Composition et recomposition opportunistes : motivations et exigences

Charles Triboulot[1,2] Sylvie Trouilhet[2]

Jean-Paul Arcangeli[2] Fabrice Robert[1]

[1]Sogeti High-Tech
Aeropark - 3 chemin Laporte
31300 Toulouse
France
{prenom.nom@sogeti.com}

[2]Université de Toulouse
UPS - IRIT
118 Route de Narbonne
F-31062 Toulouse
France
{prenom.nom@irit.fr}

RESUME

Dans les approches traditionnelles de développement, les composants logiciels sont assemblés en fonction d'un besoin explicite. La composition est alors *ad hoc*, mais difficilement adaptable lorsque l'environnement du logiciel évolue, par exemple quand il intègre de nouveaux composants dynamiquement. La prise en compte de l'opportunisme, c'est-à-dire la capacité de composer et de recomposer des composants logiciels disponibles dans l'environnement mais qui n'ont pas été spécifiquement développés pour les applications qu'ils composent, permettrait l'émergence d'applications et ouvrirait ainsi de nouvelles perspectives en matière de conception et d'évolution de logiciels. Il est possible de recenser un certain nombre de situations de composition et de recomposition opportunistes. Ces dernières diffèrent entre elles par l'élément déclencheur qui peut être en lien avec la mobilité d'un composant, son état ou l'environnement. Ces situations peuvent se décliner dans de multiples scénarios, notamment dans le cadre ambiant, comme c'est le cas pour le scénario que nous présentons. Nous nous interrogeons sur les avantages d'une telle approche pour la flexibilité et la réutilisation, ainsi que sur les exigences que devra satisfaire l'infrastructure capable de la mettre en oeuvre : pour construire et maintenir de telles compositions, le système développé devra être distribué, autonome, adaptatif et capable de contrôler de manière décentralisée la combinatoire. Une étude approfondie des systèmes de composition automatique existants montre que peu de travaux portent sur ces approches *bottom-up* et qu'aucun ne propose une solution qui réponde à toutes les exigences et donc qui soit capable de composition opportuniste efficiente.

Mots-clés

Composants logiciels, composition automatique, approche ascendante, systèmes ambiants

1. INTRODUCTION

La rationalisation du développement de logiciels nécessite la prise en compte de problèmes liés d'une part à la productivité et la réutilisation, et d'autre part à l'évolution. Les approches à base de composants logiciels, comparées à celles basées sur le concept d'objet, facilitent la réutilisation et la composition : la formalisation des interfaces requises et fournies permet notamment un contrôle du couplage et simplifie l'assemblage des composants. Cependant, dans les approches traditionnelles, la composition est déclenchée par l'explicitation d'un besoin. Les composants sont sélectionnés un par un et composés afin de former une application qui réponde à ce besoin exprimé. Cette démarche, dans laquelle la satisfaction d'un besoin explicite est à l'origine de la composition, est dite *top-down*. Dans ce cadre, la réalité de la pratique conduit cependant à utiliser et à composer des composants existants, en les modifiant plus ou moins, dans des applications pour lesquelles ils n'ont pas été initialement conçus. Ainsi, les processus de développement traditionnels intègrent également une part de démarche ascendante ou *bottom-up*.

Une fois développés, ces logiciels sont plongés dans des environnements changeants : évolution des besoins, modification du contexte opérationnel... Ils doivent donc évoluer et, par conséquent, avoir été conçus pour que les évolutions puissent être apportées à moindre frais. Cela suppose, pour le concepteur, d'identifier *a priori* les aspects du logiciel qui peuvent varier. Il s'agit là d'une exigence majeure en terme d'architecture logicielle et d'une importante source de difficultés même si les *design patterns* [Gamma et al., 1995] apportent des solutions en matière de flexibilité. Par ailleurs, le concept de service apporte d'autres réponses en matière de sélection, de composition et d'utilisation, en particulier dans un contexte dynamique (on pourra se reporter à [Amirat et al., 2014] pour une comparaison des concepts d'objet, de composant et de service). À un autre niveau, les processus de développement itératifs et les méthodes agiles prennent

en compte les questions liées à l'évolution.

Face à ces problèmes de conception et d'évolution, nous imaginons une approche pour la fabrication de logiciels à base de composants, alternative à l'approche traditionnelle, dans laquelle ce n'est plus le besoin qui pilote la composition et l'évolution mais l'opportunité d'assembler (et de réassembler) des composants logiciels disponibles. C'est cette approche, délibérément *bottom-up*, que nous nommons « composition et recomposition opportunistes » et que nous explorons dans ce travail de recherche.

1.1 Composition et recomposition opportunistes en environnement ambiant

De manière générale, les logiciels sont de plus en plus souvent plongés dans des environnements dynamiques et changeants. Ils doivent continuellement satisfaire les besoins de leurs utilisateurs mais leur durée de vie est telle qu'il est de plus en plus difficile d'anticiper les évolutions. Ce problème se pose de manière aiguë pour les systèmes ubiquitaires et mobiles. L'intelligence de ces systèmes doit leur permettre, en fonction du contexte, d'agir de manière adaptée et de réagir à des situations, d'anticiper les besoins des utilisateurs et d'aller au delà d'un cadre comportemental défini à l'avance. Il s'avère que les approches traditionnelles répondent difficilement à cette exigence.

