Le Contexteur : Capture et distribution dynamique d'information contextuelle

Gaëtan Rey, Joëlle Coutaz

CLIPS-IMAG
385 rue de la Bibliothèque, BP 53
38041 Grenoble cedex 9, France
{Gaetan.Rey, Joelle.Coutaz}@imag.fr

RESUME
En l'absence d'infrastructures logicielles, la mise en œuvre de systèmes interactifs sensibles au contexte est une tâche complexe réalisée au cas par cas. Dans cet article, nous proposons la notion de contexteur, une abstraction logicielle qui fournit un support opérationnel à la notion de contexte d'interaction. Nous montrons comment les contexteurs s'organisent en niveaux d'abstraction et comment ces niveaux s'insèrent dans Arch, un modèle d'architecture de référence pour les systèmes interactifs. Déployée selon les principes P2P, notre infrastructure de contexteurs vise les requis de mobilité et d'ubiquité. Similaire, dans l'esprit, à la Context Toolkit, nous explicitons nos différences.

Mots Cles
Contexte d'interaction, modélisation du contexte, architecture logicielle, Interaction Homme-Machine, informatique diffuse.

ABSTRACT
Without the support of adequate software infrastructures, the implementation of context sensitive interactive systems is hard to achieve in a sound way. In this article, we propose the notion of contextor, a software abstraction that supports the operational deployment of interaction contexts. We show how contextors are organized into levels of abstraction, and how these levels fit within the Arch architecture reference model for interactive systems. Based on the P2P paradigm, the contextor infrastructure is intended to support both mobility and ubiquity. Similar in spirit to the Context Toolkit, we make the differences explicit.

Keywords
Interaction context, context modeling, software architecture, Human Computer Interaction, ubiquitous computing.

Categories and Subject Descriptors
D.2.11 **[Software Engineering]**: Software Architectures - *Data abstraction, Domain-specific architectures*, H.5.2 **[Information Interfaces and Presentation]**: User Interfaces - *Prototyping*.

General Terms
Theory, Human Factors

1. INTRODUCTION
Bien que l'importance du contexte d'interaction soit acquise en conception d'Interface Homme-Machine (IHM)[1], les données contextuelles identifiées en phase amont du processus de conception sont perdues au cours du développement ou, au mieux, modélisées de manière implicite dans le système interactif final. Par exemple, toutes les IHM graphiques actuelles font l'hypothèse implicite que l'utilisateur est face à un écran vertical. Dans cet exemple, les relations spatiales entre les dispositifs d'interaction et l'utilisateur sont implicites et modélisées de manière statique. L'absence d'un modèle explicite du contexte d'interaction dans le système final a peu d'importance tant que l'usage prévu du système correspond à l'usage effectif. Avec l'émergence de l'informatique diffuse, il en va tout autrement.

Disposant de moyens de communication universels, de processeurs puissants et de mémoires miniaturisées, pour un coût d'achat accessible, l'utilisateur déporte une partie des tâches usuelles « de bureau » en des lieux inhabituels comme le train, la rue, chez soi. De cerné, le contexte d'interaction devient imprévisible. Dès lors, les systèmes interactifs doivent embarquer un modèle explicite du contexte d'interaction. C'est dans le but de faciliter ce travail de conception et de mise en œuvre que nous proposons la notion de « contexteur ».

Cet article est structuré comme suit : nous présentons notre position sur la notion de contexte, largement discutée dans la littérature sans qu'aucun consensus n'émerge. Ayant précisé notre point de vue, nous sommes en mesure dans la section qui suit de proposer un modèle computationnel du contexte avec notre concept de contexteur. Nous en détaillons ensuite la réalisation actuelle ainsi que son utilisation au moyen d'un exemple.

2. CONTEXTE D'INTERACTION
Si la notion de contexte n'est pas nouvelle en informatique, il n'existe pas de définition consensuelle en informatique diffuse. Toutefois, l'état de l'art, et notamment les *questions quintiliennes* de Schilit [14], les définitions de Pascoe [9] et de Dey [4] ou nos propres propositions en la matière [11],[2], partagent les quatre principes suivants.
1) Il n'y a *pas de contexte sans contexte*. Autrement dit, la notion de contexte se définit en fonction d'une finalité. Pour notre part, la finalité recherchée est l'adaptation dynamique des capacités interactives du système. Aussi, nous parlons de contexte d'interaction (dont on précisera la définition ci-dessous). 2) Le *contexte est un espace d'informations qui sert l'interprétation*. La capture du contexte n'est pas une fin en soi, mais les données capturées doivent servir un objectif. Dans

notre cas, l'interprétation des données est assurée par le système dans le but de servir l'utilisateur. 3) Le contexte est *un espace d'informations infini et évolutif*. Par conséquent, il n'existe pas a priori de « *contexte absolu* », mais un espace qui se construit au cours du temps. 4) Le contexte est *un espace d'informations partagé par plusieurs acteurs*. Les deux acteurs au centre de nos préoccupations sont l'utilisateur et le système. Comme le montre la figure 1, le *contexte d'interaction* est l'ensemble des informations que l'utilisateur et le système ont en commun.

