Contexte en Interaction Homme-Machine : le contexteur

Gaëtan Rey

THÈSE
présentée par

Gaëtan Rey

pour obtenir le titre de
DOCTEUR de L'UNIVERSITE JOSEPH-FOURIER-GRENOBLE I
(arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)
Spécialité : **Informatique**

Contexte en Interaction Homme-Machine : le contexteur.

Composition du Jury :

Directeur de thèse :	Pr. Joëlle Coutaz
Rapporteurs :	Pr. Simon Dobson Pr. Michel Beaudouin-Lafon
Jury :	Pr. James L. Crowley Pr. Jean Vanderdonckt

Thèse préparée au sein du laboratoire
Communication Langagière et Interaction Personne-Système
Fédération IMAG
Université Joseph Fourier - Grenoble I

Remerciements

Je voudrais profiter de cette page pour remercier les personnes qui m'ont aidé tout au long de ces années, car c'est grâce à elles si j'ai réussi à réaliser ce manuscrit.

Tout d'abord *MERCI* :

- à mes parents pour m'avoir soutenu, supporté, aidé, conseillé... durant ces 28 années,
- à mon frère (Martial) et mes amis (*Sheep*, *Dukenuke*, *Cochon*, *Fifou*, *Taki, Modeste, Marc, Rémi, Seb, Thomas*, *Luc,* ...) pour les soirées, pubs, voyages, jeux ... auxquels nous avons participés ensemble. J'en profite aussi pour remercier *madame Hekken Ann* et *monsieur Hart Rick* pour leurs contributions «modérées» à nos soirées :-),
- à Laurence pour m'avoir fait découvrir la vraie face de l'IHM, et m'avoir donnée ma chance lors de mon stage de MST ESI,
- aux «kids» de l'équipe (Alexandre, Benoit, Bérangère, Christophe, Julien, Jullien, Lionel, Nicolas, Olfa, Philippe, Sylvain, Sylvie, Tung), aux moins «kids» (François, Gaëlle, Sophie) et aux anciens (Chaouki, Dave, Fred, Manu, Yann) pour leurs conseils, pour les soirées et les parties de baseball ...
- Enfin, merci à Joëlle pour les heures de «Grand beau» qu'elle a perdu lors de la rédaction de nos papiers, la correction de mes présentations et la relecture de ce manuscrit.

Ensuite je voudrais remercier :

- le botaniste Jules Planchon de la faculté de pharmacie de Montpellier pour sa découverte de 1868, pour l'implantation de *Vitis Labrusca* en France et pour l'idée de greffer notre bon vieux *Vitis Vinifera* sur des pieds américains résistants (*V. Riparia, V. Rupestrix, V. Berlandieri*) ,
- les personnes suivantes pour leurs participations dans le développement de l'informatique: Vinton Cerf (TCP/IP), Douglas

Engelbart (la souris), Marcian Hoff (le microprocesseur) et François Gernelle (micro-ordinateur),
- Hayao Miyazaki, Georges Lucas et les studios Pixar pour leurs films.

Enfin, des remerciements spéciaux
- au personnel du centre joseph fourier de Valence pour leur accueil durant mes trois ans de monitorat, je pense particulairement à Michel et à Xavier,
- aux enseignants et non enseignants de l'IUT 2 de Grenoble où j'ai effectué mon ATER,
- aux membres du projet Gloss,
- aux chercheurs du groupe de travail francais «Mobilité et Ubiquité».
- à tous ceux dont je n'ai pas parlé et qui mériteraient d'être dans ces pages.

Table des matières

Table des Figures

A mes Parents,

Introduction

The idea of ubiquitous computing first arose from contemplating the place of today's computer in actual activities of everyday life. In particular, anthropological studies of work life teach us that people primarily work in a world of shared situations and unexamined technological skills.

Weiser, 1993, Some Computer Science Issues in Ubiquitous Computing. Communications of the ACM, p76.

1. Sujet

Ces travaux de recherche doctorale s'inscrivent dans le domaine naissant de l'informatique ambiante avec, comme sujet d'étude, la notion de contexte. L'objectif est double : aboutir à une définition de la notion de contexte qui soit opérationnelle pour les besoins de l'Interaction Homme-Machine et offrir une infrastructure logicielle qui facilite le développement de systèmes interactifs en informatique ambiante.

L'informatique ambiante, que l'on décline sous différents termes[1], correspond à cette (r)évolution technique envisagée il y a une quinzaine d'années par Weiser [Weisser 91]. Par opposition à l'informatique traditionnelle, la nouveauté tient aux capacités de mobilité et d'intégration des systèmes dans le milieu physique [Lyytinen 02], et ceci de manière spontanée, à de multiples échelles, de la planète au micro-, voire nano-objet. Sur le plan technique, cette vision se traduit par la composition de services de toutes sortes s'appuyant sur une infrastructure à granularité et géométrie variables. La finalité est l'amplification de l'espace physique au moyen de services numériques formant un tout adapté à l'homme, c'est-à-dire adaptable et adaptatif.

