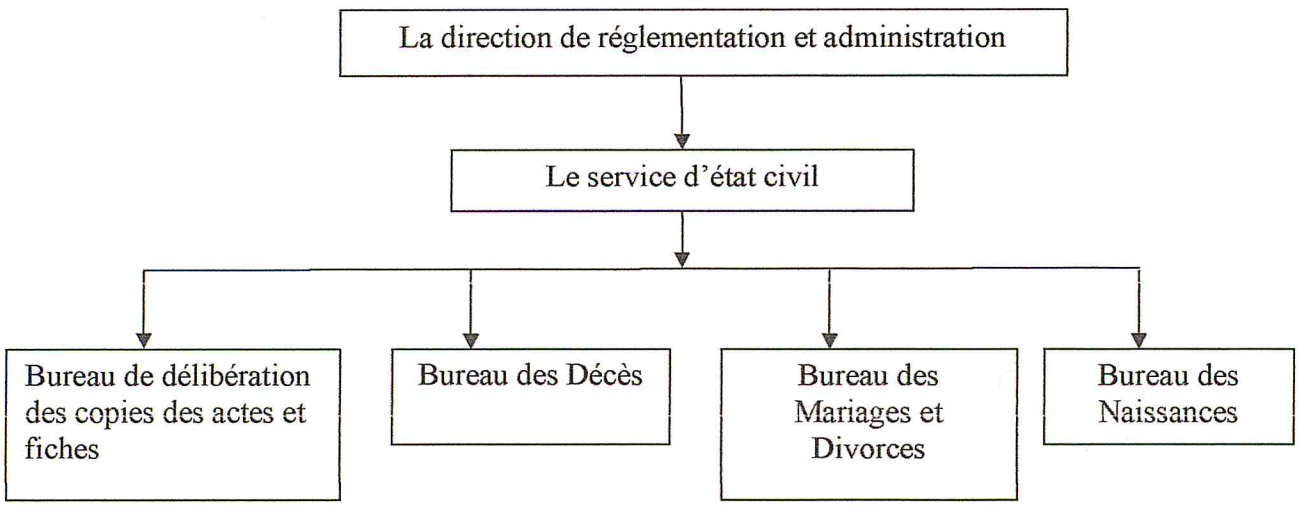


Figure II.1: L'organisation du service d'état civil

3.2. Gestion de service d'état civil

Le travail dans le service d'état civil est basé sur deux choses :

- La responsabilité et le rôle d'un officier d'état civil
- Les registres

3.2.1. Officier d'état civil

Les Officiers de l'état civil sont : le président, les vices présidents de l'assemblée populaire communale, les chefs de missions diplomatique pourvus d'une circonscription consulaire et les chefs de postes consulaires.

Le président peut déléguer à un ou plusieurs agents communaux occupants des postes permanents, âgés au moins de 21 ans les fonctions qu'il exerce en tant qu'officier d'état civil :

- Pour la réception des déclarations des naissances, décès et le reste des événements et pour la transcription.
- La constatation et l'établissement des actes.
- D'apposer les mentions qui doivent, d'après la loi, être faites dans certains cas, en marge des actes de l'état civil déjà inscrits ou transcrit et transcrire le dispositif de certains jugements
- De veiller à la tenue et la conservation des registres courants et de ceux des années antérieures déposés aux archives de la commune.
- De délivrer, à ceux qui ont le droit de les requérir, des copies ou extraits des actes figurant sur les registres.
- D'établir les statistiques mensuelles, trimestrielles, annuelles et décennales des naissances, mariage et décès.
- De lister les futurs candidats de service national chaque année.

Les officiers d'état civil exercent leur travail sous un double contrôle la surveillance judiciaire qui est assuré par le procureur général et la surveillance administrative qui est assuré par le wali donc la responsabilité de chaque officier d'état civil est une responsabilité personnelle.

3.2.2. Les registres

Tous les événements qui concernent les citoyens (naissance, mariage, décès) sont enregistrés en 2 copies. Donc nous avons 3 types de registres :

- Un registre pour les naissances
- Un registre pour les mariages
- Un registre pour les décès

Un registre concerne une année bien précise. Avant son exploitation effective, toutes les pages du registre sont numérotées séquentiellement et cachetés par le président de tribunal.

Au premier janvier de chaque année, ces registres sont pris pour enregistrer les événements séquentiellement selon leur type. Le 31 décembre, les registres doivent être fermés par un officier d'état civil. Une copie reste au service d'état civil, l'autre doit être transférée au tribunal.

3.3. Le fonctionnement dans le service d'état civil

3.3.1 La constatation des événements

C'est la déclaration d'un événement par un citoyen, et son inscription dans les registres par l'OEC après avoir vérifié sa validité.

En général le déclarant doit présenter les documents et papiers nécessaires pour s'identifier au pré de l'OEC et même des témoins s'il le faut.

3.3.1.1 Pour actes de naissance

Si la déclaration de naissance a eu lieu en une période ne dépassant pas 5 jours après la naissance l'acte sera directement transcrit en la présence du déclarant bien sur sinon (après avoir dépassé la période), mentionner le jugement dans le registre.

3.3.1.2 Pour acte de mariage

Si l'acte a eu lieu à l'APC avant le mariage il sera directement transcrit et nécessitera la présence des témoins. Sinon c à d après un certain temps de mariage, la transcription de l'acte portera la mention du jugement.

3.3.1.3 Pour acte de décès

Si la déclaration de décès a eu lieu dans 24 Heures après l'heure de décès, l'acte sera directement transcrit et nécessitera la présence du déclarant bien sur accompagné d'un certificat médical de décès. Sinon c.à.d. après dépassement de cette période, la transcription de l'acte portera la mention du jugement.

3.3.1.4 Pour acte de divorce

L'acte n'est pas établi sans l'acte de jugement de divorce.

3.3.1.5 Pour certificat de preuve d'individualité

Le certificat n'est pas établi sans l'acte de jugement et la présence des témoins à l'établissement du certificat : preuve d'individualité.

Le certificat est établi dans le but de corriger les erreurs faites lors de l'établissement des actes transcrits sur les registres ce sont des erreurs dans les noms ou les dates de naissance.

3.3.1.6 Pour certificat de résidence

Le certificat de résidence est délivré suite à la fourniture de l'un des documents suivants :

- La quittance de loyer
- la quittance d'électricité ou
- la quittance d'eau

3.3.1.7 Pour certificat de non divorce

Vérification dans les registres de mariage que la veuve n'a pas divorcé avant le décès de son marie.

3.3.1.8 Pour fiche individuelle

Vérification que l'intéressé soit en vie ainsi que validité des informations du livret de famille.

3.3.1.9 Pour fiche familiale

Vérification de la validité des informations du livret de famille. Pour délivrer des copie intégrales, il devrait y avoir une vérification si l'acte délivré est transcrit sur les registres de l'APC dans la quelle se présente l'intéressé.
Pour le reste des fiches d'état civil la vérification concernera la disponibilité du livret de famille.

3.3.2 L'établissement des actes

Les actes d'état civil énoncent :

- L'année
- Le jour et l'heure où ils sont reçus
- Les noms et prénoms de tous ceux qui y sont dénommés
- Les professions et domiciles de tous ceux qui y sont dénommés
- Les dates et lieux de naissance des pères et mères dans les actes de naissance
- Les dates et lieux de naissance des époux dans les actes de mariage
- Les dates et lieux de naissance du décès

L'officier d'état civil (OEC) établit trois types de fiches d'état civil :

- Etablissement des actes transcrits sur l'un des registres d'état civil
- Etablissement des fiches et certificats non transcrits sur les registres d'état civil
- Etablissement des fiches depuis le livret de famille

3.3.2.1 Etablissement des actes transcrits sur l'un des registres d'état civil

Toutes les informations se trouvent dans les registres d'état civil

3.3.2.1.1 Etablissement d'acte de naissance

Selon deux cas (cas avec jugement, cas sans jugement), l'acte contient les informations :

cas sans jugement

- Nom, prénom, date, heur et lieu de naissance du nouveau né
- Nom, prénom, date, heur et lieu de naissance des parents
- Nom, prénom du déclarant de naissance

cas avec jugement

- Nom, prénom, date, heur et lieu de naissance du nouveau né
- Nom, prénom, date, heur et lieu de naissance des parents
- Nom de jugement
- Tribunal de jugement

3.3.2.1.2 Etablissement d'acte de mariage

cas sans jugement

- Nom, prénom, résidence, profession, date et lieu de naissance des époux
- Nom, prénom des témoins
- Date de mariage

cas avec jugement

- Nom, prénom, résidence, profession, date et lieu de naissance des époux
- Date de mariage, tribunal de jugement

Suite à l'établissement de l'acte de mariage, l'OEC effectue les taches suivantes :

- Mentionner le mariage dans les marges du registre de naissance
- Délivrer un livret de famille

3.3.2.1.3 Etablissement d'acte de décès

cas sans jugement

- Nom, prénom, résidence, profession, âge, date et lieu de naissance du décédé
- Nom, prénom du déclarant du décès
- Date, heure de décès

cas avec jugement

- Nom, prénom, résidence, profession, âge, date et lieu de naissance du décès
- Date de décès, tribunal de jugement

Suite à l'établissement de l'acte de décès, l'OEC mentionne le décès dans les marges du registre des naissances.

3.3.2.2 Etablissement des fiches non transcrites sur les registres

Ce sont des fiches prouvant certains états et situations vis-à-vis des citoyens

3.3.2.2.1 Etablissement de certificat de résidence

- Nom, prénom, date et lieu de naissance de l'intéressé
- Lieu de résidence de l'intéressé (commune de résidence).

3.3.2.2.2 Etablissement de certificat de preuve d'individualité

- Nom, prénom et résidence de l'intéressé
- Nom et prénom des témoins
- Nom (date de naissance) erroné et le nouveau (nouvelle) nom (date de naissance) rectifié.

3.3.2.2.3 Etablissement de certificat de non divorce

- Date de mariage
- Nom, prénom de l'intéressé et l'époux décédé
- Date de décès de l'époux

3.3.3 Apposer les mentions et transcrire certains jugements

3.3.3.1 Mentionner le mariage

- Nom et prénom de l'époux
- Date et numéro d'acte de mariage
- Mentionné dans registre de naissance et le registre de mariage

3.3.3.2 Mentionner le décès

- Date et numéro d'acte de décès

Mentionné dans registre de naissance et le registre de mariage

3.3.3.3 Mentionner et transcrire le jugement de divorce

- Date, Tribunal de jugement
- Numéro d'acte mariage

Mentionné dans le registre de mariage

Mentionné dans le registre de naissance

3.3.3.4 Mentionner et transcrire le jugement de preuve d'individualité

- Ancien et nouveau (nom ou date de naissance)
- Date et Tribunal de jugement

Mentionné dans le registre de naissance

3.3.4 La tenue et la conservation des registres

3.3.4.1 Tenue des registres

Les actes de l'état civil sont transcrits, dans chaque commune, sur 3 registres tenus en double :

- un registre des actes de naissance
- un registre des actes de mariage
- un registre des actes de décès

Chaque registre doit comporter une marge permettent l'apposition des mentions marginales. Les registres sont cotés par première et dernière, et paraphés sur chaque feuille, par le président du tribunal ou le juge qui le remplace. Le président du tribunal dresse un procès-verbal d'ouverture du registre, qui est consigné sur ce dernier et qui précise le nombre de feuilles le composant.

Les actes sont inscrits sur les registres, de suite, sans aucun blanc ni interligne les ratures et les renvois sont approuvés et signé de la même manière que le corps de l'acte il n'y est rien écrit par abréviation et aucune date n'y est mise en chiffres Les registres sont clos et arrêtés par l'officier de l'état civil, à la fin de chaque année, dans le mois qui suit, l'un des doubles est déposé aux archives de la commune, l'autre au greffe de la cour .

3.3.4.2 Conservation des registres

Les registres de l'état civil doivent être conservés au siège de la commune et au greffe pendant cent ans à compter de leur clôture. Après ce délai, les registres des greffes sont versés dans les wilayas où ils sont conservés indéfiniment.

La consultation directe des registres par les personnes autres que les agents de l'état habilités est interdite. Toutefois, la consultation des registres datant de plus de cent ans est soumise aux règles qui régissent la consultation des archives communales.

Les dépositaires des registres sont tenus de les communiquer :

- aux procureurs généraux et à leurs substituts pour leur permettre d'exercer leur contrôle et d'obtenir tout renseignement

- aux walis, aux chefs d'arrondissement et à leurs délégués pour leur permettre de procéder à certaines opérations administratives
- aux administrations qui seront déterminées par décret En outre, les registres sont déplacés en vue de leur consultation :
- par les juridictions, lorsqu'une décision de justice ordonne leur communication
- par les procureurs généraux ou les magistrats qu'ils ont délégués pour opérer leur control annuel.

Chapitre 4 : Conception orientée Aspect et par composant d'eAPC

Dans ce chapitre nous allons présenter une conception orientée aspect et par composants d'eAPC service d'état civil, nous utiliserons la notation IASA. Nous commencerons par donner un aperçu global du système puis nous procéderons par raffinement successif pour détailler à un niveau de granularité le plus fin possible. Mais avant d'entamer la conception, on va présenter brièvement comment les composants sont décrits avec IASA.

4.1. Description de l'architecture avec IASA :

Le composant composite dont on veut décrire la vue interne est représenté par un cadre qui englobe les autres éléments, la partie de contrôle est située en bas à droite et la partie aspect en bas à gauche.

Une interface (port) est représentée avec IASA sous la forme d'un petit carré de couleur grise pour les interfaces fournies, et de couleur blanche pour les interfaces requises.

