

Interfaces adaptatives

Frédérique Laforest

LIRIS UMR CNRS 5205 Bat Blaise Pascal INSA de Lyon 69621 Villeurbanne Cedex France

frederique.laforest@liris.cnrs.fr

Résumé.

Dans cet article, nous présentons les résultats du projet SEFAGI (Simple Environment For Adaptable Graphical user Interfaces) dont l'objectif est de fournir un outil de génération automatique du code des interfaces graphiques d'un système d'information, pour différents types de terminaux, qu'ils soient fixes ou mobiles. Cette génération automatique se fonde sur une description des fenêtres désirées, indépendamment des terminaux cibles. La description est faite par un utilisateur non informaticien mais connaissant les services offerts par le système d'information ; elle est outillée d'un assistant interactif qui guide l'utilisateur dans sa description, en lui fournissant la liste des services disponibles et les différentes formes d'interaction disponibles. Le générateur construit alors automatiquement le code des fenêtres décrites, pour le(s) type(s) de terminal désiré(s).

Mots-clés : informatisation du dossier patient, interfaces homme-machine, adaptation, mobilité

1 Problématique

Le patient étant par essence mobile et rencontrant divers épisodes de santé, son dossier est forcément distribué dans les différentes entités qui l'ont pris en charge. Si hier la transmission de documents sous forme papier résumant les informations principales était la principale et presque unique voie de transmission de l'information entre entités, le suivi des patients est aujourd'hui devenu une problématique de l'informatique distribuée. En effet, on ne se contente plus d'un résumé des actes effectués par d'autres entités ou d'autres professionnels de la santé, il est nécessaire d'accéder à l'information rapidement, à sa source, et quelle que soit sa localisation géographique, bien sûr tout en conservant les règles de confidentialité et de sécurité définies par la loi. L'extension de la prise en charge des patients à domicile (par exemple dans le cadre d'hospitalisations à domicile) et l'intérêt toujours grandissant pour la prise en compte de la qualité des soins et de leurs coûts orientent les regards vers des outils permettant la gestion des données distribuées de façon transparente à l'utilisateur.

Mais on ne s'arrête pas là : le professionnel de santé est lui aussi mobile et désire avoir accès aux dossiers des patients là où il se trouve (à son bureau, mais aussi au lit du patient à l'hôpital, chez le patient en hospitalisation à domicile, dans l'ambulance...). Cette nouvelle étape requiert l'intégration de terminaux mobiles dans le système d'information médical distribué. Cette intégration pose encore aujourd'hui de nombreux problèmes, allant de la connectivité (accéder au meilleur réseau disponible), à la gestion des données (gestion de caches et de cohérence), en passant par les interfaces adaptatives. Les interfaces adaptatives ont pour but de permettre la présentation des informations et l'interaction avec le système d'information d'une façon adaptée au dossier traité, au terminal utilisé et plus généralement au contexte de l'utilisateur. Ce problème est complexe car la diversité des cas d'utilisation est importante. En effet, si l'on pouvait restreindre le nombre de cas à une petite dizaine, on pourrait développer plusieurs versions de la même application. Mais dès que le nombre de cas augmente, il n'est plus possible de maintenir autant de versions d'un même système d'information, d'autant plus que les systèmes d'information médicaux ne sont pas des applications triviales.

La diversité des acteurs du système de santé est également une contrainte forte à prendre en compte. Nous ne pouvons pas construire une unique application pour l'ensemble des acteurs. Chaque type d'acteur ayant des droits et des besoins en information très divers, il est indispensable de construire des modules « clients » (apparaissant sur le terminal) adaptés au besoin de chacun. Ainsi, l'application « dossier

médical » est décomposée en divers modules clients, un module (composé de fenêtres) étant dédié à un type d'utilisateur donné et à un type de dossier médical donné (par exemple, module pour le médecin spécialiste de l'hôpital pour des dossiers de cardiologie, différent du module pour l'infirmière pour des dossiers de la même spécialité, différent du module pour un autre spécialiste ou pour un généraliste ou pour un kinésithérapeute...). Nous avons même dans notre proposition la possibilité de personnaliser les interfaces graphiques à chaque utilisateur.

