Utilisation des services et des aspects pour la réutilisabilité du logiciel d'un automate pour l'analyse de plasma

Assia AIT ALI SLIMANE, Muhammad Usman BHATTI

Université Paris 1 Panthéon Sorbonne
Centre de Recherche en Informatique
90 rue de Tolbiac
75013 Paris, France
{assia.ait-ali-slimane,muhammad.bhatti}@malix.univ-paris1.fr

RÉSUMÉ. Le but principal de l'ingénierie logicielle est la qualité logicielle. Les approches traditionnelles telles que les approches objet fournissent un support important de décomposition permettant d'offrir la réutilisation. Cependant, quelques préoccupations, appelées préoccupations transverses dans la terminologie orientée aspect, entravent la réutilisation des modules. Il existe plusieurs travaux de recherche relatifs à la programmation orientée aspect (POA), mais il y a un besoin important d'aligner la réalité industrielle aux abstractions de recherche concernant la POA, en présentant des études de cas concrètes. Ce papier est un retour d'expérience d'un cas industriel relatif au logiciel d'un automate analysant le plasma sanguin. Ce papier montre le code transverse, relatif aux patrons conceptuels, aux connecteurs architecturaux, et aux préoccupations fonctionnelles et non fonctionnelles, qui entravent la réutilisation et l'évolution du logiciel de l'automate. Pour une meilleure modularité, nous proposons d'opter pour une décomposition en service, et une encapsulation des préoccupations transverses dans des aspects, motivant le besoin d'adopter la POA, et les services dans les systèmes industriels.

ABSTRACT. Software engineering advocates decomposing software into separate modules for better reusability. Certain functional and non-functional properties in complex software systems cannot be encapsulated using traditional approaches such as functional decomposition, design by contract, and encapsulation. The Aspect-oriented paradigm attempts to solve this problem by modularizing functionality that commonly pertains to multiple functional units, also called cross-cutting concerns. Supporting the case of the AOP paradigm, languages and techniques have been proposed to align new systems with the paradigm. But there is a need to align the ground realities with research abstractions for AOP by presenting concrete case studies. This industrial experience paper evaluates the software written for an industrial system using the traditional object oriented techniques. In this paper we present crosscutting concerns found dispersed or "lurking" in the system as an obstacle for its reuse and evolution. This case study shows the crosscutting code related to the design patterns, architectural components, and functional and non-functional concerns of the software to motivate the need for the adoption of AOP in industrial systems. We also depart from the notion of non-functional concerns and identify concerns such as display components, display policies, parameter reading and architectural-related issues to be crosscutting and the need to encapsulate them in separate modules for better modularity. For better modularity, reuse and evolution, we propose an approach based on service decomposition and aspects.

MOTS-CLÉS : *préoccupation transverse, programmation orientée aspect, services, réutilisation, évolution.*

KEYWORDS: *crosscutting concern, aspect oriented programming, services, reuse, evolution.*

1. Introduction

Le but principal de l'ingénierie logiciel (plus particulièrement dans les systèmes critiques comme celui de notre étude de cas) est la qualité logicielle. L'orientation objet fournit un support important pour améliorer la qualité logicielle, grâce à l'encapsulation, le polymorphisme, l'héritage et la délégation.

La qualité logicielle n'est pas une idée aussi simple ; elle est vue comme une notion à multiples facettes, décrite par un ensemble de facteurs ou de critères [Meyer88]. Les facteurs clés pouvant influer la qualité logicielle sont la compréhensibilité et la réutilisabilité.

La compréhensibilité consiste à comprendre aisément un module logiciel indépendamment du reste du logiciel. La réutilisabilité consiste à produire des modules logiciels cohérents, autosuffisants pouvant être réutilisés à tout moment sans avoir à les recréer à zéro (*from scratch*).

Nous participons à la réalisation d'un logiciel d'automate pour l'analyse de plasma sanguin au sein de l'entreprise STAGO. L'objectif de l'entreprise est d'améliorer la qualité de ses logiciels, et plus particulièrement de réutiliser le logiciel d'un automate pour en fabriquer de nouvelles gammes.

Pour améliorer la qualité des logiciels, l'Architecture Orientée Service (AOS) [Papazoglou03] est proposée comme étant le style architecturel adéquat. Les services sont les éléments fondamentaux de conception d'une AOS. La caractéristique principale des services est leur nature de faible couplage qui minimise les dépendances entre les modules logiciels, facilitant la maintenance, la réutilisation, et l'évolution.

Cependant, la séparation des préoccupations en utilisant la dimension « service » implique des préoccupations transverses, résultat de la tyrannie de la décomposition dominante [Tarr99], appelées les préoccupations transverses [Kiczales97]. Ces dernières représentent des préoccupations qui sont *scattered* et *tangled* [Berg05] à travers les multiples services composant le logiciel, entravant la réutilisation des modules logiciels. Le développement logiciel orienté aspect [DLOA] est un ensemble bien adapté de techniques permettant de gérer les préoccupations transverses en les regroupant dans des unités de code, appelées aspects.

Ce papier est de contribution multiple, à savoir: (i) présenter une étude de cas relative au domaine médical : logiciel de l'automate qui analyse le plasma sanguin, (ii) diagnostiquer le logiciel de l'automate dans le but de mettre en évidence les préoccupations transverses qui entravent la réutilisation des modules du logiciel, et (iii) proposer une approche de développement logiciel à base de service et d'aspect pour concevoir des modules de petite granularité, facilitant ainsi la réutilisation et la maintenance.

Le reste du papier est structuré de la manière suivante : La Section 2 donne un aperçu de notre étude de cas. La Section 3 met en évidence les préoccupations transverses liées à notre étude de cas, et la Section 4 se focalise sur notre solution. Les travaux reliés sont présentés dans la Section 5 et nous terminons par une conclusion dans la Section 6.

