Composition d'entités de contexte de ressources système

Denis Conan
GET / INT, CNRS UMR SAMOVAR
9 rue Charles Fourier, 91011 Évry, France
Denis.Conan@int-evry.fr

RÉSUMÉ

Dans ce papier, nous présentons le cadriciel COSMOS pour la composition d'entités de contexte de ressources système. Cette composition est optimisée et isolée des manipulations de plus haut niveau d'abstraction. Nous appliquons l'approche composants à la manipulation d'entités de contexte, et pour optimiser ces compositions, mettons à profit les capacités de gestion des ressources système des cadriciels construits autour du modèle de composant Fractal.

Mots clefs

Mobilité, contexte, ressources système, composants.

ABSTRACT

In this paper, we present the framework COSMOS, which composes context entities from system resources. This composition is optimised and isolated from the most abstract context manipulations. We apply the component approach to the manipulation of context entities, and in order to optimise these manipulations, we benefit from the system resource management capabilities of the frameworks available with the Fractal component model.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Data abstraction, Domain-specific architectures, Patterns

General Terms

Design

Keywords

Mobility, context, system resources, components.

1. INTRODUCTION

UbiMob'06 September 5–8, 2006, Paris, France

Les applications réparties s'exécutant dans des environnements ubiquitaires doivent continuellement s'adapter pendant l'exécution. Selon le dictionnaire Le Petit Robert, l'adaptation est *l'aptitude à modifier sa structure ou son comportement pour répondre à des situations nouvelles.* Les situations sont définies et identifiées à partir d'informations de contexte [6, 14]. Le contexte est lui-même constitué d'entités de différentes catégories (ressources système, préférences utilisateurs, capteurs...), de rôles de ces entités et de relations entre entités. Dans ce papier, nous ne considérons pas les changements de contextes, par exemple l'apparition d'une entité de contexte comme une nouvelle interface ou un nouveau type de réseau, mais nous intéressons à la transformation d'information de contexte pour l'identification de situations dans un contexte donné, par exemple la perte de connectivité du réseau WiFi.

L'approche empruntée ici consiste à séparer les fonctionnalités d'une architecture de gestion de contexte pour des raisons d'hétérogénéité (différentes sources de contextes), de puissance d'expression (différents opérateurs d'abstraction d'entités de contexte de haut niveau), d'adaptation aux applications (différents motifs de conception et domaines applicatifs) et d'efficience (différents besoins en ressources système pour la collecte, la transformation et l'exploitation de ces informations de contexte). Cette séparation des préoccupations implique la décomposition de l'architecture d'un gestionnaire de contexte en plusieurs cadriciels.

Dans ce papier, nous définissons un cadriciel pour la composition d'entités de contexte de ressources système (processeur, mémoire, réseau...) réifiées, appel COSMOS pour *COntext entitieS coMpositiOn and Sharing.* Cette composition est optimisée et isolée des manipulations de plus haut niveau d'abstraction. Nous appliquons l'approche composants à la manipulation d'entités de contexte, et pour optimiser ces compositions, mettons à profit les capacités de gestion de ressources système des cadriciels construits autour du modèle de composants Fractal [2].

La suite du papier est organisée comme suit. La section 2 motive la conception d'un cadriciel dédié à la composition d'entités de contexte de ressources système. Puis, la section 3 présente le cadriciel COSMOS. Enfin, les sections 4 et 5 positionnent nos travaux par rapport à ceux de la littérature, et concluent en donnant l'état d'avancement du prototype et les perspectives de ce travail.

2. MOTIVATIONS ET OBJECTIFS

L'architecture présentée dans la figure 1 (sans les parties grisées) est inspirée de [6, 9, 16]. Les briques de base sont

la réification des ressources système, des capteurs à proximité du terminal et des préférences de l'utilisateur, la distribution des informations de contexte, puis le traitement des informations de contexte par ce que nous appellerons un *processeur de contexte*, et enfin, la sensibilité au contexte de l'application, par exemple, orientée composants. La réification permet de collecter des données brutes, souvent numériques, comme la qualité du lien réseau. Le processeur de contexte en déduit des informations de contexte de plus haut niveau, souvent des données symboliques comme le mode de connectivité (connecté, partiellement connecté ou déconnecté) et permet d'identifier des situations comme la perte prochaine de la connectivité réseau. Les politiques d'adaptation, quant à elles, sont généralement gérées en collaboration avec l'application : par exemple, l'exploitation consiste à avertir l'utilisateur du mode de connectivité à l'aide d'un icone dédié.