L'adaptation des logiciels n'est pas un problème nouveau. Mais l'imprévu, qui prévaut en informatique ambiante, lui confère un relief particulier. D'où le retour au premier plan d'une notion introduite dans les tout premiers systèmes d'exploitation : le contexte. Pour ces systèmes, le contexte dénotait l'ensemble des informations nécessaires et suffisantes pour traiter une interruption et reprendre l'exécution du programme interrompu. En informatique ambiante, cet ensemble est difficile à cerner puisque le nécessaire et le suffisant sont conditionnés par la nature de l'amplification souhaitée qui elle-même est indéfinie. Le scénario de la figure 1 illustre l'ampleur de la tâche.

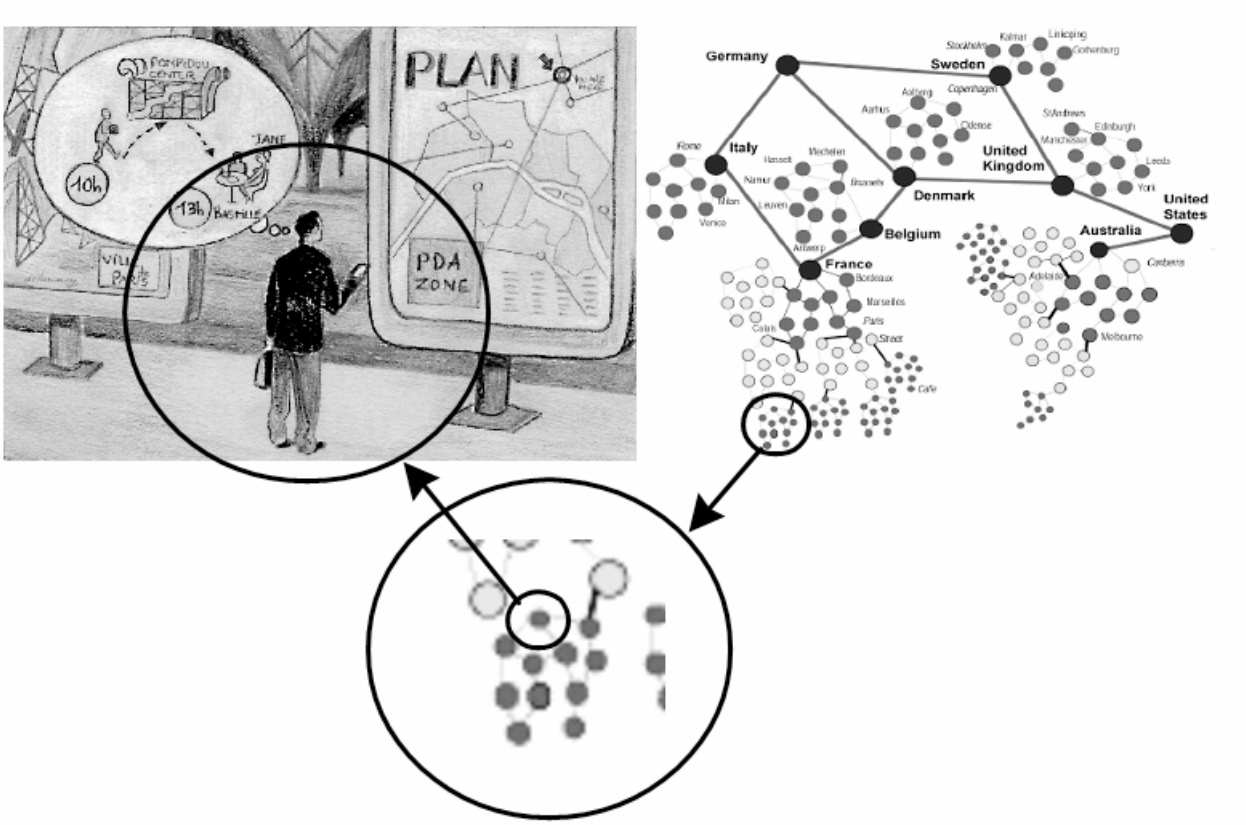
Un voyageur pense avoir un peu de temps pour visiter le musée Beaubourg avant de prendre le train du retour. Il s'approche du plan du quartier, énonce (ou saisit) "Beaubourg" au moyen de son assistant personnel. En réponse, le plan montre le chemin optimal, transférable sur le dispositif personnel. Le système indique en privé le temps que notre voyageur peut raisonnablement dédier à sa visite. Dans cet exemple, la

1. On relève également les termes : informatique ubiquiste (ubiquitous computing), ordinateur évanescent (disappearing computer), informatique pervasive (pervasive computing), informatique augmentée (augmented computing), ou encore objets communicants et systèmes embarqués (embedded systems), chacun mettant en relief une propriété particulière de ces futurs systèmes (qui ne formeront peut-être, plus qu'un!). Cette diversité de nomenclature traduit l'instabilité des concepts, un espace problème aux contours mal cernés, en tout cas, un domaine qui se construit.

connexion de l'assistant personnel avec le plan est réalisée automatiquement, par proximité. La réponse à la demande dépend de la localisation du voyageur (quelque part à Paris). Elle dépend aussi des contraintes notées dans son agenda (ne pas oublier de passer à une pharmacie), et de l'heure de départ du train enregistrée sur le billet électronique.