2. Etude de cas : automate pour l'analyse du plasma sanguin

L'automate est composé de deux partie principales : la partie *hardware* concernant la mécanique et l'électronique comme les bras, les tiroirs, et la partie *software* gérant le *hardware* dans le but d'analyser le plasma sanguin. L'utilisateur de l'automate (opérateur, biologiste, etc.), après authentification, charge un ou plusieurs tubes de plasma, ainsi que des produits, dans les tiroirs de l'automate, associe un test d'analyse à chacun des tubes, et lance l'analyse. Le système d'information de laboratoire (LIS) communique avec l'automate pour lancer des analyses et/ou récupérer des résultats d'analyse. Il est possible de demander des résultats via divers medias de communication tels que téléphone mobile, PDA, et l'internet. Le logiciel doit pouvoir afficher une interface homme machine (IHM) selon le profil de l'utilisateur. L'accès à l'automate est protégé des mauvais usages grâce aux mécanismes de sécurité. Le logiciel de l'automate gère l'authentification, la persistance, les mécanismes de transaction, etc.

Dans le but de supporter l'évolution et la réutilisation successive des automates, nous proposons d'utiliser les services afin de promouvoir une architecture flexible. Cependant diverses préoccupations telles que l'authentification, et la gestion des exceptions sont des préoccupations transverses entravant la réutilisation du logiciel.

La section suivante présente quelques préoccupations transverses rencontrées lors de l'analyse du logiciel de l'automate.

3. Les préoccupations transverses

Nous tenons à préciser que vu l'importance du logiciel de l'automate et de sa complexité (environs 1000 classes) nous n'avons pas analysé le logiciel en entier, mais nous nous sommes intéressé à quelques parties dont le métier de l'automate (analyse plasma sanguin et les données associées), et la partie interface homme machine (IHM). Le logiciel de l'automate est implémenté selon le paradigme objet en utilisant le langage C#.

Plusieurs artéfacts d'un logiciel, tels que la spécification des exigences, l'architecture, les composants, et le code, peuvent faire l'objet de réutilisation. Dans notre cas, nous optons pour la réutilisation des composants ou des classes. De la décomposition par objet (d'autres approches sont aussi concernées par cette problématique) résulte les préoccupations transverses. Kiczales et al ont étudié l'existence du code transverse, et considèrent la technique des aspects, à savoir la programmation orientée aspect (POA), comme étant la solution permettant de bien gérer les préoccupations transverses [Kiczales97]. La POA permet de mettre en facteur certaines fonctionnalités dont la réalisation est dispersée à travers le code du logiciel.

Il existe trois types de préoccupations transverses: A transverse B correspond à trois éventualités, à savoir : i) A et B sont des préoccupations métiers, ii) A et B sont de préoccupations non fonctionnelles (techniques), et iii) A est une préoccupation métier, et B une préoccupation non fonctionnelle [Rosenhainer04].

Au départ, notre intérêt s'est porté sur le code transverse liés aux préoccupations non fonctionnelles, mais d'autres préoccupations liées à l'architecture du logiciel sont apparues. La réutilisabilité est aussi entravée par un autre aspect important qui est la hiérarchie des classes: lorsqu'une classe est réutilisée, ses super classes et ses sous classes doivent être étudiées également afin de comprendre le comportement global du module comportant ces classes : c'est l'effet Yo-Yo qui entrave les efforts de réutilisation des modules logiciel [Gamma95].

Afin d'évaluer réellement la réutilisabilité du logiciel existant, on a besoin de trois types d'informations : i) l'architecture du logiciel global à savoir les interactions à travers les modules, ii) les préoccupations que nous suspectons comme transverses, et iii) la hiérarchie des classes. Ce papier aborde les premiers deux points.

Nous utilisons les techniques d' *Aspect Mining* [Deursen03] [Kellens05] pour identifier le code transverse. Plusieurs navigateurs tels que *Concern Graph* [Robillard02], *JQuery* [Janzen03], et *Aspect Browser* [Griswold99] facilitent cette tâche. Dans la littérature, il existe trois types d'outils pour observer le comportement transverse :

- Analyse lexicale : Ce type d'outil se base sur la recherche de mots clés suspectés comme préoccupations transverses dans le logiciel. L'hypothèse d'utilisation de l'outil est que le code du logiciel respecte les règles générales de nommage : *Naming Conventions*. *Aspect Browser* appartient à cette catégorie d'outils.

- Analyse de types : Ce type d'outil se base sur la recherche de types d'objets se trouvant dans le code logiciel. Par exemple, on peut chercher le type bouton pour chercher les variables qui correspondent à ce type. *Aspect Mining* Tool (AMT) [Hannemann01] appartient à cette catégorie d'outil.

- Analyse dynamique : Ce type d'outil observe la trace d'exécution d'un logiciel pour dénommer les préoccupations transverses. *DynAMiT (Dynamic Aspect Mining Tool)* [BreuKrin04] est un outil de ce type qui analyse les traces d'exécution d'un logiciel pour chercher les exécutions récurrentes.

Nous optons pour l'utilisation d'*Aspect Browser* afin de rechercher le code transverse. L'outil *Aspect Browser* est choisi pour les raisons suivantes:

- C'est un outil qui permet de chercher dans les fichiers C# et ne dépend pas du compilateur JAVA,

- Il permet de rechercher des mots clés, et offre une bonne présentation des résultats. En plus, le logiciel en question a été écrit en suivant les règles des nommage : *Naming convention*,

- Nous avons testé plusieurs outils d'*Aspect Mining,* mais nous n'avons pas trouvé de support pour la langage C#.

Nous présentons dans ce qui suit les diverses fonctionnalités transverses que nous avons recensé. Il faut savoir que la liste des fonctionnalités transverses n'est pas exhaustive, vu l'ampleur du logiciel de l'automate.