- interface fournie
- interface requise

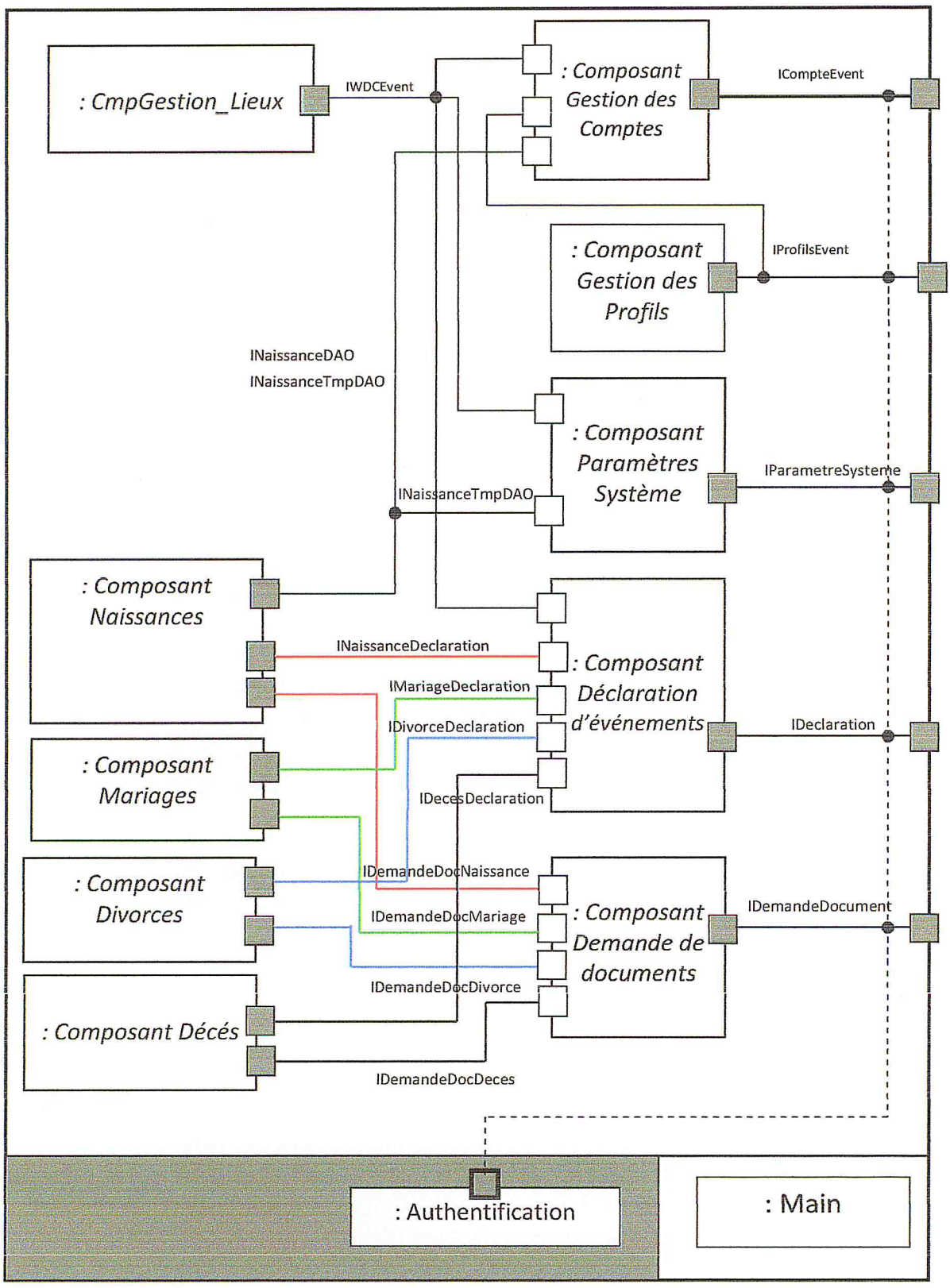
Les sous composants faisant partie du métier sont définis dans la partie opérationnelle, ils sont définis sous forme de boites contenant le nom du sous composant et ses interfaces (sur le contour). Les boites ayant un contour pointillé sont des composants externes qui ne font pas partie du composant décrit, mais comme ce dernier les utilise, alors on les intègre dans sa description pour que son fonctionnement devienne plus clair. Les aspects sont définis de la même façon que les sous composants mais à l'intérieur de la partie aspect.

Un connecteur est défini par une ligne joignant deux interfaces, et la coupe d'un aspect est définie par une ligne pointillée entre l'aspect et le point de jonction (un port spécifique).

- le composant utilisateur CmpUtilisateur
- un composant externe (gestion des lieux) : CmpGestion_Lieux
- Un connecteur ————
- Une coupe ----------

4.2. Conception du System E_APC, Le composant Civile State :

Architecture globale de l'application e-APC : Civil State Component (Partie 1)



Architecture globale de l'application e-APC : Civil State Component (Partie 2)

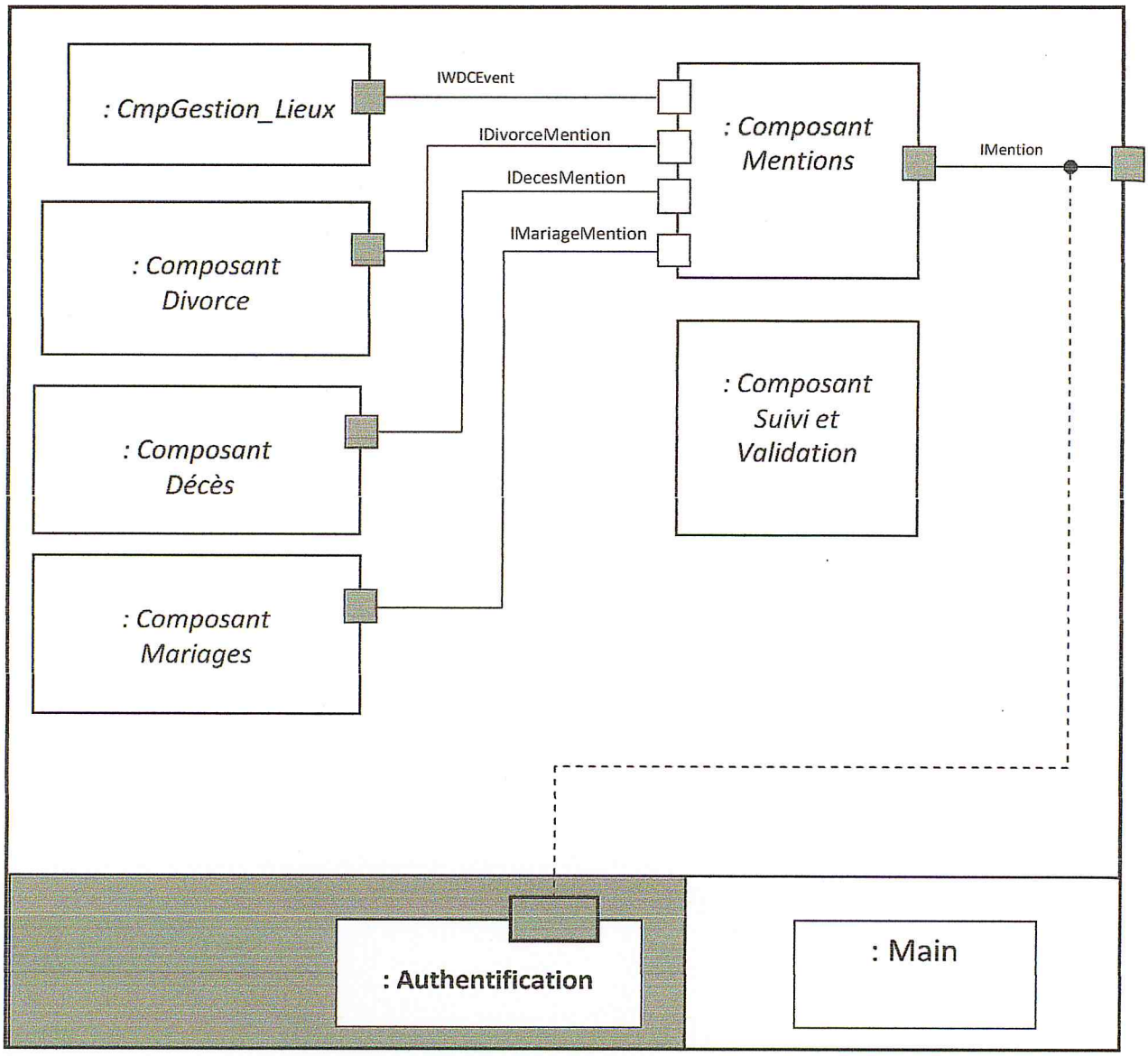


Figure 4.1 : Architecture globale d'eAPC

4.3. Raffinement des composants d'eAPC :

On a définit l'architecture globale d'eAPC, maintenant on va décrire la vue interne de certains de ses sous composants composites pour donner un aperçu sur leur structure.

4.3.1 Composant Gestion des comptes

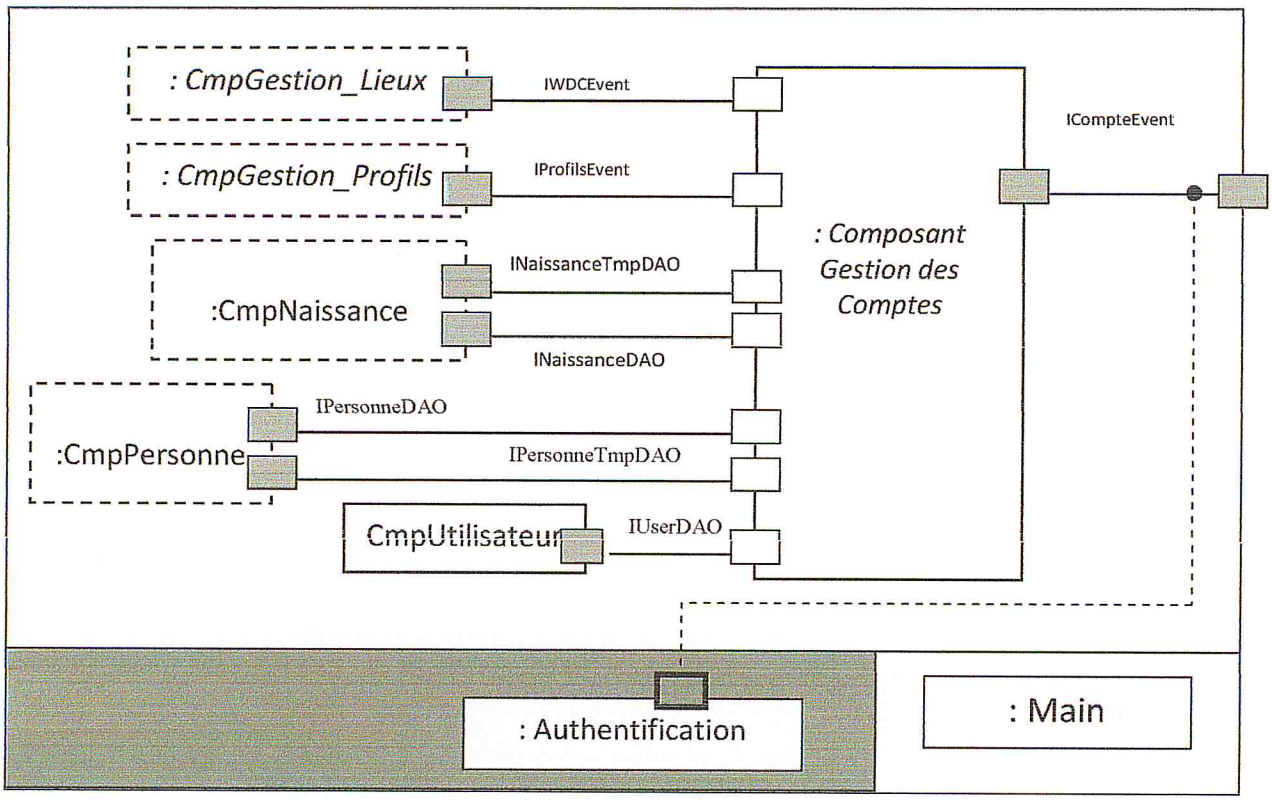


Figure 4.2 : Composant gestion des comptes

Le composant gestion des comptes est un composant composite qui se charge de :

1. La validation des comptes par les utilisateurs autorisés.
2. La création d'un nouveau compte utilisateur.
3. La suppression des comptes.
4. La modification des comptes.

Ces taches seront effectuées à l'aide de deux composants externes Gestion des lieux et gestion des profils pour le processus de création des comptes, ainsi la sauvegarde de données personnelles sera effectuée à travers les composants CmpNaissance, CmpPersonne et CmpUtilisateur.