Cet exemple montre l'importance des informations contextuelles pour une réponse adaptée. Quelles sont-elles ? Intuitivement, sont nécessaires dans ce scénario : 1) la détection de la proximité de l'utilisateur avec le plan en sorte que le système ambiant découvre l'arrivée d'un assistant personnel et de là, établisse une connexion avec un service d'aide à la navigation ; 2) la connaissance du lieu et de l'heure de la demande, les contraintes et préférences de l'utilisateur afin d'éviter au voyageur de saisir des paramètres ; 3) les caractéristiques des ressources d'interaction (plan et assistant personnel) pour que le système ajuste la visualisation du chemin mais aussi juge du caractère privé ou public du dispositif de rendu. Mais ces informations sont-elles suffisantes ? Seront-t-elles toujours

Figure 1
Exemple tiré du scénario du projet européen GLOSS
Adossés à une infrastructure multi-échelle (à droite de la figure) [Kirby 02], des services de renseignements intégrés dans un mur actif obtenus par connexion spontanée avec un dispositif personnel (à gauche de la figure)[Coutaz 03].



disponibles en ces mêmes circonstances ? En d'autres circonstances, aurons-nous affaire au même ensemble ?

L'exemple, pourtant simple, de la figure 1, laisse entrevoir la difficulté de l'entreprise. C'est que la notion de contexte est utile pour de nombreux services logiciels et ceci quel que soit leur niveau d'abstraction. Reprenant notre exemple, nous pouvons citer : les services de connexion à l'infrastructure et la découverte de ressources d'interaction, l'authentification et les contrôles d'accès, la protection de l'espace privé, la composition dynamique de services d'information (système d'information géographique et navigation, agenda et préférences personnels, achat de titres de transport).

En réponse à la complexité que laisse entrevoir notre exemple, cette thèse envisage la notion de contexte sous l'angle de son utilité pour le développement de services liés à l'Interaction Homme-Machine (IHM).

2. *Contexte et Interaction Homme-Machine*

En IHM, le contexte est exploité dans les étapes amont du processus de conception d'un système tout comme à l'exécution. Dans les deux cas, il s'agit d'améliorer la qualité de l'interaction personne-système. L'analyse qui suit sur l'utilité du contexte en IHM comprend trois volets : le contexte et son rôle dans le processus de développement, le contexte pour la prise en compte de la diversité des dispositifs et des lieux d'interaction, le contexte et l'interaction implicite pour enrichir la bande passante de l'interaction explicite.

2.1. CONTEXT ET PROCESSUS DE CONCEPTION

L'importance du contexte d'interaction est reconnue depuis longtemps en conception de systèmes interactifs. La conception contextuelle (contextual design) de Beyer et Holtzblatt, fondée sur la collecte et l'organisation de données de terrain (contextual inquiry), en est un exemple type [Beyer 98]. L'analyse des données permet d'identifier la façon dont les individus travaillent en situation, de détecter les insuffisances et d'engager une reconception de la pratique professionnelle par suite de l'introduction d'un nouveau support informatique.

La psychologie cognitive appliquée à l'Interaction Homme-Machine est encore largement marquée par le modèle GOMS [Card 83] selon lequel l'individu agit de manière rationnelle suivant un plan déterminé de tâches (ou buts) décomposables en sous-tâches (ou sous-buts). Cette approche ne laisse aucune place au comportement opportuniste, c'est-à-dire à l'improvisation, que justifie notamment l'inattendu. Les modèles de

l'Action Située (Situated Action) [Schuman 87], la Théorie de l'Activité [Bardram 97] ou encore la Cognition Distribuée (Distributed Cognition) [Halverson 94] visent autre chose que la décomposition stricte de GOMS en considérant le contexte comme pièce maîtresse.

Si l'importance du contexte est acquise en méthode de conception, en pratique, les systèmes relevant de l'informatique conventionnelle sont conçus pour un contexte donné pré-établi. Par manque d'outils, les informations contextuelles se perdent au cours du processus de développement. Et pourtant, si le contexte vient à changer, l'utilisabilité, voire l'utilité du système risquent d'être compromises. Par exemple, un système d'achat de billets de train conçu pour fonctionner sur des bornes interactives visera à réduire la trajectoire d'interaction (le train part dans cinq minutes : il s'agit de faire vite !) et optera pour une sérialisation des tâches (un guidage bien pensé réduit les sources d'erreur). Cette même interface utilisée chez soi, restera efficace en termes de trajectoire d'interaction, mais ne fournira pas l'accès aux bons plans (option inutile pour le voyageur de la gare, mais indispensable pour celui qui souhaite flâner sur la Toile). Chez soi, l'utilité du système s'en trouvera donc diminuée.