Dans la section suivante, nous présentons l'architecture logique de SEFAGI. Ensuite, nous présenterons un exemple de terminal généré et nous terminerons par un état d'avancement, ce qui nous permettra de conclure et de tracer les perspectives de recherche de ce projet.

2 Architecture générale de SEFAGI

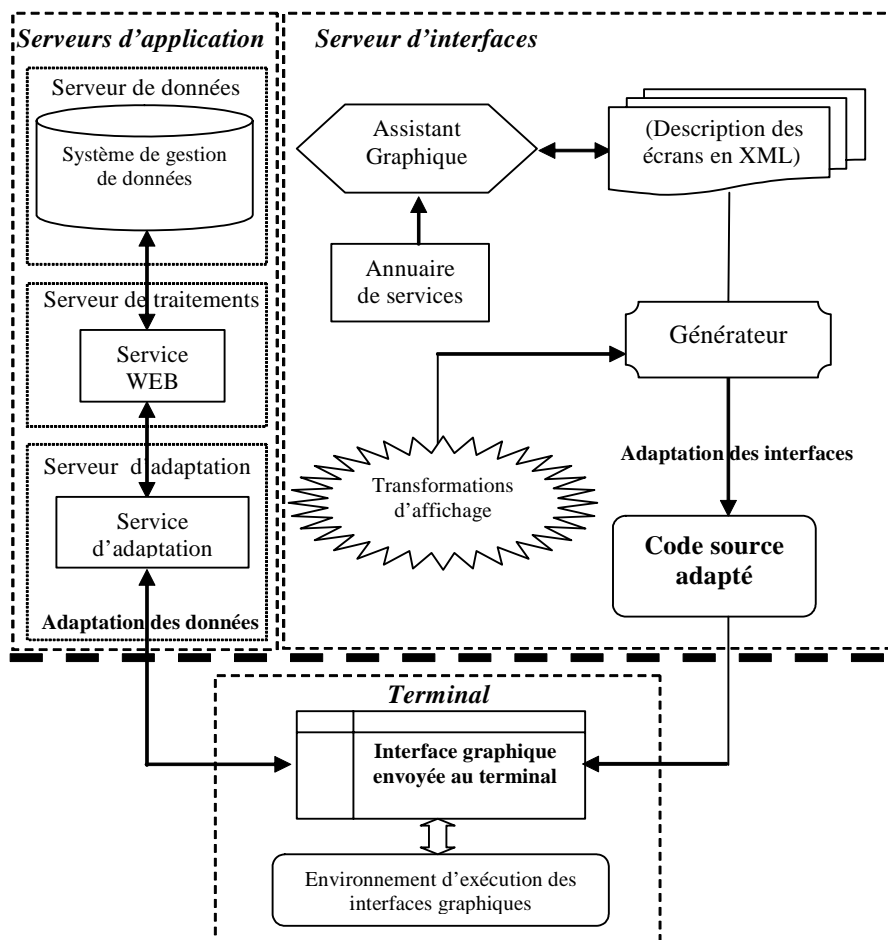


Figure 1 Architecture de SEFAGI

La figure 1 présente l'architecture de SEFAGI. Elle est composée de trois blocs que nous décrivons ci-dessous.

Le bloc serveurs d'application représente le système d'information de l'organisation. Nous avons choisi, dans notre prototype, d'implémenter ce système d'information sous la forme de services Web (W3C, 2006), gérés par le middleware OSGi (OSGI, 2006). Puisque les données peuvent être consultées sur différents types de terminaux et avec différentes qualités de réseau, nous avons adjoint un ensemble de services d'adaptation des données, qui permettent de transformer les données de telle manière qu'elles soient adaptées au contexte actuel.

Chaque terminal qui voudra interagir avec les serveurs d'application doit dans un premier temps télécharger un environnement d'exécution dédié au type de terminal concerné. En choisissant la technologie Java pour développer cet environnement d'exécution et les codes des fenêtres générées, nous avons restreint les types de terminaux à trois catégories : les terminaux à haute capacité (tels des PC) avec la norme J2SE, les terminaux mobiles à faible capacité (tels les téléphones mobiles java) avec la norme J2ME et le profil CLDC, et les terminaux mobiles à haute capacité (tels les assistants personnels ou autres PDA) avec la norme J2ME et le profil CDC. Pour chaque type de terminal, un fichier XML de description des API des interfaces graphiques et une bibliothèque de composants graphiques de haut niveau doivent être fournis au générateur. A partir de ces descriptions, le générateur construit complètement et de façon entièrement autonome le code des environnements d'exécution.