4.3.2 Composant paramètres du système :

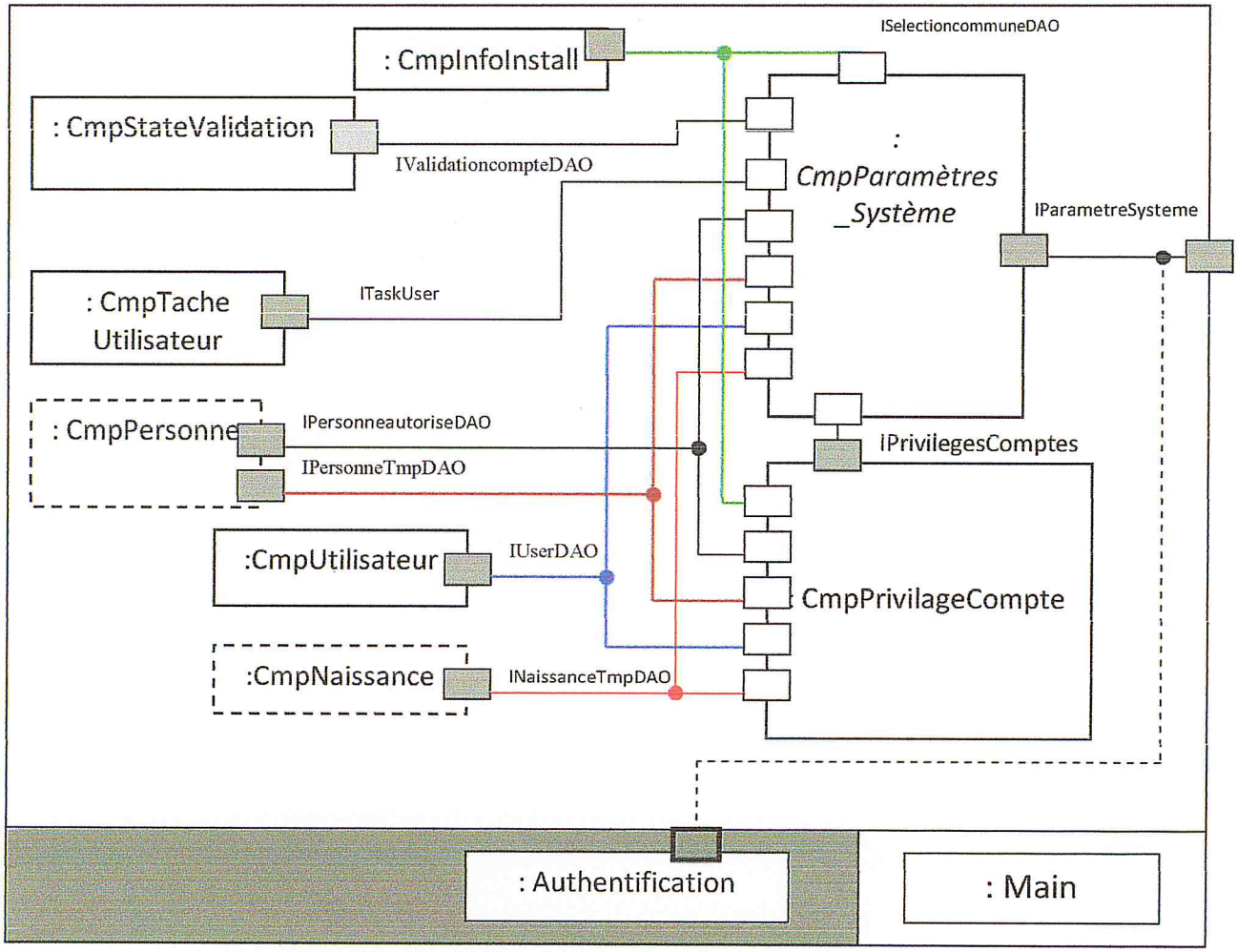


Figure 4.3 : Composant Paramètres du système

Le composant *paramètres du système* est un composant composite qui se charge de :

1. La spécification du type de validation.
2. La spécification du nombre de validation.
3. Les événements : modification, suppression ou ajout d'une wilaya, daïra ou commune.
4. La configuration d'une nouvelle tâche.
5. La modification de l'année de découpage administrative.
6. De spécifier les personnes autorisées à faire sortir des documents à partir d'un autre compte

Le composant *paramètres du système* va interagir avec le composant *Tâche utilisateur* pour la configuration d'une nouvelle tâche. Le composant authentification est instancié dans la partie aspect ce service sera appliqué au composant paramètres système.

4.3.3 Composant Gestion des profils

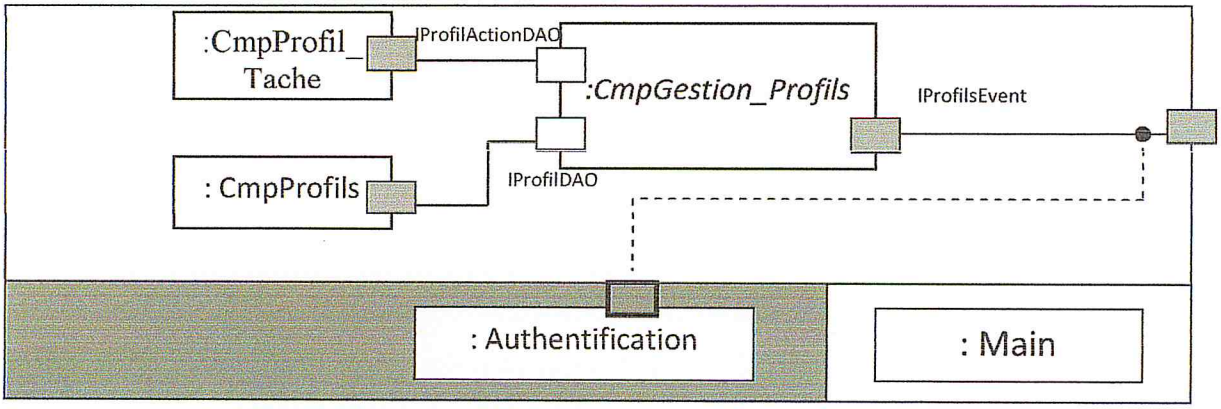


Figure 4.4 : Composant Gestion des profils

Le composant gestion des profils se charge de :

1. Spécifier les taches attribuées aux profils.
2. De créer de nouveaux profils grâce au composant CmpProfils.
3. La suppression et la modification des profils ainsi que les taches attribuées.

Le composant interagit avec des composants qui sont liés à la base de données tels que :

CmpProfil_Tache, CmpProfils.

4.3.4 Le Composant gestion des lieux :

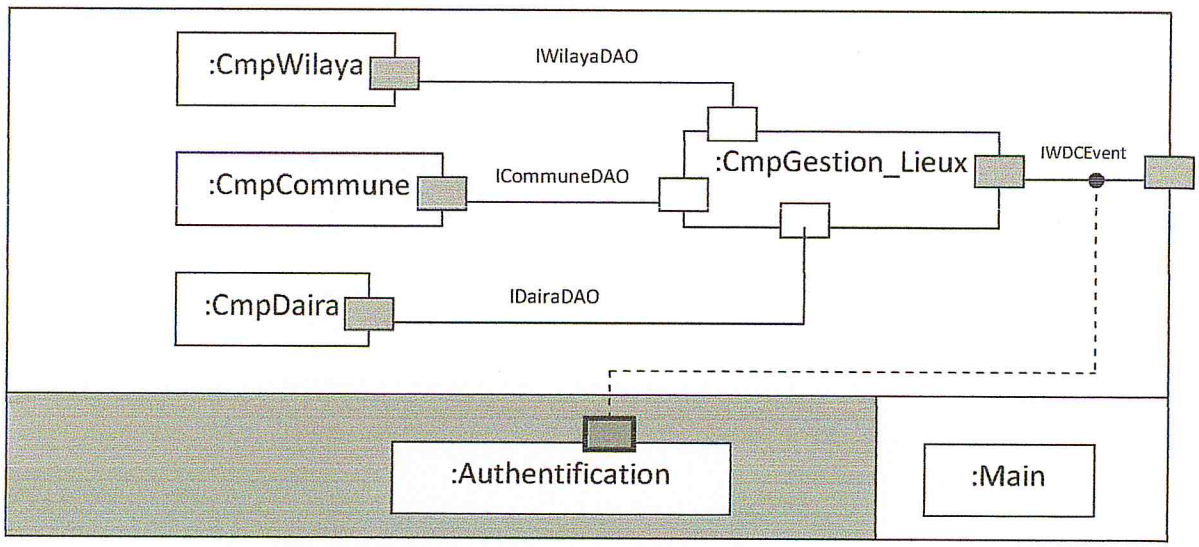


Figure 4.5 : Composant Gestion des lieux

Le composant gestion des lieux se charge de :

1. Modifier une wilaya, daïra ou commune.
2. Supprimer une wilaya, daïra ou commune.
3. Ajouter une wilaya, daïra ou commune.

Ce composant effectue ces différentes fonctions grâce aux composants CmpWilaya, CmpCommune, CmpDaira ; ces composants interagissent avec la base de données.

4.3.5. Le Composant Demandes de documents :

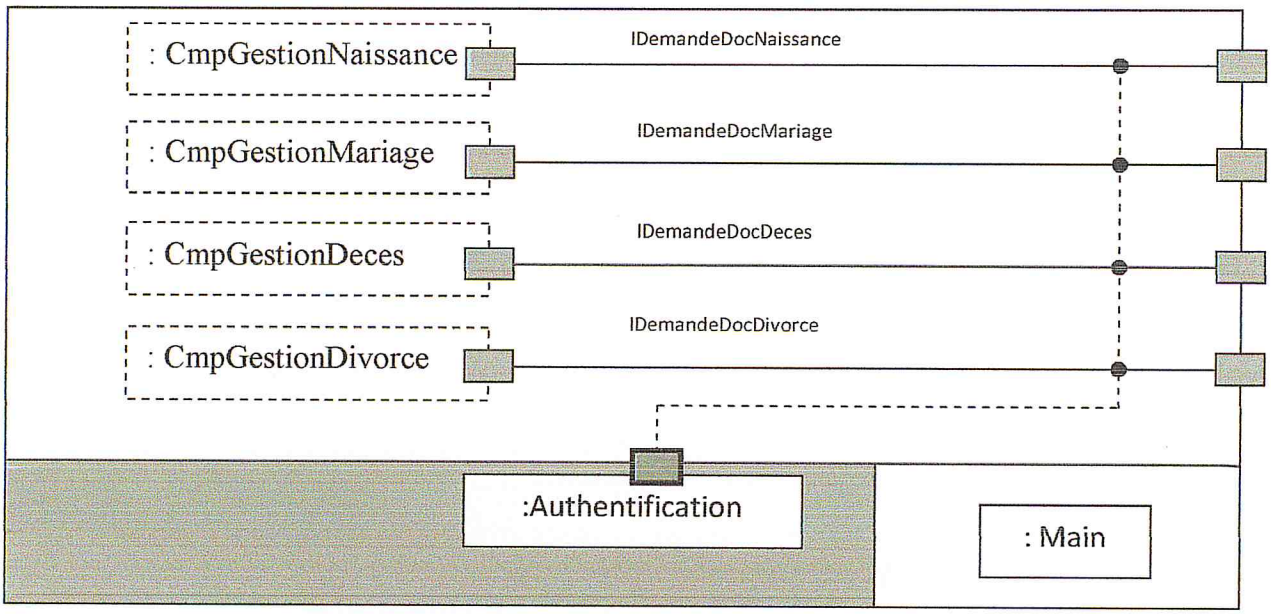


Figure 4.6 : Composant Demandes de documents

Le composant Demande de documents se charge de :

1. Recevoir les demandes de documents.
2. Modifier, Supprimer une demande.
3. Validation d'une demande de documents.

4.3.6. Le Composant Déclarations d'événement :

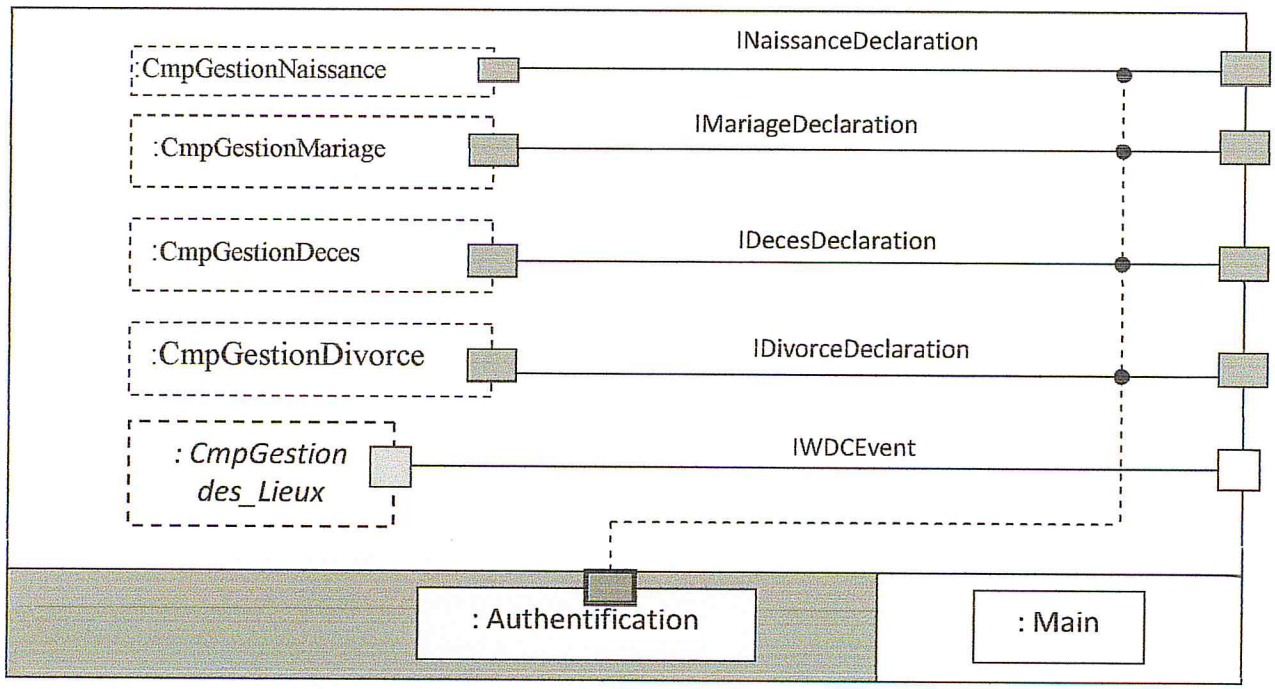


Figure 4.7 : Composant Déclarations des événements

Le composant Déclaration de événement se charge de :

1. Recevoir les événements de déclaration de documents.
2. Modifier, Supprimer une déclaration.
3. Validation d'une déclaration.

