Ethylene : composants dynamiques pour la mise en œuvre d'IHM plastiques en informatique ambiante

Lionel Balme

Université de Grenoble, CNRS, LIG
385, rue de la Bibliothèque,
BP 53, 38041 Grenoble Cedex 9, France
Lionel.Balme@imag.fr

Joëlle Coutaz

Université de Grenoble, CNRS, LIG
385, rue de la Bibliothèque,
BP 53, 38041 Grenoble Cedex 9, France
Joëlle.Coutaz@imag.fr

RESUME
L'informatique ambiante impose de nouvelles contraintes sur la manière de construire les interfaces Homme-Machine (IHM). Traditionnellement centralisée sur un unique dispositif, l'IHM doit maintenant être distribuable de manière opportuniste sur un ensemble dynamique de dispositifs hétérogènes. Ces nouvelles IHM, dites *plastiques*, sont abordées dans cet article sous l'angle du génie logiciel (GL). Ce sont des logiciels répartis, dynamiquement adaptables, capables de s'affranchir de l'hétérogénéité des dispositifs et des logiciels. Les solutions actuelles de l'état de l'art en GL et systèmes répartis ne tiennent pas compte de la spécificité de l'interaction homme-machine. Nous proposons donc *Ethylene*, un cadre conceptuel et technique pour développer des IHM plastiques en informatique ambiante. Notre solution s'appuie sur l'intégration des approches dirigées par les modèles, d'une combinaison particulière de l'approche à composants et de l'approche à service et d'une manière originale de s'affranchir de l'hétérogénéité des modèles de communication inter-composant.

MOTS CLES : IHM, informatique ambiante, IHM plastique, adaptation dynamique d'IHM, adaptation à l'exécution, composant orienté service, assemblage hétérogène de composants logiciels.

ABSTRACT
Ubiquitous computing introduces new constraints on the way we build interactive systems. Traditionally centralized, user interfaces (UI) can be distributed across a dynamic set of heterogeneous devices. In this article, we examine these new kinds of UIs, named *plastic* UIs, from a software engineering perspective. They are dynamic adaptable distributed software, and they able to deal with device heterogeneity at multiple levels of abstraction. However, the generic solutions from distributed software do not take into account the specificity of human-computer interaction. To address these limitations, we propose Ethylene, a conceptual and technical framework to develop UIs for ubiquitous computing. Our solution is based on the integration of model driven engineering with a particular combination of component and service oriented approaches, as weel as an original way to overcome the heterogeneity of communication models between components.

IHM 2009, 13-16 Octobre 2009, Grenoble, France

CATEGORIES AND SUBJECT DESCRIPTORS: H.5.2 User Interfaces---Theory and methods, D.2.2 Design Tools and Techniques--- User interfaces.

GENERAL TERMS: Human factor, Design.

KEYWORDS: HCI, ubiquitous computing, UI plasticity, UI dynamic adaptation, adaptation at runtime, service oriented component, assembly of heterogeneous software components.

INTRODUCTION

L'informatique ambiante a pour vocation de fournir des services capables de répondre en toute circonstance à des besoins individuels, collectifs ou sociétaux. Pour l'Interaction Homme-Machine, cette ambition signifie que l'enjeu n'est plus seulement de fournir des produits finalisés pour des tâches données en un lieu et des ressources prédéterminés, mais de permettre à l'utilisateur de recourir de manière opportuniste à des services et à des ressources d'interaction très diversifiés. En conséquence, les interfaces homme-machine de ces services doivent devenir plastiques : avoir la capacité de s'adapter au contexte d'usage[1] dans le respect d'une valeur attendue par l'utilisateur cible [33, 15].

Sur le plan technique, la plasticité des IHM s'inscrit dans le problème général de l'adaptation dynamique des logiciels, elle-même traitée par plusieurs communautés de

[1] Dans ce cadre, le contexte d'usage (également appelé contexte de l'interaction) se définit par le triplet constitué par la l'utilisateur, la plate-forme et l'environnement.

recherche : l'Interaction Homme-Machine, l'Intelligence Artificielle (IA) et le Génie Logiciel et systèmes distribués (GL). Avec l'exemple de l'album photos PhotoBrowser, nous identifions des requis logiciels et montrons dans les sections qui suivent que l'état de l'art en IHM, IA et GL ne couvre que partiellement ces requis. Ce constat nous a amenés à identifier des principes logiciels pour appréhender, en conformité avec les requis, l'adaptation dynamique des IHM plastiques à l'exécution. Nous proposons ensuite Ethylene, un modèle à composant dynamique d'abord sur le plan conceptuel, suivi de sa mise en œuvre technique.

UN EXEMPLE : PHOTOBROWSER

PhotoBrowser est un système interactif réalisé selon l'approche Ethylene . Il s'agit d'un album photos dont l'IHM découle de l'assemblage dynamique de composants hétérogènes éventuellement répartis sur plusieurs machines. Au moyen d'une IHM de contrôle (voir fig. 1-a), l'utilisateur a la possibilité de transformer, à l'exécution, l'IHM de PhotoBrowser, en décidant de la combinaison de ses composants.

