HAL
open science

PLACID: un planificateur pour composer dynamiquement des Services IHM

Yoann Gabillon, Gaëlle Calvary, Humbert Fiorino

To cite this version:

Yoann Gabillon, Gaëlle Calvary, Humbert Fiorino. PLACID: un planificateur pour composer dynamiquement des Services IHM. IHM'14, 26e conférence francophone sur l'Interaction Homme-Machine, Oct 2014, Lille, France. pp.123-129, 10.1145/2670444.2670448 . hal-01090427

HAL Id: hal-01090427

https://hal.science/hal-01090427v1

Submitted on 3 Dec 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PLACID : un planificateur pour composer dynamiquement des Services IHM

Yoann Gabillon
CRP - Gabriel Lippmann
41, Rue du Brill
L-4422 Belvaux, Luxembourg
gabillon@lippmann.lu

Gaelle Calvary
Univ. Grenoble Alpes, LIG
F-38000 Grenoble, France
CNRS, LIG, F-38000
Grenoble, France
gaelle.calvary@imag.fr

Humbert Fiorino
Univ. Grenoble Alpes, LIG
F-38000 Grenoble, France
CNRS, LIG, F-38000
Grenoble, France
humbert.fiorino@imag.fr

RÉSUMÉ
La Composition Dynamique de Services a pour but de composer un système interactif à partir d'un ensemble de services disponibles correspondant à des composants. Un composant est constitué d'une partie fonctionnelle et d'une partie Interface Homme-Machine (IHM). En Génie Logiciel, la grande majorité de la littérature se concentre sur la composition dynamique de services fonctionnels. Si l'on fait l'hypothèse qu'un service IHM peut aussi être composé, cela entraine une nouvelle problématique de recherche en Interaction Homme-Machine : la Composition Dynamique de Services IHM. Cet article présente un algorithme de planification permettant de résoudre le problème de la composition dynamique d'IHM pour produire le modèle de tâches de l'IHM composée permettant à l'utilisateur d'atteindre son objectif.

ACM Classification Keywords
H.5.2 User Interfaces: Ergonomics, Graphical user interfaces (GUI), Prototyping, User-centered design.

General Terms
Design, Human factors, Algorithms

Mots Clés
Composition Dynamique, Interface Homme-Machine (IHM), service IHM, composition d'IHM, Formalisation, Algorithme

INTRODUCTION
La Composition Dynamique de Services (DSC pour Dynamic Service Composition) a pour objectif de composer un système interactif à partir d'un ensemble de services disponibles et d'un objectif utilisateur à satisfaire. Le processus DSC peut être décomposé en deux étapes. Premièrement, le plan (i.e. une sélection et un ordonnancement d'un ensemble de services) est calculé. Deuxièmement, les composants sont composés conformément au plan calculé pour produire un système interactif exécutable.

© ACM, 2014. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Actes de la 26ième conférence francophone sur l'Interaction Homme-Machine, 2014.
http://dx.doi.org/10.1145/2670444.2670448

Un système interactif comprend un noyau fonctionnel (FC pour Functional Core) et une partie Interface Homme-Machine (IHM) [5]. De la même manière, une distinction peut être effectuée entre service fonctionnel et IHM [20]. Les services fonctionnels sont fournis par des composants fonctionnels. Les services IHM sont fournis par les composants IHM. Si l'on fait l'hypothèse que les services IHM peuvent être composés comme les services fonctionnels, cela entraine une nouvelle problématique de recherche en Interaction Homme-Machine : la Composition Dynamique de Services IHM.

En Génie Logiciel (GL), la majeure partie des travaux s'intéresse à la composition dynamique de services fonctionnels. Le reste du temps, les services IHM sont soit considérés comme des services fonctionnels, soit l'IHM est composée dans un deuxième temps. Cependant, pour promouvoir une composition dynamique centrée utilisateur, les services IHM doivent aussi être composés dynamiquement pour produire un plan d'exécution.

L'article présente comme contribution principale la formalisation et l'implémentation d'un algorithme de planification pour produire un plan de service IHM sous la forme d'un arbre de tâches. En effet, la composition dynamique nécessite une technique d'Intelligence Artificielle (IA) pour sélectionner et ordonner les services à composer. En conséquence, nous avons utilisé les concepts de planification automatique déjà utilisées en DSC pour produire un plan sous la forme d'un modèle de tâches. L'algorithme PLACID (pour « PLAnner for dynamically Composing user Interfaces Descriptions ») permet de calculer un plan de services IHM. PLACID est intégré dans le prototype COMPOSE qui compose dynamiquement une IHM à partir de ce plan.

ÉTAT DE L'ART ET POSITIONNEMENT
La Composition de Services peut être *statique* ou *dynamique* [3]. La composition statique débute à partir d'un plan (un schéma de composition de services) pour produire un système composé à partir de composants préexistants alors que la composition dynamique a pour objectif de calculer ce plan à partir d'un objectif utilisateur et de serices disponibles [3]. La DSC repose sur l'hypothèse que chaque service peut être vu comme une action décrite par ses préconditions, postconditions, et effets [22]. Ces actions sont utilisées pour calculer un plan selon le contexte d'usage courant (propriétés concernant l'utilisateur, sa plate-forme et son environnement [24]).

Cette section propose une vue d'ensemble des techniques d'IA utilisés en GL et en IHM pour composer dynamiquement des services fonctionnels et IHM.