4.3.7. Le Composant Mention :

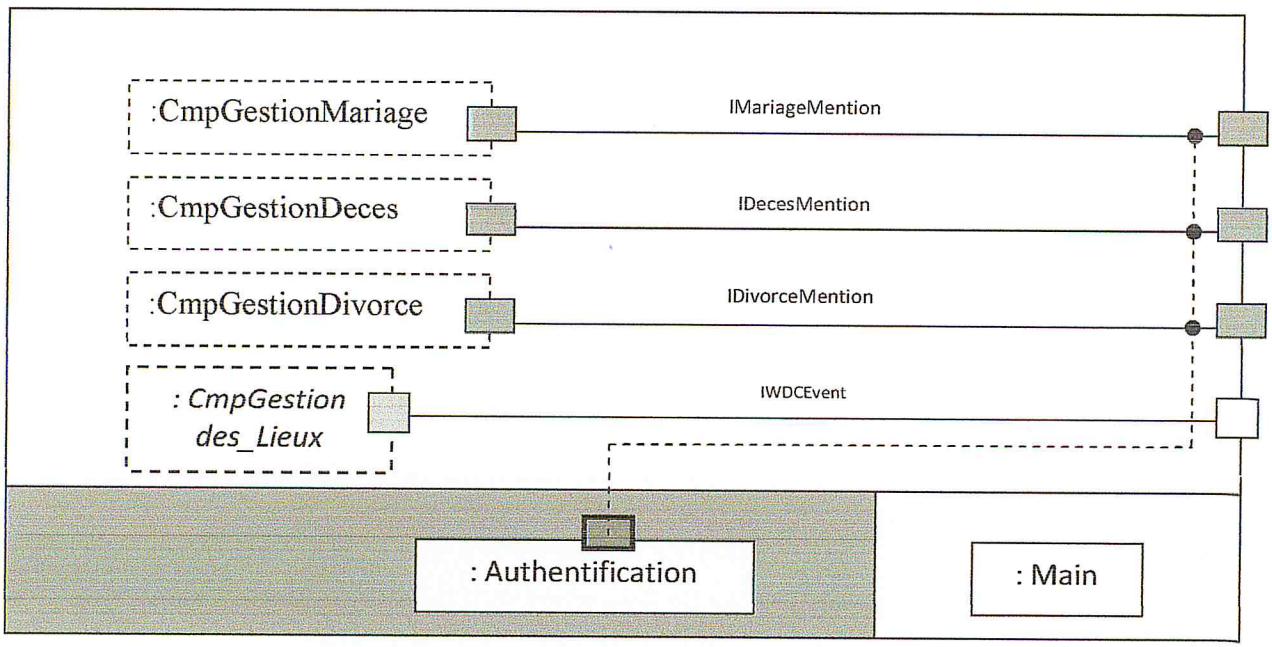


Figure 4.8 : Composant Mention

Le composant Mention se charge de :

1. Recevoir les événements de mention.
2. Ajouter, Supprimer, Modifier une mention.
3. Validation des mentions.

4.3.8. Raffinement du Composant Naissances :

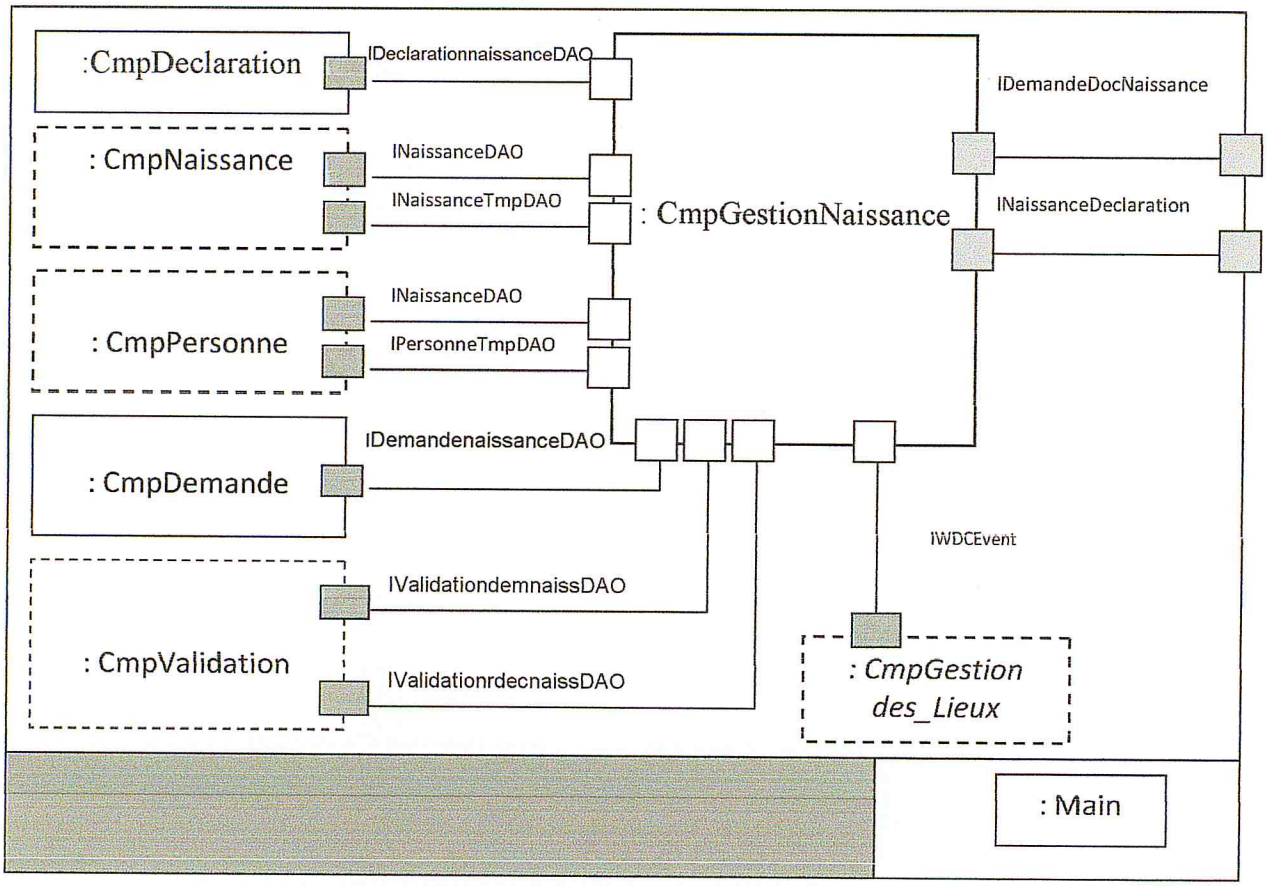


Figure 4.9 : Composant Naissance

Le composant naissance se charge de ces fonctions :

1. La suppression, création et la modification d'une déclaration ou d'une demande
2. La validation d'une déclaration ou d'une demande

Ces opérations ce font a travers les composants naissance et personne qui interagissent avec la base de données.

4.3.9. Raffinement du Composant Mariage:

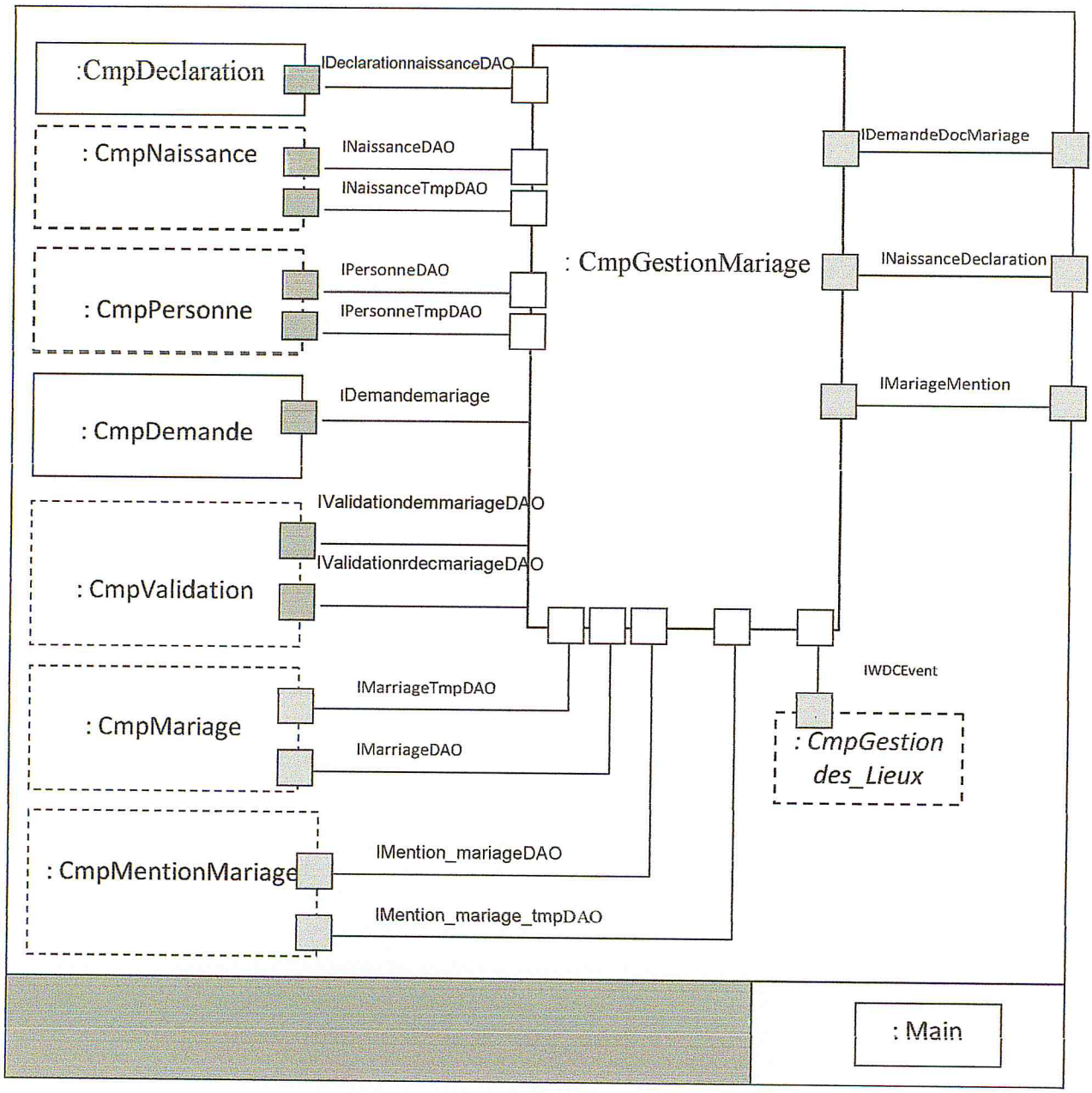


Figure 4.10 : Composant Mariage

Le composant mariage se charge de ces fonctions :

1. La suppression, création et la modification d'une déclaration ou d'une demande
2. La validation d'une déclaration, demande ou mention

Ces opérations ce font a travers les composants naissance et personne, mariage qui qui interagissent avec la base de données.

4.3.10. Raffinement du Composant Décés

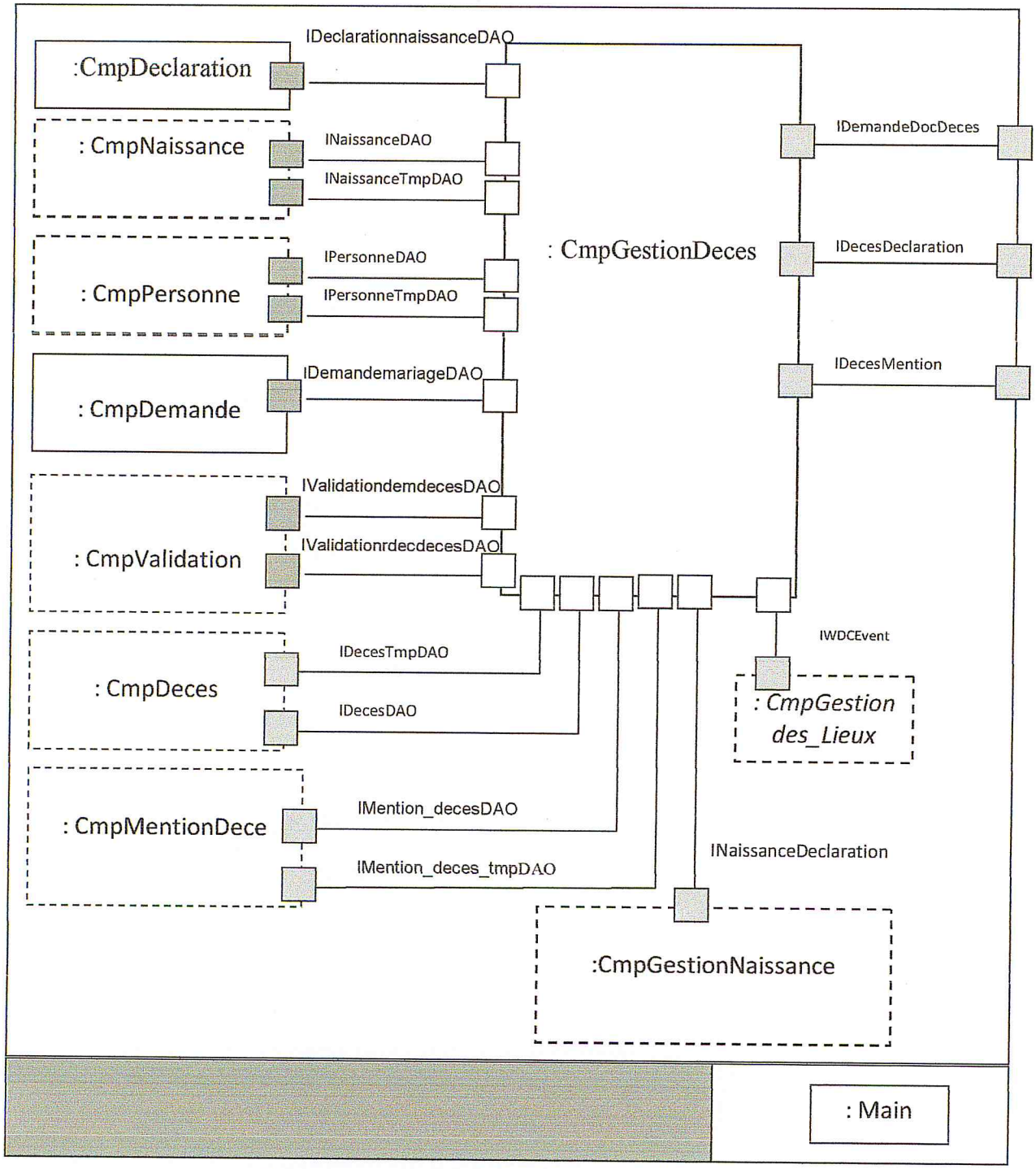


Figure 4.11 : Composant décés

Le composant décès se charge de ces fonctions :

1. La suppression, création et la modification d'une déclaration ou d'une demande
2. La validation d'une déclaration, demande ou mention

Ces opérations ce font a travers les composants naissance et personne, dece qui interagissent avec la base de données.