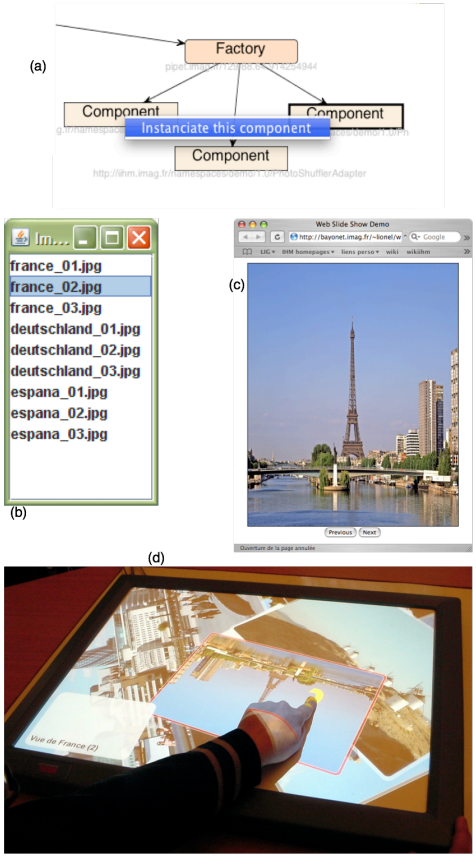

Figure 1: Les différents composants de PhotoBrowser

Dans ce prototype de PhotoBrowser, trois composants IHM sont disponibles dynamiquement : ils apparaissent et disparaissent au cours de l'exécution du système, au gré des variations de la plate-forme de l'utilisateur. Le premier (fig. 1-b) est implémenté sous la forme d'une classe Java et présente à l'utilisateur la liste des photos disponibles. Il permet la sélection d'une photo à visualiser. Le deuxième (fig. 1-c) est écrit en html/ajax et affiche à l'utilisateur la photo courante ainsi que deux boutons pour passer à la photo suivante ou précédente. Le troisième (fig. 1-d) est programmé en Tcl avec la boîte à outils post-WIMP GML-Canvas [6] et permet de naviguer, de réorienter et de zoomer les photos à plusieurs sur une table interactive. Cet exemple illustre nos requis : les IHM sont réparties sur plusieurs dispositifs d'interaction dont la disponibilité est dynamique, leur adaptation est dynamique et est réalisée sous le contrôle de l'utilisateur, les entités logicielles qui les constituent sont disponibles dynamiquement et s'inscrivent dans des espaces technologiques [24] très différents (dans l'exemple : Java, Tcl, html/ajax). Que nous offre l'état de l'art en réponse aux requis des IHM plastiques en intelligence ambiante ?

SOLUTIONS ACTUELLES EN IHM ET EN IA

La question de la plasticité a été abordée de multiples façons par la communauté de l'Interaction Homme-Machine. On constate une nette partition entre une première catégorie de travaux qui se concentrent sur les phases de conception et une deuxième qui s'intéresse à la phase d'exécution.

Plasticité en phase de conception

Les travaux centrés sur les phases de conception adoptent généralement une approche centrée sur les modèles. C'est le cas des outils de génération d'IHM multicibles proposés par [34] (ArtStudio), [19] (Graceful Degradation), [28] ou encore la chaîne d'outils UsiXML[2] [8, 27]. Ces travaux ont en commun l'utilisation en entrée de modèles à haut niveau d'abstraction (modèle de tâche notamment), puis la transformation de ces modèles pour obtenir en sortie des IHM exécutables sur un ensemble de plates-formes cibles prédéterminées.

Au regard de nos requis, ces approches souffrent de deux limitations. D'une part, les contextes d'usage doivent être parfaitement connus dès la conception, ce qui est difficile lorsque la disponibilité des ressources d'interaction et des composants logiciels est dynamique. D'autre part, les IHM produites se limitent à des interacteurs WIMP conventionnels.

Plasticité en phase d'exécution

Les travaux centrés sur les phases d'exécution traitent le problème de la plasticité des IHM selon trois angles principaux : celui des gestionnaires de fenêtres, celui des boîtes à outils d'interacteurs et celui des infrastructures logicielles. Tous ces travaux jouent sur deux moyens d'adaptation des IHM : le remodelage et/ou la redistribution [4]. Toutefois, les travaux permettant le remodelage traitent peu, voire excluent, la redistribution (et réciproquement) ; les travaux qui traitent l'hétérogénéité ne

[2] http://www.usixml.org

s'intéressent principalement qu'à l'hétérogénéité des plates-formes élémentaires[3] alors qu'en informatique ambiante, l'interopérabilité doit être assurée entre des éléments logiciels issus d'espaces technologiques différents (comme dans le cas de PhotoBrowser).

À la croisée des travaux « centrés phase-de-conception » et « centrés phase-d'exécution », quelques travaux tentent d'estomper cette dichotomie traditionnelle.

Plasticité et rapprochement des phases de conception et d'exécution

Une façon d'estomper les phases de conception et d'exécution est de considérer un système interactif, non pas sous le seul angle de son code exécutable, mais sous la forme d'un graphe de modèles reliés par des transformations [31]. Chaque modèle décrit une perspective sur le système, depuis le modèle de tâche jusqu'au code exécutable qui n'est jamais qu'un modèle comme les autres. Dès lors, il est possible, à l'exécution, d'envisager des adaptations de l'interface homme-machine à tout niveau d'abstraction, par exemple supprimer ou réorganiser des tâches jusqu'à des ajustements cosmétiques comme le permettent les CSS. Cependant, comme pour toute approche dirigée par les modèles, les IHM innovantes de type post-WIMP ne sont pas traitées.

Dans une perspective différente, Lewandowski vise à décrire des composants logiciels au moyen de modèles de tâches [25]. Ces composants logiciels orientés tâches (COTs) sont alors bien plus faciles à réutiliser et à intégrer au sein de développement logiciel. Cependant, l'approche de Lewandowski n'envisage pas, pour l'instant, le cas de l'adaptation dynamique des assemblages de composants, cette tâche revenant au développeur.

Plasticité selon l'IA

Jusqu'ici, les travaux en IA se sont principalement intéressés à une forme d'adaptation à l'utilisateur. Par exemple, Browne [9] traite de l'adaptation de l'aide et des messages d'erreur. Plus récemment Lieberman [26] propose Roadie, un agent IHM capable de deviner les objectifs de l'utilisateur pour générer une aide contextuelle à initiative mixte. Ou encore, les réseaux Bayésiens sont mis à contribution pour identifier des situations clefs (chez soi, au travail, etc.) et de là, le système construit la page d'accueil du téléphone mobile avec les applications auxquelles l'utilisateur fait appel en ces situations [20].

SOLUTIONS ACTUELLES EN GL

En GL, les travaux visent principalement l'adaptation aux changement de ressources afin d'optimiser les performances du système de manière autonome. Pour ce faire, plusieurs approches sont explorées parmi lesquelles les architecture dynamiques (par ex. [1][36]), la réflexivité et la reconfiguration dynamique de composants orientés services (par ex. [12][17]) et le tissage de comportements par aspects (par ex. [16][13]). Au-delà de leur capacité d'adaptation dynamique, les IHM plastiques, tel PhotoBrowser, peuvent être réparties et doivent être capables de s'affranchir de l'hétérogénéité des espaces technologiques d'implémentation. La nature logicielle des IHM s'inscrit donc maintenant à la croisée de trois domaines : celui des intergiciels qui traite de la répartition, celui des approches à composants et à services qui traite de l'adaptation dynamique et celui de l'IDM qui aborde le problème de l'hétérogénéité des technologies d'implémentation.

