Une Approche Métier au Développement d'Applications pour Milieux Pervasifs

Thomas Morsellino, Tegawendé F. Bissyandé

Une Approche Métier au Développement d'Applications pour Milieux Pervasifs

Thomas Morsellino et Tegawendé F. Bissyandé

LaBRI, CNRS, Université de Bordeaux, 351 cours de la libération, 33405 Talence - France.

Contact : {morsellino,bissyand}@labri.fr

Résumé
Nous décrivons dans cet article différents éléments de prospection pour la conception d'une plateforme d'automatisation des décisions clés d'implémentation d'applications dédiées pour les environnements pervasifs. Nous proposons la définition d'un langage dédié de haut niveau permettant au développeur non expert d'écrire des spécifications qui tiennent compte du contexte d'exécution et des capacités des périphériques hôtes.

Abstract
This paper aims at describing the design of a generative solution for enabling novice programmers to easily and safely develop applications that sit over ubiquitous environments. Our proposition relies on the definition of a new Domain Specific Language (*DSL*) that allows developers to write specifications which account for the execution context properties and the capabilities of hosting devices as well.

Mots-clés : Informatique ubiquitaire, Génération automatique, Protocoles de communication.

Keywords: Ubiquitous computing, automatic generation, network protocols

1. Introduction

Notre univers quotidien est progressivement peuplé d'entités communicantes amenées à interagir de façon collaborative afin de fournir un service plus ou moins complexe aux utilisateurs. L'hétérogénéité matérielle et la diversité des capacités dont disposent ces différentes entités contribuent cependant à restreindre les possibilités d'interaction. Par ailleurs, les environnements ubiquitaires ainsi constitués se caractérisent par la volatilité des ressources qui apparaissent lorsqu'un périphérique intègre l'environnement puis disparaissent, soit suite à une panne, ou volontairement suite à la mise en service d'un périphérique potentiellement plus performant. Dans un tel contexte pervasif, les applications évoluant dans sur les supports matériels mobiles ou fixes doivent être en mesure de s'adapter aux ressources disponibles dans leur voisinage. Cet état de fait a un impact significatif sur la conception des applicatifs et introduit des contraintes majeures dans leur développement.
La multiplicité des implémentations d'applications fournissant des services identiques est largement due au manque d'interopérabilité protocolaire. En effet, le code d'une application de communication donnée est intrinsèquement lié à la combinaison de système/réseau/protocole pour laquelle elle est conçue. De plus, la diversité des technologies requiert des compétences spécifiques difficilement maitrisables dans leur globalité par un même développeur. Par ailleurs, constituer une application qui gérera différentes technologiques n'est pas une solution viable dans cette ère du tout-performant.
Le modèle de développement que nous introduisons dans cet article est celui qui permet de rendre tous les supports matériels *équipotent*s vis-à-vis des applications à déployer et sans tenir compte des différences fondamentales entre les protocoles et les réseaux. Il s'agit ainsi de permettre à un programmeur d'écrire une application de communication en ignorant les spécificités de la plateforme à laquelle elle est destinée et les protocoles à utiliser. Ces *détails*, certes indispensables, ne

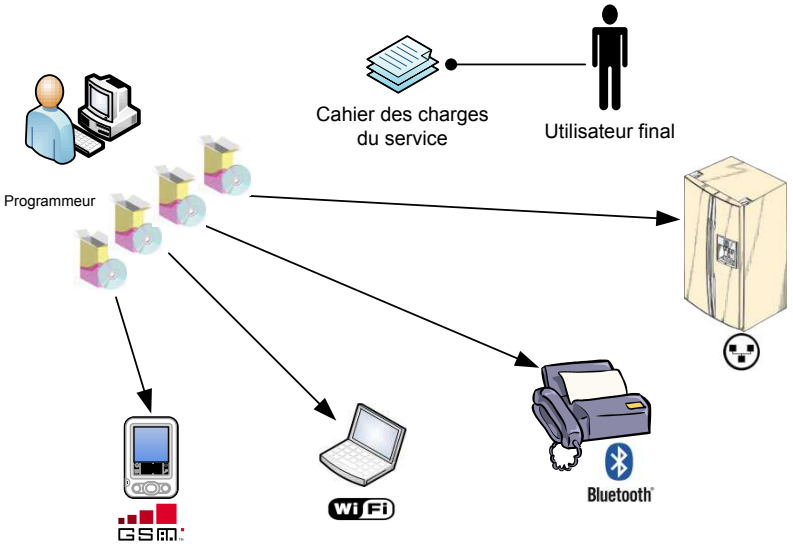

FIGURE 1 – Schéma usuel de commande d'applications pour réaliser un service dans un environnement ubiquitaire.