3.1 Réutiliser le module IHM authentification

Dans un premier temps, nous présentons un cas simpliste de réutilisation du module pour interface d'authentification. L'utilisateur est authentifié au système en fournissant son nom utilisateur et son mot de passe. Si l'utilisateur fournit le bon nom utilisateur et mot de passe, l'application est lancée selon le profil de l'utilisateur. Ce module contient trois classes:

- Accueil : cette classe est utilisée pour afficher l'interface.
- Erreur : cette classe affiche une boite de dialogue concernant l'erreur.
- IHMControleur : cette classe encapsule la logique concernant l'interaction entre la couche présentation et la couche données du logiciel.

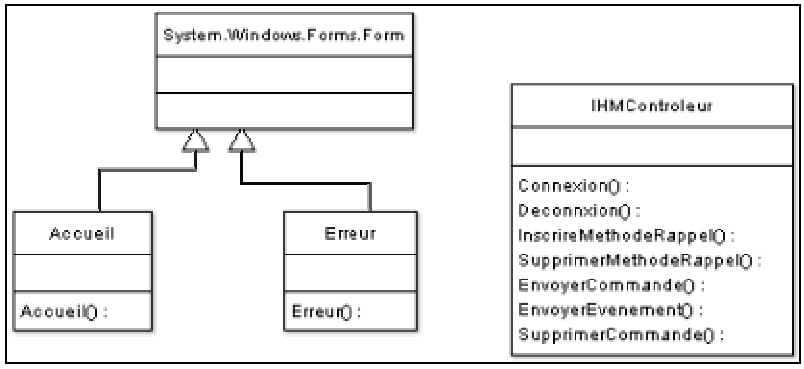

Figure 1. *Diagramme de classe du module IHM authentification.*

3.1.1 Les obstacles rencontrés

Nous souhaitons réutiliser le module interface authentification avec les exigences suivantes, à savoir :

- ne pas réutiliser la librairie graphique appelée *Infragistics*, car la licence d'utilisation de la librairie est payante.
- ne pas réutiliser l'architecture du logiciel (voir Figure 2), basée sur le paradigme client serveur. Le module IHM authentification est le client. Il envoie des requêtes sous forme d'évènements et de commandes au serveur. Ce dernier prend en charge les requêtes, les traite (il existe aussi des modules de traitement des modules client), et renvoie des évènements au client (le résultat des requêtes).
- séparer les diverses préoccupations pour rendre le module plus flexible. Toutes les préoccupations présentées précédemment seront décrites ultérieurement.

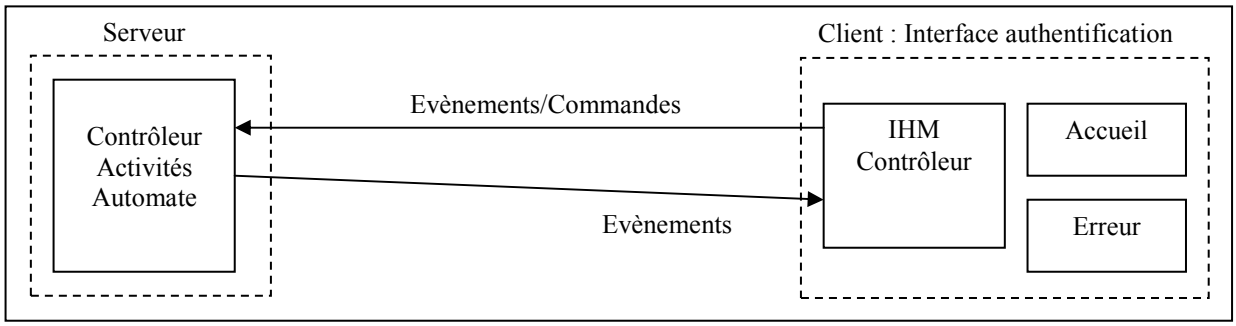

Figure 2. *Architecture Client Serveur du module authentification.*

Les différentes préoccupations relatives aux exigences citées préalablement sont recherchées avec l'outil *Aspect Browser*. La Figure 3 montre le résultat de la recherche, dans le module IHM authentification (voir section 3.1), des mots clés :

- Infragistics : Concernant la librairie de l'affichage
- Evenement : Concernant la préoccupation liée à l'architecture du système. Le module IHM authentification communique avec le serveur via les événements.
- Parametres : Concernant la lecture des paramètres du logiciel sauvegardés dans des fichiers XML.

Infragistics - Le logiciel utilise pour sa couche présentation les propriétés des contrôles graphiques tels que les boutons, les libellés,…etc des librairies *Infragistics*. La recherche du mot clé *Infragistics* dans le module authentification montre la transversité de la librairie à travers les classes *Accueil.cs* et *Erreur.cs* (couleur orange dans la Figure 3).

La transversité de *Infragistics* ne concerne pas que le module IHM authentification, mais concerne toutes les classes de la couche présentation (l'interface du logiciel)). Par conséquent, avant de décider de réutiliser un module, il faut soit reprendre la librairie *Infragistics*, soit utiliser ses propres contrôles nécessitant d'effectuer des modifications intrusives du code source du logiciel.