Au bilan, si le contexte est une notion admise depuis longtemps en conception de systèmes interactifs, en pratique il n'est utilisé qu'en phase amont du processus de développement. Tout ce qui relève du contexte se dilue progressivement au cours du processus de développement et le triangle classique en IHM " utilisateur-tâche-machine " ne fonctionne que pour un contexte statique prévu à l'avance. Or, la technologie aidant, l'utilisateur est mobile, les dispositifs d'interaction (téléphones et assistants numériques personnels) aussi.

2.2. CONTEXTE, DIVERSITÉ DES DISPOSITIFS ET DES LIEUX D'INTERACTION

Les dispositifs d'interaction se diversifient par la forme et la finalité. L'ordinateur tout usage de type calculateur de bureau se voit prolongé d'appareils dédiés comme les assistants personnels (PDA) et les téléphones mobiles. Inversement, avec la convergence télévision-informatique, un objet à finalité bornée devient un dispositif tout usage. Les PDA, comme des Legos, incluent progressivement les services de téléphonie et inversement, les fabricants de téléphone font du " cellulaire " un véritable PDA. Avec l'ordinateur évanescent [Weisser 93], l'espace et les objets de la vie courante prennent le statut de ressources d'interaction [Nabaztag].

Il en résulte un foisonnement de solutions techniques. Dans cet espace, l'utilisateur veut avoir le choix. Dès lors, il convient d'envisager l'utilisation mixte du gros calculateur comme du petit dispositif, du fixe comme du mobile, de l'ordinateur palpable à l'ordinateur évanescent. Nous assistons progressivement à la création d'un tissu informationnel et

computationnel extrêmement malléable. Mais cette plasticité [Thevenin 99] dépend de l'accès au contexte pour la découverte dynamique de ressources d'interaction et de leurs caractéristiques.

La miniaturisation des dispositifs d'interaction favorise la mobilité et de là, la possibilité d'interagir avec un système en des lieux distincts. En conséquence, le système doit être capable d'identifier le lieu actuel d'interaction. De plus, en un même lieu, les conditions d'interaction peuvent varier : dans une salle, lorsqu'une réunion commence, la sonnerie du téléphone devrait passer automatiquement en mode vibreur. La migration de cette tâche de configuration vers le système éviterait à l'utilisateur d'interrompre son activité (ou d'oublier de la réaliser !). Dans ce cas, on dit que l'adaptation sert l'humain de manière directe. L'adaptation peut aussi le servir de manière indirecte. Par exemple, un système de suivi par vision par ordinateur qui s'adapte aux changements de conditions lumineuses assure la continuité du service de suivi. Elle sert l'humain de manière indirecte.

La perception du contexte permet aussi de modéliser l'interaction implicite.

2.3. CONTEXTE ET INTERACTION IMPLICITE

En communication interpersonnelle, les gestes qui accompagnent un discours, les expressions faciales, les attitudes corporelles, les expériences partagées, le vécu journalier, etc., sont autant de facteurs implicites qui participent à la compréhension mutuelle des échanges qui, eux, sont explicites (ils sont voulus). En bref, l'utilisation d'informations contextuelles implicites améliore la bande passante de la communication explicite. Elle la rend plus efficace.

Actuellement, les systèmes interactifs d'usage courant, sans accès au contexte, n'ont pas les moyens d'élaborer de l'information implicite. Au mieux, en proposent-ils des versions appauvries. La liste des derniers fichiers ou options utilisés des éditeurs est un exemple commun. Il s'agit d'une connaissance implicite puisqu'elle évite à l'utilisateur de spécifier explicitement certains paramètres, mais le service est pour le moins embryonnaire. Donner accès au contexte devrait permettre d'améliorer la bande passante des canaux d'échange entre l'utilisateur et le système pour faire migrer vers le système des tâches et actions peu gratifiantes.

En résumé, cette rapide analyse démontre la nécessité pour le système de gérer un contexte afin de proposer un service adapté à la situation, qu'il s'agisse de renseigner l'utilisateur à bon escient, de déléguer au système les tâches subalternes, de migrer sans discontinuité entre dispositifs d'interaction. L'objectif général vise une meilleure efficacité interactionnelle, une amélioration de la disponibilité du système, un élargissement de l'éventail des choix, en bref un progrès dans l'utilité et

l'utilisabilité des systèmes. Les objectifs de cette thèse s'inscrivent dans cette ligne de pensée.

3. Objectifs et approche

En IHM, toute analyse, tout modèle et plus généralement tout outil, adopte un point de vue qui est, soit centré sur l'utilisateur, soit centré sur le système. Dans cette étude, l'accent est mis sur la dimension système. Précisons la couverture des objectifs et leurs limites, sachant que la visée est essentiellement système.