seront pris en compte qu'avant le déploiement en tenant compte des préférences du développeur et des capacités du support.
Nous proposons ici une approche de génération automatique pour le développement des applications, afin de généraliser autant que possible la conception des applications ayant pour vocation d'échanger des données sur un réseau quelconque. Nos contributions sont les suivantes :
– Nous définissons un schéma global de l'approche proposée en montrant les avantages des choix de conception,
– Nous présentons les pistes à emprunter pour une mise en oeuvre de l'approche,
– Nous décrivons quelques travaux liés à la génération automatique et à la l'uniformisation de paradigmes de communications afin de mieux démontrer la faisabilité de notre approche.
Le reste de cet article est organisé comme suit. La section 2 décrit de manière plus détaillée notre approche. La section 3 soulève quelques questions de conception et d'implémentation générale de l'approche. La section 4 permet d'établir la pertinence et la faisabilité de notre proposition. Enfin, nous concluons l'article à la section 5.

2. Approche générative pour la conception d'applications pour l'ubiquitaire

La maison du futur est un environnement ubiquitaire dans lequel évoluent une diversité de terminaux. Ces entités communicantes sont en constante interaction suivant des configurations imposées par leurs capacités réseaux, leurs offres applicatives et les besoins des utilisateurs. Afin d'introduire au mieux l'approche défendue dans cet article, nous proposons d'examiner un scénario de gestion d'aliments frais dans une maison moderne.
La maison est équipée d'un ordinateur et d'un réfrigérateur connectés à Internet et d'un fax. La personne en charge de l'approvisionnent de la maisonnée dispose d'un smartphone. Les entrées et sorties d'aliments sont suivies par l'ordinateur intégré dans ce réfrigérateur *intelligent*. En cas de pénurie imminente, plusieurs configurations de communication sont envisageables pour remédier au problème :

1. Le réfrigérateur fait la liste des achats en consultant les meilleurs prix sur Internet, lance la commande et faxe la facture au gestionnaire.
2. Le réfrigérateur établit la liste des produits manquants, recherche les magasins les plus proches et envoie les informations sur le smartphone du gestionnaire qui s'occupera de faire les achats.
3. Le réfrigérateur alerte l'ordinateur de la famille et, en fonction de l'agenda du gestionnaire,

déduit la meilleure façon de gérer la pénurie.

Afin mettre en oeuvre un tel système, une armée de développeurs serait nécessaire pour coder les différentes applications sur les différents terminaux dont un sous-ensemble est représenté sur le schéma de la Figure 1. De plus, la communication inter-terminaux peut s'avérer complexe, lorsque ceux-ci reposent sur des protocoles incompatibles. Pourtant, d'un point de vue architecture logicielle, toutes les applications qui participent à ce scénario offrent le même service qui consiste à récupérer une liste d'achats et à prendre des décisions de prise en charge. On pourrait tout aussi bien concevoir un cas dans lequel les ordres de commandes sont envoyés depuis le smartphone vers le réfrigérateur pour vérification et traitement.
Dans le cadre du scénario précédent, il est pertinent d'imaginer la même application déployée sur l'ensemble des divers supports reliés grâce à des technologies hétérogènes et souvent incompatibles. Pour ce faire, il faut définir et mettre en place une plateforme de développement qui propose des outils conséquents pour offrir une efficacité dans l'implémentation et le déploiement d'applications.

