Vers une approche flot de données pour supporter la composition d'interfaces homme-machine

Christian Brel, Sébastien Mosser

Vers une approche flot de données pour supporter la composition d'interfaces homme–machine

Christian Brel[⋆] — **Sébastien Mosser**[†]

⋆*: I3S, UMR 6070 CNRS – Université Nice Sophia Antipolis*
†*: INRIA Lille–Nord Europe, LIFL, UMR 8022 CNRS – Université de Lille 1*

RÉSUMÉ. *Les approches «orientées services» permettent la création d'applications complexes par réutilisation et assemblage de services existant. Au niveau des interfaces hommes–machines, cette réutilisation n'est que peu supportée, obligeant les concepteurs d'interfaces à redéfinir complètement les interfaces des assemblages, sans pouvoir réutiliser les interfaces associées aux services élémentaires. Nous proposons dans cet article l'utilisation d'un méta-modèle de flot de données dédié à la composition, permettant une telle réutilisation.*

ABSTRACT. *Service–oriented approaches support the definition of complex systems through the reuse of existing services, as assemblies. Unfortunately, the human–computer interfaces associated to existing services cannot be easily reused. Designers must rewrite interfaces from scratch, without being able to reuse existing artifacts. In this paper, we propose to use a data–flow based meta–model to support such a reuse.*

MOTS-CLÉS : *Composition, Interface homme–machine, Flot de données*

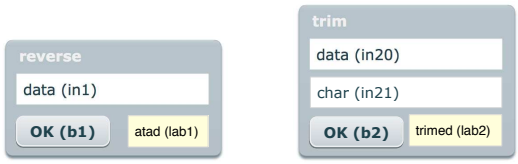
KEYWORDS: *Composition, Human–Computer Interface, Data–flow*

1. Introduction

L'avènement du Web 2.0 a transformé l'utilisateur, qui de simple consommateur passif est devenu producteur de contenu. Cette transformation s'est accompagnée d'une multiplication de services disponibles sur le Web. Des approches de composition supportent l'utilisateur dans la construction de nouveaux services complexes (*e.g.*, «*mashups*»), par assemblage de services existants (Merrill, 2006). Bien souvent pour développer une telle application, la construction de l'enchaînement des services et la construction de l'interface associée sont deux tâches séparées. Un des leviers permettant d'augmenter la rentabilité ainsi que la rapidité de développement, réduisant le «time–to–market» et préservant l'ergonomie et l'utilisabilité de ces applications est de favoriser la réutilisation de l'existant. L'objectif de cet article est d'esquisser une approche dirigée par les modèles permettant la définition d'opérateurs de compositions d'interfaces. Contrairement aux métamodèles usuels d'interfaces (qui mettent l'accent sur le positionnement des éléments, (Limbourg *et al.*, 2005)), nous proposons ici l'usage d'un métamodèle dédié à la composition[1], basé sur la modélisation du flot de données existant implicitement entre l'interface et le noyau fonctionnel.

2. Scénario d'illustration

Nous présentons dans cette section un cas d'utilisation typique de l'approche proposée, sur un exemple simplifié pour des raisons de concision. Nous considérons deux services existants (`reverse` et `trim`, implémentés par des services Web) permettant *(i)* de renverser une chaîne de caractères et *(ii)* de supprimer une lettre donnée dans une chaîne de caractères. Les interfaces associées sont représentées en figure 1. Pour `reverse` par exemple, l'activation du bouton «`OK`» provoque l'envoi de la donnée saisie vers le champ $data$ dans le service sous-jacent, et la donnée obtenue en retour est affichée dans le champ $atad$.



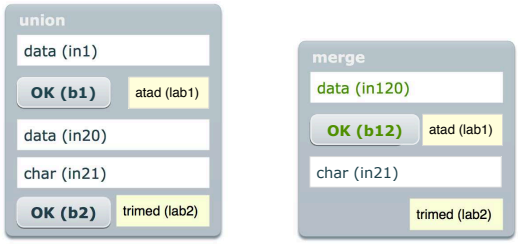
(a) `reverse` (b) `trim`

Figure 1. *Interfaces existantes, associées aux services `reverse` et `trim`.*

Sur la base de ces deux interfaces, plusieurs compositions automatiques peuvent être proposées à l'utilisateur. Nous considérons ici *(i)* l'`union` des deux interfaces, où

1. «*A model of a system is a description or specification of that system and its environment for some certain purpose.*» (Miller *et al.*, 2003)

les éléments pré-existant sont juxtaposés dans la même interface, et *(ii)* leur fusion (`merge`). Dans ce cas, les éléments «équivalents» (au sens d'une composition de modèle, *e.g.*, (Fleurey *et al.*, 2007)) sont unifiés dans l'interface composée. La figure 2 représente les interfaces obtenues à l'aide de ces opérateurs.