4.3.11. Raffinement du Composant Divorce

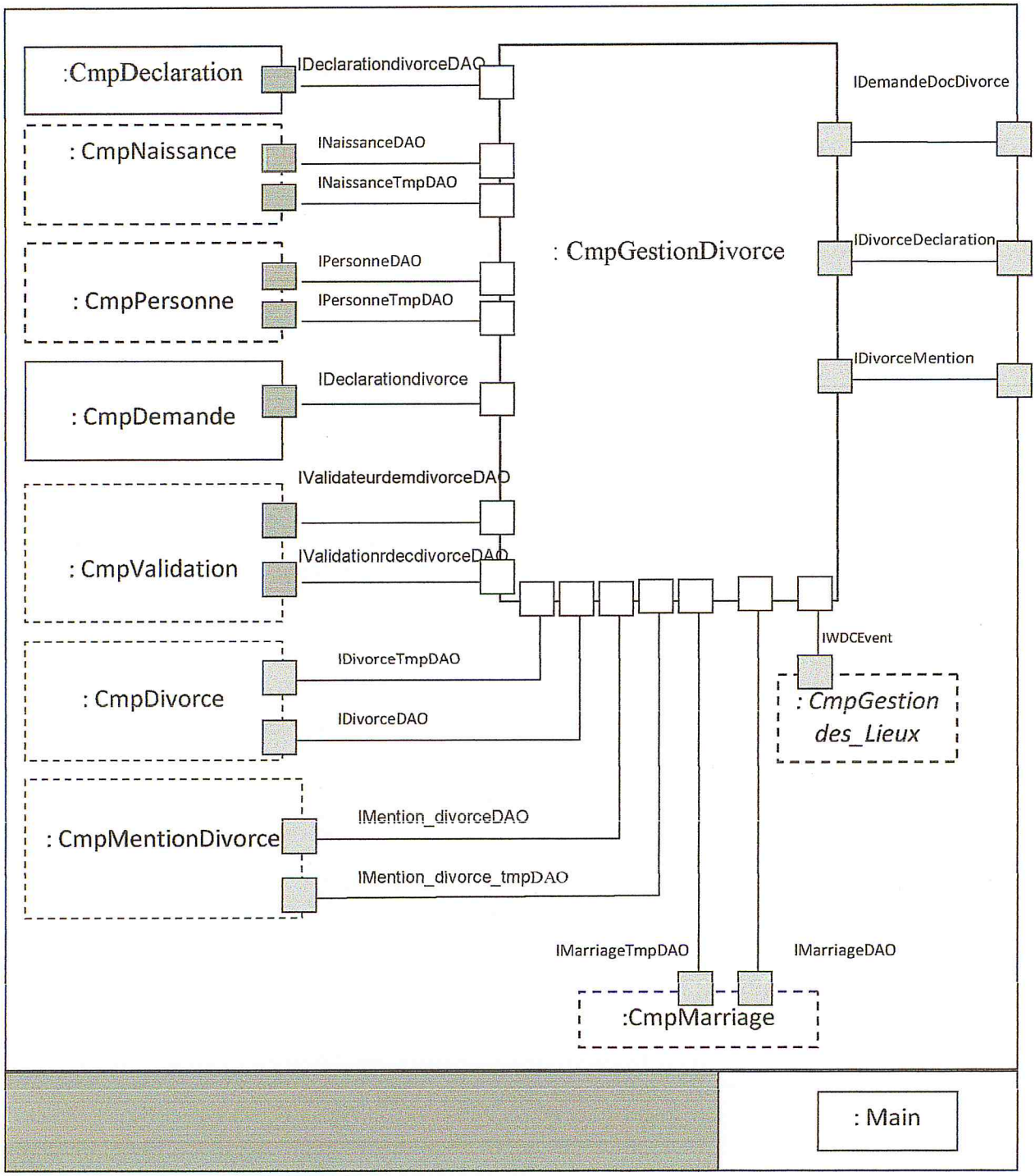


Figure 4.12 : Composant divorce

Le composant divorce se charge de ces fonctions :

1. La suppression, création et la modification d'une déclaration ou d'une demande
2. La validation d'une déclaration, demande ou mention

Ces opérations ce font a travers les composants naissance et personne, divorce qui interagissent avec la base de données.

4.3.12. Raffinement des Composants : CmpMentionDeces, CmpMentionMariage, CmpMentionDivorce, CmpDivorce, CmpValidation, CmpPersonne, : CmpNaissance :

AUD = ADD, UPDATE, DELETE.

CmpMentionDeces : défini par deux primitifs pour effectuer des opérations AUD sur les données de la Base de données.

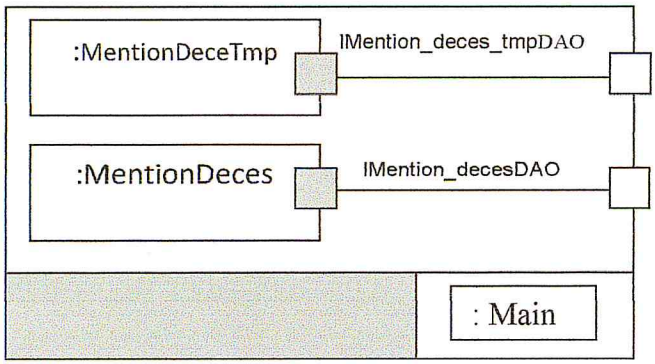


Figure 4.13 : Composant CmpMentionDeces

CmpDivorce : défini par deux primitifs pour effectuer des opérations AUD sur les données de la Base de données.

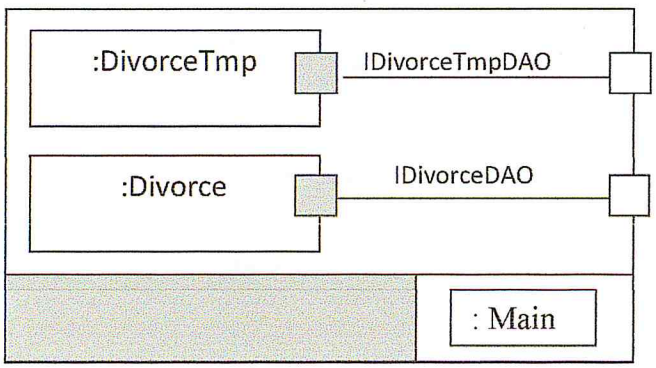


Figure 4.14 : Composant CmpDivorce

CmpMariage : défini par deux primitifs pour effectuer des opérations AUD sur les données de la BD

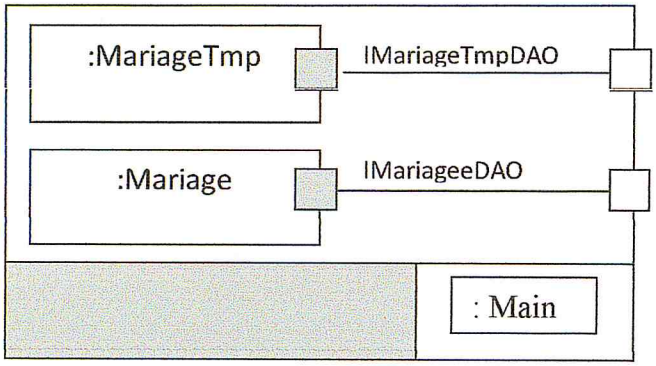


Figure 4.15 : Composant CmpMariage

CmpDeces : défini par deux primitifs pour effectuer des opérations AUD sur les données de la Base de données.

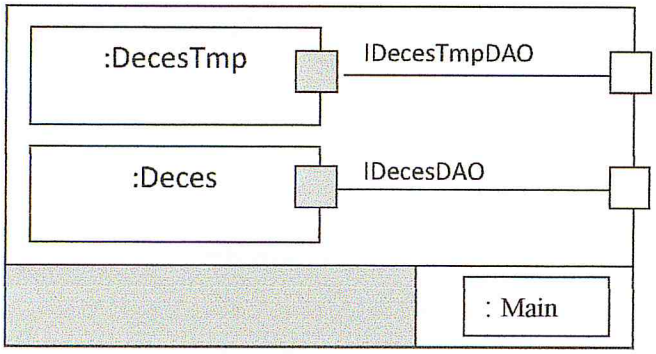


Figure 4.16 : Composant CmpDeces

CmpPersonne : défini par trois primitifs pour effectuer des opérations AUD sur les données de la BD

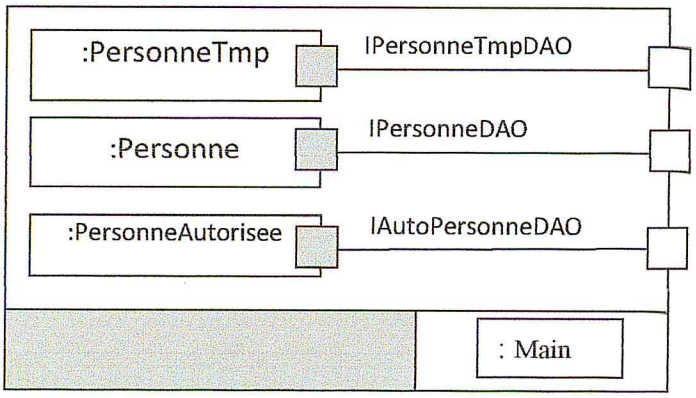


Figure 4.17 : Composant CmpPersonne

CmpNaissance : défini par deux primitifs pour effectuer des opérations AUD sur les données de la BD

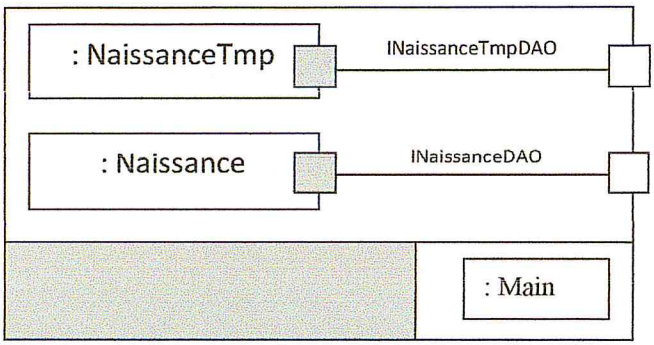


Figure 4.18 : Composant CmpNaissance

CmpMentionMariage : défini par deux primitifs pour effectuer des opérations AUD sur les données de la BD

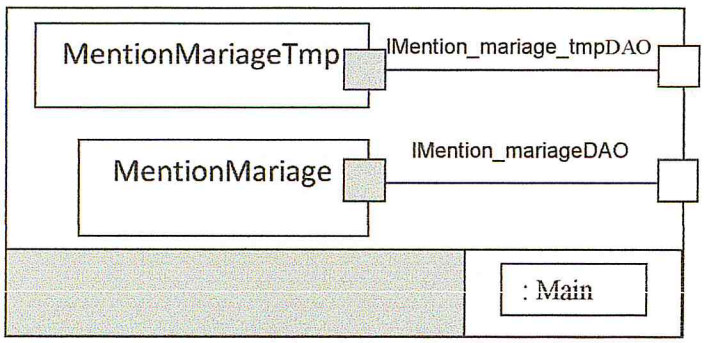


Figure 4.19 : Composant CmpMentionMariage

CmpMentionDivorce : défini par deux primitifs pour effectuer des opérations AUD sur les données de la BD

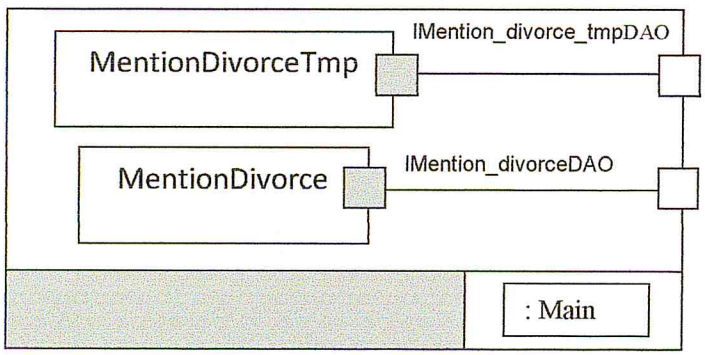


Figure 4.20 : Composant CmpMentionDivorce

Chapitre 5 : Implémentation

5.1. Introduction

Dans ce chapitre on présentera les détails d'implémentation du modèle de composants que nous avons choisi, ainsi que l'outil de connexion qui a été réalisé: ***l'Assembleur***.

5.2. Exemple d'introduction

Avant de se lancer dans la description de l'assembleur, un exemple simple sera utile pour éclaircir le mécanisme d'assemblage. Le modèle utilisé pour la représentation des composants est le modèle IASA puisqu' il répond aux exigences de l'application E-APC qui se veut orientée Aspect.

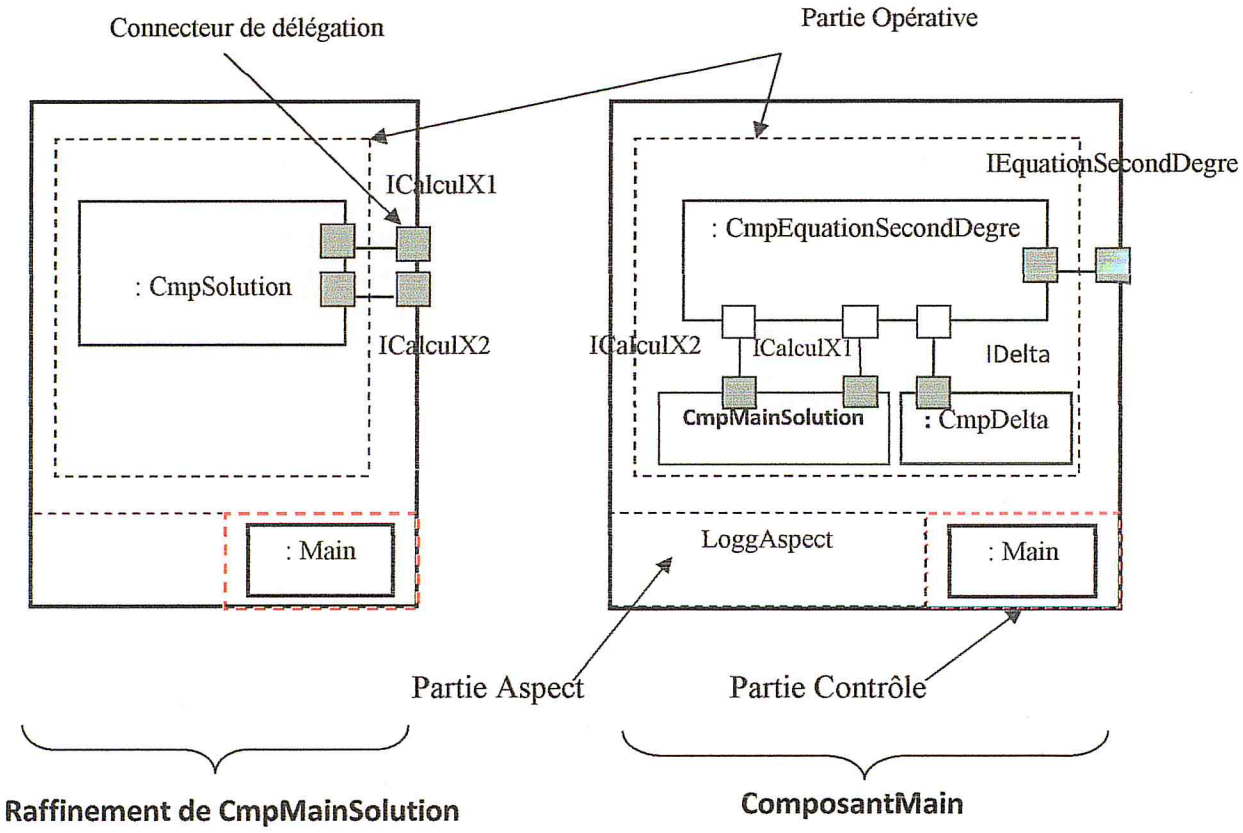


Figure 5.1 : Exemple: Solution Equation Second Degré

Dans la représentation précédente le composant *ComposantMain* est Constitué de trois parties :

- Partie Contrôle : *Main*
- Aspect Part : Représenté par *LoggAspect*
- Operative Part : Constituée de deux composants primitfs CmpEquationSecondDegre et CmpDelta.

