Utilisation de BPEL pour la gestion des processus métier d'une application locale
Étude de cas du calcul des allocations familiales luxembourgeoises

Nicolas Biri*, Pascal Bauler*, Fernand Feltz*, Nicolas Médoc*, Céline Thomase*

*Centre de Recherche Public - Gabriel Lippmann
41, rue du Brill
L-4422 Belvaux
Luxembourg
{nom}@lippmann.lu

Résumé. Nous présentons ici les différentes étapes et techniques de développement choisies pour l'informatisation du traitement des prestations familiales des travailleurs frontaliers du Luxembourg. Après avoir expliqué le cadre, les enjeux et le déroulement global du projet, nous présentons notre utilisation d'un moteur de processus basé sur BPEL et de développement dirigé par des modèles, ces techniques étant utilisées dans le cadre d'un développement agile. A travers le travail réalisé, nous pointons les atouts et les désavantages de l'utilisation de BPEL pour gérer les flux de données d'une application locale ainsi que les enjeux scientifiques intéressants dans ce cadre.

1 Introduction

La création d'un gouvernement électronique (ou e-government) est considérée comme un chantier important par la communauté européenne pour améliorer la qualité de ses services et pour accroître l'intérêt de la population pour l'internet. Si la façade la plus visible d'un gouvernement électronique est la mise à disposition pour les citoyens de portails internet pour gérer leurs demandes administratives, une part nécessaire au bon fonctionnement est également l'automatisation du traitement de ces demandes.

Le sujet est d'autant plus complexe au Luxembourg puisque, par sa taille et son activité économique, les échanges avec les pays voisins sont primordiaux pour la gestion des tâches administratives. En effet, de l'attractivité économique du pays et de son besoin de main d'œuvre résultent la présence de nombreux travailleurs frontaliers. La gestion de leurs droits nécessite naturellement un accord avec l'administration de leur pays de résidence.

Conscient du problème, la CNPF[1] a décidé depuis de nombreuses années de renforcer l'informatisation de son système d'information afin de maîtriser le temps nécessaire à la gestion de ses prestations. Le Centre de Recherche Public - Gabriel Lippmann (CRP-GL) est largement impliqué dans cette restructuration (Bauler et al. 2006, Hitzelberger et al. 2006). Dans cet article, nous présentons la partie de ce projet concernant le calcul des allocations pour les allocataires frontaliers, actuellement en cours de finalisation. Sur le plan technique,

[1] CNPF : Caisse Nationale de Prestations Familiales

cette réalisation est l'occasion de partager un retour d'expérience sur l'intégration d'un moteur de processus BPEL dans une application locale et dans un processus de développement agile.

Après avoir expliqué le contexte du projet, nous présentons les phases de sa réalisation et les difficultés rencontrées pour sa mise en place. Nous abordons ensuite les choix technologiques effectués pour la réalisation du projet (utilisation d'un moteur de workflow, de modèles pour l'aide au développement - MDSD[2]) et en partageant notre expérience sur les bénéfices et les problèmes liés à ces choix.

2 Description du projet

Le projet décrit ici consiste à mettre en place l'automatisation du calcul du montant des allocations familiales devant être payées par la CNPF luxembourgeoise aux travailleurs frontaliers. Afin de comprendre les problèmes posés par ce calcul, nous exposons ici les principes de fonctionnement de ces allocations au sein de la communauté européenne et les spécificités du Luxembourg sur ce point.

Par décision de la communauté européenne, les allocations familiales sont exportables. Cela signifie qu'elles sont disponibles pour chaque personne travaillant au Luxembourg, qu'elle soit ou non résidante. De plus, chaque travailleur doit toucher l'allocation la plus élevée entre celle de son pays de résidence et celle du pays où il travaille. Comme les allocations luxembourgeoises sont majoritairement plus élevées que celles des pays voisins, la situation est en partie simplifiée. Ainsi chaque foyer dont les revenus proviennent exclusivement du Luxembourg peut être considéré comme une famille luxembourgeoise pour le calcul de ses droits. La situation est plus complexe pour un foyer percevant des revenus du Luxembourg et d'un autre pays. Dans ce cas, la procédure actuelle consiste à un règlement mensuel des allocations par le pays de résidence selon son propre barème. Deux fois par an, la différence entre ces allocations et les allocations luxembourgeoises est calculée et est payée au foyer en question. Ces différences sont appelées compléments différentiels.