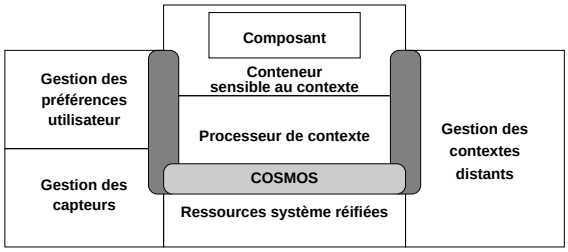

Figure 1: Architecture de gestion de contexte

Dans la littérature, le processeur de contexte est le point central de cette architecture, rassemblant les données brutes, les interprétant en données symboliques et identifiant des situations d'adaptation. Suivant le principe de séparation des préoccupations, nous proposons de réduire son rôle à la collecte de données symboliques de bas niveau pour les interpréter en données symboliques de haut niveau et identifier des situations. Autrement dit, nous introduisons des briques logicielles (les parties grisées de la figure 1), que nous appelons des *processeurs d'entités de contexte de ressources*, et qui sont en charge de la transformation des données brutes sur les ressources (système, capteurs, préférences de l'utilisateur) en données symboliques de bas niveau.

La première raison de ce choix est que les cycles de vie des éléments observables sont très différents. En effet, la collecte des données brutes (découverte, filtrage, gestion de l'historique, précision...) dépend du type de ressources. Ces spécificités sont gérées par les nouveaux cadriciels. En outre, les ontologies des collecteurs de contexte sont diverses et variées. Les transformations d'ontologies doivent être effectuées lorsque nécessaires sans engorger le processeur de contexte avec ces transformations. Par ailleurs, les ressources système affectées à la collecte et aux manipulations des données brutes doivent être finement contrôlées. À titre d'exemple, les activités (en anglais, *threads*) créées pour la collecte des données brutes peuvent inhiber et rendre inutilisable le terminal de l'utilisateur si leur nombre est trop grand. La même remarque s'applique pour d'autres ressources système comme l'espace mémoire (pour les historiques). Les nouveaux cadriciels sont donc isolés et leurs ressources contrôlées. Une autre raison pour l'isolation des collecteurs de contexte du processeur de contexte est d'éviter que les requêtes (synchrones) des applications se traduisent par la collecte (elle-aussi synchrone) des informations brutes avec appels (eux-aussi synchrones) au système d'exploitation. Enfin, les cadriciels de collecte de données brutes étant potentiellement nombreux, les processeurs de contexte de ressources servent aussi à interfacer ces cadriciels avec un processeur de contexte. Cela permet d'interfacer des cadriciels avec des styles architecturaux différents : par exemple, selon [6], orienté processus ou orienté données.

3. COSMOS

Dans cette section, nous présentons trs succintement le cadriciel COSMOS de composition de contextes de ressources système réifiées. La section 3.1 expose les principes et la section 3.2 en donne les principaux composants.

3.1 Principes de conception

De manière classique, nous appliquons les principes de base de la construction d'intergiciel, c'est-à-dire le cadriciel doit être construit à partir d'éléments génériques et spécialisables, modulaire afin de composer plutôt que de programmer (d'où l'orientation composants avec le modèle de composants hiérarchiques et partagés Fractal), extensible pour facilement prendre en compte de nouveaux éléments (ressources observables, rapports d'observation, opérateurs de composition), et dynamique pour permettre l'adaptation en cours d'exécution.

Les principes de conception qui suivent sont spécifiques à la gestion de contexte. Les entités de contexte sont organisées en une hiérarchie avec possibilité de partage. Tous les composants de la hiérarchie sont accessibles. Les feuilles de l'arbre sont des composants primitifs, les autres nœuds des composites. Les composants des nœuds feuilles de l'arbre encapsulent les objets réifiés (dans notre cas, avec SAJE [11]) et les relations entre les nœuds sont des relations d'encapsulation. Chaque nœud peut être passif ou actif, avec exécution périodique de tâches dans des activités. Les rapports d'observation contenant les informations de contexte circulent de bas en haut de la hiérarchie, mais la circulation peut s'effectuer soit à la demande d'un nœud parent (c'est une observation), soit à l'initiative d'un nœud enfant (c'est une notification). Lors d'une observation ou d'une notification, le composant qui traite la requête peut être passant ou bloquant. Lors d'une observation, un composant de contexte passant demande d'abord un nouveau rapport d'observation à ses enfants, puis calcule un rapport d'observation pour le transmettre. Lors d'une notification, un composant de contexte passant calcule un nouveau rapport d'observation avec la nouvelle notification, puis passe vers le haut ce rapport en notifiant ses parents.