L'opportunisme désigne la capacité à assembler des composants logiciels disponibles dans l'environnement ambiant, mais qui n'ont pas été développés spécifiquement pour les applications qu'ils composent. Ainsi, la composition n'est pas dirigée par la satisfaction d'un besoin global exprimé mais par le contexte. Elle s'effectue localement lorsqu'une opportunité de composition se présente. Les composants ne sont plus sélectionnés pour faire partie d'une application précise, mais ils sont composés parce qu'ils sont en situation de l'être. Les applications « émergent » donc de ces compositions, puis évoluent et s'adaptent dynamiquement par recomposition en fonction d'autres opportunités. Dans la section 2, nous présentons un scénario concret et réaliste qui illustre l'intérêt de la composition et de la recomposition opportunistes en environnement ambiant.

Bien sûr, cette approche pose à son tour différents problèmes. Ceux-ci concernent en particulier la réalisation de la composition et de la recomposition opportunistes, la pertinence des compositions et la maîtrise de la combinatoire, la sémantique de l'application produite et son utilité.

Dans ce travail, nous nous intéressons à la question de la réalisation de la composition et de la recomposition opportunistes. Outre le scénario, la contribution de ce papier est aussi d'identifier un certain nombre d'exigences pour cette réalisation. Parmi celles-ci, on trouve l'automatisation de la composition. Si la tâche d'assemblage est traditionnellement à la charge du développeur, plusieurs travaux se sont intéressés à l'automatisation de la composition de composants logiciels. Nous les présentons en section 4.

1.2 Plan du papier

Le scénario présenté dans la section 2 motive l'approche de composition et de recomposition opportunistes. Il sert aussi de base à notre réflexion et à mettre en évidence différentes situations de composition et de recomposition. Ce scénario n'est qu'une illustration parmi d'autres de situations génériques pouvant se décliner dans différents contextes. Dans la section 3, une analyse conduit à une définition plus formelle de l'approche proposée et diverses exigences pour la réalisation d'une telle approche sont extraites. Nous présentons ensuite dans la section 4 un état de l'art de la composition automatique de composants logiciels : nous avons retenu neuf solutions et étudié comment elles répondent aux exigences de réalisation de la composition et de la recomposition opportunistes. La section 5 conclut et présente nos perspectives en matière de réalisation.

2. SCÉNARIO

Dans cette partie, nous présentons un scénario dans lequel un système de composition opportuniste assiste un ou plusieurs utilisateurs humains dans leur vie quotidienne et leur travail. Ce scénario illustre des situations pouvant survenir dans un environnement dynamique et montre le comportement attendu du système en réponse à ces situations, mais ne décrit pas le système lui-même. Le système rend possible les compositions et les recompositions sans qu'elles aient été nécessairement explicitées ; celles-ci résultent de la présence de certains composants dans l'environnement ainsi que de la pertinence de leur composition.

Le scénario met en scène Plip, qui est ingénieure dans une entreprise disposant d'une salle d'expérience sécurisée. Elle travaille avec un bras robotisé muni d'un laser dont le rôle est d'émettre un rayon vers une surface. Le scénario se déroule en trois actes de deux scènes chacun ; chaque scène est le théâtre d'une ou de plusieurs situations de composition et de recomposition effectuées de manière opportuniste. Une analyse de l'approche opportuniste à travers ce scénario est faite en section 3.

Acte 1

Scène 1 - Plip travaille sur son PC fixe dans son bureau sur un nouveau modèle expérimental. Une fois celui-ci peaufiné, elle lance sur sa tablette une application permettant d'estimer les résultats d'un tel modèle. Cette application détecte alors rapidement le composant qui implémente le modèle sur l'ordinateur de Plip, grâce à une communication WiFi entre les deux appareils, et lui propose de l'utiliser afin de lancer l'estimation. Comme cela correspond à son besoin, elle accepte. L'estimation ressemble à ce qu'elle envisageait pour son modèle, elle est donc prête à lancer l'expérience réelle. Elle se rend alors dans la salle d'expériences, sa tablette en main, afin de comparer les résultats réels aux estimations. La figure 1 propose une vue architecturale simple des compositions effectuées.

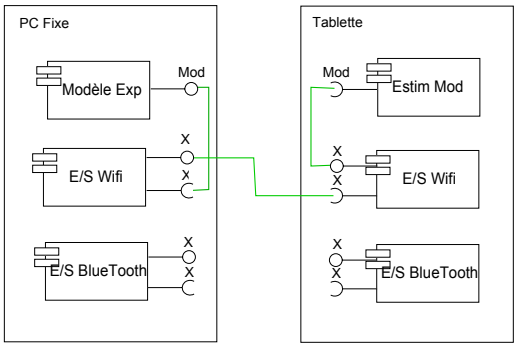

Figure 1: Acte 1 Scène 1 ; Vue Composants

Scène 2 - La salle d'expérience est une salle sécurisée récemment protégée contre l'entrée ou la sortie d'ondes WiFi. Plip n'est pas au courant de cette nouvelle mesure et y pénètre avec sa tablette, qui communique toujours en WiFi avec son PC. Le système détecte alors que le composant WiFi de la tablette n'est plus utilisable vers l'extérieur. Il recompose donc l'assemblage qui permettait d'utiliser le modèle expérimental stocké sur la machine de son bureau et utilise un composant Bluetooth, dont la fréquence est autorisée à circuler à travers les murs de la salle. Son application d'estimation fonctionne toujours, sans que Plip ne remarque une différence. La figure 2 illustre le résultat du point de vue des composants.

