Le Contexteur : une Abstraction Logicielle pour la Réalisation de Systèmes Interactifs Sensibles au Contexte

Gaëtan Rey

CLIPS-IMAG
385 rue de la Bibliothèque
38041 Grenoble cedex 9, France
Gaetan.Rey@imag.fr

Joëlle Coutaz

CLIPS-IMAG
385 rue de la Bibliothèque
38041 Grenoble cedex 9, France
Joelle.Coutaz@ imag.fr

RESUME

En l'absence d'outil, les connaissances sur le contexte d'interaction, pourtant identifiées dans les phases amont du processus de conception, se diluent progressivement au cours de la mise en œuvre. Dans cet article, nous proposons une définition de la notion de contexte d'interaction. Nous présentons ensuite le contexteur, une abstraction logicielle qui fournit un support opérationnel au contexte d'interaction. Nous indiquons comment les contexteurs se composent en niveaux d'abstraction et comment ces niveaux s'inscrivent dans le modèle de référence Arch. Nous montrons une architecture de mise en œuvre des contexteurs fondée sur les principes P2P. Similaire dans l'esprit au Context Toolkit, nous explicitons nos différences.

MOTS CLES : contexte d'interaction, modélisation du contexte, architecture logicielle, Interaction Homme-Machine, informatique diffuse.

ABSTRACT

Important information about interaction context is developed during the design process. In this article, we propose a formalism to encode and preserve this information during system implementation. We begin by proposing a definition of interaction context. We then present the "Contextor", a software concept that provides developers with a tool for modeling context. We show how contextors can be composed into layers of abstraction and how contextors fit within the Arch software architecture model. We present an implementation architecture based on the Peer to Peer computing

KEYWORDS : Interaction context, context modeling, software architecture modeling, Human Computer Interaction, ubiquitous computing.

INTRODUCTION

L'étude du contexte constitue l'un des piliers du processus de conception de systèmes centrés sur l'utilisateur. L'approche *contextual design* de [3], fondée sur la récolte (*contextual inquiry*) et l'organisation de données de terrain, est un exemple représentatif de l'utilisation du contexte en conception. De même, l'Action Située (*Situated Action*) [19], la Théorie de l'Activité [1] ou encore la Cognition Distribuée (*Distributed Cognition*) [11] offrent des vues complémentaires à la décomposition hiérarchique des modèles de tâches à-la-GOMS [8], en considérant le contexte comme pièce maîtresse du processus de conception.

Si l'importance du contexte est acquise en conception d'IHM, les systèmes interactifs sont, en pratique, réalisés pour une situation d'usage donnée. En l'absence d'outil, les connaissances sur le contexte, pourtant identifiées dans les phases amont du processus de développement, se diluent progressivement au cours de la mise en œuvre. En conséquence, le triangle classique « utilisateur-tâche-machine » ne fonctionne que pour un contexte figé prévu à l'avance : une classe d'utilisateurs représentatifs, accomplissant des tâches en un lieu donné, au moyen d'un ordinateur-type dont les ressources d'interaction sont fixes. Or, la technologie aidant, l'utilisateur est mobile, les dispositifs d'interaction (téléphones et assistants numériques personnels) aussi. Dans ces conditions, un système doit être sensible au contexte

Un système interactif est *sensible au contexte* s'il est capable d'identifier les circonstances qui entourent l'action utilisateur en vue d'offrir des *services contextualisés*: migration de tâche, offre sélective d'information, décorations contextuelles pour archivage et recherche ultérieure d'information [9]. Dans cet article, nous traitons du contexte et du logiciel de base nécessaire à la mise en œuvre de systèmes sensibles au contexte. Comme pour le développement des Interfaces Homme-Machine graphiques traditionnelles, le logiciel de base pour systèmes sensibles au contexte inclut une infrastructure de type intergiciel (middleware), des boîtes à outils, voire des squelettes d'application. Dans cet article, nous nous intéressons à l'aspect boîte à outils et à son utilisation dans les systèmes interactifs sous forme d'architecture logi-

cielle. Nos travaux relèvent du même esprit que les études pionnières de Dey et al. sur le sujet. On relèvera donc des similitudes, mais aussi des différences et des compléments aux propositions originales de Dey et al.

Cet article est structuré comme suit : dans la section qui suit, nous précisons la notion de contexte et nous proposons une définition opérationnelle qui sert la conception et la mise en œuvre de systèmes interactifs. Nous sommes alors en mesure de proposer un modèle computationnel conceptuel avec la notion de contexteur. Nous en proposons ensuite une mise en œuvre suivie d'un exemple d'application.