Pour passer du modèle abstrait vers un modèle concret, on crée un fichier descripteur pour chaque composant composite. Ce fichier décrit la vue interne du composite. Il s'agit d'une description détaillée du composant contenant:

- Le nom du composant.
- Le type du composant.
- Les interfaces qu'il requiert ou qu'il fournit, leurs noms, rôles et signatures.
- La classe implémentant la partie contrôle.
- Les aspects qui y seront tissés.
- Les sous composants du composant décrit et les liaisons qui se font entre ces derniers (connecteurs d'assemblage) ou entre un sous composant et le composant décrit (connecteur de délégation).

Le fichier est écrit en XML, il contient un ensemble de balises qui sera détaillé plus loin dans ce chapitre. On montre la description du composant **ComposantMain** dans l'exemple suivant :

On commence la description en déclarant le composant décrit, c'est en créant une balise ***définition,*** à l'intérieur de cette balise on spécifié :

- les interfaces du composant, à l'aide de la balise ***interface***
- les sous composants, avec la balise ***component,*** cette dernière contient des balises ***interface*** et ***content*** qui sont propres au sous composant.
- la partie contrôle, avec la balise ***content.***
- les aspects, avec la balise ***aspectconfig.***
- les connecteurs, avec la balise ***binding***.

```
<definition name="ComposantMain" type="composite" >

    <interface name="m" role="server" signature="IEquationSecondDegre" />

    <component name="CmpEquationSecondDegre" type="primitive">
        <interface name="m" role="server" signature="IEquationSecondDegre" />
        <interface name="d" role="client" signature="IDelta" />
        <interface name="s1" role="client" signature="ICalculX1" />
        <interface name="s2" role="client" signature="ICalculX2" />
        <content class="Test.EquationSecondDegre.EquationSecondDegre" type=""/>
    </component>

    <component name="CmpDelta" type="primitive">
        <interface name="d" role="server" signature="IDelta" />
        <content class="Test.EquationSecondDegre.Delta" type=""/>
    </component>

    <component name="CmpMainSolution" type="composite">
        <interface name="s1" role="server" signature="ICalculX1" />
        <interface name="s2" role="server" signature="ICalculX2" />
        <content class="Test.Solution.Main" type="controller" />
    </component>

    <content class="Test.EquationSecondDegre.Main" type="controller" />
    <aspectconfig name="applicationContext" />
```

Dans ce descripteur XML on a spécifié un composant regroupant 3 composants :

- Deux composants primitifs (CmpEquationSecondDegre, CmpDelta).
- Un composant composite (CmpMainSolution).

On ajoute les connexions inter composants avec la balise ***binding,*** et on déclare la fin du fichier :

```
    <binding client="ComposantMain.m" server="CmpEquationSecondDegre.m" />
    <binding client="CmpEquationSecondDegre.d" server="CmpDelta.d" />
    <binding client="CmpEquationSecondDegre.s1" server="CmpMainSolution.s1" />
    <binding client="CmpEquationSecondDegre.s2" server="CmpMainSolution.s2" />

</definition>
```

Cette description est utilisée par ***l'Assembleur*** comme référence pour configurer dynamiquement l'architecture d'un composant lors de sa création.

5.3. Solution Proposée

5.3.1. Le descripteur XML

Le fichier descripteur a un ensemble de balises bien défini :

1. <definition name="*Nom du composant*" type="*composite*">

Cette balise définit le composant global, l'attribut ***name*** est le nom du composant et l'attribut ***type*** peut être soit composite/primitive.

2. <interface name="*nom*" role="*client*" signature="*Nom de l'interface*" />

Cette balise décrit une interface, l'attribut ***name*** est le nom attribué a l'interface, l'attribut ***role*** c'est le rôle de l'interface dans la liaison (client/serveur) et ***signature*** c'est la signature de l'interface (code).

3. <content class="*Nom complet de la classe*" type = "*controller*" />

Cette facette décrit la classe que possède le composant avec l'attribut ***class***. La classe peut décrire le contrôleur du composant externe

4. <binding client="NomComposant.interface" server=" NomComposant.interface" />

Cette facette décrit la liaison entre le composant qui demande le service et le composant fournisseur du service

5. <aspectconfig name="*nom du descripteur des aspects*" />

Cette facette détient une référence vers le descripteur aspect des composants. Le descripteur des aspects a ses propres facettes appartenant à SprigAOP.

5.3.2. L'Assembleur :

L'assembleur est un outil développé pour permettre l'assemblage et la configuration dynamiques des composants dans une application, selon une description fournie.

Lorsqu'un composant est instancié, sa partie contrôle fait appel à l'assembleur pour charger le fichier descripteur propre au composant, le parser et configurer les dépendances de ce composant comme décrit dans le descripteur.

5.3.3. Le composant dans le modèle concret :

Les primitifs sont implémentés par des classes java implémentant les interfaces fournies par le composant. Pour éliminer les dépendances vers d'autres composants, le code source d'une telle classe ne contient aucune référence directe vers un type de composant externe. Au lieu de ça, il utilise des références sous forme d'interfaces, qui sont définies conceptuellement en tant qu'interfaces requises. Comme la vue interne de ces composants est inaccessible, ces composants n'ont pas de fichier de description d'architecture.

Les composites sont implémentés sous forme de package contenant une classe principale (contrôleur), des classes représentant les composants primitifs et souvent d'autres composants composites. Ces composants ont chacun un fichier descripteur d'architecture à fin de les configurer. La classe principale étend la classe **Controller** et implémente l'interface ***Binding*** pour pouvoir utiliser l'API de l'assembleur. Elle implémente aussi les interfaces fournies par le composite puis les délègue aux sous composants chargés de les réaliser.

5.3.3.1. La délégation :

La délégation est un comportement lié à l'interception des appels entrants et de les rediriger vers le sous composant approprié.

La délégation est assurée par l'implémentation de l'interface ***Binding***, dans nos exemples et notre application nous avons opté pour que la classe principale du composite soit la classe qui l'implémente.

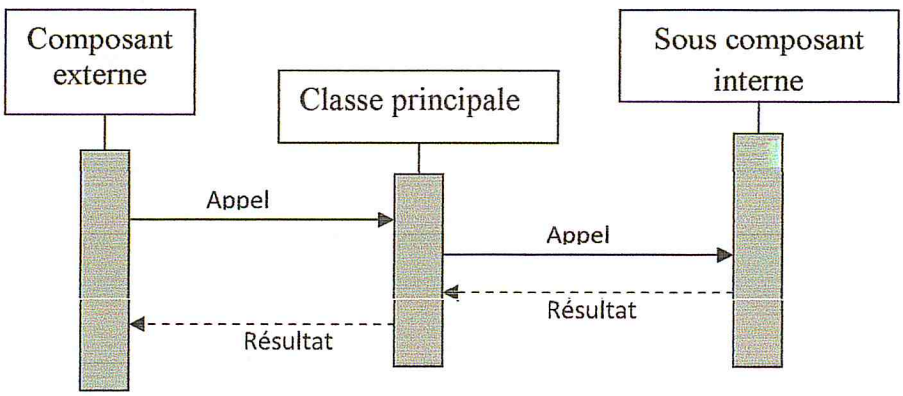


Figure 5.2. Diagramme de séquence montrant un Exemple de la délégation

5.3.3.2. Le Contrôleur

Le contrôleur est un objet qui est chargé de traduire la configuration fournie dans le fichier descripteur en une connexion concrète entre des sous composants déterminés. Dans nos exemples et notre application on a opté pour que la classe principale du composite soit le contrôleur de ce dernier. La fonction du contrôleur est décrite dans la section « 5.3.4. Mécanisme d'assemblage ».

Listing : Contrôleur du composant CmpEquationSecondDegre

```
@Descriptor("Test.EquationSecondDegre.EquationSecondDegre")

public class Main extends Controller implements Binding {
        IEquationSecondDegre equationSecondDegre;

        public void bind(String cl, Object sr) {

          if(cl.equals("m")){  equationSecondDegre= (IEquationSecondDegre)sr; }
        }

        public Object lookUp ( String cl ) {

                if(cl.equals("m")){ return equationSecondDegre; }
                return null;
        }
 }
```

L'annotation dans la classe Main définit un lien vers le fichier descripteur.

5.3.4. Le mécanisme d'assemblage

Un composant communique avec les autres composants grâce à ses interfaces, ce qui fait que ces interfaces sont en quelque sorte des identifiant pour identifier les autres composants. De ce principe, l'assembleur charge le fichier descripteur et en construit un arbre de liaisons, cet arbre est traduit plus tard en un mappage qui met en correspondance chaque interface d'un composant avec le composant qui l'utilise réellement. Lorsqu'un composant veut communiquer avec un autre composant, il demande ce composant au mappage en utilisant l'interface correspondante (lookup). Les composants sont construits en utilisant les mécanismes de réflection de Java, les classes utilisées pour l'instanciation sont définies dans le fichier descripteur, la réflexion, la configuration et la communication basée sur les interfaces rendent les

composants indépendants de l'implémentation de leur contexte d'exploitation et améliore la réutilisabilité.

5.4. Spécification des aspects d'un composant

A fin de tisser les aspects dans notre application, nous avons intégré SpringAOP dans l'assembleur. En effet, la spécification des aspects d'un composant se fait dans le fichier descripteur grâce à la balise *aspectConfig* qui comporte une référence vers le descripteur d'aspects SpringAOP. Ce dernier décrit les aspects à la manière de SpringAOP qui sera utilisée par l'assembleur pour tisser les aspects.

SpringAOP

Le Framework Spring fournit un léger conteneur de beans (bean factory) qui implémente le principe d'inversion de contrôle (IoC ou injection de dépendance). Ce principe permet une séparation entre la définition du bean et sa configuration et la gestion de ses dépendances, en résumé lorsque l'application a besoin d'un bean elle demande à l'usine qui s'occupe de le créer et le configurer selon un contexte d'application (spécifié dans un fichier de configuration en XML). Spring applique le même principe à la POA. En effet, l'usine retourne un bean qui sera manipulé à travers un proxy AOP qui se charge du tissage des aspects lors de l'exécution.

Le greffon (advice code) est implémenté dans un ***Interceptor***, qui sera relié à une ***coupe*** par le biais d'un ***Advisor***. Un aspect est défini avec Spring sous la forme d'un ensemble d'***Advisors***.

Comme la construction d'un objet est à la charge de l'usine, Spring reconnait **dangereux** d'intercepter les constructeurs, et même d'intercepter les setters/getters d'un attribut d'un objet, ce qui fait que le seul type de points de jonction possible avec Spring AOP est l'interception des méthodes. Les coupes sont définies avec une syntaxe utilisant les expressions régulières PERL5. Les types de greffons possibles sont les types : BEFORE, AROUND, THROWS et AFTER RETURNING, qui s'exécutent respectivement avant, autour du point de jonction, lorsque le point de jonction lève une exception ou après le retour du point de jonction.

Le mécanisme d'introduction de Spring (***The Mix-in***) ne permet d'introduire que des méthodes, ça se fait en utilisant un type spécial d'intercepteurs : ***l'intercepteur d'introduction***, cet intercepteur implémente une interface qui contient les méthodes introduites, et délègue les autres appels de méthodes à un autre intercepteur ou à l'objet intercepté. Spring AOP implémente l'API AOP Alliance, et même si la majorité de ses fonctionnalités sont introduites par le biais d'un fichier de configuration, il est possible aussi de personnaliser le modèle AOP par programmation, ou même mieux : l'étendre.

Exemple : Injection d'un greffon avant le calcul de delta dans le composant *CmpEquationSecondDegre avec SpringAOP*

La configuration des aspects avec SpringAOP suit les étapes suivantes :

D'abord on déclare un objet :

```
<bean id="Target" class="Test.EquationSecondDegre.EquationSecondDegre" />
```

Puis on déclare on lui associe un intercepteur

```
<bean id="CmpEquationSecondDegre"
class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="target">
                <ref local="Target" />
        </property>
        <property name="interceptorNames">
                <value>tracingBeforeAdvisor</value>
        </property>
</bean>
```

A la fin on associe une coupe à l'intercepteur :

```
<bean id="tracingBeforeAdvisor"
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">

        <property name="advice">
                <ref local="theTracingBeforeAdvice" />
        </property>
        <property name="pattern">
                <value>Test.EquationSecondDegre.EquationSecondDegre.calculdelta
                </value>
        </property>
</bean>
```

Les classes cités dans l'exemple sont implémentées en Java et les beans sont créés et configurés par le conteneur de beans de SpringAOP à partir de ce fichier descripteur.