L'évolution du nombre de travailleurs frontaliers est importante à connaître pour comprendre l'importance de ces échanges. En 2005, le Luxembourg employait 121.200 frontaliers alors qu'il compte une population d'environ 460.000 habitants alors qu'il ne comptait que 90.000 frontaliers en 2000 et 35.000 en 1990 (source : STATEC, 2007). Cette augmentation rapide des travailleurs frontaliers nécessite une adaptation des moyens de traitement des dossiers. L'expérience décrite ici concerne la CNPF, qui est directement touchée par cette augmentation, la taille de l'administration ne pouvant suivre l'augmentation du nombre de dossiers

[2] MDSD : Model Driven Software Development

3 La modernisation du calcul des allocations des travailleurs frontaliers

Lors de notre arrivée, la situation de la CNPF était critique. Le nombre d'allocataires était tel que le calcul manuel des compléments différentiels était fastidieux et sujet à erreurs. Un plan en trois phases fut élaboré.

La première phase consiste en l'établissement d'une procédure provisoire de calcul semi-automatique pour les frontaliers français. Afin de répondre à la situation de la CNPF, il s'agissait de fournir rapidement une solution opérationnelle. Cette phase fut réalisée en 2005 et fut étendue aux frontaliers belges et allemands en 2006.

La deuxième phase consiste à mettre en place une solution totalement automatisée pour les travailleurs frontaliers français. Cette solution doit effectuer mensuellement le calcul du complément différentiel à verser. Cette phase a été mise en production progressivement au troisième trimestre 2007. Enfin, la troisième phase visera à généraliser les résultats de la deuxième phase afin d'intégrer le calcul des prestations pour les allocataires luxembourgeois. Seul la deuxième étape du projet sera détaillée dans cet article.

Compte tenu des délais de développement assez courts et de l'environnement en pleine évolution de la CNPF, nous avons choisi d'utiliser une méthode de développement agile (Martin, 2002) pour ce projet. Le principe central étant de définir une priorité des besoins afin d'établir plusieurs cycles de développement incrémentaux, chaque cycle nécessitant la validation d'un certain nombre de tests opérationnels et fonctionnels. Une telle approche, fondée sur des cycles successifs courts, est particulièrement adaptée pour s'intégrer à un milieu changeant et pour fournir rapidement des solutions exploitables.

3.1 Réalisation

La seconde phase du projet consiste donc à l'établissement d'une solution entièrement automatisée pour le calcul automatique des compléments différentiels. Cette automatisation permet un calcul mensuel du complément afin d'améliorer la prestation pour les allocataires. De plus, l'automatisation des échanges, liés à un système efficace de détection des anomalies, permet d'éviter les payements incorrects ou la duplication des payements.

La nécessité de collaboration avec les pays partenaires implique une mise en place progressive du système, pays par pays. Comme pour la phase précédente, il est choisi par la CNPF de s'occuper en premier lieu des travailleurs frontaliers français. La mise en place de l'échange des données étant plus lourde et demandant une participation du pays partenaire plus accrue que pour la phase semi-automatique, l'extension du système aux autres pays frontaliers du Luxembourg ne fait pas l'objet de cette phase. De plus, tout comme la CNPF, la CNAF désire profiter de ce prototype pour définir une base de discussions pour leurs échanges avec d'autres pays voisins.

Le protocole de communication repose sur trois aspects principaux :

- L'échange des données nécessaires sur les personnes, permettant de définir si la personne à droit à une allocation et, le cas échéant, si le calcul et le payement des allocations sont nécessaires ou doivent être suspendus pour un mois donné, ainsi que les informations requises pour le calcul du montant de ce payement.