3.2 Architecture de quelques composants

COSMOS utilise le modèle de composant Fractal du consortium ObjectWeb[1]. Comme représenté dans la figure 2, un composant Fractal est une entité logicielle qui fournit et requiert des services regroupés au sein d'interfaces. Le modèle distingue les interfaces dites *serveur* (sur la gauche), des interfaces dites *client* (sur la droite) et des interfaces de contrôle (sur le haut). Un composant Fractal possède un contenu, qui peut être composé d'un ensemble de sous-composants. Dans ce cas, le composant est dit composite.

[1] `http://fractal.objectweb.org`.

Au dernier niveau de récursion, les composants sont dits primitifs. Les hiérarchies de composants Fractal sont des graphes quelconques (ce ne sont pas nécessairement des arbres). Les composants inclus dans plusieurs composites sont dits partagés. Cette notion représente de façon naturelle le partage de ressources système (segments de mémoire, activités. Les composants Fractal sont assemblés à l'aide de liaisons représentant un chemin de communication entre deux composants, plus précisément entre une interface requise (client) et une interface compatible fournie (serveur).

Toutes les entités de contexte sont des composants étendant le composant composite abstrait `ContextNode` (cf. figure 2). Hormis pour la gestion des ressources consommées, l'architecture d'un `ContextNode` est similaire à celle d'un contexteur [15, 7, 14] : les interfaces `Pull` et `Push` sont les interfaces pour l'observation et la notification, respectivement. Le composite abstrait `ContextNode` contient au moins un opérateur (composant primitif abstrait `ContextOperator`) et les composants partags d'accès aux gestionnaires de messages et d'activités de Dream[2]. Les rapports d'observation sont des messages Dream, constitués de morceaux (en anglais, *chunks*) et de sous-messages. Dans COSMOS, toutes les informations lmentaires des rapports d'observation des observables du canevas logiciel SAJE donnent lieu un morceau de message typ: par exemple la qualit du lien rseau WiFi est mmorise dans un morceau de message de type `LinkQualityChunk`. Donc, au-dessus des nœuds gestionnaires de ressources qui constituent les feuilles de la hirarchie, les infrences de contexte de plus haut niveau sont indpendantes du canevas logiciel utilis pour la collecte des donnes brutes. En outre, le composant `ContextNode` est spécialisable via des attributs pour la gestion des activités pour l'observation et la notification, et pour la propriété passant/bloquant. Par défaut, les nœuds sont passifs ($isActiveXxx = false$) et passants ($xxxThrough = true$) pour l'observation et la notification.

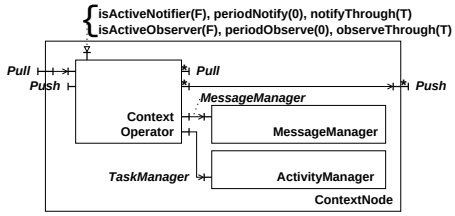

Figure 2: Composite abstrait `ContextNode`

Tous les nœuds de la hiérarchie peuvent être gérés finement, tant au niveau de leur cycle de vie, qu'au niveau de leur accès et de la gestion des ressources qu'ils consomment. Par exemple, pour la gestion des activités, les composants de contexte `ContextNode` actifs enregistrent leurs tâches au service de gestion d'activités. Le service de gestion des activités, lui-même paramétrable, peut créer une activité par tâche (observation ou notification) par composant actif ou bien une activité par composant actif ou encore une activité pour toute la hiérarchie.

Les nœuds de la hiérarchie se classent ensuite en deux catégories : les feuilles et les autres. Pour les feuilles, le `ContextNode` est étendu pour contenir, en plus de l'opérateur et du générateur d'identifiants, un composant primitif qui encapsule un objet SAJE réifiant une ressource système. Pour les autres nœuds de la hiérarchie, le `ContextNode` est étendu pour y ajouter un ou plusieurs nœuds composites de type `ContextNode`. Dans la suite de cette section, nous présentons un exemple de chaque catégorie.