Génie Logiciel

La Composition Dynamique de Services (web services) nécessite une technique de l'IA pour calculer un plan. Par exemple, la Planification Automatique [9] développe des algorithmes pour atteindre un objectif à partir d'actions disponibles. En GL, différentes approches (techniques d'IA) ont été explorées pour calculer un plan de services comme le Calcul de Situations, la Planification à base de Règles, la Preuve de Théorème ou la Planification Automatique [22].

Calcul de Situations. Par exemple, [14] adopte le langage Golog pour calculer automatiquement un plan qui est une séquence d'actions. Golog est un langage de programmation qui représente les changements ou les évolutions de situations, actions ou objets. L'objectif utilisateur est exprimé en prédicats de la logique de premier ordre. Les services sont des actions. À partir de règles logiques et de contraintes, le plan est calculé à l'exécution.

Planification à base de Règles. Par exemple, [15] calcule un service composite à partir de règles de composabilité pour vérifier si les services peuvent être composés ou non. Premièrement, à partir d'une description de haut niveau, l'ensemble des compositions possibles est décrit. Deuxièmement, à partir des règles de composition, les descriptions de haut niveau sont composées. Troisièmement, les services sont sélectionnés à partir des descriptions pour obtenir un plan.

Preuve de Théorème. Par exemple, [21] utilise un prouveur de théorèmes. Les services disponibles et l'objectif utilisateur sont traduits en prédicats du premier ordre. Puis le plan est calculé par le prouveur de théorèmes qui utilise ces prédicats.

Planificateur PDDL. Classiquement, un planificateur PDDL manipule trois entrées : un but, (ex : « J'ai besoin d'aller de mon hôtel au bureau du médecin. »), l'état initial du monde (ex : Victor est à Philadelphie) et un ensemble d'actions (ex : voyager en voiture). Une action est définie par des préconditions (un ensemble de prédicats devant être vrais) et les effets sur l'état du monde (ex : pour l'action « travel by car », les effets seront que Victor sera dans le bureau du médecin). En retour, les algorithmes de planification automatique calculent un plan (i.e., une séquence d'actions) pour atteindre le but. Par exemple, [12] utilise un planificateur PDDL pour produire un plan de services permettant à l'utilisateur d'atteindre son objectif.

Planificateur HTN. Hierarchical Task Network (HTN) [6] est une approche hiérarchique de planification automatique. En HTN, les dépendances entre actions peuvent être données sous la forme d'un réseau. En conséquence, HTN permet l'utilisation de descriptions hiérarchiques pour décrire les actions abstraites (appelée méthodes) en sous-actions pour obtenir des actions élémentaires (i.e., correspondant aux actions PDDL). Une méthode a des préconditions et un ensemble de sous-actions. Par exemple, la méthode « voyager » peut être décomposée en deux sous-actions « voyager en voiture » ou « voyager en avion ». À partir d'un état initial, un planificateur HTN essaye d'appliquer les méthodes et les actions jusqu'à ce que l'objectif utilisateur soit réalisable par une séquence d'actions. Par exemple, [23] utilise un planificateur HTN comme JShop [6] pour produire un plan à partir d'une hiérarchie d'actions disponibles.

De même, le projet européen ATRACO (Adaptive and Trusted Ambient Ecologies) [1] utilise un planificateur HTN pour permettre de décrire une hiérarchie de tâches pour différencier les tâches pouvent être exécutées en parallèle ou en séquence. En effet, ce planificateur utilise aussi des concepts de Partial-Order Planning pour produire un plan d'actions concrètes partiellement ordonnées. Ainsi, il est possible de composer des services IHM en parallèle, par contre, seules les actions apparaissent dans le plan (les méthodes sont omisent).

Pour produire une IHM, la composition de composants IHM peut également être effectuée après la DSC. L'idée est de tirer avantage du plan calculé dynamiquement pour composer, dans un second temps, les IHM des services. Par exemple, [7] propose de composer les IHM des services sélectionnés dans une même fenêtre. De cette manière, l'IHM composée se déduit du plan de services fonctionnels.

CRUSe [20] introduit la notion de services IHM. Il s'intéresse à la composition de composants IHM à partir de services IHM en proposant une architecture système. Cependant, CRUSe ne se concentre pas sur la sélection des services IHM (i.e. le calcul du plan). En effet, il suppose le plan déjà calculé.

Interaction Homme-Machine

À haut niveau d'abstraction, une IHM est modélisée par un *modèle de tâches* [18, 4]. Le modèle de tâches (TM pour Task Model) est une description récursive de la *tâche utilisateur* en sous-tâches parallèles ou séquentielles jusqu'à obtenir des *tâches élémentaires* (i.e., des tâches qui ne peuvent être décomposées qu'en actions physiques). Par exemple, pour obtenir une assistance médicale « get medical assistance », l'utilisateur doit appeler le médecin « call the office » et suivre le trajet pour s'y rendre « find route information ». Une tâche peut être vue comme un *service IHM* fourni par un composant IHM. En conséquence, pour composer dynamiquement un nouveau service IHM, le plan calculé doit être un arbre de tâches au lieu de seulement une séquence. La correspondance entre les services et les actions est effectuée à l'aide d'ontologies [10].

Une IHM adaptative est une IHM capable de s'adapter ou d'être adaptée au contexte d'usage courant. Beaucoup de travaux se concentrent sur l'adaptation d'une IHM à l'exécution [4, 16]. Cependant, tous ces travaux démarrent à partir d'un modèle de tâches complet. Or le modèle de tâches, i.e., le plan en termes de services IHM, doit être calculé à l'exécution.