- La gestion des erreurs de protocoles. Les situations non définies comme correctes sont détectées, les allocations suspendues et les deux organismes doivent être prévenus afin d'éviter des payements incorrects. La qualité de ces échanges est cruciale car en cas d'erreur de payement, le caractère international des transactions rend le recouvrement du trop perçu très difficile.
- Enfin, dans le cas normal de fonctionnement, le système doit informer le pays partenaire du montant des allocations qu'il doit financer ainsi que le montant de la contre-partie à verser par la CNPF (soit pour le mois courant, soit rétroactivement pour les régularisations).

La version actuelle utilise des fichiers « plats » structurés pour la communication. Une version utilisant un format de données XML est prévue dans une deuxième phase du projet. Par la suite, une communication sous forme de service web est envisagée. Ces possibilités sont déjà prévues d'un point de vue fonctionnel dans notre solution. Elles demandent toutefois un investissement plus lourd du coté français car le fonctionnement actuel du mainframe nécessite des efforts plus importants pour être modifié.

De plus, afin de préparer la troisième phase du projet, la solution proposée devait être suffisamment générique afin de pouvoir intégrer les échanges avec les autres pays et le calcul des prestations familiales pour les foyers luxembourgeois.

3.2 Contraintes

Ce projet faisant parti d'un ensemble important de projets de modernisation du système d'information de la CNPF, il était nécessaire que la solution proposée soit facilement intégrable aux autres projets en cours de réalisation. Les détails concernant l'évolution technique du système d'information de la CNPF et la solution retenue sont exposés au chapitre 4.

Du fait du partenariat avec la France et de la collaboration nécessaire avec l'équipe de développement de l'administration française, des contraintes organisationnelles sont apparues ici de manière plus marquée que lors de la première phase. L'accord sur le protocole de communications entre le Luxembourg et la France devait être adapté aux systèmes d'informations existants dans les deux administrations. Cette contrainte était plus souple du coté luxembourgeois où l'environnement est en cours de modernisation mais elle était plus importante pour l'administration française où le mainframe est en production depuis une vingtaine d'années et où chaque changement doit être intégré avec précaution. Les différences méthodologiques furent aussi cause de difficultés. L'administration française ayant choisi une méthode de développement de type waterfall, leurs cycles étaient plus longs et leurs besoins d'analyse et de documentation plus conséquents. Il apparaît dans ce cas que la collaboration avec une méthode imposant des cycles de productions courts est complexe. Nous avons donc dû nous adapter aux méthodes du partenaire pour la partie protocole et découpler cette partie du reste de la solution.

3.3 État actuel et bilan

La mise en production de la deuxième phase est entrée en production depuis la fin du troisième trimestre 2007 et la troisième phase, intégrant le calcul des prestations pour les Luxembourgeois, devrait suivre courant 2008. De plus, comme indiqué précédemment, la

solution qui a été initialement déployée pour les frontaliers français devrait elle aussi évoluer, notamment pour obtenir un échange plus efficace des données.

4 Intégration d'un moteur BPEL dans une application locale

Une première définition des processus métiers (*business process*) fut effectuée en collaboration avec la CNPF. Durant cette phase, il était important de distinguer ce qui était spécifique aux travailleurs frontaliers français de ce qui était générique pour le traitement de tout travailleur frontalier. Ces descriptions furent réalisées à l'aide de diagrammes EPC (*Event-Driven Process Chain*) d'ARIS. Nous avons ensuite traduit ces diagrammes sous forme de processus BPEL afin de les rendre exécutables par un moteur de processus métiers. Le choix d'utiliser un tel moteur permet notamment de définir des processus spécifiques pour les différents pays frontaliers et de s'adapter ainsi plus facilement aux spécificités de chacun. BPEL est un langage de processus au format XML défini initialement par IBM et standardisé par OASIS[3].

Le choix de BPEL est naturel à la vue de la restructuration du système d'information de la CNPF sous forme d'une architecture SOA. Dans ce type d'architecture, l'utilisation de BPEL comme solution d'orchestration est souvent recommandée, notamment lorsque les flux sont entièrement automatisés (Silver, 2005).