CONTEXTE D'INTERACTION

Si la sensibilité au contexte est reconnue comme un facteur de qualité, il n'existe pas en ce jour de consensus sur une définition précise conduisant à une exploitation opérationnelle de la notion de contexte. L'analyse de la littérature appelle les deux conclusions suivantes : 1) Il n'y a *pas de contexte sans contexte*. Autrement dit, la notion de contexte se définit pour une finalité donnée. Dans le cadre de notre étude, la notion de contexte désigne le contexte d'interaction. 2) Pour un contexte d'interaction donné, nous distinguons l'état contextuel instantané de la capitalisation de ces états au cours du temps. Cette distinction est présentée ci-dessous après un rapide état de l'art.

Brève revue de l'état de l'art

La présentation qui suit est organisée selon l'ordre chronologique, démontrant une progression dans la compréhension de la notion de contexte.

Pour Schilit, étudier le contexte c'est répondre aux *questions quintiliennes* « Où es-tu ? », « Avec qui es-tu ? », « De quelles ressources disposes-tu à proximité ? » [18]. Schilit définit donc le contexte comme les changements de l'environnement physique, utilisateur et computationnel. Ces idées seront reprises par Pascoe [12] puis par Dey [10].

Brown restreint d'abord le contexte aux éléments de l'environnement de l'utilisateur que le *système informatique connaît* [5]. Puis, il considère que le contexte est plutôt « la localisation et l'identité des personnes qui entourent l'utilisateur ainsi que l'heure, la saison, la température ... » [4].

Parallèlement aux travaux de Brown, des définitions émergent avec l'introduction explicite du *temps* et la notion *d'état*. Ryan définit le contexte comme l'environnement, l'identité et la localisation de l'utilisateur ainsi que le temps [15]. Ward voit le contexte comme les états des environnements possibles de l'application [24].

Avec la définition de Pascoe, on retrouve la notion d'état à laquelle s'ajoute celle de *pertinence* : le contexte est « un sous-ensemble des états physiques et conceptuels ayant un intérêt pour une entité particulière » [12]. Puis Dey précise la notion *d'entité* : « le contexte couvre toutes les informations pouvant être utilisées pour caractériser la situation d'une entité. Une entité est une personne, un lieu, ou un objet qui peut être pertinent pour l'interaction entre l'utilisateur et l'application, y compris l'utilisateur et l'application eux-mêmes» [10].

Au bilan, tous les auteurs font référence à la localisation, à l'environnement physique, mais aucun ne considère les dispositifs d'interaction, ni l'activité utilisateur. Or les entités pertinentes ne peuvent se définir qu'en relation avec l'activité. Pour tous, le temps est un facteur décisif et se trouve fortement lié à celui d'état. L'état permet de caractériser ce qui est pertinent à un instant donné, mais l'évolution de cet état au cours du temps constitue à son tour une information qui peut être influente. Or aucune des définitions ci-dessus ne fait référence à la capitalisation des états au cours du temps. Cette analyse nous conduit à formuler la définition suivante.

Contexte d'interaction : définitions

Le contexte d'interaction n'existe pas en tant que tel. Il existe pour une finalité. Dans notre cas, la finalité est de permettre à un système interactif de fournir des services contextualisés à un utilisateur donné engagé dans une activité à un instant t.

Contexte d'interaction = combinaison de situations
Etant donné un utilisateur U engagé dans une activité A, le contexte d'interaction à l'instant t est la composition des situations entre les instants t_0 et t de conduite de A par U.

La notion de situation traduit les « circonstances qui entourent l'action » à l'instant t qui peuvent avoir un impact sur la conduite de l'activité A par U. La composition de situations traduit les relations reliant les situations au cours du temps à partir d'une date d'observation de référence t_0.

Situation = ensemble de variables et leurs relations en t
Etant donné un utilisateur U, une activité A et un instant t, la situation à l'instant t est l'ensemble des états des variables périphériques à A pouvant influencer A à l'instant t et/ou ultérieurement, et de l'ensemble des relations entre ces variables en t.

La distinction entre *périphérique* et *central* est assurée dans l'étape amont du processus de conception. Les techniques de type « Contextual Design » [21], [6] ont notamment pour mission d'identifier les entités pertinentes et, parmi ces entités, de distinguer celles qui sont ma-

nipulées au cours de l'activité (les entités centrales) des entités qui sont potentiellement influentes (les entités périphériques) [7].

Dans l'exemple d'une billetterie électronique, les notions de date, de place et de tarifs sont centrales pour la tâche de réservation, tandis que le lieu (hall de gare ou chez soi) est périphérique. Néanmoins, le fait que ce lieu puisse être public ou privé présente un impact potentiel, maintenant ou plus tard. (Suite à une mauvaise expérience en un lieu, la tâche sera peut-être effectuée autrement la prochaine fois, voire pas du tout !)