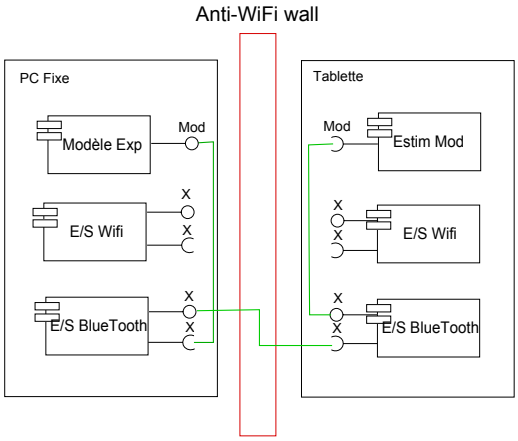

Figure 2: Acte 1 Scène 2 ; Vue Composants

Acte 2

Scène 1 - Plip démarre l'expérience correspondant à son nouveau modèle. Pour cela, elle allume les différents appareils, oscilloscopes et écrans de la salle d'expérimentation, ainsi que le PC central permettant de diriger l'expérience. Le système ayant préalablement appris à se composer pour convenir aux besoins expérimentaux de Plip, les divers appareils se composent automatiquement afin de créer tout l'environnement dont elle a besoin. Plip en profite d'ailleurs pour installer un nouvel oscilloscope holographique récemment acquis par le laboratoire. Le système réagit immédiatement et automatiquement, afin de remplacer une vieille machine et utiliser à sa place le nouvel arrivant, bien plus performant. La figure 3 donne une représentation en composants de l'environnement expérimental.

Scène 2 - Plip signifie ensuite qu'elle souhaite utiliser son modèle pour l'expérience. Celui-ci, stocké sur son PC fixe, communique alors avec le PC central via la tablette de Plip. Il est alors mis en contact avec le composant de commande permettant de lancer la simulation. Ce composant, contenu dans le PC central, a une interface requérant un modèle. Cette même interface est fournie par le composant implémentant le modèle développé par Plip. Les deux interfaces se composent donc. L'expérience peut alors commencer. L'assemblage complet est décrit dans la figure 4.

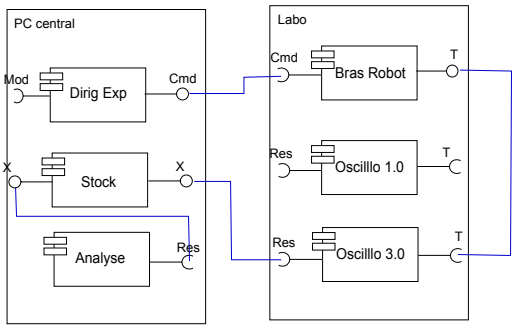

Figure 3: Acte 2 Scène 1 ; Vue Composants

Acte 3

Scène 1 - En plein milieu de l'expérience, un événement malheureux survient : le PC central tombe en panne ! Ce PC avait deux fonctions essentielles. D'un côté, il permettait, en se basant sur un modèle, de diriger un bras robotisé muni d'un laser. De l'autre, il récupérait les résultats numériques de l'expérience et les stockait avant de les analyser. De nombreux composants, indispensables au bon fonctionnement de l'expérience ont donc disparu. Le système va alors chercher des composants de remplacement, afin de continuer l'expérience, le temps que le PC central soit de nouveau fonctionnel. Le bras robotisé, n'ayant plus rien pour le commander, va chercher à composer son interface de commande. La tablette de Plip, toujours en communication avec son PC personnel, apprend par son intermédiaire qu'une copie du composant de commande est rangée dans une clé sans fil qui se trouve dans l'un des tiroirs du bureau. La tablette met alors en communication ce composant avec le bras robotisé. Ces derniers forment ainsi à eux deux une nouvelle application permettant de continuer l'expérience.

Scène 2 - Le composant censé envoyer des données doit lui aussi être remplacé. En l'absence du PC central, c'est un autre appareil électronique disposant d'une mémoire suffisante, par exemple le nouvel oscilloscope, qui va s'occuper de stocker temporairement ces résultats. Ne disposant pas de composant d'analyse, cette machine ne pourra pas les exploiter, mais se chargera de les transmettre au PC central aussitôt celui-ci réparé. L'expérience ne sera donc pas à recommencer, au grand soulagement de Plip. La figure 5 illustre le résultat en terme d'architecture des recompositions effectuées.

3. ANALYSE DU SCÉNARIO ET DE L'APPROCHE DE COMPOSITION

3.1 Typologie des situations de composition et de recomposition

Six situations de composition et de recomposition différentes peuvent être identifiées. Ces situations peuvent survenir en nombre et en ordre quelconques, le système de composition doit pouvoir supporter l'ensemble. Ces situations sont, le plus souvent, déclenchées par la mobilité d'un composant (2), (4) ou par la modification de son état (1), (3), (5). Elles sont aussi sensibles au contexte et au profil de l'utilisateur (6). Dans le scénario précédent, cinq de ces si-