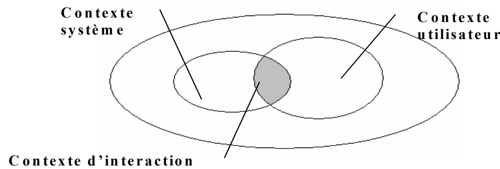

Figure 1 *: Le contexte d'interaction est un espace d'informations partagé entre l'utilisateur et le système.*

Au bilan, le contexte est un espace infini de variables en constante évolution dont la nature et l'exploitation par les systèmes interactifs ne sont pas connues par avance. Dans ces conditions, en l'absence d'outils, le développement de systèmes interactifs « sensibles au contexte » est une tâche difficile nécessairement réalisée au cas par cas.

Par analogie avec les infrastructures dédiées au développement d'Interface Homme-Machine, il convient de mettre à la disposition des développeurs une infrastructure générale de type *intergiciel* capable de fournir à façon les données contextuelles nécessaires à chaque système interactif. Cette infrastructure doit satisfaire les requis suivants :1) Elle doit permettre la mobilité de l'utilisateur. 2) Elle doit fonctionner sur de petits dispositifs. (Même si les PDA et téléphones mobiles voient leur puissance augmenter, ils ne sont pas encore adaptés au fonctionnement de gros intergiciels.) 3) Elle doit autoriser les connexions et déconnexions à la volée et mesurer la qualité des données captées ou calculées. (Nous ne sommes plus dans un monde au fonctionnement certain, mais dans un espace opportuniste marqué par les incertitudes et les dysfonctionnements, notamment des capteurs.)

Depuis peu, nous voyons émerger des infrastructures pour la réalisation de systèmes sensibles au contexte : « Strathclyde Context Infrastructure » [6], le « TEA Context-Awareness Module » [5], et la Context Toolkit, infrastructure pionnière la plus aboutie [3].

3. LA CONTEXT TOOLKIT

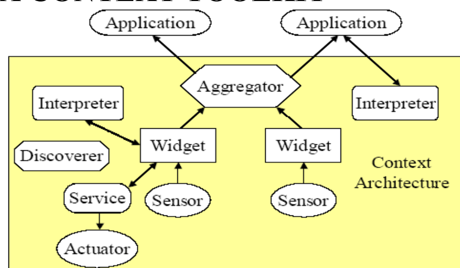

Figure 2 *: Architecture de la Context Toolkit extraite de [3].*

La Context Toolkit est une infrastructure de type *flot de données* fondée sur cinq catégories de composants logiciels (voir figure 2) et deux types de composants matériels.
Les *Context-Widgets* servent de lien entre les capteurs physiques (*Sensors)* et les applications, tout comme les *Widgets* graphiques servent d'intermédiaires entre l'utilisateur et le système interactif. Les *Context-Aggregators*, qui concatènent les données des *Context-Widgets* en une unité d'abstraction, fournissent aux applications une vue simplifiée de l'espace d'informations contextuelles. La liaison entre *Context-Widgets*, les *Context-Aggregators* et les applications se fait grâce au *Discoverer*. Le *Discoverer* maintient la liste des composants disponibles. Il peut être interrogé par les applications cherchant tels ou tels composants et/ou services. Maintenant que les applications reçoivent les données enregistrées par les capteurs via les *Context-Widgets* et les *Context-Aggregators* il ne reste plus qu'à donner un sens à ces données. Pour cela, les *Context-Widgets*, comme les applications, peuvent interroger des *Context-Interpreters*. Une dernière classe de composants logiciels, les *Services*, permet d'exécuter des actions (via les *Actuators*) au nom de l'application.

Cette infrastructure, bien adaptée à la distribution des informations contextuelles, présente cependant quelques limitations : 1) Elle ne prend pas explicitement en charge les métadonnées censées renseigner l'application sur la qualité et/ou la précision des informations fournies. 2) Bien que les composants de la Context toolkit soient répartis, la machine qui héberge le *Discoverer* doit être connue de tous. Le *Discoverer*, élément central, constitue donc le point faible de cette infrastructure tant du point de vue du passage à l'échelle que de la mobilité.

Notre infrastructure à contexteurs, inspirée des bons principes de la context toolkit, vise à éliminer ces limitations : absence de composant central, gestion explicite de métadonnées.