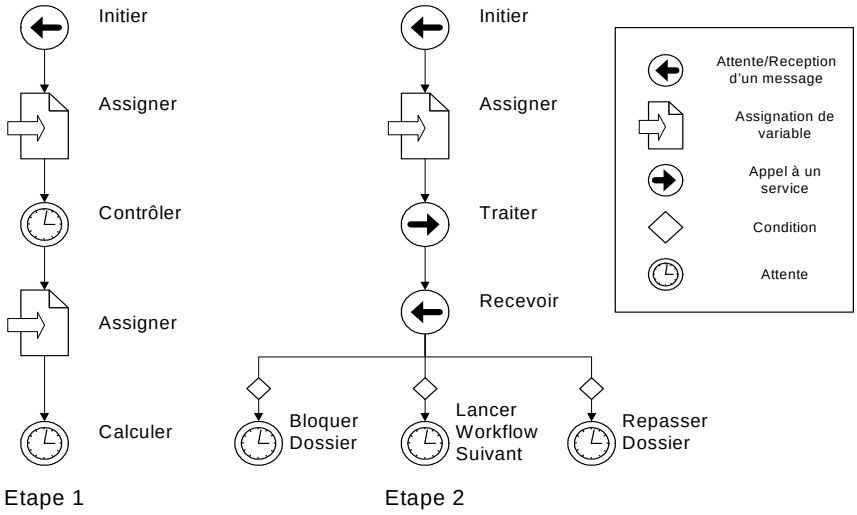

FIG. 1 – *Exemple de processus – premières étapes*

[3] OASIS : Organization for the Advancement of Structured Information Standards

4.1 BPEL dans un environnement agile

Il nous semblait également intéressant de juger de la pertinence de l'utilisation de BPEL dans le cadre des cycles de développement courts et de l'ordonnancement des tâches préconisés par la méthode agile. Dans ce contexte, il était important d'avoir la possibilité d'adapter les processus définis en fonction des réalisations prioritaires du développement agile. Pour mettre en place rapidement un environnement fonctionnel, il était en effet nécessaire de pouvoir obtenir des processus rapidement exploitables. Concernant l'ordonnancement des tâches, il est nécessaire de pouvoir exécuter les processus même si la totalité des services appelés n'est pas disponible. Pour parvenir à gérer ce problème, les processus BPEL ont été enrichis en cours de développement pour intégrer les différentes tâches finalisées. De manière similaire, il est apparu plus pertinent d'intégrer au sein de ces processus certaines vérifications de données initialement prévues pour être définies sous forme de service distant ou d'inclure des cas d'erreurs. La séparation claire entre les processus métiers et le reste du code nous a été très utile pour accomplir cette tâche.

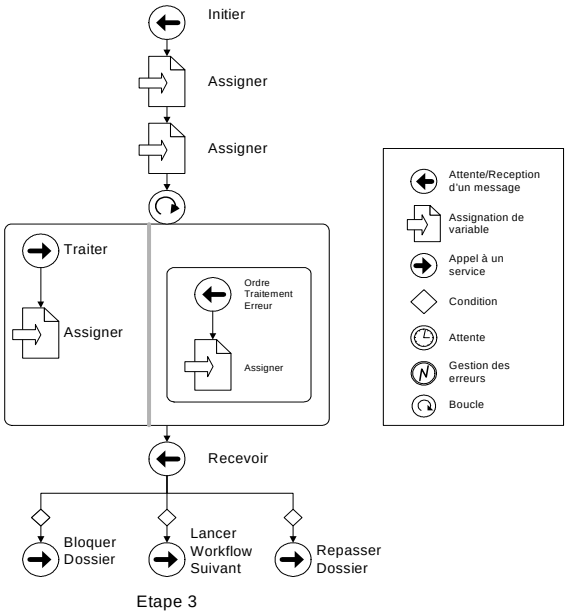

FIG. 2 – *Exemple de processus métier – étape finale*

Les figures 1 et 2 présentent un exemple d'évolution d'un processus métier. Les deux premières étapes d'évolution du processus sont présentées en figure 1 alors que la figure 2 présente la dernière version du processus. En figure 1, le passage de l'étape 1 à l'étape 2 a permis d'intégrer les composants de traitements réalisés lors des cycles de développement suivants, de regrouper certains contrôles prévus initialement séparément en un seul appel de service et d'affiner les tâches à réaliser lors de l'étape suivante. Le passage à la dernière