Les intergiciels

Les intergiciels constituent la solution reconnue au problème du logiciel réparti lorsque l'infrastructure réseau est fixe. La multitude des dispositifs et des sytèmes introduits par l'informatique ambiante pose le problème de l'hétérogénéité des intergiciels et de leur interopérabilité [29]. À notre connaissance, l'interopérabilité des intergiciels a été traitée de deux manières : par réflexivité (par ex. dans [22]) et par traduction de protocole (par ex dans [2]).

L'intergiciel réflexif ReMMoC permet à un client ReMMoc d'interagir avec des services logiciels implémentés au-dessus d'autres intergiciels [22]. Cependant, ReMMoC ne permet pas la réciprocité : un service ne peut pas être développé au-dessus de ReMMoC pour être exploité par des clients implémentés au-dessus d'autres intergiciels. Dans le cadre de la plasticité des IHM, cette limitation est blocante : vis-à-vis du noyau fonctionnel, une IHM est à la fois cliente (elle permet à l'utilisateur de manipuler le noyau fonctionnel) et service (elle rend observable à l'utilisateur l'état du noyau fonctionnel). L'approche par traduction de protocole adoptée par le projet AMIGO permet de dépasser cette limitation.

La traduction de protocole retenue dans AMIGO repose sur des *générateurs* qui, à partir de la spécification de protocoles, produit une unité de traduction pour chaque composant engagé dans une interaction logicielle. Une *unité* est composée d'un *analyseur* et d'un *composeur*. L'analyseur transforme les données émises par un composant source en événements qui sont transmis, via un *mandataire*, au composeur de l'unité de traduction associée au composant destinataire. Ce composeur transforme les événements reçus dans le format du protocole compris par le destinataire.

Si cette solution permet l'existence d'entités logicielles à la fois clientes et services, elle est en revanche gourmande en puissance de calcul, notamment lors de la génération des traducteurs. Ce coût de calcul peut entraîner des problèmes de latence perceptibles pour l'utilisateur lors-

[3] Une *plate-forme élémentaire* est l'association au sein d'un dispositif physique indivisible, d'un ensemble de ressources d'interaction. Voir [4], page 29.

qu'en situation de mobilité, les ressources de calcul s'avèrent insuffisantes.

Si les intergiciels traitent des questions de distribution et d'interopérabilité, celle de l'adaptation dynamique des logiciels est traitée, en GL, par les approches à composants et les approches à services.

Les approches à composants

Un composant est une unité de déploiement et de composition [32]. Plusieurs composants reliés par des *connecteurs* forment un *système* [30], [21]. D'abord simple outil de spécification pour le concepteur de système, l'architecture logicielle accède aux phases d'exécution en devenant dynamique.

Cette dynamicité des architectures logicielles est obtenue de façons diverses : chez Allen [1] des transitions spécifiées à la conception permettent de passer, à l'exécution, d'un sous-sytème préconfiguré à un autre. Wile [36] rend les architectures modulables en spécifiant des degrés de liberté explicites et Batista [5] va plus loin en donnant la possibilité de spécifier la reconfiguration d'un système lors de son exécution. D'autres travaux s'intéressent plus précisément aux aspects technologiques de la reconfiguration dynamique : SOFA 2.0 introduit la notion de patron de reconfiguration [23], et Bruneton [10] définit sept requis clefs qu'une technologie doit satisfaire pour permettre l'implémentation des systèmes distribués hautement flexibles et dynamiquement configurables. Il propose Fractal, un modèle à composant qui respecte ces requis. À un autre niveau d'abstraction, SAFRAN [16] et WCOMP [13] explorent de façon très différente les techniques de programmation par aspects logiciels pour conduire les phases de reconfiguration dynamique des systèmes.

Cependant, les reconfigurations dynamiques envisageables avec les approches précédentes restent limitées, parce que la notion de disponibilité dynamique des composants en est absente [12]. Ainsi l'ensemble des composants susceptibles de jouer un rôle dans les reconfigurations d'un logiciel doit être déterminé à la conception. Au contraire, la notion de disponibilité dynamique des composants implique qu'un système soit en mesure de découvrir dynamiquement de nouveaux composants et de faire face à la disparition imprévisible de certains d'entre eux en cours d'utilisation. Le concept de disponibilité dynamique est central dans les approches à services.

L'approche à services

Un service logiciel se résume à la définition contractuelle d'une interaction entre deux entités logicielles [7]. L'approche à services se distingue par la découverte des fournisseurs de services par leurs clients à l'exécution. Une liaison entre un client et un service est établie tardivement, c'est-à-dire à l'exécution, au moment où elle devient nécessaire, à la suite d'une demande explicite du client. Pour assurer l'indépendance technologique du client vis-à-vis du service qu'il utilise, l'interaction logicielle entre le client et le service s'exprime par un contrat rédigé dans un formalisme indépendant des technologies d'implémentation du client et du service.

Les *Services Web* [35] constituent le principal exemple d'application de l'approche à services. Dans cet proposition, une distinction est faite entre le *service*, qui existe sous la forme d'une spécification qui caractérise une fonction, et *l'agent*, l'entité logicielle qui implémente cette fonction. Typiquement, les Services Web utilisent les protocoles et formats WSDL (description d'interfaces programmatique), SOAP (format d'échange de messages), UDDI (découverte de services) pour la spécification des services, et laissent le libre choix de la technologie d'implémentation aux développeurs d'agent.

Ces agents de service peuvent être développés au dessus d'OSGi[4], un cadre technologique fondé sur Java. Dans OSGi, l'agent est empaqueté au sein d'une archive java (JAR) appelée *bundle*. OSGi utilise les bundles comme vecteur de déploiement de service : un bundle peut être installé à distance sur une passerelle d'accueil, il peut voir ses dépendances résolues automatiquement et être démarré à distance par un intégrateur humain ou un logiciel. OSGi offre au développeur les mécanismes minimaux pour la disponibilité dynamique des agents de service. Cependant, pour en tirer parti convenablement, la tâche est complexe pour le développeur. Pour lui faciliter le travail, [12] propose un premier « modèle à composants orientés services » en mixant l'approche à composant et l'approche à service.