4. LE MODELE DES 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.

4.1 Description

Inspiré des « context-handling components » de Salber [13], un contexteur comprend trois classes d'éléments : entrées, sorties et un corps fonctionnel (voir figure 3).

Les entrées sont de 2 types : Les données d'entrée et le contrôle d'entrée. Les *données d'entrée* correspondent aux données que le contexteur a la charge de traiter. Toute donnée d'entrée est intimement associée à une *métadonnée* 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 calculé à partir des caractéristiques des capteurs. 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. 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 lors-

qu'il a été reconnu déficient, ou négocier sa Qualité de Service (QoS).

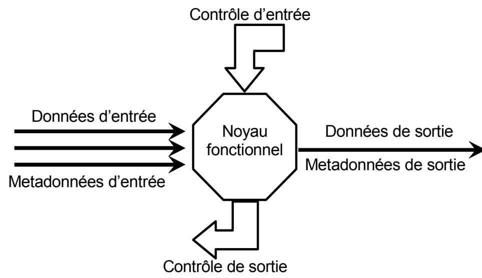

Figure 3 : *Modèle de contexteur.*

Symétriquement*, **les sorties*** sont de 2 types. 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 métadonnées qui en expriment la qualité. 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 (par exemple, un contexteur C1 recevant un ordre d'arrêt provenant d'un contexteur C2 peut choisir de suspendre la communication avec C1 de manière à pouvoir continuer à fournir ses données à un contexteur C3). Le contrôle de sortie permet aussi de demander une modification des QoS préalablement négociées : changement des modes de transmission des données par exemple (ces modes seront présentés en détail dans la section 5.2).

Le corps fonctionnel désigne la fonction que le contexteur remplit. On trouvera dans [9] une classification des contexteurs qui étend la proposition de Dey [3] : contexteur élémentaire (qui, encapsulant un capteur physique, ne possède pas de données d'entrées), contexteur à mémoire (qui mémorise un historique des données et métadonnées d'entrée), contexteur à seuil (qui teste le franchissement d'un 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 exprimée au travers des métadonnées), contexteur d'abstraction (qui produit des informations de plus haut niveau d'abstraction). On notera que ces classes peuvent être étendues dynamiquement.

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

4.2 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. [10]. Pour ces raisons, un contexteur doit être réflexif et doit pouvoir répondre à de nouveaux contrats de QoS.

Un contexteur est réflexif : il est capable de s'autodécrire, 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 nouveaux requis de qualité de service ou à de nouveaux requis fonctionnels comme se mettre en veille s'il n'est plus utilisé.

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 pour représenter d'autres variables. Nous présentons, ici, deux mécanismes de composition : l'assemblage hiérarchique et l'encapsulation.

4.3 Assemblage Hiérarchique : composition dynamique

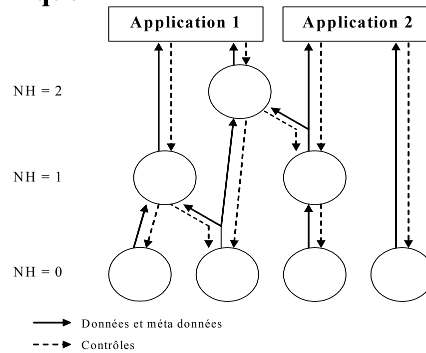

Figure 4 : *Exemple de composition hiérarchique de contexteurs.*

Cette méthode d'assemblage permet une composition dynamique des contexteurs. Comme le montre la figure 4, les contexteurs s'assemblent 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.

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.

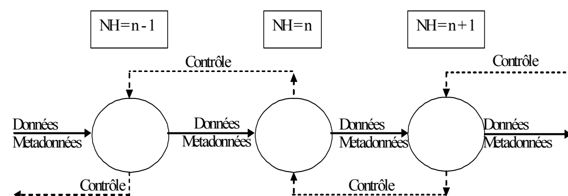

Figure 5 : *Schéma montrant la symétrie entre les flots de données et de contrôle entre trois contexteurs*

La Figure 5 détaille 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.

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.

4.4 Encapsulation : assemblage statique à la conception

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. L'encapsulation permet d'optimiser les couplages des contexteurs encapsulés de manière à réduire la latence et les risques de rupture des liaisons. Ces liaisons (en pointillé sur la figure 6) sont des liens statiques correspondant à des appels de méthodes (pour la version encapsulée) alors que les liaisons dans la composition hiérarchique de la figure 4 sont des connexions réseau créées dynamiquement.