Nature des variables d'état
Afin de faciliter l'analyse d'une situation donnée, nous proposons d'organiser les variables (attributs des entités) en trois classes :

- Environnement physique et social
- Plate-forme d'interaction
- Profil de l'utilisateur

L'environnement physique et social couvre l'ensemble des attributs des entités constituant des lieux d'interaction : espace physique (salle, train, parc), objets physiques s'y trouvant, personnes présentes (excepté l'utilisateur).

La plate-forme inclut tous les attributs caractérisant les ressources d'interaction et de calcul disponibles, depuis la station usuelle, au PDA, et de manière plus nouvelle, les surfaces augmentées comme le tableau magique [2], I-land [20], les Data Tiles [13]…

L'exemple suivant d'une activité de bureau illustre les trois classes de variables :

- Le bureau, lieu d'interaction, est une entité de l'environnement physique. Ses attributs incluent : sa dimension, localisation, etc. Il contient des objets physiques (stylos, chaises, tables, etc.) qui ont des attributs (couleurs, taille, position …) qui, à leur tour, participent à la définition de l'environnement physique.
- Le profil de l'utilisateur inclut les informations concernant le « propriétaire » du bureau (son agenda, sa fonction, ses goûts, ses habitudes, etc.).
- Les personnes qui visitent occasionnellement le bureau voient leur profil (informations les concernant) s'ajouter à l'environnement social d'une situation qui correspond à leur séjour dans le bureau.
- La plate-forme d'interaction est composée des attributs des différents ordinateurs des utilisateurs et des autres personnes se trouvant dans le bureau. Ces attributs représentent, par exemple, la taille du disque dur, la mémoire disponible, la vitesse des processeurs ou encore la bande passante du réseau.

Notre définition du contexte vaut selon plusieurs points de vue que nous explicitons dans la typologie qui suit.

Typologie du contexte d'interaction
Comme le montre la figure 1, il convient de distinguer trois points de vue (omniscient, système, utilisateur) sur le contexte et deux étapes essentielles du processus de développement : l'étape de conception/analyse et l'étape d'exécution.

On dénomme *contexte brut* (omniscient) tous les faits universels absolus en relation avec A et U à l'instant t. Le *contexte système* est la modélisation du contexte par le système en relation avec A et U à l'instant t. Le *contexte utilisateur* est la modélisation du contexte par l'utilisateur U en relation avec A à l'instant t.

Par définition, les contextes système et utilisateur sont inclus dans le contexte brut. Mais les contextes système et utilisateur ne se recouvrent pas nécessairement : l'utilisateur a un vécu (les variables et leurs relations qu'il modélise mentalement) que le système ignore (à tort ou à raison). Inversement le système dispose de variables et de relations que l'utilisateur ignore.

Le *contexte net* correspond à l'intersection des contextes système et utilisateur. Il représente ce que le système et l'utilisateur ont en commun pour la conduite de A en t.

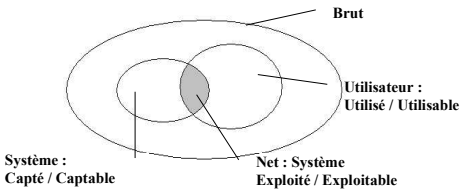

Figure 1 : Relations ensemblistes des contextes Brut, Système, Utilisateur et Net.

Ces trois points de vue peuvent s'analyser selon les deux étapes de déploiement du contexte :

- A l'étape de conception, les analystes et concepteurs identifient les requis et les confrontent aux contraintes techniques et sociales de faisabilité, etc. Il en résulte un ensemble de « souhaitables ».
- A l'exécution, les « effectifs » ne sont pas nécessairement conformes aux « souhaitables » : cas des pannes, cas de détournement de la technologie par les utilisateurs, etc.

Pour cette raison, on distinguera : Pour le contexte système : le *contexte captable* du *contexte capté*. Le contexte captable dénote le contexte système tel qu'il a été défini à l'étape d'analyse/conception pour l'activité A de U en t. Le contexte capté est le contexte tel qu'il est modélisé en pratique à l'exécution par le système : des capteurs ont pu tomber en panne.

Pour le contexte utilisateur : le *contexte utilisable* et le *contexte utilisé*. En analyse amont, les concepteurs ont défini le contexte utilisable : l'ensemble des facteurs (les variables périphériques) auxquels l'utilisateur risque de faire appel dans la conduite de A. Le contexte utilisé dénote les facteurs influents en pratique.

A l'intersection des points de vue système et utilisateur : le *contexte net exploitable* et le *contexte net exploité* obtenus respectivement à l'intersection des contextes captable et utilisable, et des contextes capté et utilisé.

La distinction entre le *–able* et le *–é* permet de fournir des bases à l'évaluation de la conception du système comme à ses performances effectives.