Un composant orienté service devient une entité logicielle disponible dynamiquement. Chaque « composant à services » est pris en charge par un *gestionnaire d'instance* qui gère de façon autonome les aspects liés à la reconfiguration dynamique (établissement des liaisons entre les entités, découverte automatique de la création et destruction des liaisons). Cependant, dans le cadre de la plasticité des IHM, il est primordial que l'utilisateur puisse contrôler la reconfiguration dynamique. Or, l'autonomie du gestionnaire d'instance, qui empêche l'intervention d'un tiers dans ce processus, en est un obstacle. iPOJO dépasse cette limitation en permettant d'étendre la couverture fonctionnelle du gestionnaire d'instances [17].

Le gestionnaire d'instance d'iPOJO devient *conteneur* qui regroupe un ensemble extensible de *handlers* tels que chaque handler gère un aspect non-fonctionnel de l'exécution d'un composant (dépendance de services, offre de services, contrôle du cycle de vie). L'extensibilité des conteneurs fait d'iPOJO un bon candidat pour

[4] OSGi : voir http://www.osgi.org

l'implémentation de systèmes interactifs plastiques : il suffirait de développer un handler qui permettrait à un tiers, et notamment à une méta-ihm [14], de participer à la gestion de la reconfiguration dynamique du système.

iPOJO est une solution possible pour la plasticité des IHM. D'autres technologies à composants orientés services pourraient être adaptées à la plasticité résultant inévitablement à un foisonnement de solutions incapables d'interopérer. L'Ingénierie Dirigée par les Modèles (IDM) offre un moyen de s'affranchir de ces barrières technologiques.

L'ingénierie dirigée par les modèles

Issue du standard industriel MDA[5] (Model Driven Architecture) prôné par l'OMG, l'IDM en est une généralisation. Le MDA vise à séparer les modèles indépendants des technologies (PIM : Platform Independent Model) des modèles qui en dépendent (PSM : Platform Specific Model). Les modèles PIM sont transformables en modèles PSM. Ainsi il est possible de capitaliser des savoir-faires indépendamment des technologies et de réduire à des transformations de modèles l'application d'un savoir-faire sur une technologie particulière. L'IDM généralise cette approche en s'appuyant sur trois concepts essentiels (modèle, métamodèle, transformation) et trois relations (représente, est conforme à, est transformé en) [18] : Un modèle *représente* un système étudié ; ce modèle *est conforme à* un métamodèle ; une transformation *transforme* modèle en un autre modèle.

Appliquée à la question de la plasticité des IHM, cette approche peut être une réponse au problème de l'hétérogénéité des technologies : les concepts et mécanismes nécessaires à la plasticité peuvent être exprimés à un niveau d'abstraction indépendant des technologies et, par transformation, mis en correspondance avec les concepts et mécanismes propres à chaque technologie. S'affranchir des frontières technologiques de cette façon permettrait de construire des systèmes interactifs par assemblage des briques logicielles issues d'espaces technologiques différents.

En synthèse, l'IDM, les approches à services et à composants apportent chacun quelques solutions au problème de la mise en œuvre d'IHM plastiques, mais aucun de ces trois domaines pris séparément n'apporte de solution satisfaisante. Si, comme le montre la littérature, certaines combinaisons sont possibles, aucune des propositions existantes ne satisfait les requis posés par la plasticité des IHM. Notamment, en GL, l'utilisateur n'est pas un concept de première classe. Il en résulte qu'aucune des solutions proposées dans l'état de l'art n'est capable de prendre en compte l'utilisateur, ses objectifs et son niveau de disponibilité pour interagir avec le système. Ainsi, si le GL offre des solutions partielles intéressantes et pertinentes dans le cadre de la plasticité des systèmes interactifs, celles-ci doivent être adaptées pour satisfaire aux spécificités de l'ingénierie de l'interaction homme-machine. C'est ce que nous proposons dans la suite de cet article avec Ethylene conçu selon les principes directeurs suivants.

L'APPROCHE ETHYLENE EN TROIS PRINCIPES

L'état de l'art que nous avons réalisé au sein des communautés IHM, IA et GL, nous a permis d'identifier trois principes directeurs :

Premier principe : *les reconfigurations interne et externe doivent coopérer*. Une IHM peut être capable d'adaptations à certains contextes d'usage envisagés par son concepteur. Lorsque le contexte d'usage courant appartient à l'ensemble des contextes d'usage connus de l'IHM, c'est-à-dire prévus par le concepteur, l'IHM est capable de s'adapter convenablement par reconfiguration interne. En informatique ambiante, les contextes d'usage imprévus sont inévitables. L'IHM doit alors être adaptable par le biais d'une logique externe qui pourra décider de remplacer l'IHM courante par une autre plus adéquate. Qu'elle soit interne ou externe, le processus de reconfiguration doit être placé sous le contrôle de l'utilisateur.

Second principe : un système interactif est un graphe de modèles qui expriment chacun un aspect du système à différent niveau d'abstraction. Conformément à l'IDM, les modèles classiques en IHM (tâches, concepts, interfaces abstraites, interfaces concrètes, contexte d'usage, etc. [11]) sont reliés par des mises en correspondance et transformations qui ont également le statut de modèle. Ces modèles, établis à la conception, restent disponibles à l'exécution pour permettre des adaptations au niveau d'abstraction idoine.

Troisième principe : *le premier principe compense le second principe et vice-versa (composition hybride des principes 1 et 2)*. À un extrême, un système interactif est composé seulement d'entités logicielles produites par un concepteur humain. Ses capacités d'adaptation sont limitées aux mécanismes de reconfiguration interne de chaque entité et aux mécanismes de reconfiguration externe d'une infrastructure logicielle sous-jacente, si elle existe. À l'autre extrême, un système interactif n'est constitué que de modèles à plus ou moins haut niveau d'abstraction. Ces modèles sont transformés à la volée par une infrastructure logicielle comme MARA [31] pour adapter l'IHM, mais celle-ci est limitée aux IHM conventionnelles WIMP que les générateurs d'IHM actuels sont capables de produire. L'application du troisième principe permet d'envisager des systèmes interactifs hybrides constitués *en partie* d'IHM codée à façon par des concepteurs humains (comme l'entité Post-WIMP de PhotoBrowser qui permet l'interaction multi-point sur une table augmentée) *et en partie* d'IHM géné-

[5] MDA : voir http://www.omg.org/mda

rée à partir de modèles (comme les entités WIMP des fig. 1-b et 1-c).