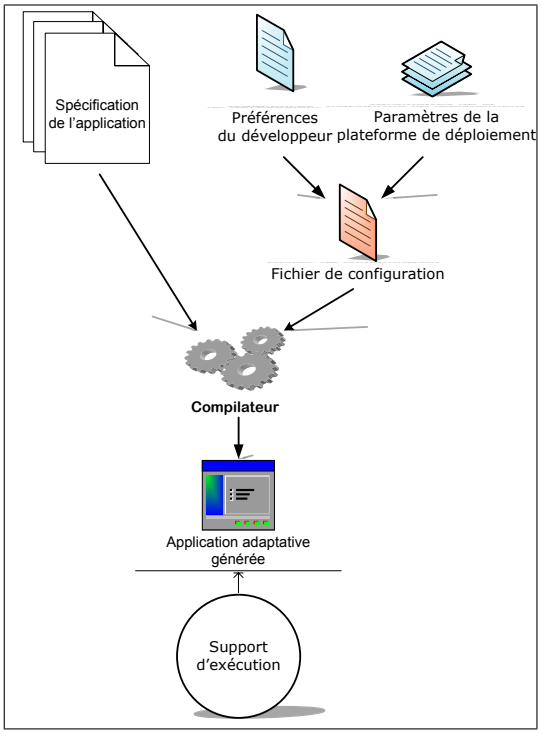

FIGURE 2 – Vue d'ensemble des phases de conception et de génération d'une application suivant l'approche proposée.

L'approche générative que nous proposons vise à cacher les incompatibilités technologiques qui obligent les développeurs à redéfinir plusieurs implémentations pour la même application. Par ailleurs, nous estimons que pour présenter des offres de services qui correspondent au mieux aux attentes de tous les utilisateurs, il faut permettre à un maximum de personnes, les experts du domaine et les utilisateurs, de participer à la conception des applications. Il est donc nécessaire de s'atteler à faire baisser le niveau d'expertise requis pour le développement d'applications ubiquitaires.
Dans le reste de cette section, nous décrivons quelques facettes de notre proposition en détaillant les éléments essentiels qui supportent l'approche.

2.1. Langage métier

Dans le cadre du développement d'applications de communication, de nombreuses fonctionnalités se font récurrentes. Ainsi, la gestion des ouvertures/fermetures de connexions et l'échange de requêtes/réponses sont autant d'actions qui partagent des primitives dans un grand nombre de protocoles réseaux, mais dont l'utilisation nécessite une expertise en programmation réseau. De plus, des connaissances approfondies des techniques de programmation système sont requises pour gérer efficacement les processus/threads différentes tâches du service. Un langage dédié, communément appelé langage métier, serait alors approprié pour cacher toute la complexité de ces opérations tout en uniformisant leur utilisation d'une technologie à l'autre. Typiquement, un opérateur `send` d'envoi de messages pourra être utilisé indifféremment pour un réseau Bluetooth, IP ou 3G. La différenciation de traitement se faisant à un autre niveau indépendant de la phase de conception.
Le langage métier offre aussi la possibilité de décrire de manière simple, mais fiable le service fourni par une application. En effet, un langage dédié offre des notations et des constructions syntaxiques taillées sur mesure pour un domaine d'applications [10]. Grâce aux primitives haut niveau du langage, le programmeur peut spécifier avec plus d'expressivité les fonctionnalités à implémenter pour fournir les services. Le schéma de la Figure 2 présente une vue d'ensemble des entrées et sorties du système.

Préférences inhérentes à l'application

Bien que conçue pour s'adapter aux changements de son environnement d'exécution, une application adaptative atteint ses meilleures performances lorsqu'elle fonctionne dans des conditions qui s'appliquent au mieux au service fourni. Typiquement, une application de transfert de fichier peut recourir à des technologies Wifi [7], Bluetooth [13] ou Zigbee [1] pour la transmission sans fil de données. Cependant, les préoccupations de débit de transfert conduisent à un schéma de préférences mettant en priorité le Wifi. En absence de WiFi, et en considérant la portée de transmission, Zigbee constituerait un meilleur choix. Une telle intelligence peut être intégrée au fonctionnement de l'application à travers les spécifications du programmeur imposant un schéma de priorité.

Configuration relatives à l'environnement

Un langage métier peut aussi permettre la définition abstraite de scénarios de fonctionnement. Ainsi, les concepteurs d'applications peuvent décrire des scénarios d'interaction en fonction des résultats d'une phase de découverte de services. En reprenant la configuration de la maison du futur décrite au début de cette section, un langage dédié simplifierait la description des orchestrations possibles entre les divers services offerts par les différents terminaux. La volatilité des terminaux et des services dans un environnement est donc bien gérée par la possibilité de reconfigurer *à la volée* le comportement des applications développées.

2.2. Vérification des spécifications et génération de code

Un langage métier est généralement défini dans le but de capitaliser sur un ensemble de connaissances acquises dans un domaine. Il est souvent accompagné d'outils pour la génération de code. Dans notre cas de développement d'applications adaptatives, les descriptions écrites par le programmeur sont analysées afin de permettre la génération du code de l'application dans un langage plus général tel le langage C.