Huddle [17] utilise un planificateur PDDL pour produire une séquence d'actions qui représente une configuration

valide des composants IHM à composer. L'objectif est de générer automatiquement une IHM. Cependant, le planificateur ne manipule pas d'actions abstraites à l'instar d'un planificateur HTN.

Paterno et al. [19] montrent comment produire dynamiquement une IHM à partir de services Web. En particulier, les auteurs présentent comment créer une description abstraite d'une IHM à partir du modèle de tâches lié avec des services Web existants. L'objectif est de gérer l'implémentation de l'IHM. De la même manière, Kritikos et al. [13] n'utilisent pas un planificateur mais supposent un modèle de tâches complet pour produire un plan qui décrit comment les services existants peuvent être composés pour générer l'IHM. En conséquence, ces deux travaux [19, 13] ne se concentrent pas sur l'algorithme pour calculer le plan.

La composition par composants IHM se base sur l'hypothèse que le schéma de composition (le plan en termes de services) est connu et propose des frameworks ou opérateurs pour aider le concepteur à produire une IHM composée. Par exemple, [11] propose de composer des composants IHM conformément au schéma de composition fonctionel et propose une formalisation de cette problématique. De manière analogue, [2] compose des composants IHM décrits et annotés à l'aide d'ontologies.

Positionnement

La contribution de cet article porte sur un algorithme permettant de calculer un plan de services IHM. Les IHM étant modélisées à plus haut niveau d'abstraction en tâches utilisateur, le plan qui décrit une IHM composée doit être un modèle de tâches comportant des tâches en parallèle et en séquence. Les travaux portant sur la composition dynamique produisent un plan sous la forme d'une séquence d'actions. A notre connaissance, seul le plannificateur du projet ATRACO[1] produit un plan partiellement ordonné. Cependant, ce plan ne comportent pas les méthodes (actions arbtraites) fournies en entrée. En conséquence, nous proposons de manière originale un algorithme de planification permettant de produire un plan sous la forme d'un modèle de tâches.

FORMALISATION DU PROBLÈME ET DE LA SOLUTION

Formalisation du problème

Le principe est de produire une IHM dont la fonction est de faire passer l'utilisateur de sa situation courante à une situation dans laquelle son objectif est satisfait. Plus formellement, la situation courante (le contexte d'usage courant) est l'*état initial* s_0 d'un graphe d'état. Un *état* est représenté par un ensemble de propositions logiques. Ici, $s_0 = \{has(Victor, Smartphone), internet(Smartphone), at(Victor, Philadelphia), Guidance_Prompting(), \ldots\}$

De la même manière, l'*objectif de l'utilisateur* est représenté par un ensemble de propositions g. L'état contenant g est appelé l'*état but* s_g : $s_g = \{found(Victor, medical_assistance), \ldots\}$.

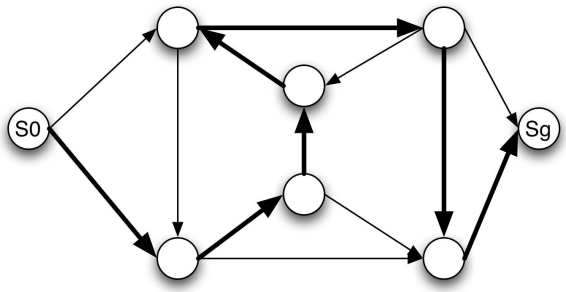

Figure 1. Graphe des changements d'état.

Le *graphe d'état* est un graphe des changements d'état dont les nœuds sont les états et les arcs sont les actions qui permettent à l'utilisateur de changer d'état à l'aide d'une IHM (figure 1).

Une *action* a est définie par le triplet $(precond(a), add(a), del(a))$ où $precond(a)$ est un ensemble de propositions logiques qui représente les conditions d'application de l'action : une action est applicable si et seulement si l'état courant contient $precond(a)$. Nous notons $precond^+(a)$ et $precond^-(a)$ respectivement les propositions logiques positives et négatives. $add(a)$ et $del(a)$ sont les effets respectivement ajoutés et supprimés à l'état courant. Si une action a est applicable, l'état suivant peut être calculé par :
$\gamma(s_i, a) = (s_i - del(a)) \cup add(a) = s_{i+1}$

En conséquence, le problème est de construire une IHM qui permet d'atteindre l'état but à partir de l'état initial. Soit π une liste d'actions de longueur k et s l'état courant. L'état suivant est défini par :

$$\gamma(s, \pi) = \begin{cases} s, & \text{si } k = 0 \\ \gamma(\gamma(s, a_1), [a2, \ldots, a_k]), & \text{si } k > 0 \text{ et} \\ & \text{a}_1 \text{ est applicable} \\ & \text{à } s \end{cases}$$

En pratique, l'utilisateur exprime ses besoins/son objectif. Ils sont traduits en un ensemble de propositions logiques g. Le système perçoit le contexte d'usage courant et le représente sous la forme d'un ensemble de propositions logiques s_0. Par exemple, le fait qu'un utilisateur nommé Victor possède un mur numérique est exprimé par la formule logique $has(Victor, Smartphone)$. De la même manière, l'ensemble des informations caractérisant le contexte d'usage courant sont traduites en propositions logiques. L'algorithme de planification connait les actions que l'utilisateur peut accomplir grâce aux IHM disponibles. Les actions sont définies en « intention » sous la forme d'*opérateurs* et de *méthodes*.