Ethylene, un modèle à composants original issu d'une association particulière des approches à composants et à services, tire partie de l'IDM pour répondre à notre trois principes directeurs.

ETHYLENE : UN MODELE A COMPOSANTS POUR L'IHM

Ethylene, satisfait les propriétés suivantes :

(P1) *Les composants sont « exécutables » et/ou sont des « modèles »*. Cette propriété découle de notre troisième principe. Ethylene permet de manipuler de manière semblable des composants développés par des concepteurs humain sous la forme de code exécutable, et des composants existant sous la forme de graphe de modèles à différents niveaux d'abstraction.

(P2) *Les composants sont disponibles dynamiquement*. Nous qualifions Ethylene de modèle à composants dynamique car il est bâti autour de la notion de disponibilité dynamique des composants. Un système interactif construit avec des composants Ethylene est adaptable à l'exécution au moyen de composants logiciels découverts dynamiquement au gré des variations de la plate-forme de l'utilisateur. Inversement, un système interactif est adaptable à l'exécution si un ou plusieurs composants le constituant venait à manquer.

(P3) *Le modèle permet l'assemblage hétérogène de composants*. Ethylene n'est pas une nouvelle technologie à composants, mais un modèle de type PIM dont la vocation est d'être une abstraction commune à différentes technologies d'implémentation (par ex. iPOJO ou WCOMP) afin que celles-ci puissent coopérer au sein d'un même système interactif. Ethylene définit donc l'ensemble des concepts et relations nécessaires et suffisants pour un modèle à composants adapté au problème de la plasticité des IHM et définit également une API à haut-niveau d'abstraction qui modélise les aspects dynamiques du modèle.

(P4) *Le modèle assure l'emploi de technologies de communication hétérogènes au sein des assemblages*. Les aspects dynamiques d'Ethylene concernent non seulement la disponibilité dynamique des composants, mais également l'établissement dynamique des liaisons entre les composants. Ces liaisons peuvent être établies entre des composants issus d'espaces technologiques différents. Ethylene incarne cette notion de liaison dans le concept de *connecteur*. Le connecteur est une entité trans-espaces technologiques : c'est lui qui permet concrètement l'assemblage de deux composants issus de d'espaces technologiques différents.

(P5) *L'utilisateur contrôle les reconfigurations*. En IHM, il est souhaitable que l'utilisateur soit en mesure de garder la possibilité d'intervenir dans la reconfiguration de son système interactif. Parmi les API offertes par Ethylene l'une donne la possibilité à un tiers extérieur de contrôler les réorganisations d'assemblages.

Cycle de vie des composants Ethylene

Les propriétés P1 et P2 concernent directement le cycle de vie des composants. Celui d'Ethylene (voir fig. 2) s'inspire fortement de celui des bundles OSGi pour les aspects de disponibilités dynamiques et embrasse les trois phases clefs de la vie d'un composant : son développement, son déploiement et son exécution.

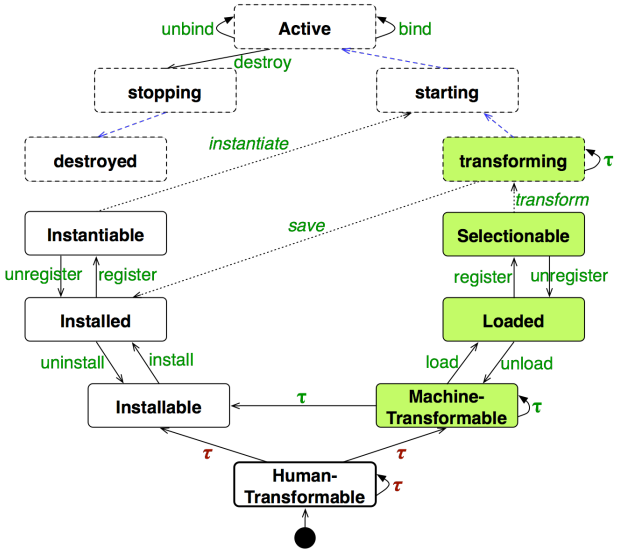

Figure 2 : Cycle de vie des composants Ethylene

Phase de développement. Qu'ils soient exécutables ou modèles, les composants ont le même état initial : ils existent sous la forme de spécifications établies par des concepteurs et sont transformables par des concepteurs *(*état *human-transformable)*. À partir de ces spécifications, il est possible de produire soit un composant exécutable (il est alors dans l'état *installable*) soit un composant composé uniquement de spécifications suffisamment formalisées pour être transformables par la machine (état *machine-transformable*).

Phase de déploiement. Un composant dans l'état installable ou machine-transformable est alors déployé. Lors de cette opération, un composant exécutable passe de l'état installable à l'état *installed*. Respectivement, un composant modèle passe de l'état machine-transformable à l'état *loaded*. La plate-forme logicielle qui les accueille est alors en charge de leur disponibilité dynamique qui est assurée par l'enregistrement de ces composants auprès d'annuaires. Cet enregistrement ef-

fectué, un composant exécutable passe à l'état *instantiable* et un composant modèle à l'état *sélectionable*. Un système interactif peut alors les découvrir et les utiliser.

Phase d'exécution. Lorsqu'un composant exécutable est recruté, il est instancié. Son instance s'initialise (état *starting*) et commence son exécution (état *active*). Pour être exécuté, un composant modèle passe par un état supplémentaire (*transforming*) qui vise à obtenir par transformation de modèle (τ) une instance d'IHM finale adaptée au contexte d'usage courant. Lorsqu'une instance de composant à l'état *active* n'est plus nécessaire, il est possible de la détruire. L'instance de composant passe par un état d'arrêt (*stopping*) avant destruction (état *destroyed*).

Ce cycle définit les étapes de la vie des composants Ethylene, et précise les opérations qui permettent de passer d'un état à un autre. En revanche, il ne décrit pas le détail des concepts mis en jeu ni leurs relations.

Concepts et relations du modèle

Les concepts et relations qui définissent le modèle Ethylene sont illustrés par le diagramme de classe (simplifié) de la figure 3.