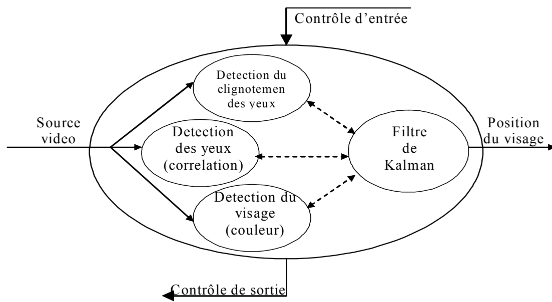

Figure 6 : *Exemple de contexteur obtenu par encapsulation pour un système de suivi du visage par vision par ordinateur*

La figure 6 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'images, 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.

4.5 Architecture Conceptuelle

Ayant présenté la composition de contexteurs (par composition hiérarchique et/ou encapsulation), voyons comment ces assemblages s'intègrent dans l'architecture d'une application. Nous reprenons pour cela la suggestion de Salber [13] fondée sur le modèle d'architecture de référence, Arch [15]. Nous affinons les deux couches fonctionnelles de Salber de la manière suivante (voir figure 7) : 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* implémente notre modèle de situation et de contexte présenté dans [2]. Dans ce modèle, un contexte est un réseau de situations et la « vie » évolue dans un réseau de contextes. La couche Identification a pour mission d'identifier la situation courante dans le contexte courant et de détecter les conditions de changement de situation et de contexte. Cette couche permet à l'application d'utiliser un langage de plus haut niveau d'abstraction pour spécifier les informations contextuelles désirées. Le module Identification prend alors en charge la traduction des requêtes de l'application en requêtes compréhensibles par les contexteurs. Actuellement non implémentée, elle pourrait l'être soit à l'aide de contexteurs (extérieurs à l'application), soit par un module (interne) de l'application. Enfin, la couche *Adaptateur* est le module de l'application qui prend en charge la gestion du contexte. Elle sert à faire le lien entre le CD et le contexte, comme l'ANF sert d'adaptateur entre le CD et le Noyau Fonctionnel. Elle permet aussi à l'application de s'abstraire de la communication avec les contexteurs et/ou la couche *Identification*. La recherche des contexteurs et la communication (réception des données et des métadonnées, émissions de requête de contrôle …) avec ces derniers sont prises en charge par l'adaptateur de contexte de manière transparente pour le développeur d'application.

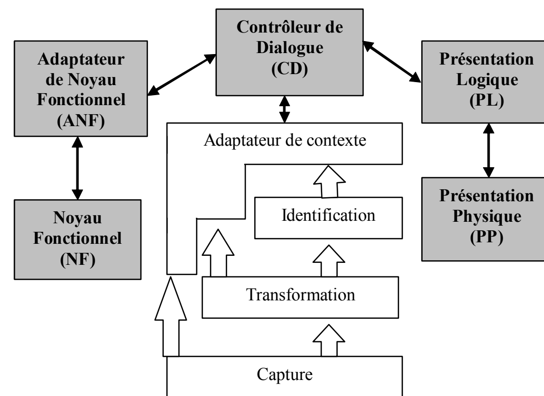

Figure 7 *: Modèle Arch étendu au contexte.*

Comme le montre la figure 7, l'adaptateur de contexte peut recevoir des informations directement des couches « capture » et « transformation ». Cette possibilité a une double justification : les couches intermédiaires peuvent ne pas exister (actuellement

la couche *Identification* n'est pas implémentée) (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.

5. 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 réseau sur laquelle il s'appuie, le protocole de communication et son cycle de vie.

5.1 Architecture d'égal à égal

En tant que composant « autonome », le contexteur est à la fois client (auprès d'autres contexteurs) et serveur (services fournis aux applications ou à d'autres contexteurs). Si on ajoute à cela le requis de distribution et de mobilité, l'utilisation d'une architecture de type égal à égal (ou dans sa dénomination anglaise peer to peer (P2P)) s'impose face à une architecture client/serveur centralisée. En effet, dans une architecture P2P, chaque composant est à la fois client et serveur, évitant ainsi les points centraux. Cependant, il n'existe pas véritablement aujourd'hui d'infrastructure P2P facilement utilisable et de taille suffisamment réduite pour être intégrée dans un contexteur. Pour cette raison, nous avons développé notre propre protocole de connexion.

Le protocole de connexion comprend deux phases. Une phase de *prospection* dans laquelle le contexteur recherche les contexteurs (ou les classes de contexteurs) dont il a besoin pour fonctionner. Une phase de *liaison* où les contexteurs devant s'échanger des données s'interconnectent. Ces deux phases prennent en charge de manière dynamique l'assemblage hiérarchique (décrit précédemment).