Le bloc serveur d'interfaces contient le générateur et l'assistant de description des interfaces graphiques. Il comporte également un accès à l'annuaire des services disponibles dans le système d'information. Pour construire une nouvelle fenêtre, l'utilisateur lance l'assistant graphique. Cet assistant, pour chaque fenêtre, demande la liste des services avec lesquels interagir, et pour chaque service le mode d'interaction choisi (pour l'instant, nous avons implanté dans notre prototype les modes suivants : présentation tabulaire des informations (chaque cellule du tableau pouvant être une zone de texte, une liste, un ensemble de boutons radio ou de cases à cocher...), affichage d'une courbe ou d'un diagramme « bâtonnets », navigation dans une liste d'images, navigation dans une liste de vidéos, barre de boutons, grande zone de texte. Des options permettent de configurer les éléments posés sur l'interface. Chacune de ces formes d'interaction correspond à un ensemble de classes Java constituant la bibliothèque de composants graphiques de haut niveau dont SEFAGI définit l'API. L'intégration d'un nouveau type de terminal nécessite donc la création des classes correspondantes pour le type de terminal incriminé.

Les choix effectués par l'utilisateur sont enregistrés dans un fichier XML dont nous avons défini la structure. Cette structure est complètement indépendante des types de terminaux cibles, et contient beaucoup moins d'informations qu'une description UIML (Ali *et al*, 2002 ; Abrams *et al*, 2002) ou XUL (Deakin, 2006) par exemple. En effet, nous avons pris le parti de mettre en place une tâche de description des interfaces graphiques la plus légère possible et donc de demander le moins de détails possible à l'utilisateur. Cela a pour conséquence que le fichier de description et la tâche de description sont légers, qu'il est réellement possible de générer entièrement le code des fenêtres correspondantes, mais il est vrai aussi que nous perdons en souplesse de configuration des interfaces générées.

Le générateur prend en entrée le fichier XML de description d'une fenêtre, le fichier XML de description de l'API du type de terminal sélectionné, et construit entièrement et de façon autonome le code correspondant. Des règles de transformation d'affichage ont été définies, qui permettent d'effectuer des adaptations aux capacités des terminaux. Par exemple, comme il n'est pas possible d'afficher un tableau sur l'écran d'un téléphone mobile, chaque affichage tabulaire est décomposé en deux écrans, et d'y associer des outils de navigation. Le cas d'étude de la section suivante présente plusieurs exemples de transformations d'affichage.

Il ne reste plus qu'à télécharger le code généré sur le terminal cible, et le tour est joué !

