Approche basée composition pour les applications sur une grille de cartes Java

Monia BEN BRAHIM[1], Feten BACCAR[1], Achraf KARRAY[1, 2], Maher BEN JEMAA[1], and Mohamed JMAIEL[1]

[1]Laboratoire ReDCAD, Ecole Nationale d'Ingénieurs, Université de Sfax, B.P 3038 Sfax, TUNISIE
[2]LaBRI, Université Bordeaux 1, 351 cours de la Libération 33405 Talence, FRANCE

{moniabrahimenis, Baccarf, karray_Achraf}@yahoo.fr
{Mohamed.Jmaiel, maher.benjemaa}@enis.rnu.tn

Résumé. Cet article présente une approche permettant le développement d'applications complexes à la base de services simples (appelés applets) installés sur une grille de cartes Java. Notre approche s'inspire de l'approche orientée service concernant la publication, la description et l'invocation de services élémentaires, et du langage BPEL[1] pour l'orchestration de services composés. Nous mettons l'accent sur le langage de composition et le moteur d'orchestration de services. Notre approche sera illustrée par le développement d'un processus métier qui fait des statistiques sur les maladies des patients sans être capable d'accéder à leurs données privées, stockées dans leurs cartes médicales.

1 Introduction

La carte à puce [1] est aujourd'hui considérée comme étant l'ordinateur le plus sécurisé [2]. Tirant profit de cette propriété, plusieurs travaux de recherche visent à l'intégrer dans les systèmes distribués afin de sécuriser des applications réparties. Le projet *Java Card Grid* [3], mis en place par le laboratoire LaBRI, est l'un de ces travaux qui a proposé une plate-forme de type grille de cartes Java (GCJ). Cette plate-forme est constituée par des cartes Java, et permet de garantir un haut niveau de sécurité grâce aux propriétés de ces cartes.

Pour faciliter l'accès aux applets installées dans les cartes Java, nous avons opté pour une architecture orientée service de la grille. Cependant le développement des applications clientes utilisant ces services ne se fait que par l'écriture du code. Ceci exige la maîtrise des APIs de programmation des applications hors carte (clientes) ainsi que du protocole APDU[2] de communication avec les cartes à puce. En effet, la

[1] Business Process Execution Language.
[2] Application Protocol Data Unit, protocole de communication avec la carte, orienté octet.

communication avec la carte se fait par des échanges de séquences de bytes (les APDUs) qui sont pauvrement structurées. La construction des messages APDUs doit absolument suivre la norme ISO7816 [11] qui décrit le format de cette commande. Ainsi, le programmeur doit faire preuve d'une bonne connaissance du protocole pour pouvoir développer des applications clientes.

Afin de faciliter cette tâche fastidieuse, nous proposons un environnement de support d'applications composées sur la GCJ.

Actuellement BPEL est le standard le plus adopté pour la description des processus métier essentiellement basés sur les services web. C'est pourquoi nous sommes inspirés de BPEL afin de concevoir un langage de description dédié pour la composition de services hébergés sur la GCJ. Nous avons également développé le moteur d'orchestration qui interprète ce langage et une interface graphique facilitant la découverte, l'invocation et la composition des services de la grille.

Nous décrivons dans la section suivante la plate-forme de la GCJ et la gestion de ses services. La section 3 présente les détails d'implémentation des outils facilitant l'exécution d'applications composées sur la grille. Après avoir illustré notre approche, dans la section 4, par un exemple d'une application composée par des services Card, nous exposerons, dans la section 5, les travaux réalisés pour faciliter l'accès aux services des cartes à puce tout en les comparant avec notre approche. Enfin, nous concluons en présentant brièvement les prochaines évolutions des outils développés.

2 La grille de cartes Java

2.1 La plate-forme de la grille de cartes Java

La plate-forme matérielle est mise en place dans une armoire en rack dans laquelle 16 lecteurs de type CCID sont connectés à un PC tournant sur Linux, à travers trois hubs USB chacun à 7 ports. Ce PC intermédiaire (appelé le serveur des cartes) constitue une passerelle entre les cartes et les utilisateurs de la grille.

L'infrastructure logicielle comprend quant à elle deux couches : une couche de bas niveau qui permet de gérer les communications entre clients et cartes. La deuxième couche, pouvant être considérée de haut niveau, a une architecture orientée service et comporte les phases de publication, découverte et invocation de services.