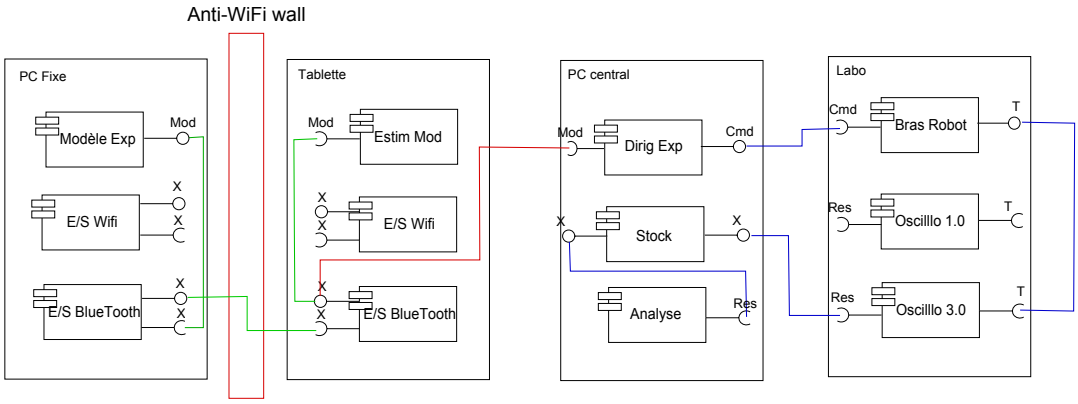

Figure 4: Acte 2 Scène 2 ; Vue Composants

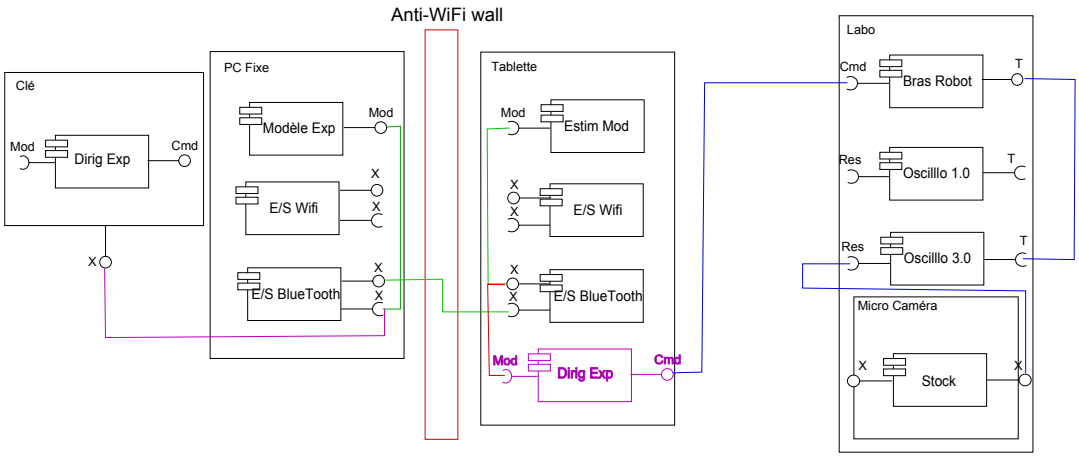

Figure 5: Acte 3 ; Vue Composants

tuations peuvent être mises en évidence ; elles sont recensées dans le tableau 1 concluant cette seconde partie.

(1) **Sollicitation d'un composant**. Un utilisateur ou le système peut solliciter un composant d'une manière ou d'une autre, c'est-à-dire signifier qu'il désire spécifiquement l'utiliser dans une application. Ce composant cherchera donc prioritairement à se composer de façon opportuniste afin de remplir au mieux sa fonction.

(2) **Apparition soudaine d'un composant**. Un composant nouvellement disponible dans l'environnement peut venir remplacer dans une application un composant de moindre qualité ou enclencher lui-même un nouveau processus de composition.

(3) **Disparition soudaine d'un composant**. Suite à une panne ou une obsolescence d'un ou plusieurs composants, le système cherchera à maintenir les applications en place en remplaçant les composants disparus par un nouveau composant ou une composition de composants.

(4) **Apparition ou disparition progressive d'un composant**. Notamment lors de situation de mobilité, des composants peuvent continuellement entrer et sortir progressivement d'un environnement. Cette évolution peut être détectée et anticipée par le système pour maintenir les applications par recomposition.

(5) **Évolution des caractéristiques d'un composant**. Les composants et leurs propriétés peuvent aussi évoluer de façon continue. Des valeurs de priorités, des valeurs de qualité de service ou d'autres caractéristiques sémantiques peuvent varier de façon imprévisible, suite à une mise à jour par exemple. Le système doit pouvoir prendre en compte dynamiquement ces modifications et adapter les assemblages.

(6) **Évolution de contexte**. Des informations de contexte comme la proximité de certains appareils peuvent mener à une décision pertinente du système. De plus, le système pourra avoir préalablement appris ces contextes et exploiter sa mémoire pour réaliser les futures compositions.

La majorité de ces situations se produit à différents passages dans le scénario précédent. La table 1 liste leurs occurrences. La situation (5), qui concerne les situations d'évolution de caractéristiques, n'apparaît pas dans le scénario présenté.

L'approche opportuniste permet d'identifier chacune de ces situations de (re)composition puis d'agir convenablement face à celles-ci. Pour chacune, il s'agira de définir des pré-conditions et des post-conditions d'assemblage afin de contrôler le processus ascendant de composition. En se basant sur ces conditions, il est alors réaliste d'envisager la réalisation d'un système de composition opportuniste qui

Acte1	*Scène 1*	(1)
	Scène 2	(4)
Acte2	*Scène 1*	(6)(2)
	Scène 2	(1)
Acte3	*Scène 1*	(3)(2)
	Scène 2	(3)

Table 1: Occurrences des situations de (re)composition

présente divers avantages comme exposés dans la section suivante.

3.2 Apports de l'approche opportuniste

L'intérêt de la composition opportuniste peut s'étudier à deux niveaux : celui de l'utilisateur et celui du concepteur de l'application. Du point de vue de l'utilisateur, ce dernier est sollicité *a minima*, le système fonctionnant en toute transparence pour lui. Du point de vue du développeur, l'enjeu est de minimiser sa charge de travail tout en obtenant une application qui s'adapte dynamiquement aux perturbations et aux nouveautés.

Trois apports principaux peuvent être mis en exergue dans l'approche opportuniste. Cette dernière facilite la conception, améliore la flexibilité par sa capacité d'adaptation et, par son aspect générique, impacte la réutilisation et la productivité.