étape (figure 2) quant à lui intègre la gestion des erreurs lors de l'appel à certains services, l'appel aux composants de fin de traitements et automatise le traitement en introduisant une boucle sur tous les dossiers. Ces diagrammes furent également utilisés pour le dialogue avec notre partenaire, afin de conforter notre analyse et de valider la conception.

4.2 Problème d'intégration

L'utilisation de BPEL pour gérer des flux de messages et orchestrer des services internes à une application constitue un détournement de son usage usuel, basé sur l'appel de services distants. Il a donc été nécessaire d'adapter l'architecture de notre solution et le comportement du moteur BPEL pour satisfaire nos besoins.

BPEL repose sur l'utilisation de services web. Toute communication avec le moteur BPEL nécessite l'utilisation de ce concept, ce qui introduit une latence supplémentaire dans les temps de traitement. Les tests réalisés ont toutefois démontré que ce temps supplémentaire était marginal comparé aux temps de traitements initiaux, il est toutefois important de ne pas multiplier les appels de services.

L'utilisation de BPEL comme cœur de notre application nécessite en outre le développement d'interfaces pour assurer la communication entre les différents éléments de l'application (notamment le moteur BPEL et les autres composants). La difficulté principale étant d'assurer une cohérence d'état entre le moteur BPEL et l'application. Comme BPEL est une technologie basé sur l'échange de message, son comportement dépend du contenu des messages et non de l'état des processus, il est donc nécessaire de trouver une solution pour intégrer la notion d'état dans les processus. Cela est réalisé par un ensemble de mécanismes qui assurent la synchronisation entre l'application et le moteur de processus. La solution retenue peut être divisé en quatre parties :

1. Un cadre de développement intitulé « State Machine », déployé sur le serveur d'applications qui surveille l'état supposé des différents processus BPEL.
2. Un composent de coordination qui permet d'interroger les processus afin de vérifier que l'état attendu par l'application est bien identique à celui du moteur de processus.
3. Un mécanisme de compensation des erreurs qui affecte à la fois les processus et les composants annexes.
4. Un module de répartition des messages qui s'assure qu'un processus ne consomme que les messages lui étant directement destinés.

5 MDSD et méthode agile

Le développement sur base de modèles est une technologie clé dans le cadre du projet décrit ici. Il nous permet de limiter les considérations techniques de l'application et de se concentrer sur les aspects business. Les modèles indépendants de la plate-forme (Platform Independent Model – PIM) décrivent les aspects techniques du projets. Ils sont agrémentés de stéréotypes afin d'obtenir des modèles spécifiques à la plate-forme (Platform Specific Model – PSM). À partir de ces modèles, on produit du code spécifique à la plate-forme choisie. Dans notre cas, il s'agit principalement du code de la couche de persistence (grâce à la génération d'Enterprise Java Beans – EJB) et de code pour l'orchestration. Nous

expliquons ici les avantages de la méthode retenue dans le contexte agile et nous détaillons le modèle « State Machine » utilisé pour l'orchestration.

5.1 Cycles de développement MDSD

La notion de développement agile a déjà été étudiée dans le cadre du développement à partir de modèles, que ce soit à travers une approche systématique comme pour le développement MDA[4] (Mellor, 2004) ou dans le cadre plus restreint de MDSD (Stahl *et al.*, 2003). Il s'agissait ici d'éprouver ces concepts dans un projet industriel et de juger de sa pertinence dans un environnement de développement où plusieurs développeurs sont amenés à utiliser les modèles.