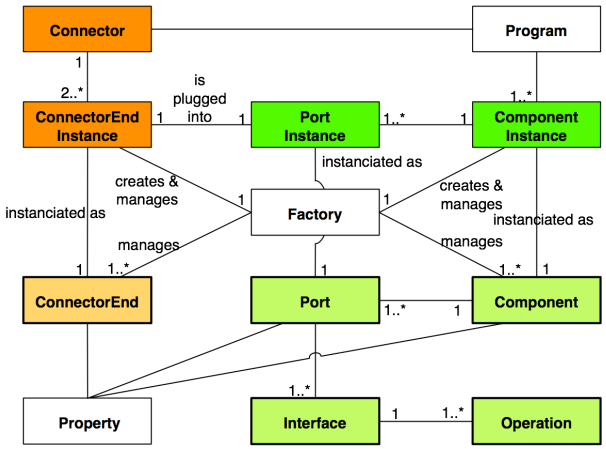

Figure 3 : Concepts et relations du modèle Ethylene.

Dans ce diagramme, un système interactif (*program*) est un assemblage d'*instances de composant*. Ces instances de composant sont reliées par *instances de port* au moyen de connecteur. Un connecteur est relié aux instances de port par des instances d'*extrémité de connecteur*. Les instances de composants, instances de port et instances d'extémités de connecteur sont les versions en exécution de leurs formes inertes que sont les *composants*, *ports* et *extrémités de connecteur*.

Un *composant* Ethylene est une entitée décrite par un ensemble de propriétés valuées. Cette entité peut-être constituée de code exécutable et/ou d'un ensemble de modèles à différents niveaux d'abstraction. Ses propriétés peuvent décrire, par exemple, la tâche utilisateur assurée par le composant (semblable aux COTs [Lewandowski07]), ses requis en termes de ressources d'interaction ou son coût physique et cognitif pour l'utilisateur. Afin d'être assemblé avec d'autres composants, un composant déclare un ensemble de ports.

Un *port* modélise l'offre d'un service logiciel par un composant ou son utilisation. Un port est défini par un ensemble d'interfaces programmatiques et de propriétés. Une interface programmatique se modélise comme un ensemble d'opérations. Le concept d'opération s'apparente à une méthode de classe (et ses paramètres d'appel et de retour), ou à un message (envoyé ou reçu). Les propriétés associées à un port qualifient le type de liaison qu'il requiert : par exemple, doit-elle être chiffrée, la bande passante doit-elle être élevée, la connexion doit-elle être fiable ? Ces propriétés sont destinées à guider, lors de l'établissement d'une liaison, le choix du connecteur à employer.

Un *connecteur* n'a pas de forme inerte qui lui correspond directement : il n'existe que par l'interconnexion d'au moins deux instances d'extrémités de connecteur. L'*extrémité de connecteur* est l'abstraction qui encapsule une technologie de communication particulière. Seules des extrémités de connecteur encapsulant la même technologie de communication sont connectables entre elles. Les caractéristiques de la technologie de communication encapsulée sont rendues explicites par des propriétés. C'est l'adéquation de ces propriétés avec celles d'un port qui permet de déterminer si une technologie de communication donnée est utilisable avec ce port. De la même manière que les composants, les extrémités de connecteur sont régies par un cycle de vie (voir fig. 4).

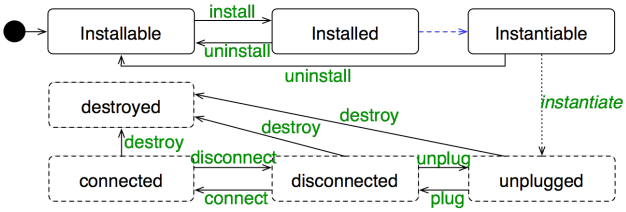

Figure 4: Cycle de vie d'une extrémité de connecteur

Au centre de ces concepts, celui de fabrique fait figure de pierre angulaire. La fabrique modélise une entité logicielle chargée de la gestion du cycle de vie des composants et des extrémités de connecteurs qui lui ont été associés lors d'opérations *install* (voir fig. 2 et 4). À ce titre, elle est en charge de l'enregistrements des composants auprès d'annuaires, de l'instantiation des composants et des extrémités de connecteurs, de leur destruction et de l'établissement de liaisons entre instances de composant.

Comme il y a peu de chance que deux fabriques hébergent exactement le même ensemble d'extrémités de connecteur, et donc de même technologies de communication, lorsqu'une liaison doit être établie entre des ins-

tances de composants issues de fabriques différentes, celles-ci s'engagent dans une phase de négociation afin d'élire, parmi les technologies de communication qu'elles ont en commun, celle qui correspond le plus précisément aux requis posés par les ports à relier. Ainsi, la fabrique est l'entité qui assure l'intéropérabilité entre les différents espaces technologiques.

De par ses fonctions, la fabrique se trouve à la frontière de deux espaces. D'un côté, elle interagit avec des composants et des extrémités de connecteurs qui s'inscrivent dans un espace technologique particulier pour lequel elle a été conçue. De l'autre côté, elle interagit avec des acteurs logiciels implémentés dans d'autres espaces technologiques : d'autres fabriques, des annuaires de composants et des entités en charge de contrôler les reconfigurations d'assemblage de composants. Pour que cela soit possible, une fabrique doit partager, avec ces autres acteurs, les API adéquates.

Les API Handling, Availability et Interoperability

Ethylene définit, à haut niveau d'abstraction, trois API asynchrones : *Handling*, *Availability* et *Interoperability*.

Handling. L'API Handling régit l'interaction logicielle entre une fabrique et une entité cliente chargée de piloter la reconfiguration d'un assemblage de composants. Handling décrit le service fourni par la fabrique.

```
PORT Handling
   IN : HandlingService
      Instantiate(      factoryId,
                        componentName,
                        instantationId);
      Bind(peers, bindingId);
      Unbind(connectorId);
      Destroy(instanceId);
   OUT : HandlingNotification
      ComponentInstantiationNotification(
                        instantiationId,
                        instanceId);
      BindingNotification(    bindingId,
                              connectorId,
                              errorCode);
```

Le client dispose de quatre opérations élémentaires : instancier un composant, le détruire, établir une liaison, la détruire. Pour toute instantiation, le client reçoit en retour l'identifiant unique correspondant à l'instance de composant ou de connecteur créée.