Les valeurs des variables (de type plate-forme, utilisateur, environnement physique et social), et leurs relations sont acquises et/ou calculées par des unités de calcul appelées contexteurs.

LES CONTEXTEURS

Un contexteur est une abstraction logicielle qui fournit la valeur d'une variable du Contexte Système. Nous en fournissons une description, nous en précisons les propriétés, puis nous indiquons comment composer les contexteurs en unités fonctionnelles plus riches.

Description

Inspiré des « context-handling components » de Salber [17], un contexteur comprend trois classes d'éléments : entrées, sorties et un corps fonctionnel.

Les entrées sont de 2 types : Le *contrôle d'entrée* permet à un autre contexteur de modifier les paramètres internes du contexteur. Ce faisant, d'autres contexteurs peuvent agir sur le comportement du corps fonctionnel. Par exemple, demander l'arrêt du contexteur lorsqu'il a été reconnu déficient, ou négocier les QoS du contexteur.

Les *données d'entrée* correspondent aux données que le contexteur a la charge de traiter. Toute donnée d'entrée est assortie d'une *méta-donnée* qui exprime la qualité de la donnée reçue. La qualité peut inclure des propriétés opérationnelles (précision, stabilité, résolution, latence, etc.), mais aussi un facteur de confiance. Ce dernier point est important : alors qu'en IHM classique, les données sont certaines (un événement souris ne peut faire de doute), il en va tout autrement avec le monde des capteurs et du raisonnement par inférence.

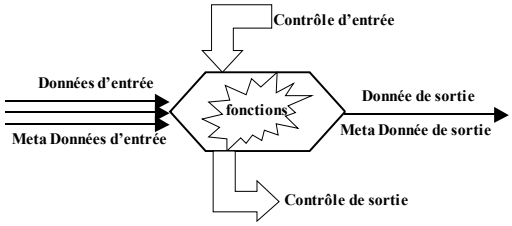

Figure 2 : Modèle de contexteur.

Symétriquement, ***les sorties*** sont de 2 types :

Le *contrôle de sortie* permet de modifier les paramètres internes d'un contexteur cible. L'ordre transmis par le contrôle de sortie est soumis à l'approbation du contexteur cible qui modifie son fonctionnement selon le message reçu. Le contrôle de sortie permet aussi de demander une modification des QoS préalablement négociés.

Les *données de sortie* sont des informations transmises soit à d'autres contexteurs soit à une entité logicielle autre que les contexteurs (une application par exemple). Chaque donnée de sortie est assortie de méta-données qui en expriment la qualité.

Le corps fonctionnel désigne la fonction que le contexteur remplit. On trouvera dans [14] une classification des contexteurs qui étend la proposition de Dey [9]: contexteur élémentaire (qui encapsule un capteur physique), contexteur à mémoire (qui mémorise un historique des données et méta-données d'entrée), contexteur à seuil (qui teste le franchissement de seuil), contexteur de traduction (sorte d'adaptateur qui change la représentation des données d'entrée sans en modifier la sémantique ni le niveau d'abstraction), contexteur de fusion (qui, à partir de plusieurs données d'entrée de même sémantique, permet d'en améliorer la qualité), contexteur d'abstraction (qui produit des informations de plus haut niveau d'abstraction).

Ayant décrit les éléments d'un contexteur, il convient maintenant d'en préciser les propriétés.

Propriétés d'un contexteur

Les propriétés des contexteurs sont motivées par un environnement d'exécution évolutif et incertain : un capteur peut tomber en panne, des ressources d'interaction peuvent être ajoutées dynamiquement par l'utilisateur ou disparaître, etc [23]. Pour ces raisons, un contexteur doit être réflexif, il doit pouvoir répondre à de nouveaux contrats de QoS, et être rémanent.

Un contexteur est réflexif : il est capable de s'auto-décrire, c.-à-d. fournir les informations suivantes : son nom, sa classe, ses interfaces d'entrée, ses interfaces de sortie, la ou les fonctions de son corps fonctionnel.

Un contexteur est capable de modifier dynamiquement son fonctionnement en réponse aux requêtes reçues sur le port Contrôle d'entrée. Ce faisant, il peut s'adapter à de nouveau requis de qualité de service.

Un contexteur est capable de se mettre en veille s'il n'est plus utilisé. Cette propriété implique que, si son état interne a une importance pour un futur redémarrage, il doit être capable de se sauvegarder. Il est donc rémanent.

Jusqu'ici, nous avons étudié le contexteur en tant qu'unité de représentation d'une variable du contexte système. Voyons maintenant comment les contexteurs peuvent se composer entre eux pour représenter d'autres variables et relations. Nous présentons, ici, deux mécanismes de composition : l'assemblage hiérarchique et l'encapsulation.