3.1. COUVERTURE DES OBJECTIFS

Cette thèse vise deux volets complémentaires. L'un sert l'étape de conception. Le second intervient dans la mise en œuvre technique de systèmes interactifs en informatique ambiante :

- *Définition de la notion de contexte sous forme d'un cadre conceptuel*. Si le contexte doit être exploité de manière rationnelle, il convient d'en comprendre la nature. Le cadre conceptuel doit permettre au concepteur de systèmes interactif d'entrevoir les dimensions du contexte, de se poser les bonnes questions, et d'en extraire les éléments pertinents pour le cas particulier du système à développer. Par exemple, pour un système de visite guidée, la localisation est une dimension essentielle.
- *Définition d'un modèle et mécanisme opérationnels* qui facilitent la mise en œuvre de systèmes interactifs et qui respectent le principe de séparation fonctionnelle. En Interaction Homme-Machine, la séparation fonctionnelle est appliquée de la manière suivante : le Noyau Fonctionnel (ou l'application) est un composant distinct de l'Interface Homme-Machine (IHM). À son tour, l'IHM se décompose en Contrôle du Dialogue et Présentation. Le premier est chargé de l'enchaînement des tâches utilisateur (par exemple, la tâche " ouvrir document " doit être effectuée avant d'entreprendre les tâches d'édition). Le second est constitué d'interacteurs (plus communément appelés widgets) qui représentent les concepts applicatifs auprès de l'utilisateur. Le principe de séparation fonctionnelle permet de modifier l'un des composants tout en minimisant les effets de bord sur les autres. Ainsi, changer d'interacteur n'a pas ou peu d'impact sur le Contrôle de Dialogue et le Noyau Fonctionnel. Nous inspirant du savoir-faire en architecture logicielle des systèmes interactifs, nos abstractions et mécanisme logiciels doivent permettre de rendre sensibles au contexte des systèmes qui ne le sont pas, ou de faire évoluer un système sensible au contexte en limitant les modifications à un seul composant.

Les propositions conceptuelles et techniques envisagées se veulent générales, mais devront servir en priorité les besoins de l'Interaction Homme-Machine. Les objectifs sont ambitieux. Il convient donc, dans ce cadre général, de préciser les limites des résultats attendus.

3.2. LIMITES ET HYPOTHÈSES DE L'ÉTUDE

Cette étude se borne à offrir un espace de conception et un support à la mise en œuvre de systèmes interactifs en informatique ambiante. Bien qu'importants, les points suivants ne seront pas traités dans le temps imparti à la thèse : l'éthique, l'évaluation des effets de l'adaptation, les erreurs d'évaluation de la situation par le système.

Ethique

La sensibilité au contexte peut s'appuyer sur la capture du comportement de l'utilisateur. On envisage par exemple, des systèmes de télésurveillance des personnes âgées pour qu'elles puissent rester à leur domicile plutôt que rejoindre une maison de retraite (exemple : le projet IMAG RESID-HIS [RESIDE-HIS]). À l'évidence, ce type de système fragilise la protection de l'espace privé.

L'évaluation des effets de l'adaptation

L'adaptation automatique n'est pas toujours bonne à prendre :

- L'adaptation peut avoir un effet disruptif. La propriété de prévisibilité [Gram 96] préconisée en IHM reste valable. Ou encore, si le comportement doit changer radicalement pour répondre à la situation, il faudra imaginer des IHM de transition qui permettront à l'utilisateur d'évaluer progressivement l'évolution.
- L'adaptation sous forme de migration de tâche n'est pas toujours souhaitable : l'utilisateur peut avoir le sentiment de ne pas contrôler la situation ou de perdre progressivement son expertise (cas des pilotes d'avion). L'analyse du système GUIDE [Cheverst 01], un guide touristique pour la ville de Lancaster, est à ce titre intéressante. Dans une première version du système, les utilisateurs n'étaient renseignés que sur les monuments proches, interdisant ainsi l'accès à d'autres informations sur la ville. En voulant simplifier la tâche de l'utilisateur (celui-ci n'avait rien à faire sauf se déplacer), les concepteurs ont créé un système beaucoup trop préemptif.

Les erreurs d'évaluation de la situation

Ces erreurs peuvent conduire l'utilisateur à douter du bon fonctionnement du système. Pour ces systèmes où la sécurité des personnes est concernée, ce problème est fondamental. Le modèle du contexte devra prendre en compte la notion d'incertitude, mais celle-ci ne sera pas validée par l'expérience à partir de capteurs instables, imprécis, et non fiables. Ces problèmes de recherche qui relèvent d'un autre domaine, sont supposés traités par d'autres équipes.

Les objectifs et leurs contours étant fixés, il convient de préciser l'approche adoptée.

4. Approche

L'approche retenue est essentiellement descendante :

- comprendre la nature du concept de contexte,
- en définir un modèle abstrait,
- traduire ce modèle en une infrastructure logicielle,
- évaluer l'infrastructure par le biais de maquettes d'application,
- puis, ajuster l'infrastructure, voire le modèle abstrait, de manière itérative.

Comprendre la nature du contexte s'appuie pour une large part sur l'analyse de l'état de l'art et l'expérimentation. C'est ce qui a été fait dans cette étude : compilation de l'état de l'art tant sur la notion de contexte que sur les techniques générales de mise en œuvre (approche par composants, architecture des systèmes répartis, découverte de ressources) suivi de la réalisation d'une application très simple (détecter l'absence de tasses à café dans la cafétéria du laboratoire).