(a) `union` (b) `merge`

Figure 2. *Interfaces obtenues par composition des interfaces précédentes.*

3. Proposition de mise en œuvre

Dans nos précédents travaux (Brel *et al.*, 2010), nous proposons un métamodèle d'interfaces homme-machine, basé sur le métamodèle MARIA (Paternò *et al.*, 2009). Ce métamodèle permet de définir la structure de l'interface ainsi que le type d'intéracteur utilisé. L'utilisateur est alors au centre de la démarche de composition, sélectionnant graphiquement dans les interfaces existantes les éléments à réutiliser dans l'interface composée. Nous proposons ici l'adoption d'un point de vue orienté «composition», permettant l'écriture d'opérateurs de composition d'interfaces.

L'idée initiale est de considérer une interface du point de vue du flot de données intrinsèque qu'elle met en œuvre. En effet, lors de l'éxécution d'une action (*e.g.*, clic souris sur un bouton), les données saisies dans l'interface sont transmises au service sous-jacent. Après traitement par le service, le résultat de son exécution est retourné à l'interface, qui traite la réponse en fonction de la logique d'affichage. Ainsi, le flot de données d'une interface peut-être représenté sous la forme d'un graphe, où les noeuds représentent les éléments constituant le système (éléments d'interface, service) et où les arcs représentent les données transférées d'un élément à l'autre. Nous représentons en figure 3 les flots de données associés aux interfaces initialement présentées, sous la forme de graphes orientés. Pour `reverse`, lors du clic sur le bouton b_1, le texte saisi dans le champ in_1 est envoyé au service $reverse$ (paramètre $data$). Après traitement, la valeur du paramètre de retour $atad$ est alors retranscrite dans le champ lab_1.

Sur la base de cette modélisation, nous pouvons définir un opérateur de composition via la manipulation des graphes originaux. Ainsi, l'opération d'union présentée précédemment est immédiatement mise en œuvre par une «simple» union de graphes.

(a) `reverse` (graphe) (b) `trim` (graphe)

Figure 3. *Graphes associés aux interfaces* `reverse` *et* `trim`.

La définition d'opérateurs plus complexes nécéssite *de facto* la manipulation des éléments du graphe (Taentzer, 2004). Considérant la formalisation d'actions de manipulation atomiques (*i.e.*, add et $delete$, (Blanc *et al.*, 2008)), nous pouvons construire des macros–actions plus complexes telles que le replacement d'un nœud par un autre, ou encore l'unification d'un ensemble de nœuds. Ainsi, un opérateur de composition peut être défini comme un *générateur* d'actions, qui seront exécutées sur les graphes initiaux afin de produire le graphe composé (Mosser, 2010). Par exemple, l'opérateur de fusion présenté précédemment repose sur l'«unification» de plusieurs nœuds identifiés équivalents dans les interfaces existantes (ici, l'équivalence est simplement calculée via le nommage des éléments). Dans cet exemple, les deux unifications suivantes sont exécutées : *(i)* $(in_1, in_{20}) \rightsquigarrow in_{120}$ et *(ii)* $(b_1, b_2) \rightsquigarrow b_{12}$. Nous présentons en figure 4 le graphe obtenu après l'application de cet opérateur de composition sur les graphes des interfaces `reverse` et `trim`. Nous pouvons de plus réutiliser des travaux existants permettant d'analyser les actions produites, supportant l'utilisateur final lors de la composition : analyse de trace (Falleri *et al.*, 2006), detection d'interférence (Blanc *et al.*, 2009), ... Une implémentation préliminaire du canevas de composition présenté ici est disponible : `http://bit.ly/idm2011`.

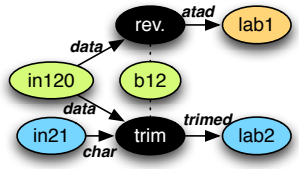


Figure 4. *Graphe obtenu par composition de* `reverse` *et* `trim` *(opérateur* $merge$*).*

4. Travaux connexes

Le framework de référence Cameleon (Calvary *et al.*, 2003) définit une conception des interfaces en 4 niveaux partant des arbres de tâches pour atteindre l'interface dite finale via la structure. Plusieurs travaux permettent la composition de la partie gra-

phique des applications suivant leur description structurelle (Lepreux *et al.*, 2006), où les auteurs proposent des opérateurs de composition basés sur la structure des interfaces. Dans (Gabillon *et al.*, 2008), l'expression en langage naturel de la composition souhaitée permettent de proposer à l'utilisateur toutes les solutions de compositions possibles, en utilisant des techniques de planification (le choix final est laissé à l'utilisateur). Contrairement à ces approches, d'une part nous gardons le lien entre l'interface et le noyau fonctionnel de l'application à travers le flot de données et d'autre part, l'utilisateur guidant la composition pas à pas, seule une application composée fonctionnelle est disponible à la fin de notre processus.

D'autres travaux composent la partie graphique des applications à travers l'arbre de tâches défini lors de la conception de cette interface (Feldmann *et al.*, 2009, Lewandowski *et al.*, 2007). Ces approches travaillent sur les arbres de tâches (attachées à des description d'interface faite avec MARIA (Paternò *et al.*, 2009) dans certains cas)afin de générer l'interface issue de l'expression de la composition des tâches. Pour cela, des opérateurs issus du monde des graphes sont proposés afin de rendre plus facile la manipulation à l'utilisateur. Une des limites des interfaces générées par ces approches est la non réutilisation des interfaces existantes, impliquant une perte de l'ergonomie de ces interfaces.