Dans le scénario présenté, les assemblages proposés à Plip n'ont jamais été demandés ni décrits explicitement, mais ils lui sont pourtant utiles et sont utilisables. En effet, il n'y a pas d'expression explicite d'un besoin ni de connaissance d'une application précise à obtenir. La composition s'effectue par opportunisme, dans une situation donnée, si se composer de telle manière est pertinent. Pour juger de cette pertinence, le système peut se baser sur les profils des composants, des informations de contexte ou une mémoire des compositions passées. La composition finale émerge donc de façon ascendante, de par les interactions entre les composants, et elle n'est connue *a priori* ni par l'utilisateur, ni par le système, ni par les composants eux-mêmes. Le système crée ainsi continuellement de nouvelles applications fonctionnelles sans que l'utilisateur ou le concepteur n'ait à fournir de besoin précis, l'expression implicite par sollicitation restant possible mais optionnelle. Dans un environnement mobile et dynamique, avec un contexte indéterminé et évolutif, il n'est que rarement possible de prévoir quels seront les besoins et les intérêts d'un utilisateur. Ce genre d'approche semble donc pertinent puisque ne reposant pas sur la connaissance d'un besoin.

Dans le scénario, plusieurs situations de recomposition montrent l'adaptation des assemblages par opportunisme, en remplaçant un composant inutile (le composant WiFi de l'acte 1 scène 2), en introduisant un composant de meilleure qualité (l'oscilloscope de l'acte 2 scène 1) ou même en remplaçant tout un assemblage (le PC qui tombe en panne de l'acte 3). L'approche opportuniste permet donc la résilience des applications, en préservant un niveau de qualité satisfaisant. En plus de proposer de nouvelles compositions, ce même système doit aussi pouvoir remettre en cause et recomposer les applications déjà établies. Cela permet de maintenir une application au sein de laquelle survient une disparition ou une obsolescence, ou encore d'introduire de nouveaux composants, le tout dynamiquement et sans directive *ad hoc*. Encore une fois, ce processus de recomposition est dynamique et permet de s'adapter en continu aux perturbations.

Le dernier avantage d'une telle approche est qu'elle est générique et reproductible dans nombre d'environnements différents. Les applications qui aident Plip à travailler sont émergentes, et ne sont à aucun moment le fruit d'un besoin exprimé ou d'un objectif global. Le système n'impose pas non plus la présence de certains appareils ou composants, et ceux-ci ne sont pas conçus initialement pour la réalisation du système final. Le système travaille uniquement à un niveau local avec les composants disponibles dans son environnement et de façon opportuniste pour établir dynamiquement les compositions jugées pertinentes, en s'adaptant à son contexte. Le système de composition, embarqué dans un autre environnement (comme une gare, un avion ou un appartement) et dans un contexte applicatif différent pourra assister un utilisateur, et ce sans aucune modification de son comportement.

La composition opportuniste et ascendante, générique et adaptative prend tout son sens dans des contextes fortement dynamiques comme le sont les environnements ambiants. Ces environnements ouverts se caractérisent par une évolution rapide des entités les composant, notamment lors de situations de mobilité. Il est donc nécessaire de pouvoir automatiquement prendre en compte les apparitions et disparitions de composants, et de pouvoir assister un utilisateur en lui proposant des applications avec la meilleure qualité disponible. Cela doit se faire sans solution pré-établie, car celle-ci pourrait rapidement se retrouver obsolète. D'ailleurs, si l'on considère la population importante des systèmes ambiants, aussi bien en terme d'humains que d'entités électroniques, il n'est pas raisonnable de penser prévoir tous les besoins possibles, ni même d'imaginer qu'ils soient tous exprimables. Voilà pourquoi une solution faisant émerger automatiquement des application pertinentes, mais sans besoin exprimé, et ce dans n'importe quel environnement, est une solution intéressante.

Toutefois, l'approche opportuniste souffre encore de limites et nombre de problèmes sont ouverts. Comment garantir l'émergence d'applications utiles et utilisables dans un environnement donné ? Quelle est la sémantique des compositions obtenues ? Quelle est la place du besoin dans ce processus de fabrication ? Comment maîtriser la complexité combinatoire ? La conception d'une infrastructure qui supporte la composition et la recomposition opportunistes doit répondre à ces questions. La section suivante précise les exigences en matière d'infrastructure.

3.3 Exigences et infrastructure de composition

Dans cette section, nous identifions les exigences concernant la réalisation de la (re)composition opportuniste. Ces exigences portent sur le système ou l'infrastructure qui va prendre en charge les assemblages et les maintenir dynamiquement. Ce système doit satisfaire des exigences fonctionnelles et non fonctionnelles pour une composition à la fois automatique, opportuniste et ascendante : il a pour objectif d'effectuer automatiquement des compositions pertinentes avec les composants disponibles dans un environnement non connu, tout en respectant les principes de l'approche opportuniste.

Le système doit dans un premier temps travailler à un niveau local et permettre l'émergence, afin que l'aspect ascendant de la composition ait un sens. Il sera d'ailleurs pertinent d'utiliser des technologies décentralisées, car d'un côté cela correspond naturellement à une vision locale. De l'autre, il n'est pas envisageable de penser qu'une entité centrale puisse gérer la forte évolution et la combinatoire d'un environnement comprenant de nombreux composants, car les risques de goulot d'étranglement ou de dysfonctionnement deviennent trop importants. Il est donc nécessaire de proposer un **système distribué**, **autonome** et **décentralisé**.

L'infrastructure utilisée doit aussi pouvoir supporter la **dynamique**, et s'adapter rapidement à divers changements, même imprévus. Elle doit pouvoir fonctionner en continu et toujours proposer de nouvelles applications, en accord avec le contexte courant, quels que soient les événements influant sur l'environnement. Elle doit donc être ouverte et posséder de fortes capacités d'adaptation pour pouvoir gérer rapidement une combinatoire importante, tout en respectant le critère de généricité prévu pour l'approche. Un système capable de **résilience** et de **maintenance** par recomposition est aussi une exigence à prendre en compte.