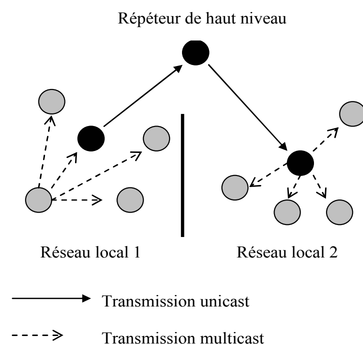

Figure 8 : *Exemple de mise en œuvre des répéteurs.*

La phase de *prospection* : Avant de lancer sa recherche, le contexteur construit un message indiquant qui il est, ainsi que l'ensemble des classes de contexteurs nécessaires à son bon fonctionnement. Il diffuse ensuite le message en utilisant le protocole multicast UDP avec un *Time To Live* (TTL) allant de 0 à n (où n est le plus petit possible). Pourquoi un tel choix ? Le TTL fixe la limite de distance maximale (en nombre de routeurs) séparant le contexteur émetteur des contexteurs cibles potentiels. Or, le multicast UDP fonctionnant par inondation, il est nécessaire de limiter son champ d'action de manière à diminuer la perturbation sur le réseau. Un TTL de 0 limite la recherche à la machine locale, un TTL de 1 au sous-réseau local, et ainsi de suite. Dans la grande majorité des cas, les informations recherchées sont locales ou dans l'environnement immédiat (réseau ou sous réseau local). Par conséquent, le protocole choisi inonde peu le réseau. Cependant, il se peut que certaines applications cherchent des contexteurs éloignés, et donc utilisent un TTL élevé avec le risque de perturber le réseau.

Pour palier l'inondation du réseau (dans le cas d'information éloignée), nous proposons une nouvelle architecture (en cours de réalisation) fondée sur la notion de *répéteur*. Cette proposition s'appuie sur l'hypothèse qu'une application et/ou un contexteur ont le plus souvent besoin d'informations fournies par des contexteurs locaux. Dans ce cas, la diffusion par multicast UDP avec un TTL maximum de 2 est utilisée. Pour atteindre des contexteurs non locaux, une requête de recherche est envoyée localement et « capturée » (de manière transparente) par le répéteur local. Une requête de recherche précise la classe de contexteurs recherchée, son groupe, le type de ses données de sortie, ses métadonnées et la *localisation* du contexteur. L'information de localisation (de type planet/continent/pays/...) permet à tout répéteur d'effectuer le routage vers le(s) répéteur(s) idoine(s) qui, à leur tour, diffuse(nt) la requête sur leur réseau local.

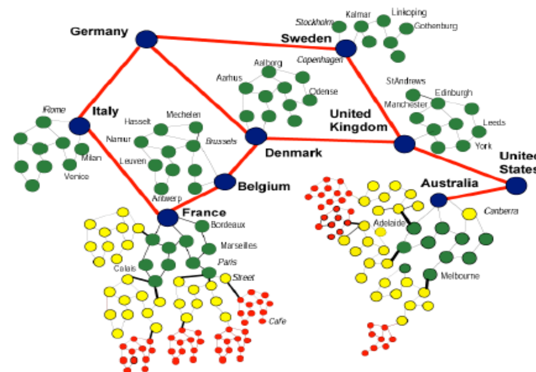

Figure 9 *: Architecture P2P hybride.*

Pour se transmettre les requêtes, les répéteurs sont organisés selon le modèle hybride de la figure 9 et décrit dans [6]. Ce modèle se présente comme une composition de deux modèles existants. D'une part le modèle hiérarchique classique (type DNS) et d'autre part, un modèle entièrement P2P pour donner un assemblage hiérarchique de réseaux P2P.

La phase de *liaison* : Chaque contexteur en attente de client (voir ci-dessous le cycle de vie pour plus de détails sur cet état) analyse l'ensemble des requêtes qui lui parviennent. S'il peut fournir l'un des services spécifiés dans la requête et qu'il ne fournit pas déjà ce service au contexteur qui le réclame, le contexteur entame la procédure de connexion. Il envoie un message direct (via TCP/IP) au contexteur « demandeur ». Ce message contient la modalité de transmission (OnChange, OnRequest, …) sélectionnée parmi la liste des modalités présentes dans la requête d'émission. Suit alors une phase d'identification entre les deux contexteurs. Cette phase correspond à l'émission d'un message de confirmation par le contex-

teur « demandeur » (client). Ce message contient des identifiants (actuellement login / mot de passe) d'accès au contexteur émetteur (source). Si l'identification réussit, les données de l'émetteur sont transmises au contexteur « demandeur » jusqu'à ce que celui-ci notifie l'arrêt des envois.