Un *opérateur* o est de la forme $(precond(o), add(o), del(o))$. Il est défini par ses préconditions qui sont un ensemble de prédicats en logique du premier ordre. $add(a)$ et $del(a)$ sont respectivement les effets ajoutés (supprimés) à l'état courant. Par exemple, les préconditions de l'opérateur $Call_the_office(?u, ?p)$ peuvent être les prédicats « $has(?u, ?p)$ »et « $internet(?p)$ » ; l'utilisateur $?u$ et

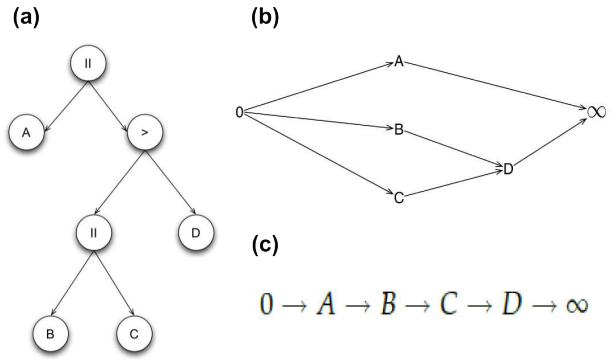

Figure 2. (a) Modèle de tâches calculé par le planificateur ; (b) La projection ρ du modèle de tâches ; (c) linéarisation.

la plate-forme $?p$ sont les variables du domaine. Donc, l'opérateur « $Call_the_office(?u, ?p)$ »peut représenter les actions « $Call_the_office(Victor, Smartphone)$ », « $Call_the_office(Victor, Wall)$ », etc.

Une *méthode* m est de la forme $(precond(m), type(m), subAct(m))$ où $precond(m)$ définit ses préconditions et $subAct(m)$ un ensemble d'opérateurs ou de méthodes à accomplir pour réaliser m. Par exemple, une méthode « se déplacer en taxi »peut se décomposer en trois opérateurs « appeller le taxi », « prendre le taxi », et « payer le taxi ». Le type de la décomposition est spécifié dans $type(m)$ qui est soit une *séquence*, soit un *parallélisme*. En conséquence, les opérateurs ou méthodes de $subAct(m)$ sont totalement ordonnés. S'ils sont en parallèle, les éléments de $subAct(m)$ peuvent être réalisés peu importe l'ordre. Par exemple, la méthode « $Get_medical_assistance$ » se décompose en parallèle par « $Call_the_office$ » et « $Find_route_information$ ».

Formalisation de la solution

À partir de l'état initial s_0, de l'objectif utilisateur g et d'un ensemble de méthodes et d'opérateurs, l'algorithme de planification calcule un modèle de tâches qui permet à l'utilisateur d'atteindre son objectif. Ce modèle de tâches est un arbre. Chaque nœud correspond à un composant IHM existant. Par exemple, la tâche « $Call_the_office$ » correspond à un composant IHM permettant d'appeller quelqu'un. Une fois calculé, le modèle de tâches permet de savoir les composants IHM à composer et comment. Par exemple, le composant « $Get_medical_assistance$ » contiendra les composants « $Call_the_office$ » et « $Find_route_information$ ».

Le modèle de tâches contient deux types de nœuds : interne ou feuille. Un nœud interne est une méthode complètement instanciée (toutes les variables sont liées avec une constante) de type séquence (« > ») ou parallélisme (« II »). Une feuille est une action, i.e. un opérateur totalement instancié (figure 2a). Nous appelons projection ρ du modèle de tâches un ensemble partiellement ordonné d'actions (figure 2b). La relation d'ordonnancement « $C > D$ » exprime que l'action C précède l'action D. 0 et ∞ sont les actions formelles qui représentent le début et la fin de l'interaction. La linéarisation de ρ est un ensemble composé des mêmes actions complètement ordonnées par la relation de précédence. Un exemple de linéarisation est présenté dans la figure 2c. L(ρ) est l'ensemble de linéarisations de ρ. Le problème de la composition est formalisé par : $P = (s_0, g, A)$ où s_0 est l'état initial, g l'objectif et A un ensemble d'actions possibles (en pratique exprimé sous la forme d'opérateurs et de méthodes). Résoudre ce problème revient à trouver une solution ρ (en pratique un modèle de tâches) tel que : $\forall \pi \in L(\rho), g \subseteq \gamma(s_o, \pi)$

Intuitivement, pour trouver une solution, il est nécessaire de calculer l'ensemble des linéarisations (toutes les séquences d'actions) qui permettent à l'utilisateur d'atteindre son objectif. En effet, une linéarisation correspond à une utilisation possible des actions (tâches) de l'IHM composée. Pour qu'une IHM composée soit solution, il faut qu'elle permette à l'utilisateur de réaliser la séquence (linérisation) qu'il souhaite. Par exemple, deux actions « $a1 = Call_the_office$ » et « $a2 = Find_route_information$ » peuvent être exécutées en parallèle si l'utilisateur peut exécuter les deux linérisations suivantes : ($a1$ puis $a2$) et ($a2$ puis $a1$). En conséquence, l'algorithme doit vérifier que toutes les linérisations sont solutions pour que le modèle de tâches le soit. Dans le cas où aucun plan n'est trouvé, le problème n'a pas de solution et aucune IHM ne peut être composée.

L'ALGORITHME PLACID

Nous proposons l'algorithme PLACID pour résoudre le problème de la Composition Dynamique de Services IHM. L'objectif de l'algorithme est de réaliser l'objectif de l'utilisateur (la tâche but).