Les nœuds feuilles de la hiérarchie sont de type `PeriodicResourceManager`. Ce composite abstrait générique est paramétrable pour que l'utilisateur fournisse, via l'attributs `resourceName`, le nom *système* de la ressource : par exemple, une interface réseau WiFi de nom `eth1`. L'opérateur `ForwarderCO` est générique et mémorise n'importe quel type de messages. Les composants génériques et spécifiques à SAJE étant fournis dans le cadriciel COSMOS, les seules lignes de code que l'utilisateur doit écrire pour obtenir un composite concret, sont les lignes en Fractal ADL suivantes :

```
<definition name="PeriodicWireless" extends="PeriodicResourceManager">
  <component name="rm" definition="WirelessInterfaceRM(eth1)" />
</definition>
```

À titre d'exemple de nœud autre qu'un nœud feuille, voici le type de nœud qui encapsule un `PeriodicResourceManager`. Il s'agit du composite abstrait paramétrable `SimpleValueCO`, qui extrait une valeur (par exemple la qualité du lien WiFi) d'un message. L'opérateur `SimpleValueCO` est générique et étend l'opérateur `ForwarderCO` : il est paramétrable via des attributs pour préciser le morceau de message à extraire. Pour la concrétisation, l'utilisateur écrit quelques lignes en Fractal ADL comme suit :

```
<definition name="LinkQuality" extends="SimpleValueManager">
  <component name="cm" definition="SimpleValueCO(LinkQualityChunk)" />
  <component name="cn" definition="PeriodicWireless"/>
</definition>
```

Enfin, le cadriciel COSMOS contient d'autres opérateurs génériques.

4. TRAVAUX CONNEXES

Parmi les premiers travaux sur la gestion de contexte, Context Toolkit [9] est un cadriciel orienté objets empruntant aux IHM les concepts de programmation événementielle et de *widget* pour la collecte du contexte des ressources. Dans le même cadriciel, sont rassemblées les autres fonctionnalités : l'interpréteur pour composer et abstraire les informations de contexte, l'*aggregator* pour la médiation avec l'application, le *service* pour contrôler les actions de l'application sur le contexte et le *discoverer* qui est un serveur de noms ou registre. La gestion des ressources système, et plus particulièrement, la gestion des activités est explicitement laissée de côté.

Une autre solution de la littérature sont les Contexteurs [15, 7, 14], qui sont des entités logicielles ressemblant à des composants avec des données et leurs métadonnées (décrivant la qualité des données) en entrée et en sortie, et des contrôles (modifications de paramètres) en entrée et en sortie. Le cadriciel est construit comme un réseau de Contexteurs, de la collecte des données brutes à l'identification de situations pour les applications qui s'y abonnent. Chaque Contexteur possédant une activité, la consommation de ressources locales pose question.

[2]Dream [12] est un canevas logiciel proposant entre autres des gestionnaires de messages et d'activités que nous utilisons dans COSMOS.

Dans ces solutions, les mêmes concepts et mécanismes sont appliqués de la collecte des données numériques jusqu'à la fourniture de contexte abstrait à l'application. Les fonctionnalités ne sont pas séparées et le contrôle des ressources système consommées n'est pas explicitement exprimé. En outre, nous ne pensons pas qu'un seul cadriciel puisse adresser la diversité des dispositifs matériels et systèmes (d'où l'existence de cadriciels dédiés à la collecte comme LeWYS [5], SAJE [11] et WildCAT [8]), pas plus que la diversité des exploitations par les applications (d'où l'existence de processeurs de contextes qui expérimentent de nouveaux opérateurs comme CARISMA [4], WComp [10], et l'existence de conteneurs sensibles au contexte qui étudient plus précisément les mécanismes de collaboration entre l'application et l'intergiciel pour l'adaptation comme CAMidO [1], Enactor [13], JADE-LEAP [3] et RCSM [16]). Une autre différence importante dans notre proposition est que le cadriciel, moyennant la programmation d'opérateurs génériques, grâce à l'approche orientée composants, permet la composition des nœuds de contexte (approche déclarative), alors que toutes les solutions précédentes suivent une approche par programmation.

5. CONCLUSIONS

Dans ce papier, nous identifions la nécessité de séparer les transformations d'abstraction de bas niveau des données brutes des ressources et les transformations de haut niveau avec identification des situations d'adaptation. Dans ce cadre, nous introduisons COSMOS, un cadriciel qui permet une première abstraction des informations sur les ressources système tout en autorisant une maîtrise fine des ressources système consommées pour ces manipulations. Une première version de COSMOS est disponible sous licence GNU LGPL à l'URL `http://picolibre.int-evry.fr/projects/cosmos`.