Faute d'accès au matériel, nous avons déployé une grille de cartes Java à petite échelle constituée seulement de 4 cartes à puce. Une photo de cette grille est montrée dans la figure 1, l'architecture matérielle correspondante est présentée dans la figure 2.

Afin de protéger les communications avec la carte, nous avons mis en place un protocole de sécurité qui consiste à établir des clés de session entre le client et l'application carte. L'objectif que nous poursuivons est de mettre en place un canal sécurisé, depuis le client vers la carte, dans lequel les messages envoyés seront cryp-

tés par ces clés de session. Les clés de session doivent être partagées seulement entre le client et la carte et doivent être gardées secrets pour les autres entités [4].

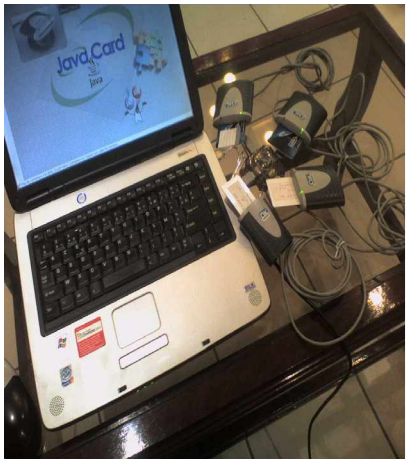

Fig. 1. Photo de la grille

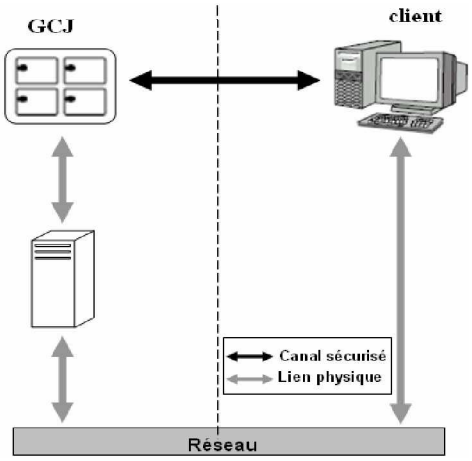

Fig. 2. Architecture matérielle de la GC

2.2 Gestion des services de la grille

Les architectures orientées services sont principalement basées sur la publication, la découverte et l'invocation de services. Dans notre contexte de la GCJ, nous avons introduit la notion d'un service Card et nous avons essayé de nous rapprocher des architectures dites orientées service, en se conformant aux trois phases précédemment citées, et tenant compte des propriétés des cartes à puce. Dans le cas d'une carte Java, un service représente une applet élémentaire installée dans la carte. Cette applet contient des méthodes qu'on peut invoquer en précisant la valeur du champ INS[3] à mettre dans la commande APDU. Chaque service Card doit avoir un descripteur pour qu'il puisse être publié par un fournisseur de services, découvert et invoqué par un client distant. Le descripteur d'un service est un fichier XML contenant une description de l'interface de l'applet implémentant le service (nom de l'applet, son AID[4], les noms de ses méthodes et les paramètres nécessaires pour les invoquer,…). Les couches logicielles de la gestion des services de la GCJ sont détaillées dans [5].

[3] Instruction byte
[4] Application IDentifier, permet d'identifier une applet dans la carte d'une façon unique

3 Mise en œuvre de la composition de services de la GCJ

Afin de mettre en œuvre un environnement de support de la composition de services de la GCJ, nous avons développé trois outils dont les rôles se complètent. Ces outils sont : un langage de définition d'un processus métier, un moteur d'orchestration d'un processus défini, et une interface graphique facilitant la découverte, l'invocation et la composition des services hébergés sur la GCJ

3.1 Le langage de définition d'un processus métier

Notre langage de définition permet de décrire un processus métier intégrant des services Card. A l'image de BPEL [6], il est basé sur XML et inclut des constructions fortement ressemblantes à celles de BPEL. Un processus métier correspond à une séquence d'opérations ou plus exactement à un flux d'activités. Ces activités peuvent faire intervenir un à plusieurs services Card. Notre langage supporte deux types d'activités : les activités de base et les activités structurées.

Les activités de base permettent d'invoquer un service Card (activité <*invoke*>) et de mettre à jour les valeurs des variables avec de nouvelles données (<*copy*>).