Dans un premier temps, le compilateur du langage dédié vérifie la cohérence des spécifications afin d'assurer que le concepteur de l'application a décrit le service de manière consistante. Typiquement, il faut s'assurer que les protocoles indiqués dans la description du service font bien partie de la liste spécifiée pour la plateforme, puis s'assurer que les opérations invoquées sont bien en rapport avec les capacités des technologies sous-jacentes.

Dans un second temps, le compilateur analyse la succession des actions, le flot des messages échangés pour détecter les incohérences de traitement.

Spécialisation en fonction de la plateforme

L'analyse du terminal hôte est une étape déterminante pour retenir les fonctionnalités à conserver dans le code de l'application à générer. En effet, pour une application pouvant supporter différentes technologies de réseau, il est contre performant de la déployer telle quelle sur un périphérique ne supportant qu'une seule technologie. Il en va de même pour les différentes fonctionnalités décrites par le développeur.

Finalement, le compilateur produit le code approprié en fonction des technologies disponibles, des préférences du concepteur et aussi des options de compilation de la plateforme d'installation.

2.3. Support d'exécution

L'abstraction fournie par l'aspect haut niveau du langage métier est rendue possible par les fonctionnalités recensées dans un support d'exécution. Ce support intègre par exemple, tout le code nécessaire à la gestion des sockets pour l'envoi de messages sur différents types de réseaux. C'est aussi le support d'exécution qui gère la transition d'une technologie à l'autre, jongle efficacement avec la multiplicité des interactions entre le terminal et les périphériques volatiles du voisinage. De plus, grâce au support d'exécution, un certain nombre d'étapes peuvent être simplement ignorées par le concepteur d'applications. Il s'agit par exemple de la découverte permanente de services dans un milieu pervasif afin de maintenir l'application consciente des configurations possibles.

3. Questions ouvertes

L'état de l'art des langages métiers distingue deux types de notations.

– *syntaxe graphique.* Un nombre de plus en plus important de langages métiers mettent à la disposition des développeurs des fonctionnalités sous forme de composants indépendants, graphiquement identifiables. Microsoft Visio© est un exemple de tels langages dont les notations graphiques sont plus accessibles à un grand nombre de personnes pour la création des diagrammes. Toutefois, la syntaxe graphique offre peu de flexibilité dans les conceptions dans la mesure où il existe une limitation dans les composants disponibles et dans les possibilités d'assemblages.
– *syntaxe textuelle.* Les notations textuelles offrent une granularité plus fine de fonctionnalités. Toutefois, leur apprentissage en tant que nouveau langage de programmation peut s'avérer hors de portée. TEX, par exemple, est utilisé dans des cercles restreints.

Le choix de la notation du langage dédié à notre approche doit donc faire l'objet d'un consensus. De nombreuses briques de base peuvent être prédéfinies en assemblant les fonctionnalités récurrentes dans les services ubiquitaires. La notation graphique peut donc permettre au programmeur de facilement, et rapidement, introduire et connecter ces fonctionnalités. Cependant, il est important de laisser une marge de manoeuvre aux développeurs experts qui souhaitent raffiner les spécifications, en leur permettant de réécrire avec une syntaxe textuelle le code qui relie différents composants.

Continuation dans les traitements.

En cas d'interruption de connectivité sur un réseau ou d'indisponibilité d'un terminal participant à l'orchestration des services, une application pour milieux pervasifs doit permettre à chaque périphérique de poursuivre, de manière transparente pour l'utilisateur, le traitement sans interruption du service. Cette contrainte soulève néanmoins de nombreuses questions dans la gestion de la continuation des traitements. Ainsi, lorsque par exemple, un transfert de données s'effectuant par le biais du service OBEX de Bluetooth vient à être interrompu à cause d'un éloignement des terminaux communicants, la question de la procédure de changement de réseau se pose. En effet, sur quelles critères faut-il se baser pour décider de recommencer le transfert, ou de détecter la coupure afin d'alerter les deux parties de la communication afin qu'elles engagent une négociation sur le réseau alternatif à utiliser.