Ces éléments de connaissance de base ont permis de prendre position et de définir un modèle conceptuel. À cette étape, la publication des premiers résultats ont permis de pré-valider nos choix [Rey 01], [Coutaz 02a], [Crowley 02], [Rey 02], [Rey 04]. Puis plusieurs maquettes applicatives utilisant les contexteurs ont été développées dans l'équipe, conduisant à des améliorations techniques.

5. Organisation du Rapport

L'organisation de ce mémoire reflète les étapes essentielles de l'approche adoptée.

- Le chapitre I analyse la notion de contexte sous l'angle des besoins et usages en Interaction Homme-Machine et en propose un modèle conceptuel. L'état de l'art appelle le constat suivant : il n'y a pas de contexte sans contexte. Autrement dit, le contexte se définit pour une finalité (ou utilité) précise. Une définition au cas par cas n'est pas envisageable : elle ne serait pas automatisable. Aussi, on se propose de définir un support conceptuel à partir duquel il est possible de représenter (ou générer) des contextes adaptés à une finalité choisie, à savoir : un réseau de contextes et de situations défini sur des ensembles d'entités, de relations et de rôles, le tout identifié au moyen d'observables.
- Le chapitre II adopte un point de vue technique avec une revue des solutions logicielles actuelles en matière d'infrastructures de gestion du

contexte. Celles-ci sont évaluées sous l'angle des services fournis complété par des facteurs qualité comme la flexibilité et la souplesse. Dans ce but, une décomposition fonctionnelle de référence est proposée : la pyramide du contexte. Ce modèle permet non seulement de comprendre les niveaux d'abstraction que recouvre le contexte mais aussi de comparer les infrastructures actuelles en termes de couverture fonctionnelle. Cette revue de l'état de l'art appelle de nouvelles solutions. Je propose le contexteur.

- Le chapitre III présente le modèle de contexteur et son extension, le répéteur. Le contexteur est abstraction logicielle, reflet des fondements conceptuels de l'ontologie du chapitre I. Il vise la couverture des niveaux bas de la pyramide du contexte selon une orientation processus (par opposition à une vision données). Le répéteur étend le service d'une famille de contexteurs au-delà de sa portée locale. L'orientation processus tient à la nécessité de gérer les fluctuations du réseau et des capteurs par une architecture flexible capable de passer à l'échelle.
- Le chapitre IV présente de manière détaillée une réalisation logicielle du modèle de contexteur. Cette mise en œuvre, dirigée par les requis de flexibilité et de fiabilité justifiés au chapitre II, se présente comme un ensemble de composants autonomiques pair à pair qui s'autoconfigurent dynamiquement. Cette infrastructure, réalisée en Java 1.4, est disponible sur toute plate-forme d'accueil (Windows, MacOS, etc.) dotée des protocoles TCP-IP et UDP.
- Au chapitre V, sont présentées des exemples d'applications utilsant l'infrastructure de contexteurs : un observatoire des activités de personnes au bureau, une Machine d'Interaction Abstraite qui permet le couplage dynamique de ressources d'interaction, Ethylene, un intergiciel pour la plasticité des Interfaces Homme-Machine.
- Une conclusion résume les contributions de ces travaux de recherche et en présente les perspectives.

Chapitre I La notion de contexte en IHM

La scène est blanche, infinie. Partout, ce n'est qu'une clarté implacable, la lumière d'un univers en incandescence, le chaos, une matière qui n'a encore ni sens ni nom ...

1996, La plus belle histoire du monde. Acte 1 - Scène 1 : Le chaos, p21

Le contexte, nous l'avons vu en introduction, n'est pas un concept nouveau en informatique : dès les années soixante, systèmes d'exploitation, théorie des langages et Intelligence Artificielle exploitent déjà cette notion. Avec l'émergence de l'informatique ambiante, le terme est redécouvert et placé au cœur des débats sans pour autant faire l'objet d'une définition consensuelle claire et définitive.

Toutefois, l'analyse de l'état de l'art conduit à ce double constat :

- Il n'y a pas de contexte sans contexte [Brézillon 02]. Autrement dit, le contexte n'existe pas en tant que tel. Il émerge, ou se définit, pour une finalité (ou utilité) précise.
- Le contexte est un ensemble d'informations. Cet ensemble est structuré, il est partagé, il évolue et sert l'interprétation [Winograd 01]. La nature des informations, de même, l'interprétation qui en est faite, dépendent de la finalité.

Il convient donc, avant tout, de cerner la finalité et de là, définir les informations nécessaires et suffisantes pour servir cette finalité. L'exemple de la figure 1, commenté dans le chapitre d'introduction, démontre la difficulté de la tâche si l'on procède au cas par cas. Aussi, il est nécessaire de définir un support conceptuel à partir duquel il est possible de représenter (ou générer) des contextes adaptés à une finalité choisie. C'est ce qui a été réalisé dans le cadre de cette thèse.