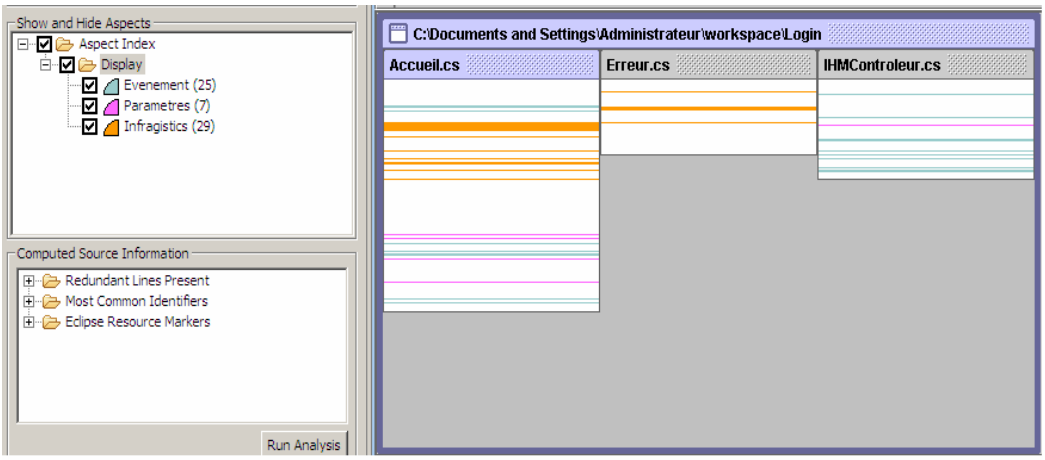

Figure 3. *Résultat de recherche de Aspect Browser dans le module authentification.*

Parametres - Le logiciel utilise des fichiers comme moyen de paramétrage. Les paramètres sont mentionnés dans des fichiers XML. Avant la réutilisation des modules affectés par le paramétrage, une décision de conception doit être établie si les paramètres doivent être introduits par une interface, sauvegardées dans un fichier XML, ou codés en dur dans le logiciel (couleur rose dans la Figure 3).

Une vision plus détaillée révèle dans la Figure 4 l'usage de *parametres* dans la classe *Accueil*.cs dans le but de lire le paramètre de configuration de l'interface, et l'affichage des boutons. Cette préoccupation transverse est à prendre en considération si nous souhaitons réutiliser le module.

```
private void BTN_BasculerDroiteGauche_Click(object sender, System.EventArgs e)
       {
         string Gaucher = "";
       Parametres.LireCleConfiguration("IHM2.exe.config","ModeGaucher",out Gaucher);
       Parametres.EcrireCle("ModeGaucher",(!Convert.ToBoolean(Gaucher)).ToString(),Path
.Combine(Parametres.CheminAppli,"IHM2.exe.config"));
       BasculerDroiteGauche(!Convert.ToBoolean(Gaucher));}
```

```
private void BTN_Valider_Click(object sender, System.EventArgs e)
             {string utilisateur = TXT_Utilisateur.Text;
             string motDePasse = TXT_MotDePasse.Text;
             string IHM;
             Parametres.LireCleConfiguration("IHM",out IHM);
             string IHMSAV;
             Parametres.LireCleConfiguration("IHMSAV",out IHMSAV);}
```

Figure 4. *Code associé à Parametres*

Evenement - Les différents modules communiquent entre eux par des évènements selon l'architecture client serveur décrite préalablement. La logique d'événements est associée au patron de conception *Observer* [Gamma95] où le sujet de changement (dans notre cas l'IHM) avertit les observateurs (le contrôleur) du changement ou des événements. La préoccupation transverse relative à la communication entre modules est montrée en couleur verte dans la Figure 3. La Figure 5 et la Figure 6 montrent un exemple d'évènement relatif à l'arrêt de l'automate.

La logique évènement est mélangée à la logique de la classe *Accueil*.cs. Ainsi, les préoccupations architecturales sont embarqués et éparpillées dans le code du logiciel entravant ainsi sa réutilisation.

```
private void BTN_Arreter_Click(object sender, System.EventArgs e)
            {if
(OKAnnuler.Instance.ShowDialog(this,MultiLangue.LireRessourceLocale(typeof(OKAnnuler)).T
raduire("ConfirmationSortie")) == DialogResult.OK)
                  {
      // envoie de l'évènement de demande d'arrêt de l'automate
      Evenement evt = new Evenement(EvenementsDef.DemandeArretLogiciel);
      evt.Type = Evenement.EvenementType.DIFFUSION;
      mIHMControleur.EnvoyerEvenement(evt);
                        TimeSpan tempArret=TimeSpan.Zero;
                        DateTime HeureFinAnalyses =
mIHMControleur.mFacade.HeureFinAnalysesEnCours();}}
```

Figure 5. *Code associé à Evenements.*

```
private void BTN_ArretUrgence_Click(object sender, System.EventArgs e)
            {
                  // signalement de l'arrêt d'urgence
mIHMControleur.EnvoyerCommande(Commun.Constantes.EvenementsDef.ArretUrgence);
      if (IHMCommun.AfficherMessages.EcranArretUrgence.Instance.ShowDialog() ==
DialogResult.OK)
      {
mIHMControleur.LancerEvenement(new Evenement(EvenementsDef.DemandeSortieArretUrgence));
mIHMControleur.LancerEvenement(new Evenement(EvenementsDef.DemandePassageSAV));
                        IHMControleur.Instance.mFacade.LancerEvenement(new
Evenement(EvenementsDef.DemandeSortieSAV)); } }
```

Figure 6. *Code associé à Evenements.*

3.2 Module Métier

Le module métier consiste en un ensemble de classes fournissant des services dont le S*ervicePatient.cs* (ajout et manipulation des données patient), le *ServiceTest.cs* (données des testes d'hémostase), et le *ServiceResultat.cs* (données résultats). Actuellement, chaque service est implémenté par une classe.

Idéalement, chaque classe implémente une préoccupation, afin de pouvoir réutiliser les classes aisément sans affecter les autres modules.

Pour le module métier, on exécute une analyse lexicale pour calculer les indices des divers identificateurs présents dans le code. *Aspect Browser* tente de calculer le nombre de lignes redondantes ainsi que le nombre de divers identificateurs présents dans le code. À partir de ces informations et grâce à notre connaissance du logiciel, nous tentons de déterminer les aspects en fournissant l'information concernant les identificateurs au système de recherche de l'outil *Aspect Browser*. Les Figure 7, 8, et 9 représentent les mots clés recherchés ainsi que les résultats de la recherche par *Aspect Browser*.