Les activités structurées utilisent les activités de base pour décrire des séquences ordonnées (<*sequence*>), des exécutions en parallèle (<*flow*>), des branchements (<*switch*>, <*if*>) et des boucles (<*while*>, <*for*>). Les activités <*flow*> contiennent des activités <*sequence*> qui s'exécutent en parallèle et qui peuvent être liées par des synchronisations. Chaque lien (<*link*>) rend une activité cible (<*target*>) dépendante d'une activité source (<*source*>). Les activités cibles peuvent être des activités de base <*copy*> et <*invoke*> qui ne peuvent pas utiliser des variables avant qu'elles soient modifiées par d'autres activités sources <*copy*> ou <*invoke*>.

Le langage permet aussi de déclarer des variables (<*variables*>) et des services (<*services*>). Les variables sont les paramètres d'entrée et de sortie du processus (<*input*>, <*output*>) ou d'autres variables locales utilisées dans le processus (<*variable*>). La déclaration d'un service se fait en indiquant le nom et la valeur du champ INS de la commande APDU relative au service, l'AID de l'applet implémentant le service, et le nom du CAD[5] et de la carte contenant cette applet. L'étude de cas traitée dans la section 4 montre la manière d'écriture de plusieurs constructions du langage de composition.

3.2 Le moteur d'orchestration

Afin d'assurer la gestion de l'exécution du processus de composition de services, le moteur d'orchestration peut être installé soit côté client soit côté serveur. Comme nous l'avons expliqué dans la section 2, nous avons mis en place un protocole qui permet d'établir un canal sécurisé de bout en bout, entre le client et la carte. La mise

[5] Card Acceptance Device, autre appellation pour lecteur de cartes

en place d'une telle sécurité implique que l'opération de cryptage/décryptage des messages ne pourra avoir lieu qu'au niveau de l'application cliente, et au niveau de la carte. D'où, il est essentiel d'installer le moteur d'orchestration au niveau de la machine cliente, afin de sécuriser les messages générés avant de les envoyer vers la grille.

Le moteur d'orchestration traite la description du processus métier d'une manière récursive et génère le code de l'application cliente utilisant l'API JPCSC[6]. Le code généré est conforme à la description de point de vue ordre d'exécution et interaction des services impliqués dans la composition. Le moteur d'orchestration crée alors dynamiquement des *threads Java* chaque fois où il y a des exécutions parallèles. Il utilise aussi le modèle producteur/consommateur pour implémenter la synchronisation entre les *threads*. Il doit convertir les paramètres d'entrée du processus métier en byte pour qu'ils soient manipulés par l'application hors carte. Pour chaque invocation d'un service Card, le moteur d'orchestration doit générer un code implémentant les étapes suivantes : 1-Sélectionner l'applet implémentant le service : Ceci est réalisé en envoyant la localisation du service (composé du nom de la carte et celui du lecteur) ainsi que l'identifiant du service. Le serveur envoie une commande Select Applet vers la carte spécifiée. 2- Etablir un canal sécurisé de bout en bout : Un canal sécurisé sera établi entre la machine cliente et la carte en question selon le protocole cryptographique que nous avons mentionné dans la section 2. 3-Construire et crypter la commande APDU qui sera envoyée vers la carte. 4- Après exécution de la commande sur la carte, il faut recevoir la réponse de la carte sous forme d'une R-APDU cryptée, la décrypter et extraire le résultat en byte afin de l'exploiter suivant la description du processus métier. À la fin de l'exécution de toute l'application hors carte, le moteur d'orchestration convertit le résultat final du byte au type spécifié pour le paramètre de sortie du processus. La figure 3 montre les principales fonctions du moteur d'orchestration, essentielles pour l'exécution du processus métier sur la GCJ.

3.3 L'interface graphique

Nous avons construit une interface graphique côté client afin de faciliter l'utilisation des services hébergés dans la GCJ. Il s'agit d'une fenêtre divisée en deux blocs. Le bloc à gauche permet d'afficher la liste de services sous forme d'une arborescence à 3 niveaux. Le premier niveau contient le nom du lecteur et la carte insérée dans ce lecteur. Le deuxième niveau visualise le nom de l'applet et son AID, et le dernier niveau affiche les noms des méthodes de l'applet du niveau supérieur. L'utilisateur peut sélectionner un service afin d'afficher une brève description de ce service, ou afin de l'invoquer. Le bloc à droite permet à l'utilisateur d'afficher le descripteur en XML d'un service, d'écrire en XML la description d'un processus métier, ou de charger une description d'un processus afin de l'exécuter. La figure 4 montre

[6] Une interface JINI (Java Native Interface) permettant d'accéder aux fonctionnalités de la carte à puce en utilisant le langage Java.

l'interface graphique du client lors de la demande d'exécution d'un processus métier chargé dans le bloc à droite.