Assemblage Hiérarchique

Comme le montre la figure 3, les contexteurs peuvent être assemblés en un graphe orienté hiérarchique dans lequel les sorties (port de données) d'un contexteur sont reliées aux entrées (port de données) d'un ou plusieurs autres contexteurs.

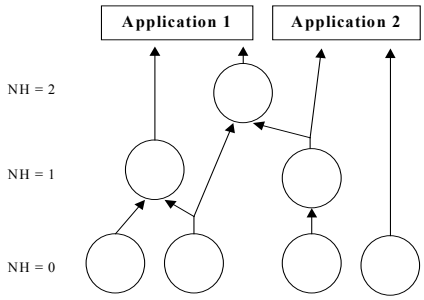

Figure 3 : Exemple de composition hiérarchique de contexteurs.

Le niveau hiérarchique d'un contexteur dans le graphe permet de caractériser sa dépendance et de là, le coût de (re)configuration du graphe lorsque ce contexteur doit apparaître (ou disparaître). Le niveau hiérarchique d'un contexteur se calcule ainsi : Le niveau hiérarchique NH des contexteurs élémentaires est NH = 0. Le niveau hiérarchique des autres contexteurs est égal au niveau hiérarchique du contexteur en entrée de niveau le plus haut + 1. Ainsi, la valeur du niveau hiérarchique d'un contexteur indique la taille (en nombre de contexteurs) de la plus grande chaîne de contexteurs le précédant.

On appelle chaîne de dépendance d'un contexteur fournissant un service à une application, l'ensemble composé du contexteur et des contexteurs dont il a besoin pour fonctionner et ceci récursivement jusqu'aux contexteurs élémentaires. On peut classer un contexteur suivant la place qu'il occupe dans sa chaîne de dépendance en trois rangs : la capture, l'interprétation, l'utilisation.

La capture. Cette étape est la première dans la chaîne. Elle se fait essentiellement au niveau des contexteurs élémentaires. Elle demande des compétences en électronique pour la création et / ou l'utilisation de capteur physique.

L'interprétation. Cette étape est traitée par les maillons intermédiaires de la chaîne de dépendance. Elle est donc essentiellement une étape de création de contexteur. Bien qu'elle ne demande pas de connaissances en électronique, elle peut nécessiter des aptitudes diverses (technique de vision, intelligence artificielle...).

L'utilisation. Cette étape est présente en fin de chaîne de dépendance. Elle est concentrée dans le maillon de niveau hiérarchique le plus élevé. Elle sert à faire le pont entre le monde applicatif classique et le monde sensible au contexte. Les compétences demandées pour les développeurs d'applications sensibles au contexte sont donc pratiquement les mêmes que pour le développement d'applications classiques.

La composition par assemblage hiérarchique peut conduire à de longues chaînes de dépendances avec, à la clef, l'augmentation des risques de pannes du service fourni par le contexteur de bout de chaîne et la latence qu'engendre la communication entre les contexteurs. Pour régler ces problèmes nous proposons d'utiliser l'encapsulation.

Encapsulation

Le principe d'encapsulation permet de réutiliser des contexteurs prédéfinis, de les assembler et d'en masquer la composition pour fournir un nouveau service. Le modèle impose que cette composition corresponde à la définition d'un contexteur (en termes de flots d'entrée et de sortie) et obéisse aux propriétés des contexteurs présentées précédemment. Mais l'encapsulation permet d'optimiser les couplages des contexteurs encapsulés de manière à réduire la latence et les risques de rupture des liaisons.

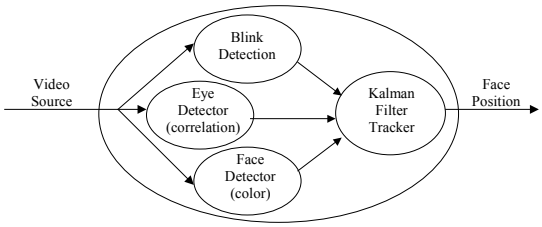

Figure 4 : Exemple de contexteur obtenu par encapsulation pour un système de suivi du visage par vision par ordinateur.

La figure 4 montre un exemple de mise en œuvre d'un contexteur qui encapsule un assemblage de contexteurs organisés hiérarchiquement. Trois contexteurs, correspondant chacun à une technique de suivi (détection de clignement des yeux par différence d'image, suivi par

corrélation avec un motif représentatif du visage, suivi par histogramme de couleur de la peau), reçoivent en entrée une image vidéo (en provenance d'un capteur élémentaire d'acquisition image. Ces trois contexteurs fournissent en sortie une information de localisation qui alimente un filtre de Kalman dont le rôle est de prédire la localisation du visage (mais aussi, via son port Contrôle de sortie de contrôler le comportement des contexteurs visuels, et notamment le placement de la région d'intérêt qui permet de réduire la latence des contexteurs visuels). On note que l'encapsulation permet de réduire le niveau de hiérarchie de contexteurs et donc le degré de dépendance.