Les cycles de développement et d'amélioration du meta-modèle et des modèles dans notre cas sont globalement calqués sur les cycles d'amélioration du projet. Chaque nouvelle implémentation nécessitait la réutilisation de modèles précédemment définis. Selon le contexte, ces utilisations pouvaient nécessiter la modification du modèle ou du code généré. Ainsi, chaque incrémentation est un test de robustesse et d'adaptation supplémentaire pour les modèles utilisés (déduction). De même la réutilisation des composants permet de gagner en temps de développement et permet d'obtenir plus rapidement un code robuste. De ce point de vue, notre expérience recouvre globalement les avantages théoriques du développement MDSD agile. Respectivement, il est aussi arrivé que des améliorations aient lieu en raisonnant uniquement sur le meta-modèle et en faisant bénéficier l'application de ces modifications. Ces différentes méthodes d'améliorations conjointes du meta-modèle et des applications l'utilisant sont présentées en figure 3.

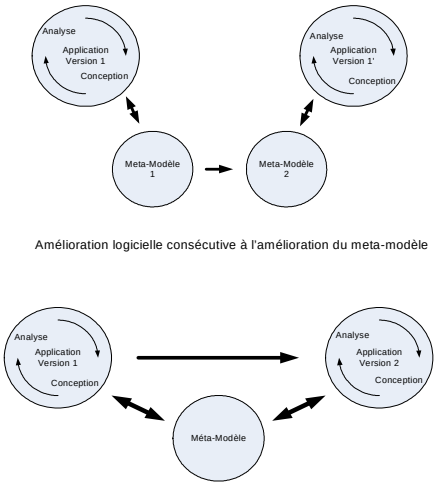

FIG. 3 – *Amélioration conjointe du meta-modèle et de l'application*

[4] MDA : Model Driven Architecture

Comme nous l'avions prévu, l'utilisation du MDSD a considérablement amélioré la productivité et la maintenance des composants souvent réutilisés. Il nous est toutefois apparu que pour les composants nécessitant une grande partie de code spécifique, l'utilisation de l'approche MDSD est trop coûteuse. Dans ces cas, il est plus efficace de coder manuellement les classes que de définir des patrons de codes non ou peu réutilisables. Dans le même ordre d'idée, la récupération de certaines données spécifiques ne pouvait se faire selon les modèles définis mais ces données n'impactaient pas un nombre suffisant de modèle pour justifier l'utilisation de MDSD pour générer le code correspondant. Là aussi, le codage direct du composant fut préféré.

Un autre problème qui est spécifique aux outils de modélisation utilisés lors du développement, est dû au partage des modèles à modifier entre les développeurs. Lors des cycles, il se peut que plusieurs utilisateurs souhaitent modifier simultanément des modèles existants. L'absence d'outils pour fusionner ces modèles sous leur forme graphique) peut être très handicapante pour le déroulement du projet. Afin de pallier ce problème, un système de planning partagé pour indiquer les modèles impactés par les changements à prévoir par chaque développeur a été mis en place. Même si ce type de procédé est peu adapté, peu de problèmes liés au partage des modèles ont été à déplorer pendant le déroulement du projet.

5.2 Le modèle de la « State Machine »

Un des modèles utilisés dans le cadre du projet vise à développer rapidement une « State Machine » sur le modèle du patron de programmation « state » proposé dans (Gamma et al. 97), ce patron est utilisable dans le contexte suivant :

- Le comportement d'un objet dépend de son état et celui-ci peut changer durant l'exécution.
- Certaines opérations contiennent des conditionnelles complexes qui permettent d'isoler le comportement à adopter en fonction de l'état de l'objet.

Cette définition correspond exactement à la situation rencontrée pour la gestion des états des processus dans notre application. Chaque flux de données doit donc contenir deux classes pour gérer son état : une classe *state* pour définir l'état courant et les transitions possibles et une classe *context* qui contient les informations nécessaires pour choisir la transition à appliquer.

La figure 4 donne un exemple d'utilisation de ce modèle. Ici, le générateur de code établit le lien avec la machine abstraite qui gère les états, créant toute la structure nécessaire pour la gestion des transitions. L'équipe de développement n'a plus qu'à définir le code business qui définit concrètement les états et les transitions (les classes ConcreteStateA et ConcreteStateB).