COSMOS constitue une brique de base pour la construction d'une plateforme d'adaptation des applications réparties en environnements mobiles : notamment par la distribution des informations de contexte pour la détection de déconnexions et l'abstraction des informations de contexte pour une exploitation dans des conteneurs de composants sensibles au contexte.

Remerciements

Ce travail a bénéficié des commentaires et de l'aide des membres du projet INRIA Jacquard, et plus particulièrement, Jérémy Dubus, Laurence Duchien, Areski Flissi, Philippe Merle, Romain Rouvoy et Lionel Seinturier.

6. RÉFÉRENCES

[1] Belhanafi Behlouli, N., Taconet, C., and Bernard, G. An architecture for supporting Development and Execution of Context-Aware Component applications. In *Proc. IEEE International Conference on Pervasive Services* (June 2006).

[2] Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., and Stefani, J.-B. An Open Component Model and its Support in Java. In *Proc. 7th International Symposium on Component-Based Software Engineering* (Edimburgh, UK, May 2004), vol. 3054 of *Lecture Notes in Computer Science*, pp. 7–22.

[3] Bucur, O., Beaune, P., and Boissier, O. Définition et Représentation du Contexte pour des Agents Sensibles au Contexte. In *Actes de la 2ème Conférence ACM Francophone Mobilité et Ubiquité* (June 2005), pp. 13–16.

[4] Capra, L., Emmerich, W., and Mascolo, C. CARISMA: Contex-Aware Reflective mIddleware System for Mobile Applications. *IEEE Transactions on Software Engineering 29*, 10 (Oct. 2003), 929–945.

[5] Cecchet, E., Elmeleegy, H., Layaida, O., and Quéma, V. Implementing Probes For J2EE Cluster Monitoring. In *Proc. OOPSLA Workshop on Component And Middleware Performance* (Oct. 2004), pp. 1–10.

[6] Coutaz, J., Crowley, J., Dobson, S., and Garlan, D. The disappearing computer: Context is Key. *Communications of the ACM 48*, 3 (Mar. 2005), 49–53.

[7] Coutaz, J., and Rey, G. Foundations for a Theory of Contextors. In *Computer-Aided Design of User Interfaces III, Proc. 4th International Conference on Computer-Aided Design of User Interfaces* (May 2002), pp. 13–34.

[8] David, P., and Ledoux, T. WildCAT: a generic framework for context-aware applications. In *Proc. 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing* (Nov. 2005), pp. 1–7.

[9] Dey, A., Salber, D., and Abowd, G. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Special issue on context-aware computing in the Human-Computer Interaction Journal 16*, 2–4 (2001), 97–166.

[10] Lavirotte, S., Lingrand, D., and Tigli, J.-Y. Définition du contexte: fonctions de coût et méthodes de sélection. In *Actes de la 2ème Conférence ACM Francophone Mobilité et Ubiquité* (June 2005), pp. 9–12.

[11] Le Sommer, N. Contractualisation des ressources pour le déploiement des composants logiciels. In *Proc. 1è Conférence Francophone sur le Déploiement et la (Re) Configuration de Logiciels* (Oct. 2004).

[12] Leclercq, M., Quéma, V., and Stefani, J.-B. DREAM: a Component Framework for the Construction of Resource-Aware, Configurable MOMs. *IEEE Distributed Systems Online* (Sept. 2005).

[13] Newberger, A., and Dey, A. Designer Support for Context Monitoring and Control. Technical Report IRB-TR-03-017, Intel Research, Berkeley, USA, June 2003.

[14] Rey, G. *Contexte en Interaction Homme-Machine : le contexteur*. PhD thesis, Université Joseph Fourier, Grenoble, France, Aug. 2005.

[15] Rey, G., and Coutaz, J. Le Contexteur: Capture et distribution dynamique d'information contextuelle. In *Actes de la 1ère Conférence ACM Francophone Mobilité et Ubiquité* (June 2004), pp. 131–138.

[16] Yau, S., Karim, F., Wang, Y., Wang, B., and Gupta, S. Reconfigurable Context-Sensitive Middleware for Pervasive Computing. *IEEE Pervasive Computing 1*, 3 (July 2002), 33–40.