Enfin, les compositions obtenues doivent avoir un niveau de pertinence suffisant pour le cadre dans lequel le système est utilisé. L'utilisateur doit pouvoir disposer d'applications utiles avec des composants adaptés. Le système doit donc pallier les limites de l'approche opportuniste et garantir un taux d'**applications pertinentes** acceptable tout en restant **générique** dans un système ouvert et dynamique. Il doit aussi fournir les moyens d'évaluation de ces applications en terme qualité. Cette évaluation peut passer par une mesure de la satisfaction de l'utilisateur sur l'assemblage complet. La prise en compte de cette mesure est difficile, mais nécessaire, car elle demande de répercuter cette information globale sur l'assemblage au niveau local des composants.

Il s'agit donc de concevoir un système qui soit ouvert, distribué et doté de capacité d'adaptation face à la dynamique de l'environnement. Il doit de plus pouvoir évaluer les compositions possibles pour gérer la **combinatoire**. Pour cela, il faut pouvoir se baser sur plusieurs données, comme par exemple des priorités, la qualité de service ou des données de **contexte et d'apprentissage**.

En conséquence, nous formalisons sept exigences qui seront utilisées comme critères d'évaluation de l'état de l'art (cf. section 4) :

- **Décentralisation** (Dc) : Capacité du système à pouvoir supporter une approche dans laquelle plusieurs entités indépendantes effectuent les compositions ensemble. Parfois, les calculs des combinaisons possibles sont effectués par un algorithme décentralisé, mais une entité centrale se charge d'interpréter les résultats pour effectuer la composition finale ; l'approche n'est alors que partiellement décentralisée.
- **Dynamique** (Dy) : Capacité du système à s'exécuter dans un environnement dynamique, dans lequel des composants peuvent apparaître, disparaître ou évoluer en cours de traitement.
- **Combinatoire** (Cb) : Capacité du système à pouvoir gérer un nombre potentiellement conséquent de compositions possibles avec les composants disponibles. Cela peut avoir un effet sur l'efficacité ou la rapidité de certaines approches. Le système doit posséder des stratégies de choix pour ne pas s'écrouler.
- **Recomposition** (Rc) : Capacité du système à maintenir l'application en production par substitution ou réorganisation des composants.
- **Apprentissage et Contexte** (AC) : Capacité du système à prendre en compte des données comme l'expérience ou le contexte pour effectuer ses assemblages.
- **Qualité du Résultat** (QR) : Capacité du système à garantir un assemblage utile et de qualité pour son utilisateur. Deux types de contrôle des compositions peuvent intervenir : en amont avec la combinatoire pour préférer des assemblages, en aval avec ce critère pour éviter de proposer à nouveau un assemblage, si la situation se représente.
- **Indépendance au besoin** (IB) : Capacité du système à opérer indépendamment de l'expression du besoin. En tant qu'éléments de comparaison, une expression implicite par désignation d'un composant correspond à un dépendance faible au besoin, tandis qu'une description précise de l'assemblage final correspond à une très forte dépendance au besoin.

4. ÉTAT DE L'ART

L'objectif de cette partie est de parcourir un état de l'art sur la composition automatique de composants afin de déterminer si les travaux existants peuvent répondre aux exigences énoncées précédemment et ainsi permettre la réalisation d'une composition opportuniste. Les travaux présentés ici concernent la création automatique d'assemblages à partir de composants, disponibles et non connus *a priori*. Ils sont, pour la plupart, supposés fonctionner dans un environnement dynamique et ouvert. En complément, on trouvera dans [Stavropoulos et al., 2011] un état de l'art sur la composition dynamique de services dans les systèmes ambiants.

4.1 Présentation des travaux existants

G. Grondin propose un modèle nommé MaDcAr permettant la composition et la recomposition d'applications à base de composants [Grondin et al., 2006]. Ce modèle est constitué d'un moteur d'assemblage possédant quatre entrées : un ensemble de composants, une description d'architecture, une politique d'assemblage et des données de contexte. La modification de l'une de ces entrées enclenche une (re)composition des composants de l'ensemble pour correspondre à la description d'architecture donnée, avec la meilleure qualité disponible. Cette solution permet de modifier dynamiquement des assemblages, mais nécessite une description spécifique et précise de l'architecture voulue, tout en s'appuyant sur un moteur centralisé, ce qui s'adapte mal à un environnement pervasif.

La problématique principale de N. Desnos est la maintenance d'un assemblage existant de composants logiciels lorsque l'un d'entre eux vient de devenir inutilisable [Desnos et al., 2007]. L'objectif est alors de remplacer le composant manquant par un nouvel assemblage de composants permettant d'offrir les mêmes fonctionnalités. Pour cela un algorithme de résolution par contrainte (CSP) est suivi pour construire progressivement le nouvel assemblage à partir des composants disponibles. Le développeur doit cependant définir des objectifs fonctionnels pertinents sur les composants afin de faire office de contraintes.

Dans [Bartelt et al., 2008], l'approche présentée est une al-

gorithme décentralisé de composition automatique de composants logiciels. Une instance initiale est sélectionnée parmi les composants disponibles, et cette dernière va envoyer une requête sur ses interfaces avec une certaine priorité. Les composants capables de subvenir à cette requête feront de même avec leurs propres interfaces. La discrimination par la priorité et la qualité de service permet de fournir une application supposée pertinente. En revanche, l'assemblage n'est pas maintenu et une nouvelle instance initiale doit être sélectionnée pour déclencher une nouvelle composition.