- les préoccupations concernant la gestion de transactions (Figure 7).
- les préoccupations concernant la recherche de glossaire dans la base de données et la logique d'événement (Figure 8).
- les préoccupations concernant la gestion du cache (Figure 9).

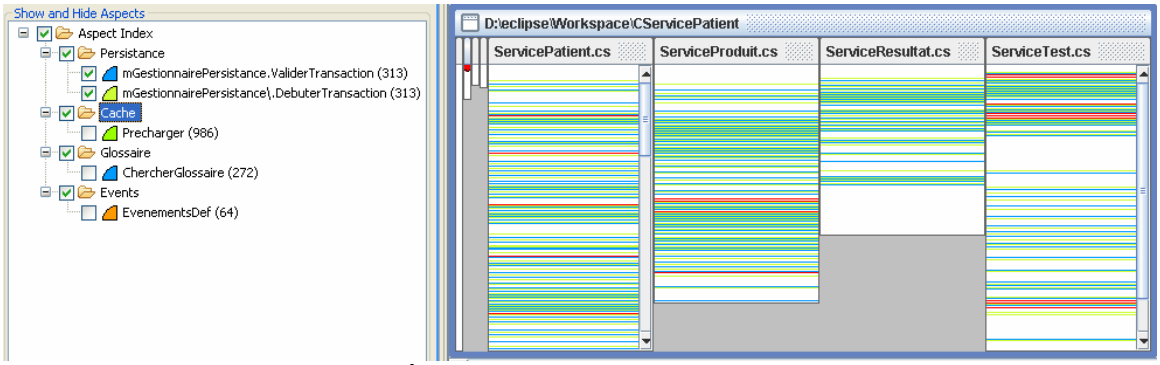

Figure 7. *Préoccupations transverses: Gestion de Transactions*

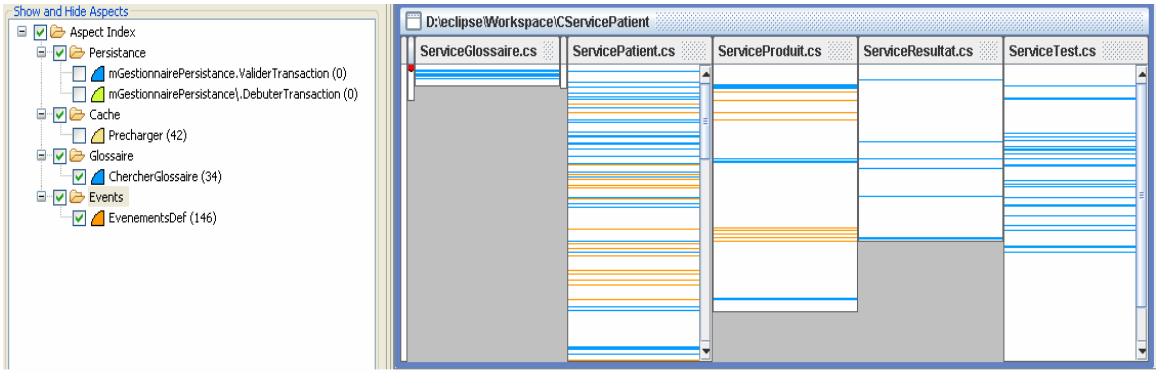

Figure 8. *Préoccupations transverses: Glossaire et Logique d'evenements*

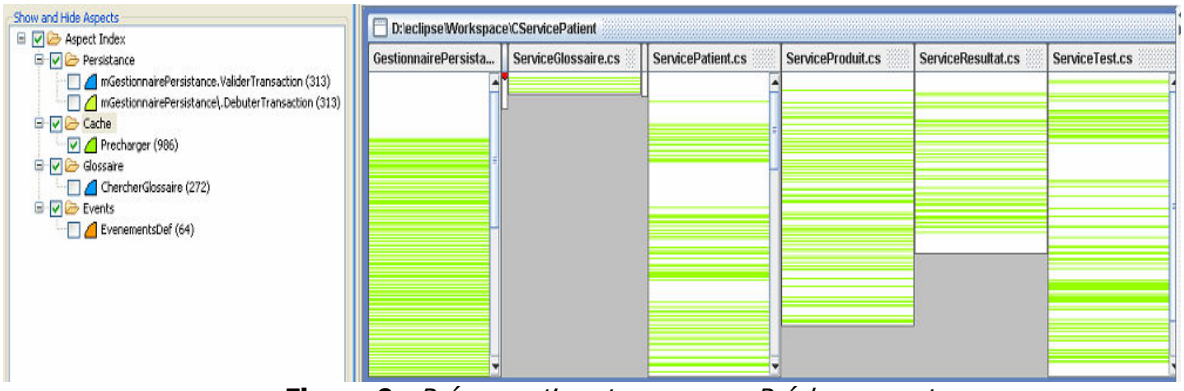

Figure 9. *Préoccupations transverses: Préchargement*

Donc, si nous souhaitons réutiliser la classe *ServicePatient.cs*, il faut aussi réutiliser le gestionnaire de persistance, la gestion du cache, et la gestion de glossaire, comme montré dans la Figure 10.

```
try
       {mGestionnairePersistance.DebuterTransaction();

mDM.QueryCriteria.And(JoinPath.dossierpatient.glossaire_by_etatdossier_glossaireid
.Columns.valeur, "EtatDossier#EnCours",MatchType.Exact);
                          //        dossierpatientCollection dp =
mDM.GetdossierpatientCollection(FetchPath.dossierpatient);
                          //        return dp;
                          System.Int32 glossaireid =
mGestionnairePersistance.AccesServiceGlossaire.ChercherGlossaireParValeur("EtatDos
sier#EnCours").glossaireid;
                          return
mGestionnairePersistance.PrechargerDossiersPatientByetatdossier_glossaireid(glossa
ireid).FilterByetatdossier_glossaireid(glossaireid); }
                    finally
                    {mGestionnairePersistance.ValiderTransaction(false);}
```