3 Cas d'étude

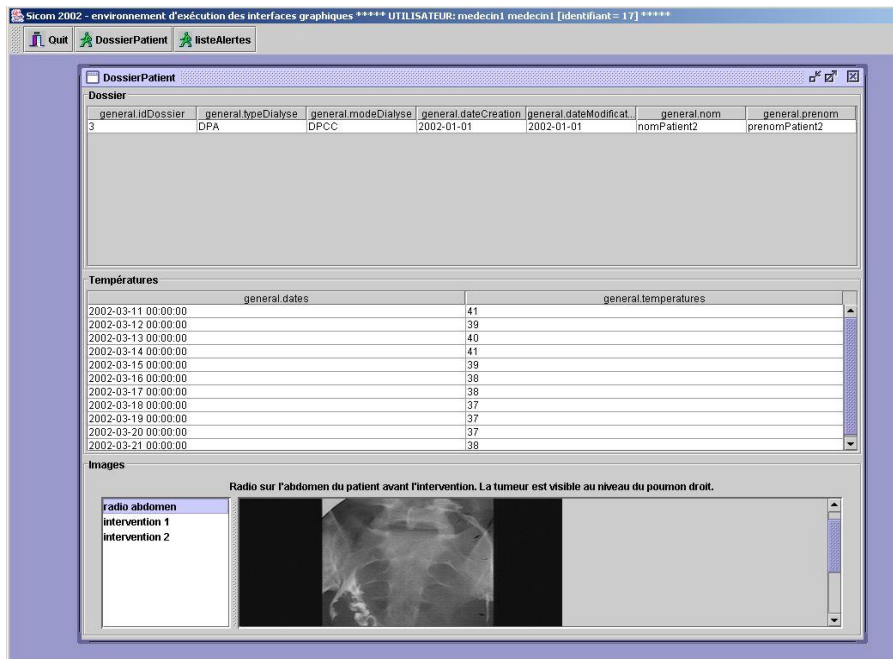


Figure 2. Interface graphique générée pour les terminaux standards

La Figure 2 présente une fenêtre générée pour les terminaux standards dans l'environnement d'exécution, c'est-à-dire comme elle est vue et manipulée par un utilisateur final. Une barre de menu permet la navigation entre les différentes interfaces générées (DossierPatient et listeAlertes). La fenêtre DossierPatient est affichée. Elle comporte 3 panneaux. Les deux premiers sont de type tableau, le troisième est de type image. Les panneaux de type tableau contiennent une ligne par n-uplet de données à afficher/saisir. Chaque élément de la ligne est ici une zone de texte simple, mais on peut trouver des listes de choix, des cases à cocher... Le panneau de type image contient trois zones : la première fournit la liste des noms des images, la seconde présente l'image sélectionnée, et la troisième (au-dessus) affiche le texte descriptif associé à l'image.

La Figure 3 donne l'équivalent pour les terminaux mobiles. La barre de menu est remplacée par un premier écran fournissant la liste des fenêtres disponibles. Le système présente ensuite le premier panneau de l'interface choisie par l'utilisateur. Le menu contextuel permet alors de passer à un autre panneau de cette fenêtre, ou de retourner au menu principal.