La plateforme MUSIC [Rouvoy et al., 2009], censée fonctionner sur des appareils mobiles, permet de construire dynamiquement des applications données, à partir de composants logiciels internes et de services ambiants. Pour cela, un planificateur détermine l'assemblage offrant la meilleure utilité en considérant plusieurs plans d'architecture prédéterminés et les composants accessibles. Cette approche permet de maintenir la meilleure qualité disponible pour une application donnée, mais peut rapidement se retrouver dépassée par une trop forte combinatoire.

Le middleware développé par M. Vallée vise la composition automatique de services en système ambiant [Vallée, 2009]. L'originalité de ces travaux réside dans l'utilisation d'une architecture décentralisée à base d'agents autonomes. On distingue trois types d'agents : superviseur, composeur et interpréteur. Les agents superviseurs sont chargés de chercher et identifier les services répondant aux fonctionnalités décrites dans une description d'application abstraite. Ils transmettent ensuite les résultats à l'agent composeur, chargé de la composition. Enfin, les agents d'interprétation offrent des informations de contexte aux autres agents. Cette approche décentralisée permet de créer dynamiquement et de maintenir des applications correspondant à une description d'application abstraite. Cependant, bien que le système soit décentralisé, il n'est pas distribué et un unique agent central se charge de l'ultime composition. Une description d'application doit aussi nécessairement être déterminée avec soin afin que le système propose des assemblages pertinents.

CODEWAN [Guidec et al., 2010] est une plateforme *peer-to-peer* pour le partage de composants au sein de réseaux non-connectés comprenant de nombreux utilisateurs mobiles. Lorsqu'un utilisateur a besoin d'une certaine application, son CODEWAN cherche alors à installer les composants nécessaires en les téléchargeant depuis les appareils des autres utilisateurs. Un algorithme de négociation décentralisé et opportuniste est alors mis en place afin de finalement installer l'application complète. Cette approche est efficace dans des systèmes mobiles mais est extrêmement dépendante de la connaissance détaillée *a priori* des applications à assembler.

Les travaux de J. Kramer et J. Magee [Kramer and Magee, 2009] concernent une architecture à trois niveaux permettant la composition automatique et dynamique de composants logiciels. Le niveau le plus bas représente les composants eux-mêmes. Le second permet de planifier et automatiser la composition en se basant sur des plans pré-calculés. Le plus haut niveau a pour tâche de créer des plans pour le niveau inférieur lorsque les plans existants ne sont plus adaptés au contexte actuel. Dans [Sykes et al., 2011], les auteurs proposent une implémentation décentralisée de la couche intermédiaire. Un algorithme de propagation d'informations (*gossip*) permet de créer de façon ascendante un assemblage résilient à partir d'un ensemble de composants. Toutefois, la potentialité d'une forte combinatoire n'est pas prise en compte. La troisième couche perd d'ailleurs en intérêt dans cette approche, puisqu'aucun plan n'est utilisé.

Dans [Vergoni et al., 2011], les auteurs font un constat similaire au nôtre en évoquant les lacunes d'une approche *top-down* et/ou centralisée dans un contexte inconnu et fortement dynamique. Ils proposent alors une approche entièrement ascendante de composants et de services ambiants à travers des orchestrations locales. Un noeud du système va, localement, orchestrer les composants disponibles et former un système local autonome. Plusieurs systèmes locaux autonomes vont ensuite communiquer et créer ensemble, de façon ascendante, une nouvelle application émergente. Cette approche décentralisée et générique correspond à la définition de l'opportuniste, mais ne vérifie à aucun moment la pertinence de ses compositions, ce qui mène probablement à des explosions combinatoires et des soucis d'utilisabilité.

Dans [Bonjean, 2013], les éléments assemblés sont des fragments de méthodes de développement. Le constat de base est que, lors d'un processus de développement logiciel, les méthodes existantes ne permettent pas toujours de satisfaire toutes les contraintes d'un projet particulier. C'est pourquoi une méthode composée de fragments de méthodes existantes pourrait être plus adéquate à certains problèmes. La plateforme SCoRe est un système multi-agent se basant sur des interactions coopératives permettant de créer de telles méthodes à partir d'une spécification des besoins de l'utilisateur et de ses préférences. La composition est ensuite complètement décentralisée. Le système est dynamique et ouvert, car les données utilisateurs comme la base de fragments peuvent changer à tout moment.

4.2 Synthèse

La table 2 synthétise l'évaluation des travaux présentés ci-dessus selon les exigences définies dans la partie 3.3. Cela permet de déterminer si ces systèmes sont capables de réaliser une approche de (re)composition opportuniste.

	Dc	Dy	Cb	Rc	AC	QR	IB
Grondin 2006	-	+	+	++	+	++	-
Desnos 2007	-	+	+	++	-	+	+
Bartelt 2008	++	+	-	-	-	+	+
Rouvoy 2009	-	++	-	+	-	++	-
Vallée 2009	+	++	+	+	+	+	-
Guidec 2010	++	++	-	-	-	++	-
Sykes 2011	+	+	-	++	+	+	+
Vergoni 2011	++	++	-	++	-	-	++
Bonjean 2013	++	+	++	+	+	+	-

Table 2: Prise en compte des exigences pour une composition opportuniste dans les systèmes étudiés

Il apparaît qu'aucun des travaux existants ne permet de répondre totalement aux exigences de la composition opportuniste. Si la dynamique est relativement bien prise en compte dans tous les systèmes, il n'en va pas de même pour l'indépendance au besoin puisque seulement trois systèmes peuvent s'affranchir d'une expression explicite du besoin. De plus, on peut constater que ces mêmes systèmes ont, en contrepartie, négligé les aspects de combinatoire, d'apprentissage et de prise en compte du contexte. Ils semblent donc être lacunaires sur le plan du contrôle des compositions.