Figure 10. *Exemple du mélange de code – trois préoccupations différentes*

3.3 Indentification des préoccupations transverses : Aspect Browser et C#

Comme cité auparavant, l'outil *Aspect Browser* est choisi, parmi d'autres, pour notre exercice d'identification d'aspect pour sa compatibilité avec C#. Nous ne pouvons pas exécuté l'analyse statique du code en question, comme souhaité, en raison du manque d'outil ayant la capacité da faire une analyse statique du code C#. Néanmoins, *Aspect Browser* nous a fourni les indices nécessaires pour découvrir les préoccupations transverses.

L'outil employé fournit un service pour l'analyse lexicale en recherchant les mots clés dans le code. Ceci nous a mené à rechercher le mot-clé qui s'avère transverse en observant le code tels que « evenements » et « parametres ». Le mot-clé « evenements » permet de montrer que la logique d'abonnement aux événements ainsi que la logique de déclenchement d'événements se trouve non seulement dans les modules en question, mais aussi dans les autres modules du système. Alors *Aspect Browser* confirme que le système d'événements est transverse au niveau du logiciel.

Le mot-clé « Infragistics » était connu car le besoin d'enlever tous les éléments graphiques utilisant les contrôles infragistics.

Quant à l'identification automatique des aspects, il est constaté que le calcul des indices de code source, comme fait par *Aspect Browser*, aide à montrer les préoccupations transverses sans entrer dans le détail du code. Une analyse lexicale est exécutée par l'outil pour calculer deux métriques pour le code source en question :

- le nombre de lignes redondantes, et
- les identificateurs répandus.

Bien que tout en calculant les indices de code, les deux métriques portent des informations inutiles tels que les commentaires et les mots-clés comme « new » et « return ». L'information utile concernant les préoccupations transverses se trouve dans les calculs et nécessite un effort manuel pour distinguer les diverses préoccupations. C'est pourquoi, il est possible d'identifier la préoccupation « glossaire » dans le code.

Etant donné que la préoccupation « glossaire » est une exigence fonctionnelle du logiciel, il est difficile de l'identifier comme préoccupations transverses et a échappé à la première observation. Le nombre d'occurrences de l'idenficateur « chercherglossaire » (265) nous a incité à étudier cette préoccupation dans le détail et à constater que les appels aux fonctions de cette méthode sont dispersés dans le code et mélangés avec d'autres préoccupations. La gestion de transactions et le cache étaient faciles à identifier et sont des préoccupations transverses bien connus de la littérature AOSD.

Politique de l'affichage de l'IHM Générale –Il est intéressant de préciser que la politique de l'affichage est trouvée transverse car les métriques de l'outil ont montré un nombre très élevé concernant cette préoccupation. Le logiciel utilise plusieurs politiques d'affichage et nous discutons deux d'entre elles qui sont les plus utilisées: (i) le style des polices pour les contrôles graphiques, et (ii) le mode d'affichage selon les droitiers et gauchers.

Chaque fois que le logiciel affiche un contrôle graphique sur l'écran, son style est changé par un appel au contrôleur de style. Nous avons trouvé 512 lignes dans le code concernant cette préoccupation dans 39 fichiers.

Le logiciel utilise aussi une politique d'affichage pour les personnes gauchères pour leur faciliter l'utilisation du logiciel.

Autant la première préoccupation est bien encapsulée dans un module pour afficher les objets selon le style, la préoccupation pour la prise en charge du mode gaucher n'est pas bien encapsulée et donne un exemple concret des mélange des préoccupation dans le code. C'est grâce à *Aspect Browser* et son système de métrique que ces deux dernières préoccupations sont déduites transverses (14 apparitions dans 10 fichiers). Un exemple de code concernant cette préoccupation (mode gaucher) dans deux fichiers différents est présenté dans la Figure 11.

```
if(IHMControleur.Instance.ModeGaucher)
{
foreach(Control c in this.Controls)
       {
c.Location = new Point((this.Size.Width - c.Location.X -
c.Size.Width),c.Location.Y);
       }
}
```

```
if(IHMControleur.Instance.ModeGaucher)
       {
              this.Location = new Point (1168, 932);
              this.BackgroundImage = pictureBox1.Image;
       }
```

Figure 11. *Préoccupation Mode Gaucher dans deux fichiers différents.*

4. Solution proposée

Nous remarquons que les préoccupations transverses entravent la réutilisation, la maintenabilité, l'évolution, et donc la qualité du logiciel. Nous avons montré, dans le cas d'un logiciel d'automate pour l'analyse du plasma sanguin, les divers types de préoccupations transverses élucidés. Nous proposons d'utiliser les services et les aspects pour remédier à cette problématique.

4.1 Pourquoi les services ?

L'avenir de l'ingénierie logicielle est de construire des systèmes logiciels se basant sur le principe de "service".

L'architecture orientée service promeut la réutilisation des modules logiciels et promet l'interopérabilité entre diverses applications hétérogènes (plateforme et langage différents). Les services sont des fonctionnalités ayant la qualité de faible couplage pouvant être facilement composés pour construire des logiciels. Les services, grâce à leur faible couplage, sont la base de conception de plusieurs systèmes dont le but est la réutilisation et l'interopérabilité. Or, les services souffrent des préoccupations transverses qui sont des propriétés qui transversent les services telles que la persistance, et la sécurité citées préalablement.