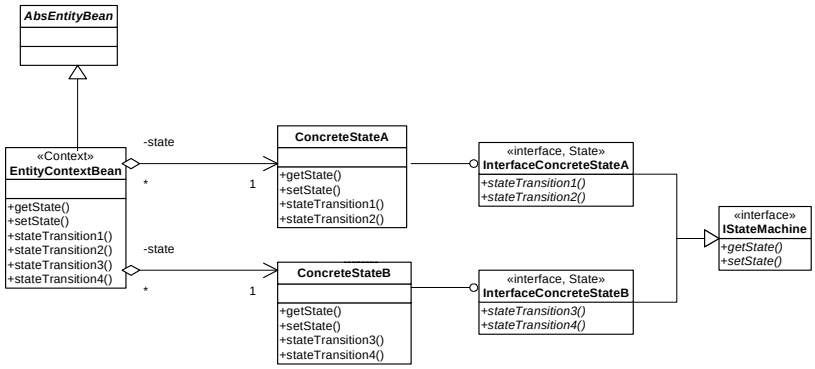

Fig. 4 – Une instance de la « State Machine »

6 Conclusions et suites du projet

La réalisation de ce projet a permis d'établir l'intérêt et les difficultés techniques dues à l'utilisation d'un moteur d'exécution de processus métier pour gérer des flux de données dans un environnement de développement agile avec des contraintes de temps fortes. L'utilisation du moteur de processus métier a permis d'offrir une plus grande souplesse lors de l'implémentation des différentes partie business. Elle permet également une adaptation plus simple des processus en production, ce qui améliore la maintenance de l'application. Une contrepartie a ses avantages est la complexité de la gestion des états des processus et de gestion de la cohérence entre les processus et le reste de l'application. Nous avons toutefois expliqué que l'utilisation de la génération de code sur base de modèle permet de simplifier cette synchronisation.

Les prochaines étapes vont mettre à l'épreuve les solutions retenues selon deux aspects. La maintenance des premières versions en production tout d'abord, qui permettra de juger des apports de l'approche MDSD pour la correction de bugs. À priori, la localisation des erreurs devrait être facilitée par le découpage imposée entre code générique et code spécifique et la robustesse du code devraient être améliorée pour les parties générées, puisque chaque amélioration du composant impacte automatiquement tous les modèles qui en dépendent. Ensuite, la phase de généralisation permettra de mieux juger de la facilité de réutilisation des services qui constituent une des raisons du choix de la technologie BPEL.

Références

Bauler P., Feltz F., Biri N., Pinheiro P. (2006). Implementing a Service-Oriented Architecture for Small and Medium Organisations. *EMISA'2006: Methoden, Konzepte und Technologien für die Entwicklung von dienstebasierten Informationssystemen*, Hambourg, Allemagne.

Gamma E., Helm R., Johnson R., Vlissides J. (1997). *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Hitzelberger P., Feltz F. (2006). An Interoperable Communication Platform for a Public Agency, *5th international EGOV conference*, Krakow, Pologne.

Martin R.C. (2002). *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall.

Mellor S.J. (2004). *Agile MDA, A white paper*, http://omg.org/mda/mda_files/AgileMDA.pdf

Mellor S.J., Scott K., Uhl A., Weise D. (2004). *MDA Distilled – Principles of Model-Driven Architecture*, Adison Wesley.

Silver B. (2005). Agile To The Bone, *Intelligent Enterprise*, http://www.intelligententerprise.com/showArticle.jhtml?articleID=57702677.

Stahl T., Völter M., Bettin J., Haase A., Helsen S. (2006). *Model Driven Software Development – Techonology, Engeneering, Management*, Wiley.

Summary

This article gives an overview of the IT based automation of the family allowances procedures for commuters, by explaining the project milestones as well as the used technologies. We start by giving a general overview of the project and its context, as well as the overall objectives. Next we explain how a workflow engine and model driven software development techniques got applied in an agile development context. Based on this experience, we show the advantages and disadvantages of BPEL to handle the workflows of a local application and conclude by a discussion about some interesting scientific questions rose during the project.