5. CONCLUSION ET PERSPECTIVES

Après avoir donné notre définition de l'opportunisme en matière de composition et de recomposition de composants logiciels, nous avons illustré l'intérêt d'une telle approche pour le développement de logiciels : flexibilité lors de la phase de conception et adaptativité lors de l'exécution.

Cette approche nouvelle que nous explorons semble prometteuse puisqu'elle s'abstrait de l'expression d'un besoin et qu'elle est adaptée à des environnements extrêmement dynamiques et ouverts comme dans le cas des systèmes ambiants, dans lesquels on attend une anticipation par le système des besoins des utilisateurs. De plus, il s'agit d'une approche générale utilisable dans différents contextes.

Elle impose des exigences en matière de processus de réalisation que les systèmes actuels n'ont pas encore réussis à satisfaire totalement. En effet, les systèmes *top-down* s'adaptent difficilement à des situations nouvelles et par conséquent ne peuvent complètement faire face à l'imprévisibilité des environnements ambiants. D'un autre coté, les systèmes sans besoin pré-requis doivent encore régler le problème de contrôle de la combinatoire et de la pertinence des compositions. Il s'agit donc de trouver une solution de conception qui supporte une approche ascendante de compositions dans un environnement dynamique et ouvert, source de multiples compositions opportunistes et, qui cependant, maîtrise le nombre et la nature des compositions pour éviter l'explosion combinatoire.

Les systèmes multi-agents coopératifs [Georgé et al., 2011], qui permettent l'émergence de fonctions par interactions locales, sont candidats à la conception d'un tel système de composition automatique, ascendante et opportuniste. Une première étude expérimentale a été réalisée et présentée dans [Denis et al., 2012]. L'idée est que, par coopération, des agents-composants peuvent décider localement et de façon autonome de s'assembler avec d'autres (réponse aux exigences Dc, Cb, AC) sans connaître explicitement un besoin (réponse à l'exigence IB). Ils maintiennent un réseau d'accointances en fonction de leur mobilité (réponse à l'exigence Dy) et construisent ainsi collectivement une architecture logicielle émergente. Les interactions de ces agents-composants permettent l'adaptation dynamique par réorganisation des compositions (réponse à l'exigence Rc). L'exigence de qualité du résultat (QR) ne pourra cependant pas être prise en compte au niveau de l'agent, et devra donc être évaluée à un macro-niveau.

Références

Abdelkrim Amirat, Hock-Koon Anthony, and Mourad Chabane Oussalah. Paradigmes objet, composant, agent et service dans les architectures logicielles. In M. C. Oussalah, editor, *Architectures logicielles, principes, techniques et outils*, chapter 1, pages 19–67. Hermes-Lavoisier, 2014.

Christian Bartelt, Benjamin Fischer, and Andreas Rausch. Towards a Decentralized Middleware for Composition of Resource- Limited Components to Realize Distributed Applications. In *3rd Int. Conf. on Pervasive and Embedded Computing and Communication Systems (PECCS 2013)*, pages 245–251, 2008.

Noelie Bonjean. *Auto-organisation de fragments pour la conception de processus de développement.* PhD thesis, Université de Toulouse, UPS, 2013.

Grégoire Denis, Jean-Paul Arcangeli, Victor Noël, Charles Triboulot, and Sylvie Trouilhet. Composition opportuniste et ascendante à base d'agents coopératifs. In *Journées francophones Mobilité et Ubiquité (UBIMOB 2012)*, pages 196–209. Cépaduès Editions, 2012.

Nicolas Desnos, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. Automated and Unanticipated Flexible Component Substitution. In *Proc. of 10th Int. Symp. on Component-Based Software Engineering*, 2007.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison Wesley, 1995.

Jean-Pierre Georgé, Marie-Pierre Gleizes, and Valérie Camps. Cooperation. In G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageogos, editors, *Self-organising Software*, Natural Computing Series, pages 7–32. Springer, 2011.

Guillaume Grondin, Noury Bouraqadi, and Laurent Vercouter. MaDcAr : An Abstract Model for Dynamic and Automatic (Re-)Assembling of Component-Based Applications. In *Component-Based Software Engineering*, number 4063 in LNCS, pages 360–367. Springer-Verlag, 2006.

Frédéric Guidec, Nicolas Le Sommer, and Yves Mahéo. Opportunistic Software Deployment in Disconnected Mobile Ad Hoc Networks. *Int. Journal of Handheld Computing Research*, 1 :24–42, 2010.

Jeff Kramer and Jeff Magee. A Rigorous Architectural Approach to Adaptive Software Engineering. *Journal of Computer Science and Technology*, 24(2) :183–188, April 2009.

Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. Music : Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software Engineering for Self-Adaptive Systems*, pages 164–182. Springer-Verlag, 2009.

Thanos G. Stavropoulos, Dimitris Vrakas, and Ioannis Vlahavas. A survey of service composition in ambient intelligence environments. *Artificial Intelligence Review*, 40(3) : 247–270, September 2011.

Daniel Sykes, Jeff Magee, and Jeff Kramer. FlashMob : Distributed Adaptive Self-Assembly. In *Proc. of the 6th Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems*, pages 100–109, 2011.

Mathieu Vallée. *Un intergiciel multi-agent pour la composition flexible d'applications en intelligence ambiante.* PhD thesis, École Nat. Sup. des Mines de Saint-Etienne, 2009.

Christophe Vergoni, Jean-Yves Tigli, Gaëtan Rey, and Stéphane Lavirotte. Construction Bottom-up d'applications ambiantes en environnements partiellement connus a priori. In *7èmes Journées Francophones Mobilité et Ubiquité (UBIMOB 2011)*, 2011.