Ce chapitre présente une synthèse de l'état de l'art sur la notion de contexte suivie de ma contribution : des définitions du dictionnaire (section 1), on passe ensuite aux points de vue des auteurs représentatifs du domaine de l'informatique ambiante (section 2). Le reste du chapitre est dédié à la contribution de cette thèse au concept de contexte : le cadre conceptuel (définition et ontologie) est prolongé, dans les sections 4 et 5, d'une typologie des contextes et de leurs entités. Les deux dernières sections de ce chapitre présentent de manière plus concrète l'apport du cadre conceptuel : la section 6 montre, par l'exemple, comment dériver un modèle du contexte à partir d'une description informelle de la finalité d'un système, et la section 7 propose, à partir des éléments du cadre conceptuel, une classification de systèmes interactifs en informatique ambiante. L'intérêt de cette dernière section est double : (a) offrir une analyse comparative synthétique des nombreuses réalisations de l'état de l'art ; (b) identifier les requis fonctionnels en matière d'infrastructures logicielles pour la mise œuvre du contexte. Ce dernier point nous permettra d'ouvrir la discussion sur les aspects logiciels du contexte.

1. Que disent les dictionnaires ?

Contexte : du latin *contextus* "assemblage", *contextere* "tisser avec". Ainsi, les origines latines du terme dénotent l'aspect constructif (assemblage) et associatif (tisser avec) de la notion de contexte.

Parmi les dictionnaires de référence, citons :

- Le Petit Robert : "*ensemble du texte qui entoure un élément de la langue (mot, phrase, fragment d'énoncé) et dont dépend son sens, sa valeur " ou encore " ensemble des circonstances dans lesquelles s'insère un fait*".
- L'encyclopédie Larousse : "*ensemble des conditions naturelles, sociales, culturelles dans lesquelles se situe un énoncé, un discours*" [Encyclopédie Larousse]. ou encore : "*ensemble des circonstances dans lesquelles se produit un événement, se situe une action*" [Encyclopédie Larousse].
- Hachette Multimédia : "*ensemble des éléments qui entourent un fait et permettent de le comprendre*" [Hachette Multimédia].
- Le Grand dictionnaire numérique en ligne : "*Énoncé dans lequel figure le terme étudié*" ou encore "*Ensemble d'un texte précédant ou suivant un mot, une phrase, un passage qui éclaire particulièrement la pensée d'un auteur*". Et, si l'on parle d'informatique [GrandDictionnaire] : "*Ensemble d'informations concernant l'action du stylet, en rapport principalement avec sa localisation à l'écran, qui permet au système d'exploitation de l'ordinateur à stylet de différencier les commandes et l'entrée des données, et de fonctionner en conséquence*."

Ces définitions partagent l'idée d'ensemble d'informations associé à quelque chose : "*ensemble […] qui entoure*", "*ensemble dans lequel se situe …*". La nature du "quelque chose" (*texte, fait, énoncé, discours, événement, action, etc.*), dépend précisément de l'utilité du contexte, confirmant le fait qu'il n'y a pas de contexte sans contexte. Et cette utilité, permettre de "*comprendre*", de "*fonctionner en conséquence*", ou de donner un "*sens*, [une] *valeur…*", peut s'exprimer de manière générique par "servir l'interprétation".

En somme, le constat énoncé en introduction de ce chapitre, est cohérent avec les définitions générales des dictionnaires. Comment le contexte est-il abordé en informatique ambiante ?

2. *Contexte et informatique ambiante*

La présentation qui suit sur les définitions de l'état de l'art en informatique ambiante est organisée selon l'ordre chronologique, démontrant une progression dans la compréhension de la notion de contexte. Elle est suivie d'une analyse et d'une synthèse.

2.1. ÉTAT DE L'ART

Schilit et Theimer introduisent l'expression *context aware* pour désigner un système doté d'un modèle du contexte. Le contexte, selon Schilit, inclut la localisation et l'identité des personnes et des objets à proximité ainsi que les modifications pouvant intervenir sur ces objets [Schilit 94a], [Schilit 94b]. Étudier le contexte, c'est répondre aux questions " Où es-tu ? ", " Avec qui es-tu ? ", " De quelles ressources disposes-tu à proximité ? ". Il définit donc le contexte comme les changements de l'environnement physique, de l'utilisateur et des ressources de calcul. Ces idées sont reprises par Pascoe [Pascoe 98] puis par Dey [Dey 99]. Un peu plus tard, Brown restreint le contexte aux éléments de l'environnement de l'utilisateur [Brown 96], puis il introduit l'heure, la saison, la température, l'identité et la localisation de l'utilisateur [Brown 97].

Parallèlement aux travaux de Brown, des définitions émergent avec l'introduction explicite du temps et la notion d'état. Ryan assimile le contexte à l'environnement, l'identité et la localisation de l'utilisateur ainsi que le temps [Ryan 97]. Ward voit le contexte comme les états des environnements possibles de l'application [Ward 97].

En 1998, Pascoe définit le contexte comme n sous-ensembles d'états physiques et conceptuels ayant un intérêt pour une entité particulière [Pascoe 98]. Nous relevons ici la référence à la notion de pertinence.

Puis Dey précise la nature des entités en question :

> *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*[1]
> [Dey 99].