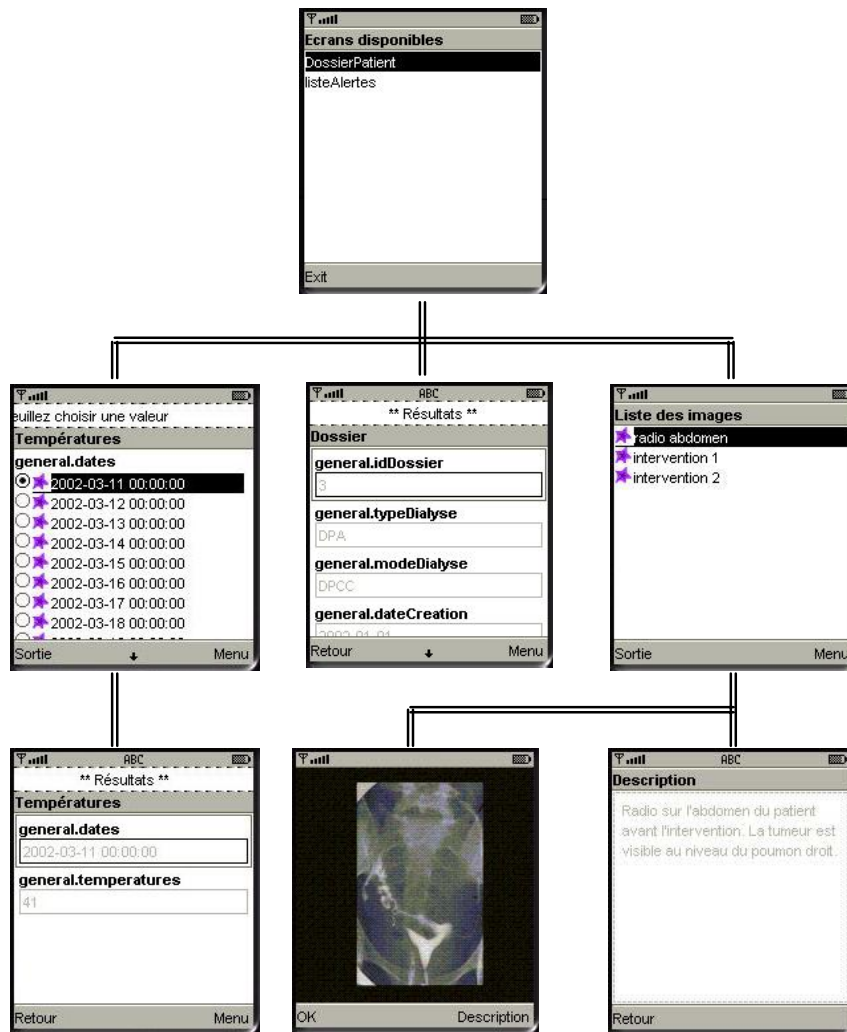


Figure 3. Interface graphique générée pour les terminaux mobiles

Le premier panneau (résultat monoligne dans un panneau tableau) est constitué d'un seul écran sur le mobile, les éléments sont affichés de façon verticale (ils sont affichés horizontalement sur un écran standard). Pour le panneau tableau multi-lignes, on accède premièrement à une liste correspondant aux valeurs de la première colonne, qui permet de sélectionner l'affichage de la ligne correspondante dans un autre écran (affichage vertical des informations). Pour le panneau de type image, un premier écran est proposé avec la liste des images. La sélection d'une image permet de passer à un écran où l'image elle-même est présentée (elle est entre temps passée par une adaptation de contenu pour être transformée au format PNG). Le menu « description » permet ensuite de passer à la description textuelle de l'image. Dans le cas où le terminal utilisé ne supporte pas l'affichage d'images, la sélection d'une image amène directement à la description textuelle correspondante.

4 Conclusion

Nous avons présenté dans ce papier l'architecture de SEFAGI, qui permet la description de fenêtres à l'aide d'un assistant graphique et la génération automatique du code correspondant. Un petit cas d'étude permet d'illustrer notre travail. La description des fenêtres et des API cibles sont externalisés du générateur, ce qui permet d'ajouter de nouveaux types de plates-formes cibles sans toucher au générateur.

Actuellement, nous orientons nos travaux sur le téléchargement automatique du code des interfaces graphiques. Ceci nécessite la description des ^profils d'utilisateurs et l'association d'ensembles de fenêtres

à chaque profil. Une étape de négociation doit s'engager à chaque connexion d'un utilisateur pour que les nouvelles fenêtres le concernant soient automatiquement téléchargées sur son terminal. Les conditions et les moyens d'action de ce téléchargement sont en cours de définition.

D'autre part, la facilité de description des interfaces graphiques a engendré un nouveau besoin : les potentiels utilisateurs à qui nous avons présenté ce travail imaginent même qu'ils pourraient définir des interfaces spécifiques à une personne en particulier, par dérivation d'une fenêtre rédigée pour son profil. La gestion des versions de fenêtres disponibles est donc un sujet sur lequel nous travaillons actuellement (notamment dans le cadre d'un stage de master recherche M2).

Finalement, nous avons l'ambition d'intégrer SEFAGI à un environnement plus vaste d'adaptation complète des applications, intégrant non seulement l'adaptation des interfaces, mais également l'adaptation des données et l'adaptation des services eux-mêmes.

Références

OSGi Alliance (2006) *Open Service Gateway Interface* Available at <http://www.osgi.org/> Last visited June 15, 2006

W3C (2006) *Web services Architecture Domain* Available at <http://www.w3.org/2002/ws/>. Last visited June 15, 2006

Abrams M., Helms J. (2002) *User Interface Markup Language (UIML) 3.0* Specification DRAFT, Document version 08 February 2002. <http://www.uiml.org/specs/docs/uiml30-revised-02-12-02.pdf>

Ali M.F., A.Pérez-Quiñones M., Abrams M., Shell E. (2002) Building Multi-Platform User Interfaces With UIML. In *CADUI'2002, 4th International Conference on Computer-Aided Design of User Interfaces*. Valenciennes, France May 2002

Deakin N., (2006) *XUL tutorial*. Available at <http://xulplanet.com/tutorials/xultu/>. Last visited Feb. 11, 2006

Remerciements

Je tiens à remercier tous les étudiants thésards, master ou autres qui ont participé au projet SEFAGI. Je citerai tout particulièrement Tarak Chaari sans qui SEFAGI ne serait pas ce qu'il est aujourd'hui.