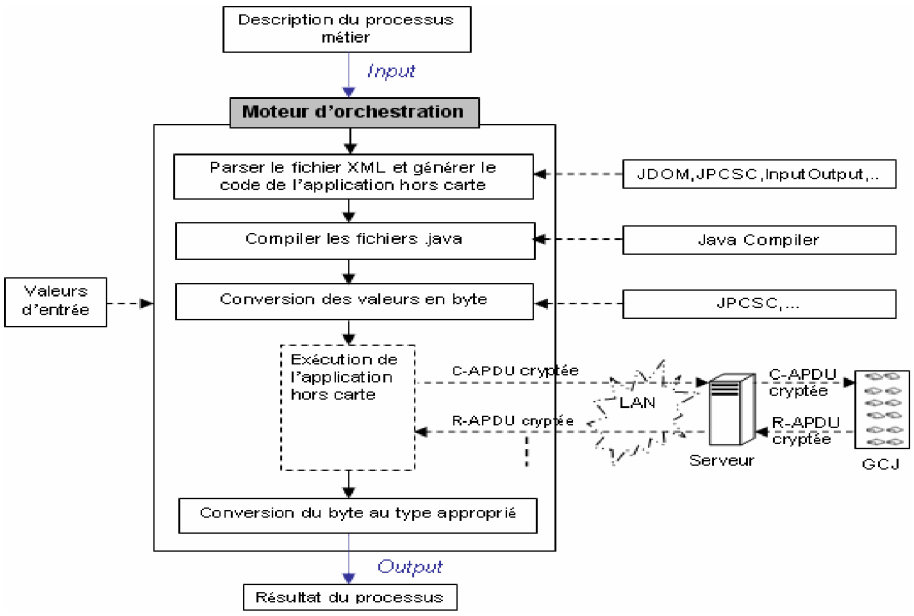

Fig. 3. Exécution d'un processus métier sur la GCJ

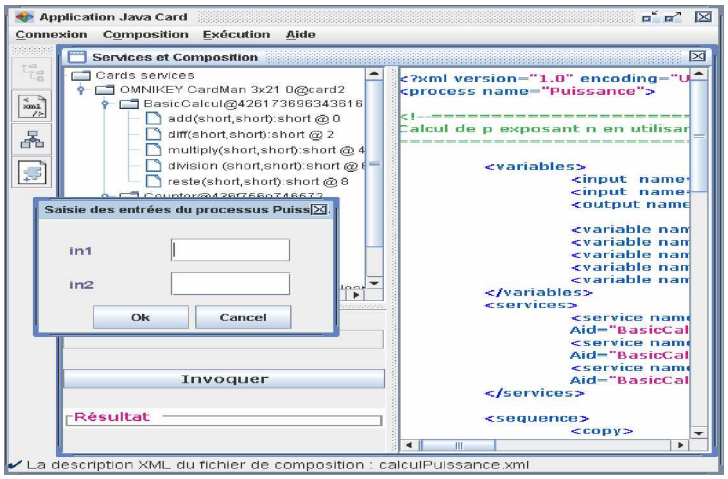

Fig. 4. demande d'exécution d'un processus métier

4 Etude de cas

Dans cette étude de cas, il s'agit de réaliser des statistiques sur une maladie *x* sans être capable d'accéder aux données privées du patient. Nous avons implémenté des applets gérant le dossier médical d'un patient, similaires à celles qui existent dans une carte médicale. L'applet *Maladies* contient plusieurs services gérant les données concernant les maladies d'un patient. Cependant, le statisticien ne peut invoquer que le service *malade* qui cherche si le patient souffre d'une maladie particulière entrée en argument. L'applet *Identité* inclut les services gérant les données d'identité du patient. Le statisticien ne peut invoquer que le service *ageInferieur* qui informe si l'âge du patient est inférieur ou égal à une certaine valeur entrée en argument. Nous avons aussi connecté à la grille une autre carte que nous avons appelée *carteStockage*. Cette carte héberge des applets calculant les statistiques des maladies. *maladieX* est l'une de ces applets, gérant les données statistiques d'une maladie *x* tel que le nombre de malades ayant l'âge inférieur ou égal à 20 ans (*nb1*) et le nombre de malades ayant l'âge strictement supérieur à 20 ans (*nb2*).Le processus métier développé par le statisticien interroge parallèlement des cartes médicales afin de savoir si le patient souffre d'une maladie *x*. Si le patient est malade, selon son âge le nombre *nb1* ou le nombre *nb2*, dans *carteStockage*, sera incrémenté. Nous présenterons dans ce qui suit des extraits du processus décrit par le statisticien :

```
//déclaration de quelques variables
       <variables>
              <input name="maladie" type= "String"/>
                     [...]
              <variable name="r1" type="boolean"/>
                      [...]
       </variables>

//déclaration d'un service
       <service name="malade" reader="OMNIKEY CardMan
3x210@patient1" Aid="Maladies@426173696343616C63756C" ins="4"/>

//invocation d'un service
       <invoke service="malade">
              <inputVariable>maladie</inputVariable>
              <outputVariable>r1</outputVariable>
       </invoke>
```