Quelques notations sont d'abord introduites pour décrire l'algorithme PLACID.

Notations

Une tâche peut être une méthode ou un opérateur. La racine du modèle de tâches (plan) à calculer est appelée la *tâche but* t_u. En pratique, la projection ρ est un ensemble d'actions représentant le modèle de tâches. $\rho(t_u)$ est la projection de racine t_u où t_u est la tâche but.

Une fois $\rho(t_u)$ calculé, PLACID est capable de calculer les préconditions et les effets de t_u. Plus formellement, pour calculer les préconditions et les effets de la tâche t_u, deux cas sont à considérer si t_u est une action ou une méthode.

- Cas 1 (t_u est une action a) : si $\rho(t_u) = \{a\}$ est solution, alors $precond(t_u) = precond(a)$, $add(t_u) = add(a)$ et $del(t_u) = del(a)$.
- Cas 2 (t_u est une méthode m) : soit $t_1, \ldots, t_n$ les sous-tâches de t_u dans $\rho(t_u)$. Si $\rho(t_u)$ est une solution, alors $precond(t_u) = precond(t_1) \cup \cdots \cup precond(t_n)$, $add(t_u) = add(t_1) \cup \cdots \cup add(t_n)$ et $del(t_u) = del(t_1) \cup \cdots \cup del(t_n)$. Intuitivement, les préconditions (resp. effets) de t_u sont l'union des préconditions (resp. effets) de ses sous-tâches.

Ainsi, l'algorithme va diviser le problème en sous-problèmes grâce aux méthodes disponibles. Une fois que l'algorithme a trouvé que toutes les sous-tâches sont des solutions, il doit vérifier que les sous-tâches puissent être appliquée soit en séquence soit en parallèle.

Pour vérifier si deux tâches peuvent être composées en parallèle, nous adaptons la notion de conflit déjà utilisée en planification PDDL. Si deux tâches ne sont pas en conflit, elles sont indépendantes. Plus formellement, deux tâches t_1 et t_2 sont indépendantes si et seulement si :
- $del(t_1) \cap (precond^+(t_2) \cup add(t_2)) = \varnothing$ et
- $add(t_1) \cap (precond^-(t_2) \cup dell(t_2)) = \varnothing$ et
- $del(t_2) \cap (precond^+(t_1) \cup add(t_1)) = \varnothing$ et
- $add(t_2) \cap (precond^-(t_1) \cup dell(t_1)) = \varnothing$.

Intuitivement, deux tâches sont indépendantes si les effets d'une tâche ne sont pas en conflit avec les préconditions de l'autre. Si les tâches en parrallèle sont applicables et ne sont pas en conflit, alors t_u est réalisable.

Algorithme

L'algorithme adopte la stratégie de diviser pour régner (divide and conquer). En effet, en partant de l'objectif de l'utilisateur (t_u), l'algorithme va récursivement décomposer cette tâche en sous-tâches grâce aux méthodes diponibles pour obtenir un arbre où les feuilles sont des actions. Si toutes les tâches sont applicables et que les tâches parallèles ne sont pas en conflit, alors l'algorithme trouve une solution : le modèle de tâches permettant à l'utilisateur d'atteindre son objectif.

Soit $P = (s_0, t_u, O, M)$ un problème où s_0 est l'état initial, t_u est la tâche but à effectuer (une tâche est une méthode ou une action, par ex : « get medical assistance »), O et M sont les actions et les méthodes disponibles. Pour calculer l'ensemble des linéarisations (une solution), l'algorithme cherche une linéarisation et vérifie s'il n'y a pas de conflit entre les tâches en parallèle. Plus formellement, $\rho(t_u)$ est une solution à P si et seulement si :
- Cas 1 (t_u est une action a) : a est applicable à s_0. Alors $\rho(t_u) = \{a\}$.
- Cas 2 (t_u est une méthode complètement instanciée m) : soit $\{t_1, \ldots, t_n\} \in subAct(m)$; il y a deux cas si la méthode est de type séquence ou parallélisme :
  - Cas 2a (m est de type séquence) : m est applicable à s_0 et $\forall t_i \in subAct(m)$, $\rho(t_i)$ est solution à $P_i = (\gamma(s_0, t_{i-1}), t_i, O, M)$ avec $\gamma(s_0, t_{i-1}) = s_0$ si $i = 1$. Intuitivement, si t_u est une séquence, une solution est une linéarisation classique de ses sous-tâches.
  - Cas 2b (m est de type parallélisme) : m est applicable à s_0 et $\forall t_i \in subAct(m)$, $\rho(t_i)$ est solution à $P_i = (s_0, t_i, O, M)$ et $\forall t_i, t_j \in subAct(m), i \neq j, t_i$ et t_j sont indépendantes. Intuitivement, l'algorithme vérifie que toutes les sous-tâches en parallèle sont applicables à s_0 et qu'il n'y a pas de conflit entre elles. L'absence de conflit est vérifié pour permettre à l'utilisateur de sélectionner lui même l'ordre dans lequel il veut effectuer les sous-tâches.

IMPLEMENTATION SUR UN EXEMPLE APPLICATIF

PLACID est illustré sur un exemple applicatif : Victor a besoin d'une assistance médicale durant ses vacances. Premièrement l'exemple applicatif est présenté. Puis, l'implémentation des opérateurs et des méthodes nécessaires est décrite. Finalement, les spécificités de PLACID sont discutées.