Ayant présenté la composition de contexteurs (par assemblage hiérarchique et/ou encapsulation), voyons comment ces compositions s'intègrent dans l'architecture d'une application. Nous reprenons pour cela la suggestion de Salber [17] fondée sur le modèle d'architecture de référence, Arch [22]. Nous affinons les 2 couches fonctionnelles de Salber de la manière suivante (voir figure 5) :

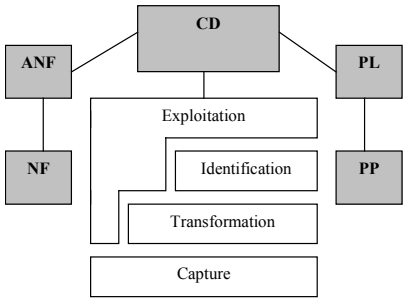

Figure 5 : Modèle Arch étendu au contexte.

La couche *Capture* correspond aux contexteurs élémentaires, c'est-à-dire à l'acquisition d'information à l'aide de capteurs. La couche *Transformation* correspond aux chaînes de contexteurs. Sa mission est de fournir les informations contextuelles au bon niveau d'abstraction. La couche *Identification* a pour fonction la détection de la situation et du contexte (composition des situations au cours du temps). Cette couche peut être modélisée soit à l'aide de contexteurs (extérieur à l'application), soit par un module (interne) de l'application elle-même. Enfin, la couche *Exploitation* est le module de l'application qui prend en charge la gestion du contexte. Elle sert d'adaptateur entre le CD et le contexte, comme l'ANF sert d'adaptateur entre le CD et le noyau fonctionnel.

Comme le montre la figure 5, la couche exploitation peut directement recevoir des informations des couches « capture » et « transformation ». Cette possibilité a une double justification : les couches intermédiaires peuvent ne pas exister (phénomène slinky de arch), ou des requis de performance exigent des entorses au modèle en couche strict (comme dans X Window).

Nous avons présenté jusqu'ici les concepts qui prévalent à la définition d'un contexteur. Du modèle conceptuel, nous passons maintenant à la réalisation technique.

MISE EN ŒUVRE

D'un point de vue implémentationnel, le contexteur est un composant (ou objet) logiciel communicant. Dès lors, il convient de préciser l'architecture réseaux sur laquelle il s'appuie, le protocole de communication et son cycle de vie.

Architecture d'égal à égal

En tant que composant « autonome », le contexteur est à la fois client (auprès d'autres contexteur) et serveur (services fournis aux applications ou à d'autres contexteurs). Cette caractéristique nous oriente vers l'utilisation d'une architecture de type égal à égal (ou dans sa dénomination anglaise peer to peer (P2P)), plutôt que vers une architecture de type client/serveur. En effet, dans une architecture P2P chaque composant est à la fois client et serveur.

Protocole de Communication

La communication entre contexteurs est asynchrone. Comme le montre la figure 6, les messages échangés tant sur les ports de données que sur les ports de contrôle sont au format XML.

```
<MSG num=3456>
        <DATA>
        ...
        <DATA>
        </METADATA>
        ...
        <METADATA>
</MSG>
```

Figure 6 : Représentation XML d'un message de donnée.

La Figure 7 montre la mise en œuvre de la composition hiérarchique. En plus des flots de données, les flots de contrôle relient les contexteurs en gardant le même schéma hiérarchique que celui créé par les flots de données mais dans le sens inverse.

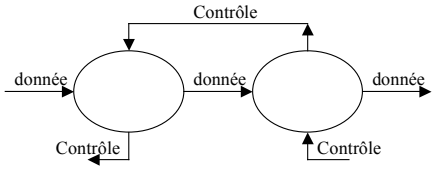

Figure 7 : Schéma montrant le parallélisme entre les flots de données et de contrôle entre deux contexteurs

Les ports de contrôle véhiculent deux types de messages : les messages de QoS, qui correspondent à une modification du contrat concernant la qualité du service rendu, et les messages de type FCM (functional core modification) correspondant à une modification du service rendu par le contexteur.

En tant qu'objet communicant, le contexteur répond à un cycle qui rythme son existence.

Cycle de vie d'un Contexteur

On distingue 5 états dans la vie d'un contexteur. La figure 8 en donne une représentation sous forme d'automate d'états finis.