5.2 Protocole de Communication

La communication entre contexteurs est asynchrone. Les messages échangés tant sur les ports de données que sur les ports de contrôle, sont au format XML. Le choix du langage XML a trois justifications : a) la portabilité du langage entre plates-formes, b) l'interopérabilité entre les langages de programmation qui permettra à l'implémentation C# des contexteurs (en cours de développement) de fonctionner avec les contexteurs actuellement réalisés en java, c) l'extensibilité qui permet l'intégration des spécificités des différents capteurs existants dans le format des données des contexteurs. L'extension peut porter sur les données elles-mêmes ou sur les métadonnées traitées par le contexteur. L'extensibilité d'XML permet aux contexteurs de permettre ces évolutions sans modifications de leur code source.

Tout comme le protocole SOAP (*Simple Object Access Protocol*), notre protocole de communication utilise XML, décrit des aspects importants des contenus de données et n'est pas lié à un langage de programmation spécifique, ni à une plate-forme informatique ou un environnement de développement. Cependant la comparaison s'arrête là. SOAP a été créé pour faire communiquer des systèmes informatiques sur un modèle de programmation familier tel que l'appel de procédures distants (RPC) et cela via le protocole HTTP. Les contexteurs utilisent plusieurs modèles de transmission de données. Le mode *On-Compute* où le contexteur fournit ses données chaque fois qu'il les recalcule. Le mode *OnChange* où le contexteur fournit ses données chaque fois que leurs valeurs changent. Le mode *On-Request* où le contexteur attend une requête du client pour chaque envoi de données (ce mode permet de faire du RPC). Le mode *Periodically* où le contexteur fournit ses données toutes les x millisecondes (x étant un paramètre du mode *Periodically*). Le mode *Once* où les données ne sont transmissent qu'une seule fois. D'autres modalités (portant sur les métadonnées) sont en cours de validation.

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

5.3 Cycle de vie d'un Contexteur

On distingue 5 états dans la vie d'un contexteur. La figure 10 en donne une représentation sous forme d'automate d'états finis.

L'état 1 correspond à la phase où le contexteur se trouve sous forme d'un fichier binaire exécutable stocké sur disque. Le chargement en mémoire de ce binaire (exécution du programme du contexteur) provoque le passage à l'état 2. Nous caractérisons cet état de *pas exécuté*, *pas approvisionné* (le contexteur n'est pas encore connecté aux contexteurs amont dont il a besoin pour fonctionner), *pas en activité* (il ne calcule pas de données de sortie) et *pas de client* (il ne fournit des données à aucun contexteur).

L'état 2 correspond à la phase d'initialisation et d'abonnement du contexteur auprès des contexteurs dont il dépend. Le contexteur est ici isolé. Cet état est transitoire : le contexteur va initialiser l'ensemble de ses paramètres (ainsi que les capteurs qu'il encapsule s'il s'agit d'un contexteur élémentaire) puis 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 (après l'initialisation de son ou ses capteurs) dans l'état 3 puisqu'il n'a pas besoin de données d'entrée pour fonctionner.

L'état 3 est la phase de mise en marche de la fonction de calcul. A l'aide des données d'entrée (ou du capteur pour un contexteur élémentaire), le contexteur calcule ses données de sortie. Une fois la première donnée calculée, le contexteur entre dans l'état 4.

L'état 4 correspond à la phase d'attente. Prêt à fonctionner, le contexteur 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. Quand le dernier contexteur (et/ou application) se déconnecte, le contexteur retourne dans l'état 4.

5.4 Evaluation de Performance

Nous avons effectué des tests préliminaires pour mesurer le coût intrinsèque des contexteurs en utilisant la plate-forme suivante.

- Java version "1.4.1_02"
- Java(TM) 2 Runtime Environment, Standard Edition (build
- Java HotSpot(TM) Client VM (build 1.4.1_02-b06, mixed
- Bi Pentium 3, 1.2GHz, exécutant Windows XP.

Nous avons déployé 5000 contexteurs fonctionnant simultanément sur une période d'un mois. Ces contexteurs s'échangeaient des données de 25Ko selon le mode de transmission *Periodically* avec une fréquence d'une seconde. Nous avons pu retirer de ce test que :

- la taille mémoire utilisée par un contexteur est comprise entre 90 Ko et 60 Ko.
- la latence intrinsèque introduite par un contexteur est de 20 ms. Ce temps correspond au temps que mettent les données pour traverser un contexteur sans que celui n'effectue de traitement.

Ces résultats sont encourageants pour l'utilisation des contexteurs sur de petites machines. D'autres expérimentations sont actuellement en cours. Elles concernent notamment les tests de l'architecture hybride (avec les répéteurs) et la mobilité de l'utilisateur dans une telle architecture. Illustrons maintenant l'utilisation des contexteurs au moyen d'un exemple : un gestionnaire d'activités.