Exemple applicatif

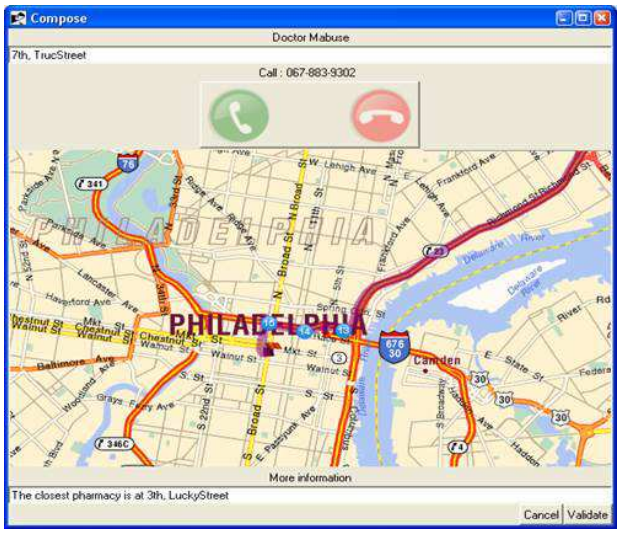

Figure 3. IHM composée pour que Victor ait une assistance médicale.

Victor vit à New York. Soudainement, il ne se sent pas bien alors qu'il est en vacances à Philadelphie. Il a besoin d'assistance médicale. Heureusement, il a un système pour l'aider. Il spécifie son objectif « Je veux une assistance médicale »sur le mur numérique de l'hôtel. En retour, PLACID compose dynamiquement un plan en termes de services IHM. Chaque service IHM décrit les services fournis par des composants disponibles. Le plan exprime donc l'IHM qui doit être composée. Par exemple, l'IHM composée permet d'abord à Victor de choisir son médecin. Une fois le médecin choisi, Victor peut appeler le médecin choisi « Dr Mabuse » (figure 3 ; le numéro est pré-composé – voir le haut de la fenêtre), être guidé jusqu'au cabinet médical (figure 3 ; voir la carte) et avoir l'adresse du pharmacien le plus proche (figure 3 ; voir l'adresse au bas de la fenêtre).

Implémentation

Pour implémenter cet exemple applicatif, il est nécessaire de disposer des composants IHM fournissant des services IHM décris par des opérateurs et des méthodes. Les opérateurs et méthodes suivants sont implémentés pour notre exemple :

Opérateur $Choose_the_doctor$ **:** les paramètres sont un utilisateur ($?u$), une plate-forme ($?p$). Le service IHM requiert que l'utilisateur ait une plate-forme d'exécution : $has\ ?u\ ?p$. Une fois que l'utilisateur a sélectionné le médecin, un prédicat est ajouté à l'état du monde : $isChosen\ ?u\ d$.

Opérateur $Call_the_office$ **:** les paramètres sont un utilisateur ($?u$), une plate-forme ($?p$) et une personne à appeler ($?pp$). Le service IHM requiert que la plate-forme ait des capacités téléphoniques : $isPhone\ ?p$. Le fait que la personne pp ait appelé quelqu'un est ajouté à l'état du monde : $isCalled\ ?u\ ?pp$.

Opérateur $Find_route_info$ **:** les paramètres sont un utilisateur ($?u$) et une plate-forme ($?p$). Le service IHM requiert que l'utilisateur ait une plate-forme connectée à Internet : $isInternet\ ?p$.

Opérateur $Find_nearest_chemist$ **:** les paramètres sont un utilisateur ($?u$), une plate-forme ($?p$) et un pharmacien ($?o$). Le service IHM requiert que l'utilisateur ait une plate-forme ($has\ ?u\ ?p$) et que l'adresse du pharmacien puisse être affichée sur cette plate-forme : $isDisplayable\ ?o\ ?p$. L'état est enrichi de : $isDisplayed\ ?p\ ?o$.

Méthode $Get_medical_assistance$ **:** les paramètres sont un utilisateur ($?u$) et une plate-forme ($?p$). Le service IHM requiert que l'utilisateur ait une plate-forme : $has\ ?u\ ?p$. Les sous-tâches sont « $Choose_the_doctor$ » et « $Contact_the_doctor$ » en séquence.

Méthode $Contact_the_doctor$ **:** les paramètres sont un utilisateur ($?u$) et une plate-forme ($?p$). Les sous-tâches sont « $Call_the_office$ », « $Find_route_info$ » et « $Find_nearest_chemist$ » en parallèle. Ce service IHM requiert que l'utilisateur ait une plate-forme d'exécution avec un écran large ($isLargeScreen\ ?p$).

A partir de ces opérateurs, ces méthodes et de l'objectif de l'utilisateur, l'algorithme calcule la solution (plan) présentée dans la figure 4. En effet, pour vérifier si la méthode « $Get_medical_assistance$ »est solution, l'algorithme vérifie si la linéarisation de « $Choose_the_doctor$ » et « $Contact_the_doctor$ » est solution. « $Choose_the_doctor$ » est applicable sur s_0 et produit s_1. Pour vérifier si la méthode « $Contact_the_doctor$ » est applicable sur s_1 et produit s_5, l'algorithme vérifie si les actions « $Call_the_office$ », « $Find_route_info$ » et « $Find_nearest_chemist$ » peuvent être appliquées sur s_1 et si les actions sont indépendantes. Si elles ne sont pas en conflit, s_5 peut être calculé à partir d'une linéarisation de ces actions.