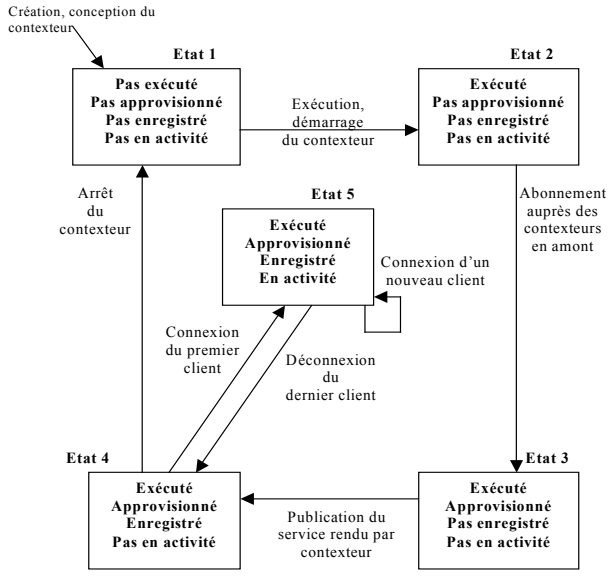

Figure 8 : Cycle de vie d'un contexteur

L'état 1 correspond à la phase où le contexteur se trouve sous forme d'un fichier binaire exécutable stocké sur un disque. Le chargement en mémoire de ce binaire (exécution du programme du contexteur) provoque le passage à l'état 2.

L'état 2 correspond à la phase d'abonnement du contexteur auprès des autres contexteurs dont il dépend. Le contexteur est ici isolé. Cet état est transitoire : le contexteur va chercher à entrer en contact avec les différents contexteurs qui lui sont nécessaires pour assurer son bon fonctionnement. Une fois la connexion établie avec tous les contexteurs nécessaires, il passe dans l'état 3. Un contexteur élémentaire passe directement dans l'état 3 puisqu'il n'a pas d'entrée.

L'état 3 est la phase de publication du service fourni par le contexteur. Nous proposons 3 types de publications de service : s'enregistrer dans un serveur de service (serveur stockant les services fournis par les différents contexteurs), diffusion multicast auprès des contexteurs du service fourni, activation d'un composant d'écoute répondant aux requêtes des autres contexteurs (qui cherchent à obtenir un service).

L'état 4 correspond à la phase d'attente. Le contexteur est prêt à fonctionner, il attend qu'un autre contexteur (et/ou application) requiert ses services. Quand cela arrive, il négocie un contrat (durée du service, QoS, …) avec le demandeur puis passe dans l'état 5.

L'état 5 correspond à la phase d'activité du contexteur. C'est durant cet état qu'il rend le service pour lequel il a été créé. Si un nouveau contexteur (et/ou application) le sollicite, il négocie un nouveau contrat avec le nouveau demandeur et crée une nouvelle instance de lui-même pour pouvoir rendre le nouveau service comme le montre la figure 9. Quand le dernier contexteur (et/ou application) se déconnecte, le contexteur retourne dans l'état 4.

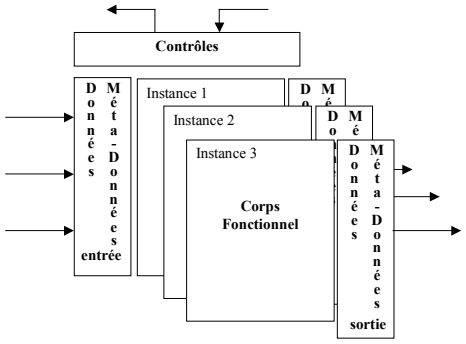

Figure 9 : Schématisation d'un contexteur.

CONCLUSION

Alors que les concepts d'ordinateur évanescents et d'informatique ubiquitaire (ubiquitous computing) ne sont pas des idées nouvelles, ce n'est que récemment que les progrès de la technologie permettent d'en envisager la mise en œuvre. Il en résulte un foisonnement de recherches autour des *systèmes interactifs sensibles au contexte*. Avec l'hypothèse que ces systèmes présenteront des avantages pour les activités humaines, cette étude a contribué à clarifier la situation en trois points complémentaires : définition, modèle, mise en œuvre.

Inspiré des travaux de Dey et de Salber, le contexteur présente comme originalité la notion de méta-donnée que nous lions étroitement aux données d'entrée et de sortie. Comme chez Salber, le contexteur distingue les données de contrôle des données proprement dites. Au-delà des travaux de Salber, nous proposons une *typologie* des contexteurs (contexteurs élémentaire, à seuil, à mémoire, de fusion, de traduction, d'extension), des *propriétés* de contexteur (réflexivité, rémanence, etc.), la *composition* de contexteurs et la notion de chaîne de dépendance qui permet de raisonner sur la modifiabilité de la composition, enfin l'*encapsulation* qui permet de considérer une composition de contexteurs comme un contexteur.