Enrichissement du support d'exécution

Le support d'exécution en soi ne pouvant être exhaustif, et au vu de la prolifération des technologies, des protocoles et des fonctionnalités de base recensées dans les environnements pervasifs, il est nécessaire de mettre en place un moyen d'étendre les capacités offertes par l'approche. Cependant, un support d'exécution modulaire peut avoir des inconvénients d'un point de vue efficacité de chargement, tandis qu'un support d'exécution monolithique nécessiterait une refonte permanente. Une solution de compromis consisterait en la réalisation d'un support d'exécution minimal, auquel il serait possible de greffer des bibliothèques , *plugins*, fournissant des fonctionnalités supplémentaires.

4. Travaux Connexes : Comparaisons et Commentaires

Dans cette section, nous présentons brièvement des travaux de recherches qui visent à fournir des solutions à une variété de problématiques abordées dans notre approche.

4.1. Abstraction des services d'un domaine

Pantagruel. Pantagruel est un langage dédié conçu par l'équipe Phoenix [1] pour permettre à des développeurs novices d'écrire des programmes d'applications pour des environnements pervasifs [5]. Toutefois, il s'agit uniquement de permettre aux développeurs, grâce à des outils visuels, d'orchestrer un ensemble de services. Nous défendons une approche visant à répartir la logique du service sur les différentes instances d'application. Ainsi, plutôt que de confier l'adaptation à une entité tierce, l'intelligence est intégrée dans le développement de l'application.

1. http ://phoenix.labri.fr

WS-AMUSE. WS-AMUSE [12] est un *framework* qui met en valeur les fonctionnalités des applications multimédias existantes en les rendant abstraites par l'utilisation des services Web. Ce système définit un ensemble de composants par défaut, utilisables par différentes applications. La conception de nouvelles applications est modélisée par des automates à états qui décrivent la logique de l'application. Dans le processus de conception, chaque sommet de l'automate représente une entité de l'application qui sera ensuite *encapsulée* par un service Web. La composition et l'interconnexion de ces sommets sont ensuite réalisées par l'utilisation du langage d'orchestration de services Web BPEL [2]. L'abstraction de la logique de l'application est donc réalisée par des automates qui sont décrits par un langage dédié SCXML [14]. La couche d'abstraction fondée sur la transformation de modèles [4] permet de créer des applications trois à dix fois plus rapidement qu'un développement classique à savoir décrire manuellement le flot BPEL de l'application. Nous partageons dans notre approche la philosophie de WS-AMUSE. En effet, nous proposons de mettre en valeur les fonctionnalités des applications pour milieux pervasifs en les encapsulant dans des composants avec une fine granularité. Toutefois, WS-AMUSE est limité par les performances des moteurs BPEL et plus particulièrement par les performances liées aux traitements successifs des messages XML dans les flots BPEL. Notre approche, quant à elle, repose sur un support d'exécution efficace, car spécialisé pour l'ubiquitaire.

4.2. Techniques de Génération Automatique

La génération automatique d'applications est un sujet fécond, car elle permet un gain en productivité - dû à la réduction du temps de développement et de débogage - et en fiabilité du code généré. La littérature regorge d'articles sur les résultats de projets de génération automatique de services Web [3, 9] et de passerelles entre protocoles incompatibles [2]. Notre approche s'inscrit dans la ligne de ses travaux pour offrir les mêmes outils dans le domaine de l'ubiquitaire. Par ailleurs, nous avons pour intention de faire effet de levier sur les résultats obtenus dans le projet z2z pour la mise en place de passerelles inter-protocoles. Bromberg *et al.* proposent une approche orientée langage pour simplifier la création de ce type de passerelles. *Z2z* est supporté par un *runtime* qui cache les détails d'implémentation aux développeurs et un compilateur qui valide ou non le code en détectant les erreurs qu'il peut y avoir dans les fichiers de description. *Z2z* est composé de plusieurs modules : des modules PS (Protocol Specification), des modules MS (Message Specification) et un module MT (Message Translation). Ces derniers permettent, respectivement, de définir le comportement des protocoles, de décrire la structure des messages et de définir la translation de chaque message passant en travers de la passerelle. Ainsi, nous avons l'intention de fournir les briques logicielles - i.e, les modules de spécifications - pour la construction de passerelles entre les protocoles réseaux usuellement présents dans les environnements ubiquitaire.

4.3. Intergiciels : Des interfaces pour résoudre les problèmes d'hétérogéneité

L'intégration de nouvelles applications dans un contexte regroupant une diversité de protocoles de communications n'est pas aisée. Le rôle des intergiciels est de fournir un support qui uniformise l'interaction entre des entités communicantes.