5.5. Configuration de l'application E-APC : Service d'état civile

eAPC est une application web développée en Java, sa conception suit le modèle MVC (Model View Control) avec Struts 2.0. Elle a besoin d'un conteneur de servlets dans notre cas on a utilisé Apache Tomcat pour l'héberger. Elle utilise MySQL comme Système de gestion de bases de données.

5.5.1 Struts 2.0

Apache Struts est un Framework libre qui facilite le développement d'applications web J2EE. Il utilise et étend l'API Servlet Java afin d'encourager les développeurs à adopter l'architecture Modèle-Vue-Contrôleur.

5.5.1.1. Fonctionnement

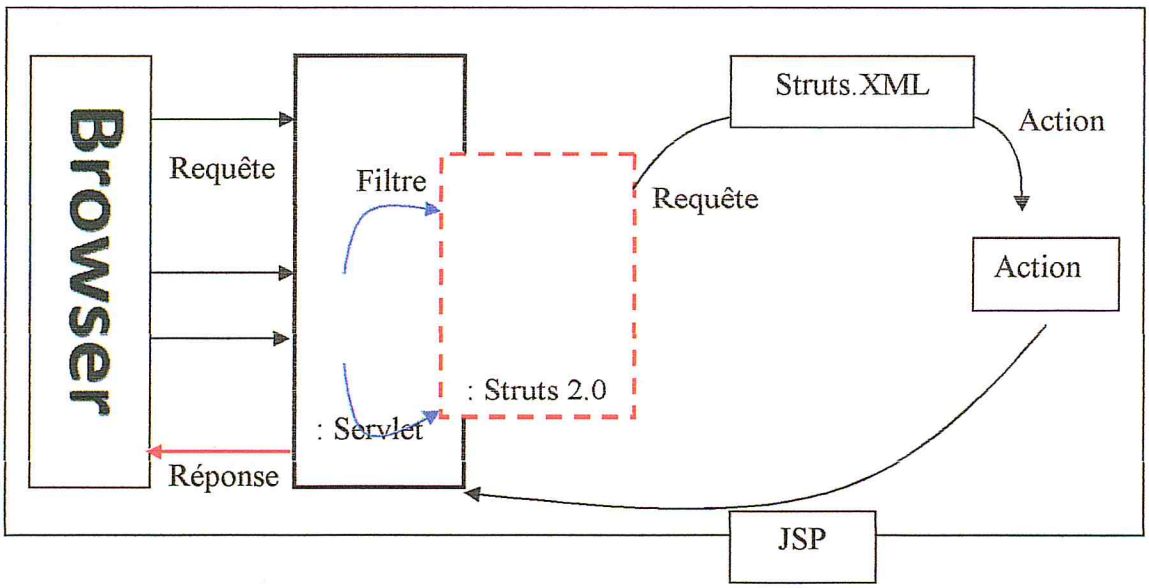


Figure 5.3: Chemin d'une requête dans Struts2

5.5.2. Le Modèle MVC

Le diagramme de collaboration ci-dessous donne un aperçu rapide de l'architecture technique à mettre en œuvre afin de respecter le Modèle Vue Contrôleur (MVC).

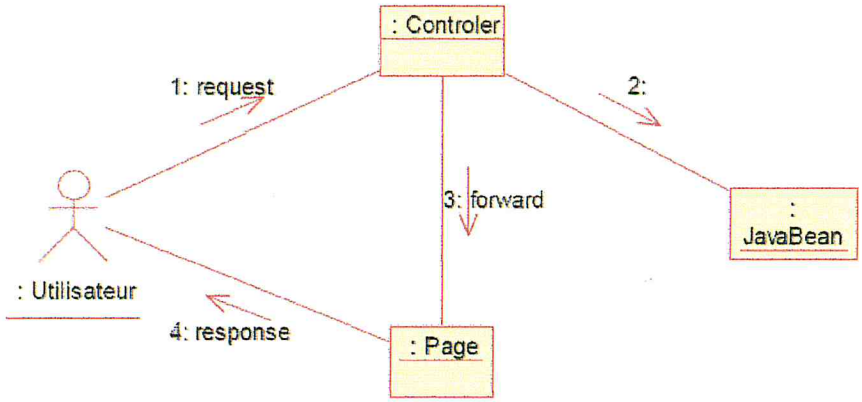


Figure 5.4. Le modèle MVC

1. L'utilisateur émet une requête auprès du serveur.
2. Le contrôleur accède aux informations demandées au travers d'un JavaBean.
3. Le contrôleur transmet le JavaBean à la page JSP chargée de la mise en forme du document.
4. La page JSP met en forme les informations et retourne la réponse à l'utilisateur.

5.6. Présentation de l'application E-APC : Service d'état civile

5.6.1. Installation du Système :

L'installation du système passe par 4 étapes Importantes (Figure 5.5)

Etape 1 : Configuration de la base de données :

- Spécifier le nom d'utilisateur.
- Spécifier L'adresse IP du SGBD.
- Spécifier le mot de passe pour accéder a la base de données
- Spécifier le port pour la connexion au serveur.

Etape 1 :

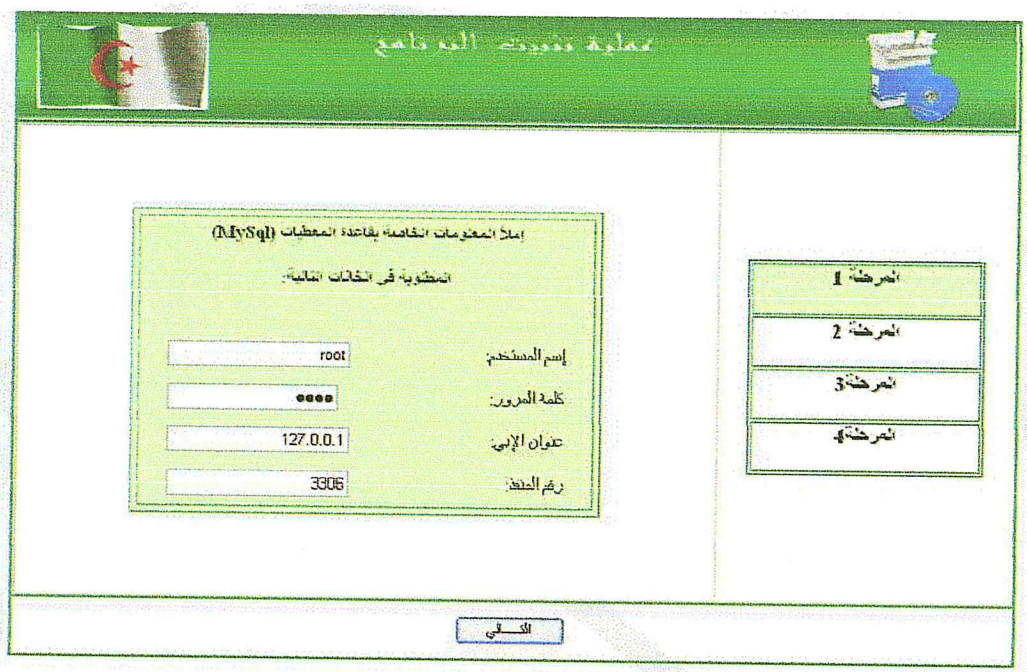


Figure 5.5. Etape 1 de l'installation

Etape 2 :

Cette étape consiste à choisir la commune sur lequel le logiciel sera installé

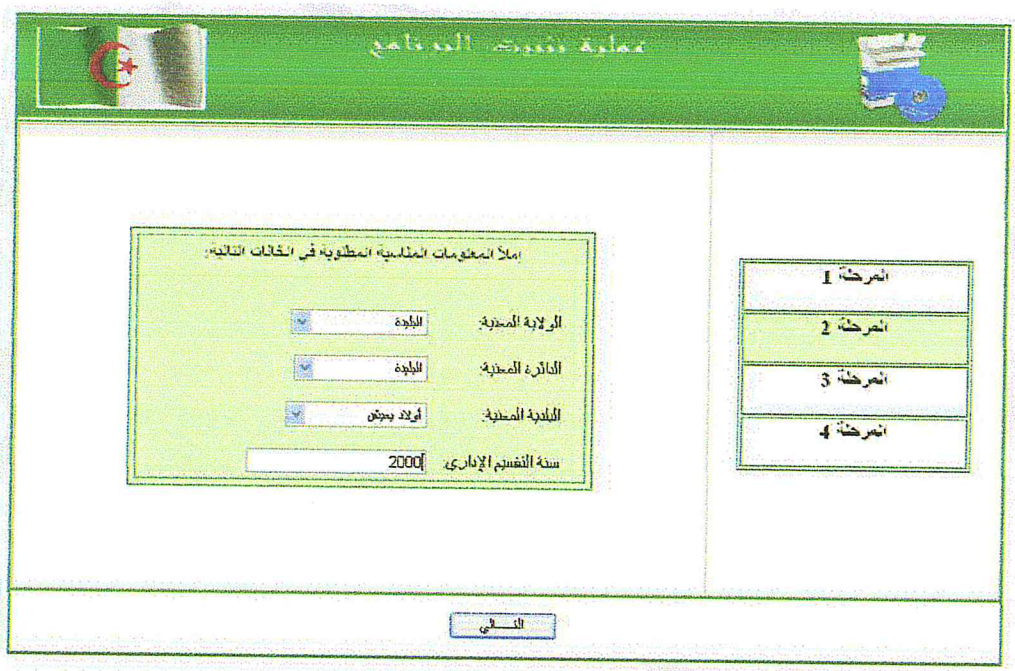


Figure 5.6. Etape 2 de l'installation

Etape 3 : Dans cette étape l'administrateur doit créer son compte

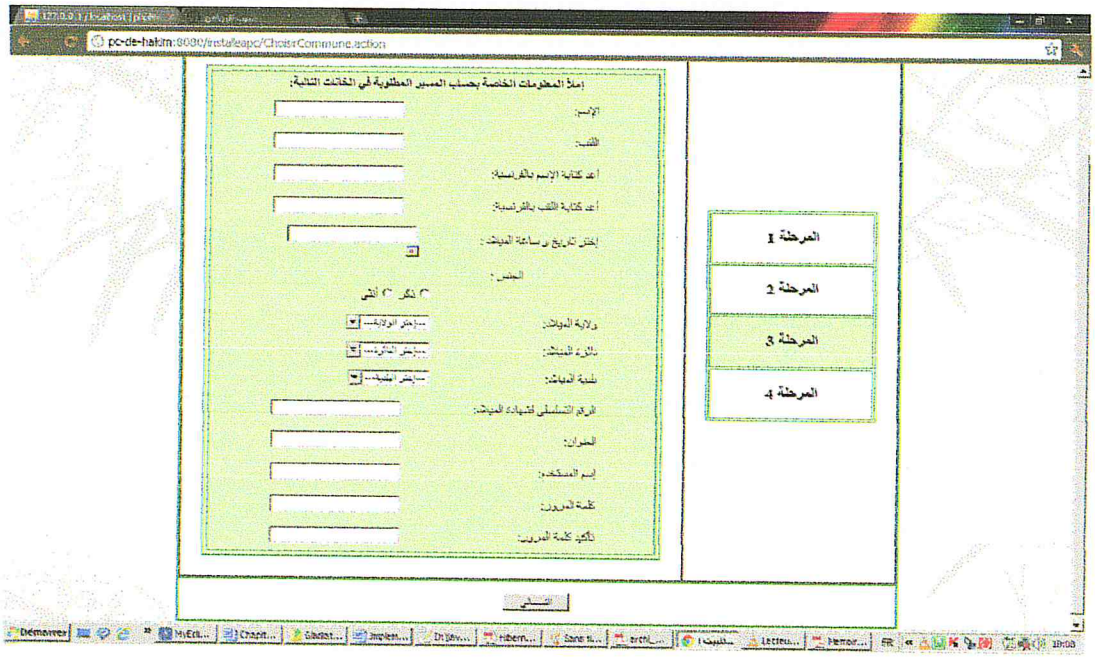


Figure 5.7. Etape 3 de l'installation

Etape 4 :

Dans cette étape l'administrateur doit un compte pour l'OEC principal de la commune courante.