5 Travaux existants

Nous avons identifié un certain nombre de projets visant à intégrer des cartes Java dans un environnement distribué. Les plates-formes OrbaCard [7] et JCRMI [8] ont utilisé respectivement les technologies Corba et RMI, qui sont des technologies largement répandues dans les systèmes distribués. L'approche JiniCard [9] a utilisé l'environnement JINI [10].

Dans tous ces travaux, l'effort était orienté vers une exposition des services de la carte vers l'extérieur, tout en assurant les aspects de transparence par rapport au

protocole de communication APDU liée à la carte. Néanmoins, aucun de ces travaux ne s'est intéressé à la réalisation des outils facilitant la composition de services élémentaires. De plus, dans notre environnement les deux le calcul et les communications avec les cartes sont sécurisés.

6 Conclusion et futurs travaux

Dans cet article nous avons présenté la plate-forme GCJ et la gestion des services dans cette grille. Nous nous sommes intéressés en particulier à la notion de composition de services dans la plate forme afin de faciliter à l'utilisateur le développement des applications utilisant les services installés dans la grille. Nous avons donc présenté les détails de mise en oeuvre de la composition de services, concernant la description d'un processus métier et son exécution.

Prochainement, nous allons créer de nouvelles constructions afin d'intégrer des mécanismes qui gèrent les exceptions et les transactions. Nous visons aussi à réaliser un éditeur graphique d'un processus métier à l'image des outils de composition des services web tel que *Oracle BPEL Process Designer*.

Références

1. Rankl, W., Effing, W.: Smart Card Handbook 2nd edition. John Wiley & Sons (2000)
2. Bobineau, C., Bouganim, L., Pucheral, P., PicoDBMS. Valduriez, P.: Scaling down database techniques for smart card. In: Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt (2000)
3. Chaumette, S., Grange, P., Sauveron, D., Vignéras, P.: Computing with Java Cards. In: Proceedings of CCCT'03 and 9th ISAS'03, Orlando, FL, USA, July 31, August 1-2 (2003)
4. Karray, A.: Calcul sécurisé sur grille de cartes à puce. Master's thesis, ENIS. University of Sfax (2004)
5. Chaumette, S., Karray, A., Sauveron, D.: Secure Collaborative and Distributed Services in the Java Card Grid Platform. The 2006 International Symposium on Collaborative Technologies and Systems (CTS 2006), Workshop on Collaboration and Security (COLSEC'06) Las Vegas, Nevada, USA (2006)
6. Wohed, P., Aalst, W. M.P. van der, Dumas, M., Hofstede, A. H.M. ter.: Analysis of Web Services Composition Languages: The Case of BPEL4WS. In: Proceedings of 22nd International Conference on Conceptual Modeling, Chicago, IL, USA, October 13-16 (2003)
7. Chan, A.T., Tse, F., Cao, J., Leong, H.V.: Enabling distributed corba access to smart card applications. IEEE Internet Computing (2002) 27–36
8. Microsystem, I.S.: Java Card 2.2 Runtime Environment (JCRE) Specification, (2002) Remote Method Invocation Service, chapter 8, pages 53–68
9. Kehr, R., Rohs, M., Vogt, H.: Mobile code as an enabling technology for service-oriented smartcard middleware. In: Internationnal Symposium on Distributed Objects and Applications. (2000) 119–130
10. The Community Resource for Jini Technology. (`http://www.jini.org/`)
11. International Organization for Standardization: ISO7816. (`http://www.iso.ch/`)