Discussion

Une fois calculé, le plan peut être mis en correspondance avec les composants IHM. En effet, chaque service IHM est fourni par un composant IHM. Une méthode exprime une manière de composer les sous-composants IHM. Par exemple, la figure 5 affiche une IHM qui peut être composée à l'aide du plan de services IHM par le prototype COMPOSE [8]. Les composants correspondant aux tâches abstraites sont en charge de composer ses sous-composants. Par exemple, le composant permettant « $Contact_the_doctor$ » est une frame qui va composer verticalement les IHM de « $Call_the_office$ », « $Find_route_info$ » et « $Find_nearest_chemist$ ».

Par rapport au planificateur classique de services fonctionnels qui ne fournissent pas les méthodes en sortie, PLACID calcule un arbre contenant les méthodes et les actions. En effet, les méthodes expriment la composition, i.e., la manière de composer les composants IHM fournis par les services IHM. Par exemple, la méthode « $Contact_the_doctor$ » exprime une fenêtre qui va contenir verticalement les composants IHM fournissant les services IHM (opérateurs) « $Call_the_office$ », « $Find_route_info$ » et « $Find_nearest_chemist$ ».

Par rapport au planificateur classique de services fonctionnels, PLACID compose des services IHM en parallèle.

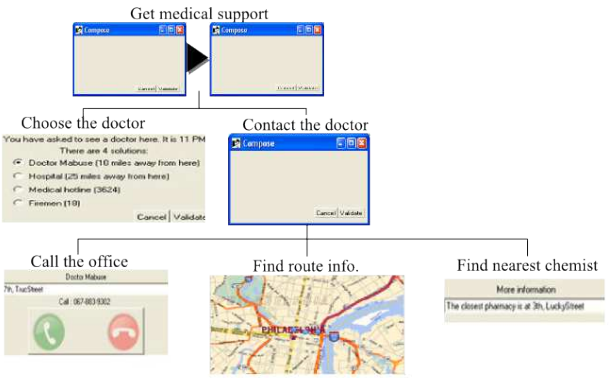

Figure 5. Composants IHM fournis par leur services IHM respectifs.

En effet, si deux services IHM ne sont pas en conflit, alors les deux composants IHM correspondant peuvent être composés dans la même temporalité (dans une même fenêtre ou onglet). Par exemple, un système de composition ne doit pas composer deux composants IHM qui nécessiteraient tous les deux les mêmes haut-parleurs.

Pour produire un système capable de composer dynamiquement une IHM, ce dernier doit disposer des composants IHM, les services IHM correspondant et d'un lien entre les deux. Un composant IHM peut être réutilisé pour fournir plusieurs services IHM. Par exemple, le composant IHM qui compose des IHM verticalement dans une même fenêtre sera souvent réutilisé.

Jusqu'à maintenant, PLACID a été appliqué à un exemple applicatif. Il est nécessaire d'étudier les performances de l'algorithme pour réaliser une composition d'IHM dynamique. Cette étude nécessite de réaliser des Benchmarks de services IHM qui n'existent pas actuellement.

Grâce à sa spécificité unique dans la littérature de retourner un arbre au lieu d'une séquence d'actions, PLACID peut également être utilisé à la conception comme générateur du modèle de tâches pour aider le concepteur à modéliser la tâche utilisateur. Cela est un intéressant effet de bord car les concepteurs d'IHM ne sont pas très familiarisés avec la modélisation en tâches utilisateur. En effet, la conception de TM est une tâche qui peut être accélérée et facilitée par l'utilisation de PLACID.

CONCLUSION ET PERSPECTIVES

Cet article a présenté une solution pour composer dynamiquement des IHM. La contribution principale est une formalisation de l'algorithme PLACID qui permet de calculer un plan à partir de services IHM. PLACID couvre la composition séquentielle et parallèle de services IHM (tâches) et qui produit un plan sous la forme d'un arbre de tâches. En effet, des services IHM peuvent être composés en séquence ou dans une même fenêtre. Pour vérifier si deux services peuvent être composés en parallèle, PLACID vérifie si les services IHM ne sont pas en conflit. Comme perspective, nous envisageons d'évaluer les performances de l'algorithme. Nous envisageons également d'évaluer PLACID comme outil d'aide à la conception rapide de modèle de tâches.