Availability. L'API Availability régit l'interaction logicielle entre une fabrique et un annuaire de composants. Availability décrit le service fourni par l'annuaire.

```
PORT Availability
   IN : DirectoryService
      Register(   factoryId,
                  componentName,
                  componentDescription);
      Unregister( factoryId, componentName);
   OUT : DirectoryNotification
      DirectoryHasArrived();
```

Une fabrique dispose de deux opérations élémentaires : enregistrer un composant et le désenregistrer des annuaires présents. Lorsqu'un nouvel annuaire arrive, il s'annonce aux fabriques déjà présentes par le biais du message *directoryHasArrived*. En réponse à ce message, les fabriques se doivent d'enregistrer auprès de lui les composants dont elles ont la charge.

Interoperability. L'API Interoperability régit l'interaction logicielle entre les fabriques. Son objectif est l'élection d'une technologie de communication lors de l'établissement d'une liaison entre des composants répartis entre plusieurs fabriques. Interoperability décrit le service fourni par la fabrique qui dirige l'élection.

```
PORT Interoperability
   IN : LeaderFactory
      SetBindingTechnologyList(
            bindingId,
            nbManagedPeers,
            technologyList);
      NotifyConnectorEndInstantiation(
            bindingId,
            nbManagedPeers,
            address,
            errorCode);
   OUT : SubordonateFactory
      InstantiateConnectorEnd(
            bindingId,
            peers,
            technology,
            address);
      NotifyConnectorExistence(
            bindingId,
            peers,
            connectorId);
```

Parmi les fabriques engagées dans la négociation, l'une est désignée dirigeante. Les autres, en fonction du nombre et de la description des ports à relier, construisent une liste des technologies de communcation adéquates dont elles disposent et l'envoient à la fabrique dirigeante. Celle-ci dispose alors des informations nécessaires pour faire le choix de la technologie de communication à mettre en œuvre. Elle précise alors aux fabriques subordonnées quelle extrémité de connecteur instancier pour cette liaison. Celles-ci s'exécutent et notifient la fabrique dirigeante. S'il n'y a pas eu d'erreur reportée lors de ce processus, la fabrique dirigeante confirme à toutes les autres l'existance du connecteur et précise son identifiant unique.

EthyleneXML et EthyleneFramework

La section précédente présente de façon synthétique le modèle Ethylene sur le plan conceptuel. Pour que celui-ci puisse effectivement être utilisé, il fallait lui fournir une incarnation technique. Celle-ci est double : d'une part, un langage XML, par définition indépendant des technologies d'implémentation, et d'autre part, un cadre de développement logiciel.

EthyleneXML[6]. Ce langage a pour objectif principal d'être un outil de description des composants. Cet outil trouve son utilité à la fois dans les phases de conception et d'exécution. À la conception, il sert à la spécification des composants. Interprétable par la machine, cette spécification peut alors servir à générer automatiquement une partie du code source d'un composant. À l'exécution, EthyleneXML est le format d'échange de description des composants. En particulier, c'est à partir d'une description EthyleneXML qu'une fabrique enregistre un composant au sein d'un annuaire ou négocie une technologie de communication à l'établissement d'une liaison.

EthyleneFramework. Ce cadre de développement a pour objectif de capitaliser une implémentation des mécanismes définis par le modèle Ethylene. Notamment, ce cadre inclut une implémentation générique du concept de fabrique et une implémentation simplifiée des concepts de composant, port et extrémité de connecteur. Actuellement disponible en C++ et en Java, ce cadre est exploitable « par héritage » : le développeur implémente des classes spécialisant à façon celles proposées par EthyleneFramework.

CONCLUSION ET PERSPECTIVES

Cet article s'intéresse à la nature logicielle des sytèmes interactifs plastiques dans le cadre de l'informatique ambiante. En nous appuyant sur un état de l'art multidisciplinaire, nous proposons Ethylene, un modèle à composants dynamiques indépendant des technologies d'implémentation, qui comble les lacunes de la litterature, au regard de nos requis pour les IHM plastiques en intelligence ambiante. Ethylene est décrit synthétiquement en termes de concepts et de leurs relations, de cycle de vie des composants et des connecteurs et en termes d'API qui modélisent les aspects dynamiques du modèle. Nous résumons ensuite les incarnations techniques d'Ethylene qui nous ont permis d'implémenter le démonstrateur PhotoBrowser.

Nos travaux actuels sur ce sujet visent à transformer ce cadre en une API C standard avec laquelle il sera possible d'écrire des *wrappers* pour un large éventail d'autres langages. Cette évolution nous permettra d'éprouver plus facilement notre solution sur de nouveaux prototypes. À plus long terme, il serait intéressant d'appliquer Ethylene à des technologie à composants existantes comme iPOJO ou WCOMP.

REMERCIEMENTS

Ce travail a bénéficié du soutien des projets FUI NOMAD et ANR CONTINUUM.

[6] Les spécifications d'EthyleneXML sont accessibles ici : http://iihm.imag.fr/namespaces/ethylenexml/1.0/