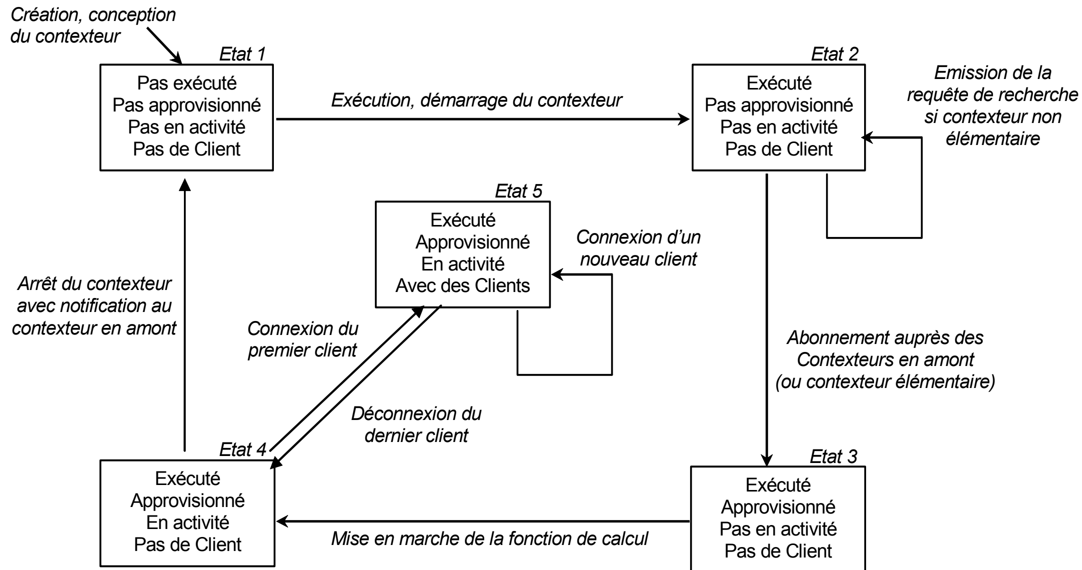

***Figure 10 :** Cycle de vie d'un contexteur.*

6. OBERVATOIRE D'ACTIVITES

Notre observatoire d'activités montre sur une page Web le niveau d'activités des utilisateurs de notre laboratoire. La figure 11 présente une vue générale du système. Sur chaque poste utilisateur dont on veut mesurer l'activité, sont installés les contexteurs suivants :

Contexteur d'activité clavier : capture le nombre d'événements clavier par seconde sur la machine où il est exécuté.

Contexteur d'activité souris : capture le nombre d'événements souris par seconde sur la machine où il est exécuté.

Contexteur d'information locale : capture le nom de l'utilisateur identifié sur la machine (ainsi que d'autres informations non pertinentes pour cet exemple).

Contexteur d'activité locale : fusionne les données de tous les *contexteurs d'activité x* se trouvant sur la machine où il est exécuté.

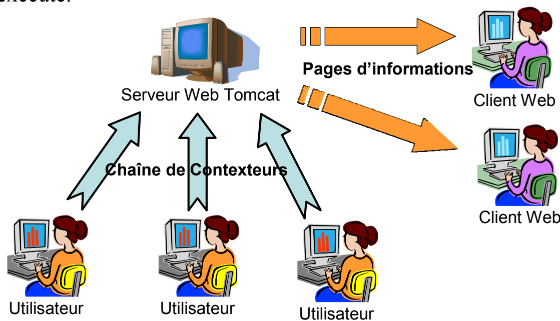

***Figure 11 :** Vue générale de l'observatoire d'activités.*

Comme le montre la figure 12, chaque *contexteur d'activité locale* fournit ses données à l'adaptateur de contexte du gestionnaire d'activités (ici un programme java sous la forme d'une servlet). L'adaptateur de contexte, comme présenté plus haut, fait le lien entre l'application et la chaîne des contexteurs souhaités par l'application. Ici l'adaptateur de contexte prend en charge la recherche des *contexteurs d'activité locale* ainsi que tous les échanges de messages (données et autres) entre l'application et ces contexteurs.