5. Conclusion

Notre approche repose ainsi sur les deux points suivants pour supporter la définition d'opérateurs de composition d'interface : *(i)* la métamodélisation des interfaces sous forme de flot de données (graphes) et *(ii)* la manipulation de ces artefacts au travers d'actions de sémantique connue (*e.g.*, une action d'unification de nœuds). Ainsi, la définition d'opérateurs de composition d'interfaces est assimilée à l'expression d'un enchaînement d'action sur les flots initiaux. De plus, contrairement aux approches basées sur des transformations de modèles «boite noire», le fait de considérer une composition d'interface comme la génération d'une séquence d'actions permet de ne plus différencier les compositions «algorithmiques» des compositions effectuées par un utilisateur au travers d'une méta–interface : dans les deux cas, nous obtenons une séquence d'actions de sémantique connue (qu'elle soit générée automatiquement ou obtenue en traçant les actions d'un utilisateur).

Sur la base de ces résultats préliminaires, nous envisageons de renforcer le lien entre services et interface via l'exploitation des arbres de tâches (graphe des actions utilisateurs et systèmes). En gardant les liens issus de la conception des interfaces entre tâches, structure de l'interface et flot de données, nos travaux futurs permettront de guider l'utilisateur dans les choix des opérateurs de compositions à appliquer.

6. Bibliographie

Blanc X., Mougenot A., Mounier I., Mens T., « Incremental Detection of Model Inconsistencies Based on Model Operations », *CAiSE*, p. 32-46, 2009.

Blanc X., Mounier I., Mougenot A., Mens T., « Detecting model inconsistency through operation-based model construction », *in* , W. Schäfer, , M. B. Dwyer, , V. Gruhn (eds), *ICSE*, ACM, p. 511-520, 2008.

Brel C., Renevier P., Occello A., Pinna-Déry A.-M., Faron-Zucker C., Riveill M., « Application Composition Driven By UI Composition », *Human Computer Software Engineering 2010 (HCSE 2010)*, 6409, IFIP, LNCS, p. 198-205, October, 2010.

Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L., Vanderdonckt J., « A unifying reference framework for multi-target user interfaces », *Interacting with Computers*, vol. 15, n° 3, p. 289-308, June, 2003.

Falleri J.-R., Huchard M., Nebut C., « Towards a Traceability Framework for Model Transformations in Kermeta », *ECMDA-TW Workshop*, 2006.

Feldmann M., Hübsch G., Springer T., Schill A., « Improving Task-driven Software Development Approaches for Creating Service-Based Interactive Applications by Using Annotated Web Services », *Proceedings of the 2009 Fifth International Conference on Next Generation Web Services Practices*, NWESP'09, IEEE Computer Society, Washington, DC, USA, p. 94-97, September, 2009.

Fleurey F., Baudry B., France R., Ghosh S., « A Generic Approach For Automatic Model Composition », *Aspect Oriented Modelling workshop at MoDELS*, 2007.

Gabillon Y., Calvary G., Fiorino H., « Composing interactive systems by planning », *Proceedings of the 4th French-speaking conference on Mobility and ubiquity computing*, UbiMob'08, ACM, Saint-Malo, France, p. 37-40, May, 2008.

Lepreux S., Vanderdonckt J., Michotte B., « Visual Design of User Interfaces by (De)composition », *Proceedings of the 13th International Conference on Interactive systems : Design, specification, and verification*, DSVIS'06, Springer-Verlag, Dublin, Ireland, p. 157-170, July, 2006.

Lewandowski A., Lepreux S., Bourguin G., « Tasks models merging for high-level component composition », *Proceedings of the 12th International Conference on Human-Computer Interaction*, HCI'07, Springer-Verlag, Beijing, China, p. 1129-1138, July, 2007.

Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., López-Jaquero V., « USIXML : A Language Supporting Multi-path Development of User Interfaces », *in* , R. Bastide, , P. Palanque, , J. Roth (eds), *Engineering Human Computer Interaction and Interactive Systems*, vol. 3425 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 134-135, 2005.

Merrill D., Mashups : The new breed of Web app—An introduction to mashups, Technical report, IBM, August, 2006.

Miller J., Mukerji J., MDA Guide Version 1.0.1, Technical report, OMG, 2003.

Mosser S., Behavioral Compositions in Service-Oriented Architecture, PhD thesis, University of Nice, Sophia–Antipolis, France, October, 2010.

Paternò F., Santoro C., Spano L. D., « MARIA : A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments », *ACM Transactions on Computer-Human Interaction*, vol. 16, n° 4, p. 19 :1-19 :30, November, 2009.

Taentzer G., « AGG : A Graph Transformation Environment for Modeling and Validation of Software », *in* , J. L. Pfaltz, , M. Nagl, , B. Böhlen (eds), *Applications of Graph Transformations with Industrial Relevance*, vol. 3062 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 446-453, 2004. 10.1007/978-3-540-25959-6_35.