BIBLIOGRAPHIE

1. Allen, R., Douence, R., Garlan, D., "Specifying and analyzing dynamic software architectures", In: Proc. of FASE'98, Lisbon, Portugal, Springer LNCS 1382, p 21, March 1998.
2. Amigo WP2, "Specification of the Amigo Abstract Middleware Architecture", In: Deliverable D2.1, AMIGO Project (IST 2004-004182), April 11th, 2005.
3. Balme, L., Demeure, A., Barralon, N., Coutaz, J., Calvary, G., "CAMELEON-RT: a Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces", In: proceedings of SOC EUSAI 2004, p. 291-302, Eindhoven, Neederland, november 8-10, 2004.
4. Balme, L., "Interfaces homme-machine plastiques : une approche par composants dynamiques", In: PhD thesis, Université Joseph Fourier, Grenoble, France, June 2008.
5. Batista, T., Joolia, A., Coulson, R., "Managing dynamic reconfiguration in component-based systems", In: Proc. of EWSA 2005, p 1-17, LNCS 3527, Pisa , Italy, June 2005.
6. Bérard, F., "The GML canvas: Aiming at Ease of Use, Compactness and Flexibility in a Graphical Toolkit", Technical Report, 2006.
7. Bieber, G., Carpenter, J., "Introduction to Service-Oriented Programming (Rev2.1)", Online document **http://www.openwings.org**, April 2001.
8. Bouillon, L., "Reverse Engineering of Declarative User Interfaces", In: PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 2006.
9. Browne, D., Totterdell, P., Norman, M., "Adaptive User Interfaces", Computers And People series, Academic Press, ISBN 0-12-137755-5, 1990.
10. Brunneton, E., Coupaye, T., Stefani, J.B., "Recursive and Dynamic Software Composition with Sharing", In: Proc. of WCOP 2002, Malaga, Spain, June 2002.
11. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J., "A unifying reference framework for multi-target user interfaces", In: Interacting With Computers, Volume 15/3, pages 289-308, 2003.
12. Cervantes, H., "Vers un modèle à composants orienté services pour supporter la disponibilité dynamique", PhD Thesis, Université Joseph Fourier, Grenoble, Mars 2004.
13. Cheung-Foo-Wo, D., Blay-Fornarino, M., Tigli, J.-Y., Lavirotte, S., Riveill, M., "Adaptation dynamique d'assemblages de dispositifs dirigée par des modèles", Proc. of the 2nd Journées sur l'IDM, 2006.

14. Coutaz, J., "Meta-User Interfaces for Ambient Spaces", Invited Talk, In: Proc. of the TAMODIA 2006, LNCS 4385, p. 11-18, Hasselt, Belgium, 2006.

15. Dâassi, O., "Les comets : une nouvelle génération d'Interacteurs pour la Plasticité des Interfaces Homme-Machine", In: PhD thesis, Université Joseph Fourier, Grenoble, France, January 2007.

16. David, D., Ledoux, T., "Une approche par aspects pour le développement de composants Fractal adaptatifs", In: Revue des Sciences et Technologies de l'Information - L'Objet, Volume 12, numéro 2-3, p. 113-132, 2006.

17. Escoffier, C., Hall, R. S., "Dynamically Adaptable Applications with iPOJO Service Components", In: Proc. of SC 2007, Braga, Portugal, March 2007.

18. Favre, J.-M., "Towards a Basic Theory to Model Model Driven Engineering", WISME 2004, joint event with UML 2004, Lisboa, Portugal, 2004.

19. Florins, M., Vanderdonckt, J., "Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems", In: Proc. of IUI 2004, p. 140-147, Funchal, Madeira Island, Portugal, 2004.

20. Ganneau, V., Calvary, G., Demumieux, R., "Learning Key Contexts of Use in the Wild for Driving Plastic User Interfaces Engineering", In proc. of Engineering Interactive Systems 2008 (2nd Conference on Human-Centred Software Engineering (HCSE 2008) and 7th International workshop on TAsk MOdels and DIAgrams (TAMODIA 2008)), Pisa, Italy, September 2008.

21. Garlan, D., Monroe, R. T., Wile, D., "An Architecture Description Interchange Language", In: Proceedings of CASCON 1997, November 1997.

22. Grace, P., Blair, G. S., Samuel, S., " Middleware awareness in mobile computing", In: Proc. of ICDCS Workshop 2003, p. 382-387, May 2003.

23. Hnětynka, P., Plášil, F., "Dynamic Reconfiguration and Access to Services in Hierarchical Component Models", In: Proc. of CBSE 2006, LNCS 4063, p. 352-359, Västerås near Stockholm, Sweden, 2006.

24. Kurtev, I., Bézivin, J., Aksit, M., "Technological spaces: an initial appraisal", In: CoopIS, DOA'2002 Federated Conferences, Industrial track, Irvine, CA, USA, October 2002.

25. Lewandowski, A., Bourguin, G., Tarby, J.-C., "De l'Orienté Objet à l'Orienté Tâches – Des modèles embarqués pour l'intégration et le traçage d'un nouveau type de composants", In: RIHM, Vol 8, n° 1, p. 1-34, December 2007.

26. Lieberman, H., Espinosa, J., "A Goal-Oriented Interface to Consumer Electronics using Planning and Commonsense Reasoning", In: Proceedings of the 2006 International Conference on Intelligent User Interfaces, IUI 2006, p.226-233, Sydney, Australia, January 2006.

27. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V., "USIXML: A Language Supporting Multi-path Development of User Interfaces", In: Proceedings of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction, EHCI-DSVIS 2004, LNCS 3425, p. 200-220, Hamburg, Germany, July 2004.

28. Paternò, F., Santoro, C., "One Model, Many Interfaces", In: Proceedings of 4th International Conference on Computer-Aided Design of User Interfaces, CADUI 2002, Kluwer Academics Pub., Valenciennes, France, May 2002.

29. Roman, M., Kon, F., Campbell, R. H., "Reflective middleware : from your desk to your hand", In: IEEE Distributed Systems Online, Volume 2, Number 5, 2001.

30. Shaw, M., DeLine, R., Klein, D. V., Ross, T. L., Young, D. M., Zelesnik, G., "Abstractions for Software Architecture and Tools to Support Them", IEEE transactions on software engineering, Volume 21, Issue 4, p 314-335, April 1995.

31. Sottet, J. S., "Malléabilité des Interfaces Homme-Machine : Méga-IHM", In: PhD thesis, Université Joseph Fourier, Grenoble, France, Octobre 2008.

32. Szyperski, C., Gruntz, D., Murer, S., "Component Software: Beyond Object-Oriented Programming, Second edition", ACM Press, ISBN 0-201-74572-0, 2002.

33. Thevenin, D., Coutaz, J., "Plasticity of user-interfaces: framework and research agenda", In: Proceedings of 7th IFIP conference on human-computer interaction, INTERACT 1999, p. 110-117, Edinburgh, Scotland, September 1999.

34. Thevenin, D., "Adaptation en Interaction Homme-Machine : Cas de la plasticité", In: PhD thesis, Université Joseph Fourier, Grenoble, France, 2001.

35. W3C, "Web Services Architecture", In: W3C Working group Note, **http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/** , February 2004.

36. Wile, D. S., "Using Dynamic Acme", In: Proceedings of a Working Conference on Complex and Dynamic Systems Architecture, Brisbane, Australia, December 2001.