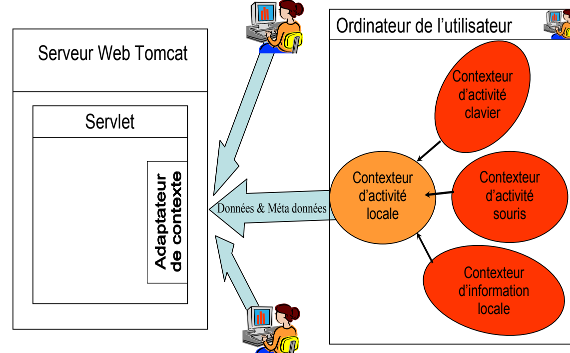

***Figure 12 :** Vue détaillée de l'observatoire d'activités.*

Au-delà de l'aspect validation du modèle des contexteurs, cet exemple soulève une question fondamentale sur les notions de confidentialité et de libertés individuelles [12]. *La capture du contexte sera-t-elle acceptée par les utilisateurs ?* En effet, la capture des actions de l'utilisateur sans que celui-ci en soit averti pose un problème d'éthique. *Que deviennent les informations ainsi capturées ? Qui a accès à ces informations ? Les informations originellement capturées pour servir l'utilisateur, ne se retourneront- elles pas contre lui ?* Avant de poursuivre le développement du modèle, il semble nécessaire de chercher des réponses à ces questions.

7. CONCLUSION

Alors que les concepts d'ordinateur évanescent et d'informatique diffuse (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 propose un modèle computationnel fondé sur une définition de la notion de contexte présentée dans [2].

Inspiré des travaux de Dey et de Salber, le contexteur présente comme originalité la notion de métadonnée 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, …), la *composition dynamique* 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 et de limiter la latence engendrée par de longues chaînes de contexteurs.

Sur le plan de la mise en œuvre, nous pensons atteindre les objectifs requis de passage à la l'échelle et de mobilité avec une architecture P2P et la notion de répéteur au cœur d'une architecture planétaire de type hybride développée dans le projet GLOSS. Au-delà de l'application *observatoire d'activités*, nos contexteurs ont été utilisés dans la plate-forme I-AM pour la découverte de ressources d'interaction dans un cluster de machines hétérogènes [8]. Nos mesures de performance initiales en terme de passage à l'échelle (mise en œuvre de 5000 contexteurs sur un Bi Pentium 3 1.2 GHz), de capacité de transfert (échange de données de 25Ko toutes les secondes entre deux contexteurs) ainsi qu'en terme de stabilité (fonctionnement sans panne durant plus d'un mois, d'une fédération de contexteurs s'échangeant des données toutes les secondes) permettent d'envisager un déploiement de notre infrastructure de contexteurs à plus grande échelle.

8. BIBLIOGRAPHIE

[1] Beyer H., K. Holtzblatt. Contextual Design. Morgan Kaufman Publ. San Francisco, 1998.

[2] Crowley, J. L., Coutaz, J., Rey, G., Reignier, P., "Perceptual Components for Context Aware Computing", UBICOMP 2002, International Conference on Ubiquitous Computing, Goteborg, Sweden, September 2002.

[3] 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, in the Human-Computer Interaction (HCI) Journal, Vol. 16, 2001.

[4] 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.

[5] Gellersen HW, Schmidt A and Beigl M "Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts", in Mobile Networks and Applications (MONET), Oct 2002

[6] Glassey R, Stevenson G, Richmond M, Nixon P, Terzis S, Wang F, and Ferguson I. Towards a Generalised Middleware for Context Management In 1st Int. Workshop for Middleware for Pervasive and Ad Hoc Computing, Middleware 2003 companion, pages 45-52, June 2003.

[7] Kirby G, Dearle A, McCarthy A, Morrison R, Mullen R, Yang Y, Connor R, Welen P, and Wilson A, First Smart Spaces, GLOBAL SMART SPACES, Project NO. IST-2000-260

[8] Lachenal C, Rey G, Barralon N. MUSICAE, an infrastructure for MUltiple Surface Interaction in Context Aware Environment. HCII 2003, Adjunct Proceedings, pp 125-126.

[9] Pascoe J. Adding Generic Contextual Capabilities to Werable Computers. 2nd International Symposium on Wearable Computers (1998) 92-99.

[10] 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.

[11] Rey G., Coutaz J., Le Contexteur : une Abstraction Logicielle pour la Réalisation de Systèmes Interactifs Sensibles au Context. In Proc. IHM2002, pages 105-112

[12] Salber D, The Need for an Applied Computer Ethics Handbook . ETHICOMP'96 conference, Porfirio Barroso, Terrel Ward Bynum, Simon Rogerson et Luis Joyane Eds., Pontificial University of Salamanca in Madrid Publ., Madrid.

[13] Salber D., Gray P. Modelling and Using Sensed Context Information in the design of Interactive applications EHCI 2001 (May 2001) 92-111.

[14] Schilit B.N., M. Theimer.Disseminating Active Map Information to Mobile Hosts, IEEE Network, (1994) 22 – 32.

[15] UIMS Tool Developers' Workshop. A metamodel for runtime architecture of an interactive system. SIGCHI Bulletin, 24(1):32{37, 1992.

[16] 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