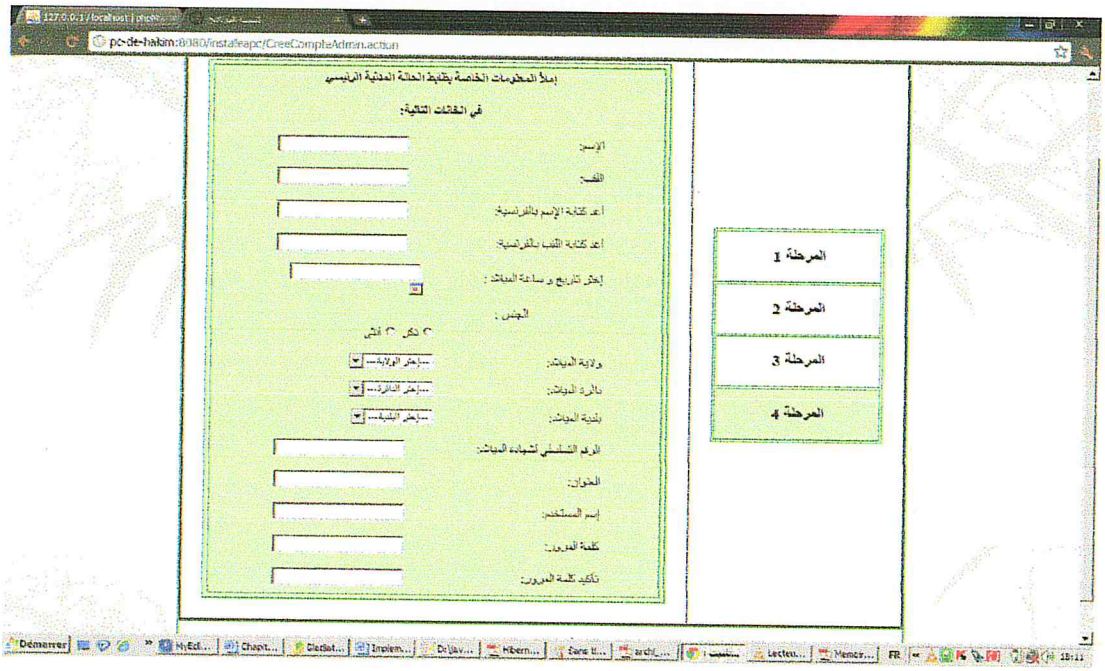


Figure 5.8. Etape 4 de l'installation

5.6.2. Création d'un compte utilisateur

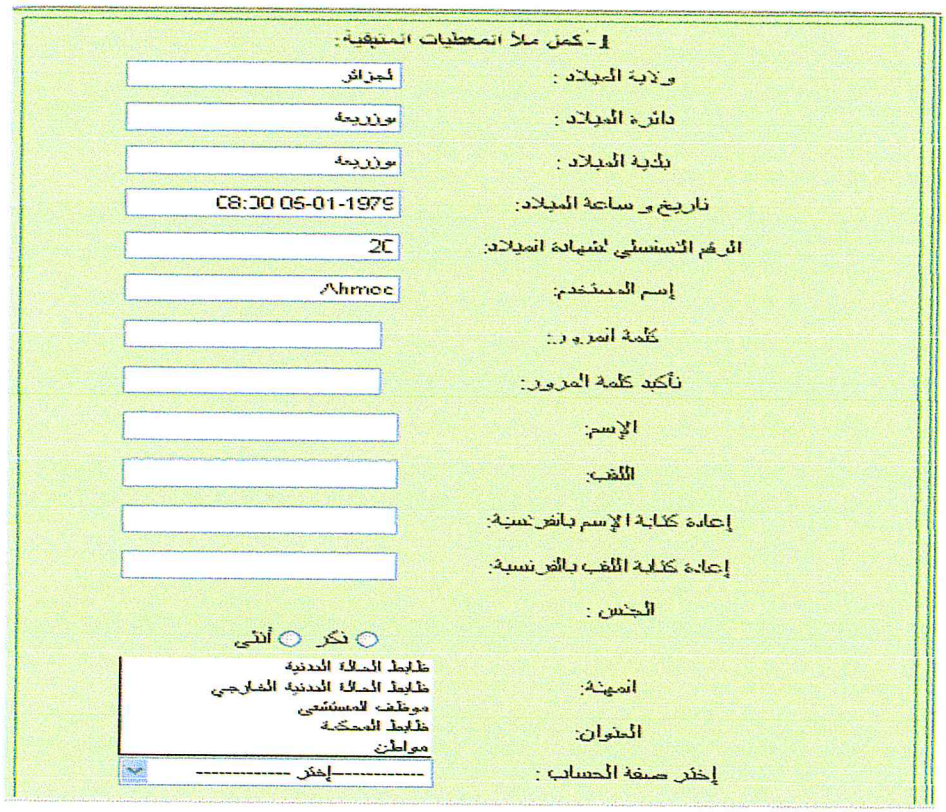


Figure 5.9. Création d'un compte utilisateur

5.6.3. Déclaration d'un événement

La déclaration comporte quatre événements : Mariage, Décès, Naissance, Divorce

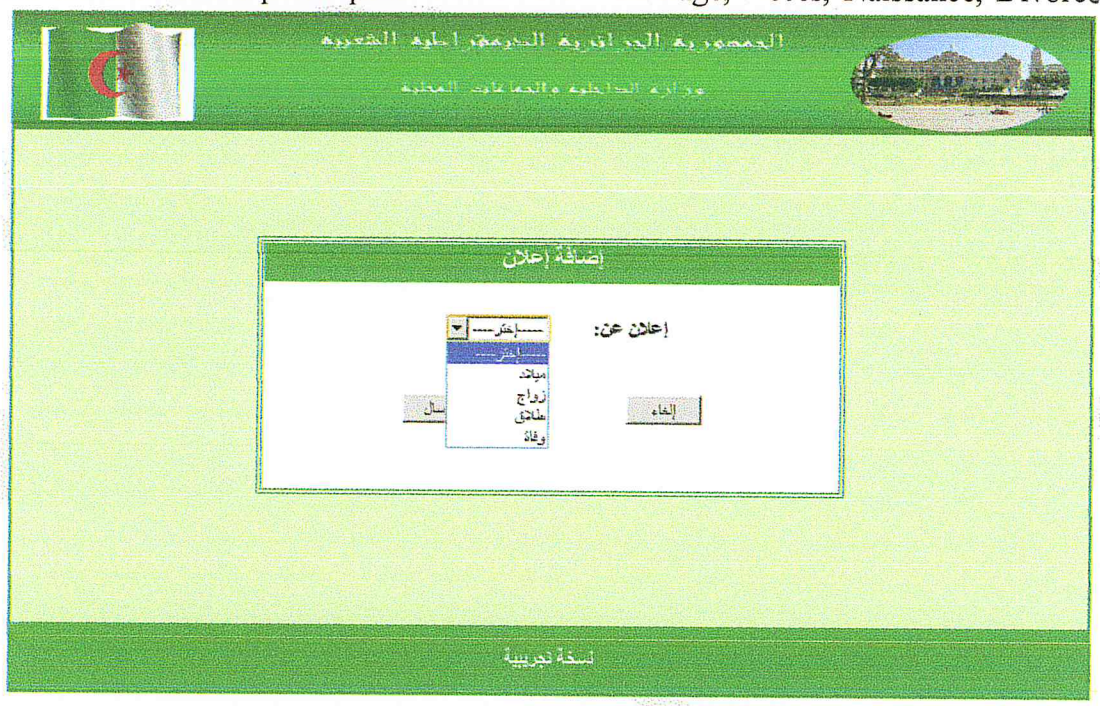


Figure 5.10. Déclaration d'un évènement

Figure 5.11. Déclaration d'un nouveau né

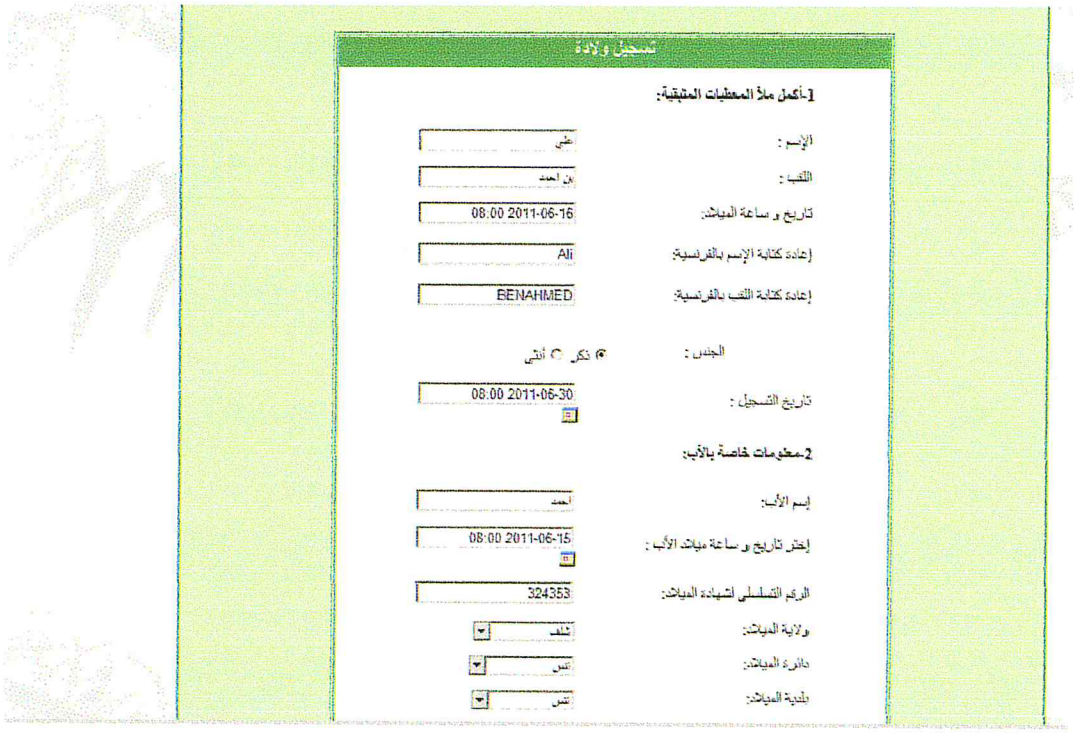


Figure 5.12 Declaration d'un nouveau né

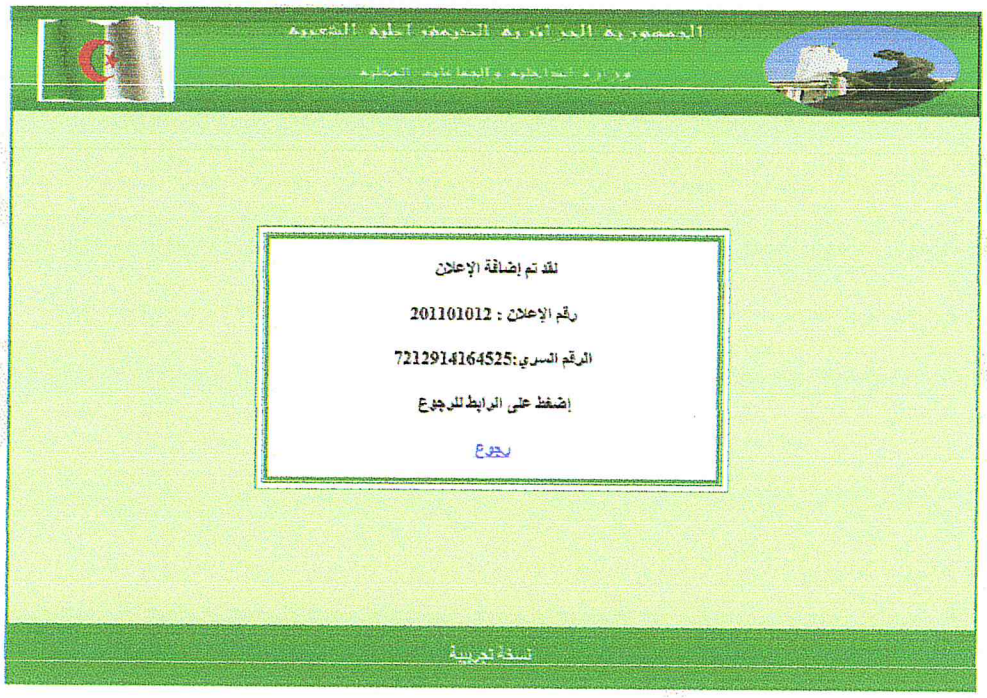


Figure 5.13 Résultat de la Déclaration d'un nouveau né

5.7 Conclusion et perspectives

Nous avons présenté dans ce travail, une conception orientée aspect par composants de l'application eAPC. On a utilisé le modèle IASA comme modèle de composants, et on a présenté une technique d'assemblage des composants qui se voit orientée aspects.

En effet, cette technique nous a permis d'assembler les composants qu'on a déterminés. L'intégration de SpringAOP nous a permis d'isoler les fonctionnalités transverses (dans notre cas l'authentification) et de les introduire à certains points de l'application sous forme d'aspects. L'application qui a été sujet de test pour cette technique fonctionne parfaitement, avec une facilité de mise à jour et une meilleure réutilisation des composants qu'on a conçu le plus fin possible pour la robustesse de l'application.

La réussite de cette technique, nous ouvre d'autres perspectives pour continuer le travail plus loin parmi lesquels on peut citer la création d'un outil de développement rapide par assemblage de composants préfabriqués et testés. Ceci facilitera la création de lignes de produits concernant différents domaines. Le développement d'applications avec l'approche ***lignes de produits,*** principalement basé sur la réutilisation des composants testés et prouvés, permet de gagner beaucoup de temps et d'efforts.

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 de l'orienté aspect. Nous espérons que notre travail aura été à la hauteur de vos espérances.

BIBLIOGRAPHIE

[MS98] Mikhajlov, L. and Sekerinski, E.: A study of the fragile base class problem. Lecture notes in computer science, 1998.

[SW09] W. Seddaoui : Conception et réalisation selon une approche MVC mettant en œuvre le Framework Struts2, d'un composant représentant l'état civil d'une APC, PFE d'Ingénieur d'état, 2009, Université Saad Dahleb de Blida.

[Mey99] Meyer, B. Onto components. In IEEE Computer, Volume 32, January 1999.

[TOHS99] Tarr, P., Ossher, H., Harrison,W., Sutton S.,M. : N degrees of separation: Multi-dimensional separation of concerns. In proceedings of international conference on software Engineering (ICSE), ACM Press 1999.

[PSR05] Renaud Pawlak, Lionel Seinturier et Jean-Philippe Retaillé : Foundations of AOP for J2EE Development. Apress, 2005.

[Szy98] Szyperski, C. *Component software beyond object oriented programming.* Addison Wesley, 1998.

[KC07] S. Krakowiak T. Coupaye Intergiciel et Construction d'Applications Réparties, 2007.

[BW96] Brown, A.W. and Wallnau K.C. *Engineering of Component-Based Systems.* Proceedings of the 2nd IEEE International Conference on Complex Computer Systems, October 1996.

[SG93] D. Garlan et M. Shaw, « *Introduction to Software Architecture* ». Advances in Software Engineering and Knowledge Engineering. World Scientific Publishing Company, 1993.

[SG96] Shaw M. and Garlan D. *Software Architecture - Perspectives on an Emerging Discipline*, Prentice Hall, ISBN: 0-13-182957-2, 1996.

[BHS09] D.Bennouar, A.Henni, A.SaadiTHE DESIGN OF A COMPLEX SOFTWARE SYSTEM USING A SOFTWARE ARCHITECTURE APPROACH,2009.

[BO05] Barais, O. Construire et Maîtriser l'Evolution d'une Architecture Logicielle à base de Composants, thèse de doctorat à l'université des Sciences et Technologies de Lille, 2005.

[ACN02] J. Aldrich, C. Chambers, and D. Notkin. ArchJava : connecting software architecture to implementation. In *Proceedings of the 24th International Conference on Software Engineering (ICSE-02)*, pages 187–197, New York, mai 19–25 2002. ACM Press.

[MDEK95] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures.*Lecture Notes in Computer Science*, 989, 1995.

[WEB01] javaworld.com --- "I want my AOP" series, article par Ramnivas Laddad, 2002.