En résumé, cette étude a permis de poser un canevas général pour l'analyse et la mise en œuvre du problème des systèmes sensibles au contexte. Mais il convient de définir plus en profondeur le protocole de communication entre les contexteurs, de décrire et caractériser les méta données et de valider le modèle par la mise en œuvre

d'exemples d'applications sensibles au contexte. En ce qui concerne ce dernier point trois applications simples ont été réalisées :

- Le contrôle de la présence des tasses à café dans une cafétéria,
- Le niveau d'activité de l'utilisateur sur sa station,
- La présence de stations de travail sur un réseau local.

BIBLIOGRAPHIE

1. Bardram J. E., Plans as Situated Action: An Activity Theory Approach to Workflow Systems. Proceedings of ECSCW'97 Conference, Lancaster UK, September 1997.
2. Bérard Francois, http://iihm/demos/magicboard/ juin 1999
3. Beyer H., K. Holtzblatt. Contextual Design. Morgan Kaufman Publ. San Francisco, 1998.
4. Brown P.J., J.D. Bovey, X. Chen. Context-Aware applications : From the Laboratory to the Marketplace. IEEE Personal Communications, 4(5) (1997) 58-64.
5. Brown P.J., The Stick-e Document: a framework for creating Context-aware applications. Electronic Publishing '96 (1996) 259-272.
6. Calvary G., J. Coutaz, D. Thevenin. A Unifying Reference Framework for the Development of Plastic User Interfaces. EHCI'2001.
7. Calvary G., J. Coutaz, D. Thevenin. Supporting Context Changes for Plastic User Interfaces. a Process and a Mechanism. HCI-IHM 2001.
8. Card, S.Moran, T.Newell,A. The Psychology of Human-Computer Interaction, Lawrence Erlbaum Associates, 1983.
9. Dey A.K., Salber D., Abowd G.D., A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications, anchor article of a special issue on Context-Aware Computing, to appear in the Human-Computer Interaction (HCI) Journal, Vol. 16, 2001.
10. Dey A.K., Salber D., Masayasu Futakawa and Gregory D. Abowd. An Architecture To Support Context-Aware Applications. GVU Technical Report GIT-GVU-99-23. June 1999.
11. Halverson, C.A. Distributed Cognition as a theoretical framework for HCI: Don't throw the Baby out with the bathwater - the importance of the cursor in Air Traffic Control. Tech Report No. 94-03, Dept. of Cognitive Science, University of California, San Diego. (1994).
12. Pascoe J. Adding Generic Contextual Capabilities to Werable Computers. 2nd International Symposium on Wearable Computers (1998) 92-99.
13. Rekimoto J., Brygg Ullmer, and Haro Oba, DataTiles: A Modular Platform for Mixed Physical and Graphical Interactions, CHI2001, 2001.
14. Rey G. Systèmes Interactifs Sensibles au Contexte. Ecole doctorale de Mathématiques et Informatique, DEA d'Informatique, Systèmes et Communications, Université Joseph Fourier et Institut National Polytechnique de Grenoble, Juin, 2001, 84 pages.
15. Ryan N., J. Pascoe, D. Morse, Enhanced Reality Fieldwork : the Context-Aware Archeological Assistant. V. Gaffney, M. van Leusen, S. Exxon Computer Applications in Archeology (1997).
16. Salber D., A.K. Dey, G. D. Abowd The Context Toolkit: Aiding the Develepment of Context-Enabled Applications, CHI 99 Conference Proceedings, pp 434 à 441.
17. Salber D., Gray P. Modelling and Usins Sensed Context Information in the design of Interactive applications EHCI 2001 (May 2001) 92-111.
18. Schilit B.N., M. Theimer.Disseminating Active Map Information to Mobile Hosts, IEEE Network, (1994) 22 – 32.
19. Schuman L. A., Plans and situated actions. The problem of human-machine communication. Cambridge: Cambridge University Press. (1987).
20. Streitz N. A., J. Geißler, T. Holmer, S. Konomi, C. MüllerTomfelde, W. Reischl, P. Rexroth, P. Seitz, R. Steinmetz i-LAND: An interactive Landscape for Creativity and Innovation. In: ACM Conference on Human Factors in Computing Systems (CHI'99), Pittsburgh, Pennsylvania, USA, May 15-20, 1999. pp. 120-127.
21. Thevenin D., J. Coutaz. Plasticity of User Interfaces: Framework and Research Agenda. In Proceedings of INTERACT'99, 1999, pp. 110-117.
22. UIMS Tool Devlopers' Workshop. A metamodel for runtime architecture of an interactive system. SIGCHI Bulletin, 24(1):32{37, 1992.
23. Want Roy, Trevor Pering, James Kardach, and Graham Kirby, The Personal Server: The Center of Your Ubiquitous World, Intel Research and Mobile Products Group white paper, May 2001
24. Ward, A. Jones, A. Hopper, A New Location Technique for the Active Office. IEEE personnal Communications (1997) 42-47.