Services Web

Les services Web constituent aujourd'hui une technologie essentielle pour résoudre les problèmes d'hétérogénéité grâce au couplage faible qu'ils imposent, à l'utilisation de protocoles simples et largement utilisés, et enfin grâce aux différents standards sur lesquels ils reposent. Il résulte de cet état de fait que de nombreux travaux de recherches sont focalisés sur la définition d'outils et de méthode pour la migration des systèmes dits *legacy* basés sur des protocoles natifs vers les services Web. Toutefois, comme nous le montrons avec quelques cas, les approches utilisées sont non génériques et requièrent du développeur une expertise sur différents paradigmes.

WSIP. Chou *et al.* ont présenté une approche qui permet de rendre accessible le protocole SIP par l'invocation de méthodes appartenant à un service Web [8]. Une couche intermédiaire composée d'un service Web cache les détails des communications SIP aux clients de l'application et fournit une interface web service pour le protocole SIP. WSIP facilite l'intégration dans des services plus complexes pour fournir plus de fonctionnalités comme dans un environnement ubiquitaire.

2. http ://www.activebpel.org/

Toutefois, dans leur implémentation, il existe un couplage fort entre le protocole natif (SIP) et le service oblige une ré-implémentation à chaque évolution du protocole. De plus, les piles SIP et Web Services évoluant de façon décorrélée, le développeur doit en permanence s'assurer de la validité des primitives d'invocation. Une approche générative aurait réduit la masse de travail du développeur, tandis qu'un langage de spécification permettrait de réduire le couplage entre la conception et l'implémentation.

4.4. Intergiciels dédiés

Intergiciels adaptatifs. Ces dernières années, de nombreuses solutions ont été proposées pour la réalisation d'intergiciels intelligents qui interfacent les protocoles et technologies natives. Zhang *et al.* ont présenté avec JIM [15] une solution de construction d'intergiciel adapté aux besoins de la plateforme de déploiement. Ainsi, une instance d'intergiciel n'est composée qu'au moment de son déploiement, i.e. lorsque le système dispose d'informations pertinentes telles que les besoins de l'application ou encore les contraintes imposées par l'environnement. Bien que flexibles et modulables, ces intergiciels perdent tout leur intérêt dès lors que l'environnement dans lequel ils sont déployés est dynamique. En ce sens, ils ne sont pas capables de s'adapter *à la volée*. Ainsi, les intergiciels adaptatifs usuels ne conviennent pas aux milieux pervasifs où les reconfigurations sont récurrentes.

Intergiciels réflexifs. Récemment, la recherche s'est enrichie de nouvelles contributions grâce notamment au intergiciels réflexifs tel ReMMoC [6]. Ces intergiciels sont capables de s'adapter à l'environnement dans lequel ils évoluent, et ce, même en cours d'exécution. Bien que ces travaux soient prometteurs, des problèmes persistent quant à la façon dont ils peuvent se reconfigurer. Pour des intergiciels déployés dans des environnements à ressources limitées, il n'est pas approprié de stocker, même de façon distribuée, les composants permettant ces reconfigurations successives.

Intergiciels coopératifs. À l'instar de solutions basées sur les services web, de nouvelles approches consistent à promouvoir la création d'intergiciels légers et décorrélés. Rellermeyer *et al.* présentent AlfredO [11] qui apporte une brique logicielle permettant l'interopérabilité entre toutes les instances d'intergiciels. Basé sur des principes de type *SOA*, AlfredO permet de faciliter l'interaction entre supports à ressources contraintes par l'utilisation de descripteurs de services ainsi que l'utilisation d'un moteur (basé sur OSGi) permettant la mise en oeuvre des services en se basant sur le principe des architectures multitierces.

Les principaux problèmes relatifs à l'utilisation d'intergiciels qu'ils soient réflexifs, décorrélés ou encore adaptatifs montrent qu'il n'existe actuellement pas de solutions permettant de généraliser la conception d'applications communicantes qui peuvent évoluer sur des réseaux hétérogènes. Nous avons essayé de montrer dans cet article que la création d'une solution basée sur un langage de haut niveau et sur des éléments de descriptions était pertinente. La génération automatique de ces applications est réalisée par un compilateur tandis que l'exécution de l'application est orchestrée par l'utilisation d'un *runtime*. Ce dernier est un intergiciel qui s'adapte à l'environnement dans lequel il est déployé ainsi qu'aux préférences de l'application. L'application devient alors flexible et est totalement découplée d'un paradigme de communication quelconque propre aux protocoles des réseaux sous-jacents qu'ils soient *IP* ou non.