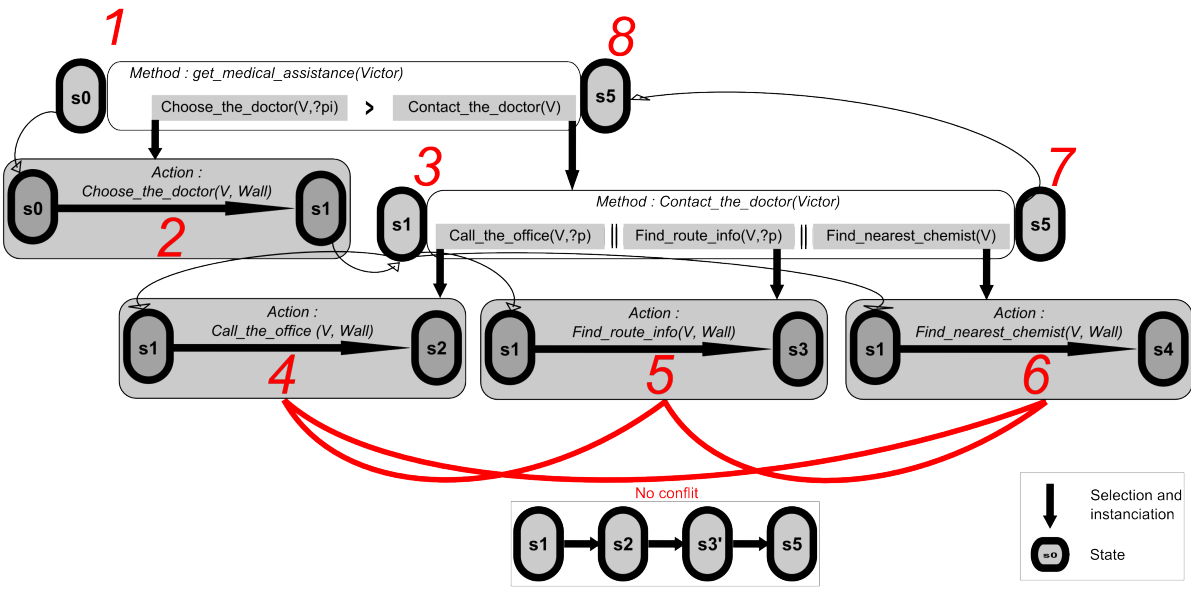

Figure 4. Exécution de PLACID sur l'exemple applicatif.

BIBLIOGRAPHIE

1. Bidot J., Goumopoulos C. & Calemis I. Using ai planning and late binding for managing service workflows in intelligent environments. In *Pervasive Computing and Communications (PerCom), 2011 IEEE International Conference on*, IEEE (2011), 156–163.
2. Brel C., Dery-Pinna A.-M., Renevier-Gonin P. & Riveill M. Ontocompo: a tool to enhance application composition. In *Human-Computer Interaction–INTERACT 2011*, Springer (2011), 588–591.
3. Bucchiarone A. & Gnesi S. A survey on services composition languages and models. In *in Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006)* (2006), 51–63.
4. Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L. & Vanderdonckt J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers 15*, 3 (2003), 289–308.
5. Coutaz J. Software architecture modeling for user interfaces. *Encyclopedia of software engineering* (1993).
6. Erol K., Hendler J. & Nau D. S. Htn planning: Complexity and expressivity. In *AAAI*, vol. 94 (1994), 1123–1128.
7. Feldmann M., Martens F., Berndt G., Spillner J. & Schill A. Rapid development of service-based interactive application using service ANNOTATIONS. 319–322.
8. Gabillon Y., Petit M., Calvary G. & Fiorino H. Automated planning for user interface composition. In *Proceedings of the 2nd International Workshop on Semantic Models for Adaptive Interactive Systems: SEMAIS'11 at IUI 2011 conference*, Springer HCI (2011).
9. Ghallab M., Nau D. & Traverso P. *Automated planning: theory & practice*. Morgan Kaufmann, 2004.
10. Heinroth T., Kameas A., Pruvost G., Seremeti L., Bellik Y. & Minker W. Human-computer interaction in next generation ambient intelligent environments. *Intelligent Decision Technologies 5*, 1 (2011), 31–46.
11. Joffroy C., Caramel B., Dery-Pinna A.-M. & Riveill M. When the functional composition drives the user interfaces composition: process and formalization. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, ACM (2011), 207–216.
12. Klusch M., Gerber A. & Schmidt M. Semantic web service composition planning with owls-xplan. In *Proceedings of the AAAI Fall Symposium on Semantic Web and Agents, Arlington VA, USA, AAAI Press* (2005).
13. Kritikos K., Plexousakis D. & Paternò F. Task model-driven realization of interactive application functionality through services. *ACM Transactions on Interactive Intelligent Systems (TiiS) 3*, 4 (2014), 25.
14. McIlraith S. & Son T. Adapting golog for composition of semantic web services. In *Principles of Knowledge representation and reasoning-Iinternational Conference*, Citeseer (2002), 482–496.
15. Medjahed B. *Semantic web enabled composition of web services*. PhD thesis, Faculty of the Virginia Polytechnic Institute and State University, 2004.
16. Motti V. G. & Vanderdonckt J. A computational framework for context-aware adaptation of user interfaces. In *Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on*, IEEE (2013), 1–12.
17. Nichols J., Rothrock B., Chau D. H. & Myers B. A. Huddle: automatically generating interfaces for systems of multiple connected appliances. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, ACM (2006), 279–288.
18. Paternò F., Mancini C. & Meniconi S. Concurtasktrees: A diagrammatic notation for specifying task models. In *Human-Computer Interaction INTERACT'97*, Springer (1997), 362–369.
19. Paternò F., Santoro C. & Spano L. D. Engineering the authoring of usable service front ends. *Journal of Systems and Software 84*, 10 (2011), 1806–1822.
20. Pietschmann S., Voigt M., Rümpel A. & Meißner K. Cruise: Composition of rich user interface services. In *Web Engineering*. Springer, 2009, 473–476.
21. Rao J., Küngas P. & Matskin M. Application of linear logic to web service composition. In *IWCS*, L.-J. Zhang, Ed., CSREA Press (2003), 3.
22. Rao J. & Su X. A survey of automated web service composition methods. *Semantic Web Services and Web Process Composition* (2005), 43–54.
23. Sirin E., Parsia B., Wu D., Hendler J. & Nau D. Htn planning for web service composition using shop2. *Web Semantics: Science, Services and Agents on the World Wide Web 1*, 4 (2004), 377–396.
24. Thevenin D. & Coutaz J. Plasticity of user interfaces: Framework and research agenda. In *Proceedings of INTERACT*, vol. 99 (1999), 110–117.