4.2 Pourquoi les aspects ?

Le développement logiciel orienté aspect (DLOA) est la paradigme permettant la modularisation des préoccupations transverses dans des unités modulaires appelées aspects. Les préoccupations transverses sont des exigences qui ne peuvent être localisées dans un composant logiciel unique et qui impactent divers composants.

Dans le langage aspect, ces exigences transversent plusieurs composants. Les langages orienté aspect tel que *AspectJ* [Kic01] définissent un aspect avec les spécifications suivantes : (i) *Join Point* correspond à la partie dans le programme affectée par les préoccupations transverses, (ii) *Point Cut* est un ensemble de *Join Points*, et (iii) *Advice* est le code à exécuter avant, après ou autour du *Join Point*. Le tisseur d'aspect (*weaver*) permet de mixer le code aspect avec le code de base afin de générer le nouveau comportement.

4.3 Exigences

Nous avons montré que les préoccupations transverses sont la source d'un code « spaghetti » ; difficile à comprendre, à maintenir, à réutiliser et à faire évoluer. Pour remédier à ça, nous optons pour l'utilisation de paradigmes offrant des modules de petite granularité. Nous proposons certaines exigences permettant d'offrir une meilleure qualité logicielle :

- Utiliser les services comme concept de base pour le développement logiciel.
- Associer aux services le concept aspects pour gérer les préoccupations transverses.
- Utiliser une approche de développement se basant sur les services et les aspects. Un premier travail dans ce sens est présenté dans [AitAliSlimane06]. Les services et les aspects sont gérés depuis l'analyse des besoins jusqu'à l'implémentation, facilitant ainsi la maintenance, la traçabilité, et la réutilisation.

Une approche à base de service et d'aspect permet de réaliser des logiciels ayant une qualité logicielle appréciable, et va permettre à STAGO de composer de nouveaux logiciels d'automate avec beaucoup de flexibilité.

5. Anciens travaux

L'architecture orientée service est considérée comme le paradigme adéquat pour améliorer la réutilisation, et l'évolution des logiciels. Cependant, l'existence des préoccupations transverses limite la qualité logicielle et plus particulièrement la réutilisation des services. Il existe plusieurs travaux de recherche relatifs à l'élucidation des aspects dans le code source logiciel, et à la gestion des aspects dans les services.

Dans ce papier, nous étudions les obstacles concernant la réutilisabilité due aux préoccupations transverses existantes dans le système. Pour cela, nous utilisons les techniques d'élucidation des aspects pour identifier les préoccupations transverses.

On commence par présenter quelques travaux concernant l'identification des préoccupations transverses dans le code logiciel via les techniques de *Aspect Mining*. *L'Aspect Mining Tool* (AMT) [Hannemann01] est l'un des premiers travaux dans ce domaine. AMT est utilisé pour faire une analyse lexicale et typée d'un programme afin de rechercher les préoccupations transverses.

Breu et al. [Breu06] ont tenté d'identifier les préoccupations transverses dans le projet Eclipse en utilisant des techniques d'analyse formelles. Une étude sur l'application de trois techniques d'extraction d'aspect pour identifier des préoccupations présentes dans *JHotDraw* a été réalisée pour étudier les forces d'identification d'aspect et les faiblesses de chaque technique [Ceccato06]. Un outil d'extraction peut ne pas suffire pour identifier les aspects car chacun emploie une méthode différente. Par conséquent, un mélange de différentes techniques est conseillé.

Durr et al. [Durr06] ont essayé de transformer les préoccupations transverses, tels que la gestion des erreurs, en aspect et ont essayé de montrer l'importance de la factorisation en aspect. Ils ont utilisé *Aspect Browser* pour identifier les aspects qui sont supposés connus à l'avance. Il est important de préciser que la majorité des études de cas étudiées, pour identifier les préoccupations transverses, sont relatives au programme *JHotDraw*, et ne concernent pas des études de cas réelles. Dans notre cas, il s'agit d'un cas industriel relatif au logiciel d'automate analysant le plasma sanguin afin de dépister des anomalies d'hémostase. Nous utilisons *Aspect Browser* pour rechercher les préoccupations transverses comme première contribution.

Voici quelques approches qui intègrent les aspects dans les services. Tomaz et al. [Tomaz 06] propose un outil de tissage d'aspect, en implémentant les aspect comme des services faiblement couplés, appelé *Aspect Services Weaver* (ASW), où les aspects sont tissés dynamiquement. Dans cette approche, les aspects sont eux même des services Web, par conséquent ils sont bien indépendants de langages et de plateformes.

Courbis et Finkeltein [Courbis 05], et Charfi et Mezini [Charfi 04] utilisent les aspects à tissage dynamique dans le moteur d'orchestration des services Web en modifiant le *Business Process Modeling Language* (BPML) pour la première approche, et le *Business Process Execution Language for Web Services* (BPEL4WS) pour la seconde, dans le but d'offrir l'adaptabilité métier, telle que l'ajout ou la modification de fonctionnalités.

Les approches citées préalablement se concentrent sur l'intégration des aspects pour améliorer la qualité des services, mais elles focalisent sur la phase d'implémentation ou de conception.

Il est primordial de préciser que le développement logiciel a besoin d'une approche se basant sur les paradigmes « service » et « aspect » comme deuxième contribution. Pour essayer de remplir ce vide, nous proposons une approche orientée service et aspect qui permet de gérer les services et les aspects depuis la phase d'analyse jusqu'à la phase d'implémentation. Le but est de faciliter la maintenabilité, la traçabilité, la réutilisation et l'évolution des composants logiciels.

6. Conclusion

Dans ce papier, nous avons présenté un cas industriel concernant l'automate permettant l'analyse du plasma sanguin.

A cause d'un manque de support concernant les techniques d'*aspect mining* pour le langage C#, nous avons utilisé *Aspect Browser* pour identifier les aspects. *Aspect Browser* exécute une analyse lexicale dans le code source afin d'identifier les indices des préoccupations transverses.