5. Conclusions et Perspectives

Nous avons présenté dans cet article des idées sur la définition d'une approche générative pour un développement plus adéquat des applications de communication fonctionnant sur des terminaux évoluant en milieux pervasifs. Une brève description des propriétés des langages métiers nous a permis de dégager quelques avantages de l'approche. Nous avons aussi soulevé de nombreuses questions pour bien montrer que la complexité du développement est déplacée vers le concepteur et développeur du langage, de son compilateur et de son support d'exécution. Le soin donné à l'analyse des travaux connexes a pour but de bien démontrer la faisabilité de l'approche en montrant des d'implémentation dans divers projets auxquels nous sommes familiers. La prochaine étape de nos travaux consiste à définir le langage puis à écrire le compilateur correspondant. Le support d'exécution est construit progressivement en fonction des fonctionnalités

que nous aurons recensées pendant les phases de conception du langage.

Bibliographie

1. Zigbee Alliance. Zigbee-2007 specification. `http://www.zigbee.org/`, 2007.
2. Yérom-David Bromberg, Laurent Réveillère, Julia L. Lawall, et Gilles Muller. Automatic generation of network protocol gateways. In *Middleware '09 : Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, pages 21–41, Urbana Champaign, IL, USA, 2009. Springer-Verlag New York, Inc.
3. Ernest Cho, Sam Chung, Daniel Zimmerman, et Menaka Muppa. Automatic web services generation. In *HICSS '09 : Proceedings of the 42nd Hawaii International Conference on System Sciences*, pages 1–8. IEEE Computer Society, 2009.
4. K. Czarnecki et S. Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3) :621–645, 2006.
5. Zoé Drey et Charles Consel. A Visual, Open-Ended Approach to Prototyping Ubiquitous Computing Applications. In *Proceedings of the 8th IEEE Conference on Pervasive Computing and Communications (PERCOM'10)*, Mannheim Allemagne, 2010.
6. Paul Grace, Gordon S. Blair, et Sam Samuel. Remmoc : A reflective middleware to support mobile client interoperability. In *Interoperability, Proc. International Symposium of Distributed Objects and Applications (DOA'03*, 2003.
7. IEEE-SA. Ieee 802.11 : Wireless lan medium access control (mac) and physical layer (phy) specifications. (2007 revision). `http://standards.ieee.org/getieee802/download/802.11-2007.pdf`, 2007 revision.
8. Feng Liu, Wu Chou, Li Li, et Jenny Li. WSIP - Web Service SIP Endpoint for Converged Multimedia/Multimodal Communication over IP. In *ICWS '04 : Proceedings of the IEEE International Conference on Web Services*, page 690, Washington, DC, USA, 2004. IEEE Computer Society.
9. Yi-Hsuan Lu, Yoojin Hong, Jinesh Varia, et Dongwon Lee. Pollock : automatic generation of virtual web services from web sites. In *SAC '05 : Proceedings of the 2005 ACM symposium on Applied computing*, pages 1650–1655, Santa Fe, NM, USA, 2005. ACM.
10. Marjan Mernik, Jan Heering, et Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4) :316–344, 2005.
11. Jan Rellermeyer, Oriana Riva, et Gustavo Alonso. Alfredo : An architecture for flexible interaction with electronic devices. *Middleware 2008*, pages 22–41, 2008.
12. Andreas Scholz, Christian Buckl, Alfons Kemper, Alois Knoll, Jörg Heuer, et Martin Winter. WS-AMUSE - web service architecture for multimedia services. In *ICSE '08 : Proceedings of the 30th international conference on Software engineering*, pages 703–712, New York, NY, USA, 2008. ACM.
13. Bluetooth Special Interest Group SIG. Core Specification v2.1 + EDR. `http://www.bluetooth.com/Bluetooth/Technology/Building/Specifications/`, July 2007.
14. W3C. State Machine Notation for Control Abstraction (SCXML). Disponible : `http://www.w3.org/TR/scxml/`, Octobre 2009.
15. Charles Zhang, Dapeng Gao, et Hans-Arno Jacobsen. Towards just-in-time middleware architectures. In *AOSD '05 : Proceedings of the 4th international conference on Aspect-oriented software development*, pages 63–74, New York, NY, USA, 2005. ACM.