Nous avons mis l'accent sur quelques préoccupations transverses existantes dans le logiciel de l'automate telles que le module d'authentification, les classes métiers, etc. Ces préoccupations transverses entravent la réutilisation des modules.

Afin de permettre le développement d'un logiciel dont les modules sont facilement réutilisables, nous proposons une méthodologie se basant sur les principes de services et d'aspects. Ces derniers sont identifiés et gérés conjointement depuis l'analyse des besoins jusqu'à l'implémentation.

Dans les futurs travaux, nous souhaitons développer une solution concrète pour séparer les diverses préoccupations montrées dans le papier selon le paradigme « services » et « aspects » pour montrer la pertinence de notre approche.

Référence

[AitAliSlimane06] Ait Ali Slimane A., Tomaz R.F., "ASOA: Aspectual Service-Oriented Approach for Developing a Blood Plasma Analysis Automaton", In proceedings of WASP06 workshop at SBES/SBBD 2006, Florianópolis, Santa Catarina Brazil, October 2006.

[AOSD] Site AOSD : http://aosd.net/

[Berg05] Berg, K., Conejero, J.M., "A Conceptual Formalization of Crosscutting in AOSD", In: Workshop on AOSD, Granada, September, 2005.

[BruKrin04] Breu, S. and Krinke, J. 2004. Aspect Mining Using Event Traces. In Proceedings of the 19th IEEE international Conference on Automated Software Engineering (September 20 - 24, 2004). Automated Software Engineering. IEEE Computer Society, Washington, DC, 310-315.

[Breu06] Breu, S., Zimmermann, T., and Lindig, C., "Mining eclipse for cross-cutting concerns", In Proceedings of the 2006 international Workshop on Mining Software Repositories (Shanghai, China, May 22 - 23, 2006), MSR '06, ACM Press, New York, NY, 94-97, 2006.

[Ceccato06] Ceccato, M., Marin, M., Mens, K., Moonen, L., Tonella, P., and Tourwé, T. 2006. Applying and combining three different aspect Mining Techniques. Software Quality Control 14, 3 (Sep. 2006), 209-231.

[Charfi04] Charfi, A., Mezini, M., "Aspect-Oriented Web Service Composition with AO4BPEL", In: Proc. of the 2nd European Conf. on Web Services, LNCS, Vol. 3250, Springer-Verlag, pp. 168-182, 2004.

[Courbis05] Courbis, C., Finkeltein, A., "Weaving Aspects into Web Services Orchestrations", In: Proc. of the 3rd IEEE Int. Conf. Web Services, Florida,USA, IEEE Computer Society Press, July, 2006.

[Deursen03] Deursen A.V., Marin M., and Moonen L., "Aspect Mining and Refactoring", In First International Workshop on REFactoring: Achievements, Challenges, Effects (REFACE), 2003.

[Durr06] Durr P., Gulesir G., Bergmans L.,Aksit M., Engelen R.V., "Applying AOP in an Industrial Context", Workshop on Best Practices in Applying Aspect-Oriented Software Development, March 2006.

[Gamma95] Gamma, Helm, Johnson, and Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[Griswold99] Griswold W. G., Kato Y., and Yuan J. J., Aspect Browser: Tool Support for Managing Dispersed Aspects, Technical Report CS99-0640, Department of Computer Science and Engineering, UC, San Diego, 1999.

[Hannemann01] Hannemann, J. and Kiczales, G., «Overcoming the Prevalent Decomposition of Legacy Code", Workshop on Advanced Separation of Concerns at the International Conference on Software Engineering (ICSE), 2001.

[Janzen03] Janzen D., and De Volder K., "Navigating and querying code without getting lost", In Int'l Conf. on Aspect Oriented Software Development, 2003.

[Kellens05] A. Kellens and K. Mens. "A Survey of Aspect Mining Tools and Techniques". Technical Report 2005-08, INGI, UCL, Belgium,2005.

[Kiczales97] Kiczales G., Lamping J.,Mendhekar A., Maeda C., Videira Lopes C., Loingtier J.M., Irwin J., "Aspect-oriented programming", Proc.11th Europeen Conf. Object-Oriented Programming (Ecoop), Springer, june 1997, pp.220-242.

[Kiczales01] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G, "An Overview of AspectJ", In Proceedings of the 15th European Conference on Object-Oriented Programming, J. L. Knudsen, Ed. Lecture Notes in Computer Science, vol. 2072. Springer-Verlag, London, 327-353, 2001.

[Meyer88] Meyer B., Object-Oriented Software Construction, First Edition. Prentice Hall, 1988.

[Papazoglou03] Papazoglou, M.P. and Heuvel, W., "Service Oriented Computing: State of the Art and Open Research Issues", In: ICSOC, 2003.

[Robillard02] Robillard M.P. and Murphy G.C., "Concern Graph: Finding and describing concerns using structural program dependencies", Proc. 24th Int'l Conf. On Software Eng., ACM Press, 2002, pp. 406-416.

[Rosenhainer04] Rosenhainer L., "Identifying Crosscutting Concerns in Requirements Specifications", Workshop on early: Aspect-Oriented Requirements engineering and Architecture Design, 2004, Vancouver, Canada, October 2004.

[Tarr99] Tarr P.L., Ossher H., Harrison W., Sutton S.M., Jr., "N Degrees of Separation: Multi-Dimensionnal Separation of Concerns", Proc. 21st Int'l Conf. Software Eng. (ICSE), ACM Press, 1999, pp. 107-119.

[Tomaz 06] Tomaz, R.F., Hamida, M.B., Monfort, V.,"Concrete Solutions for Web Services Adaptability Using Policies and Aspects", In: JDIM - Journal of Digital Information Management, September, 2006.