Dans leur étude sur la plasticité des IHM, Thevenin et al. aboutissent à une définition assez proche avec la notion de contexte d'interaction. Le contexte d'interaction est un triplet <plate-forme - environnement - utilisateur> où l'environnement est l'ensemble des entités (objets, personnes et événements) périphériques à la tâche courante et pouvant

1. Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.

avoir un impact sur le comportement du système ou de l'utilisateur [Thevenin 99]. Nous relevons ici l'idée de tâche.

Après avoir détaillé trois catégories regroupant les différentes approches sur le contexte (les théories positivistes, phénoménologiques et critiques), Dourish, dans [Dourish 01], conçoit le contexte comme une forme d'information. Il le définit comme stable et définissable (on verra à la section 4 de ce chapitre que même si l'on peut déterminer les éléments pertinents du contexte durant la phase de conception, rien ne nous garantit que ceux-ci soient accessible à l'exécution) et insiste sur le fait que le contexte et l'activité (de l'utilisateur) sont deux éléments séparés. Pour lui, l'activité a lieu à l'intérieur du contexte. Nous reprendrons cette idée dans notre propre définition énoncée dans la section 3.

Enfin, les termes environnement et situation sont souvent utilisés comme synonymes de contexte, démontrant une absence de rigueur ou de maîtrise de ces notions.

2.2. ANALYSE ET SYNTHESE

Les définitions de l'état de l'art font toutes référence à la localisation et à l'environnement physique, ce qui n'est pas surprenant dans le " contexte " de l'informatique ambiante. Mais aucune ne couvre à la fois les notions d'entité pertinente, d'état et de temps. Plus surprenant, aucun auteur ne considère la capitalisation des états alors que l'évolution de l'état d'une entité pertinente constitue à son tour une information pertinente. À l'exception de la proposition de Thevenin, aucune définition du contexte ne fait référence à la tâche utilisateur alors que l'utilisation d'un système, que ce soit de manière explicite (par exemple, rédiger un article à l'aide d'un traitement de texte) ou de manière implicite (cas d'une personne télésurveillée), a lieu en relation avec une activité humaine présente ou future.

Rares sont les définitions où la plate-forme d'interaction est vue comme une information contextuelle. Or, nous l'avons vu dans le chapitre d'introduction, la variabilité des capacités de calcul, de mémorisation, de communication, et la nature des dispositifs d'interaction (absence de clavier ou d'écran, ordinateur évanescent) ont un impact sur la nature de l'interaction aussi bien du point de vue du système que du point de vue de l'utilisateur.

Le constat présenté en introduction de ce chapitre, résume la situation :

- Le contexte n'est pas une fin en soi [Fisher 01]. Il émerge, ou se définit, pour une finalité (ou utilité) précise. La section précise les grandes classes de finalité. Dans le cadre de cette thèse, la finalité visée est la perception de l'environnement physique et des actions utilisateurs en vue d'améliorer la qualité de service des logiciels constituant les interfaces homme-machine.

- Le contexte est un ensemble d'informations. Cet ensemble est structuré et partagé ; il évolue et sert l'interprétation [Winograd 01]. Dans le cadre de cette recherche, le partage de l'espace d'informations est envisagé entre le système interactif et les utilisateurs et l'interprétation des informations contextuelles devrait améliorer la qualité de l'interaction entre le système et l'utilisateur. En particulier, l'ancrage de l'interaction dans le monde physique [Mackay 96] est un facteur d'amélioration qu'il convient de considérer.
- Les concepts fondateurs manquent. Aussi procède-t-on au cas par cas par énumération des éléments contextuels selon une approche extensionnelle. La proposition qui suit vise davantage de généricité.

3. Proposition

En premier lieu, précisons le domaine de validité de la notion de contexte sachant que la finalité visée est la perception artificielle pour améliorer les services d'Interface Homme-Machine. Nous illustrons la discussion au moyen d'un cas d'école : le gestionnaire de lumière.

3.1. LE DOMAINE

Le diagramme UML de la figure 1 précise les concepts du monde couvert dans ces travaux de recherche. L'ensemble s'appuie sur la notion d'observable. Un observable est une donnée captée par le système ou calculée à partir de données captées. Dans ce monde, on relève les notions d'utilisateur, d'activité, de tâche, d'entité, de rôle, de relations.

Un utilisateur[1] est une entité du monde vivant. Il a une activité. L'activité d'un utilisateur est un couple <ensemble des tâches courantes, ensemble des tâches de fond>. L'ensemble des tâches courantes regroupe les tâches que l'utilisateur est en train de réaliser en même temps (il s'agit de parallélisme vrai, non pas d'entrelacement de tâches). Les tâches de fond correspondent aux tâches que l'utilisateur a placées en attente. Parmi ces tâches, nous retrouvons les tâches routinières, les tâches prévues de la journée, etc. Cet ensemble de tâches est en perpétuelle évolution. Une tâche courante peut laisser la place à une tâche de fond qui devient alors une tâche courante et la tâche courante prend place dans l'ensemble des tâches de fond.

Une tâche peut mettre en jeu des entités qui, à leur tour, relèvent du monde physique vivant ou du monde inerte. En section 5 (“Classification

1. Dans ce qui suit, le terme utilisateur désigne un ou plusieurs utilisateurs de système interactif.

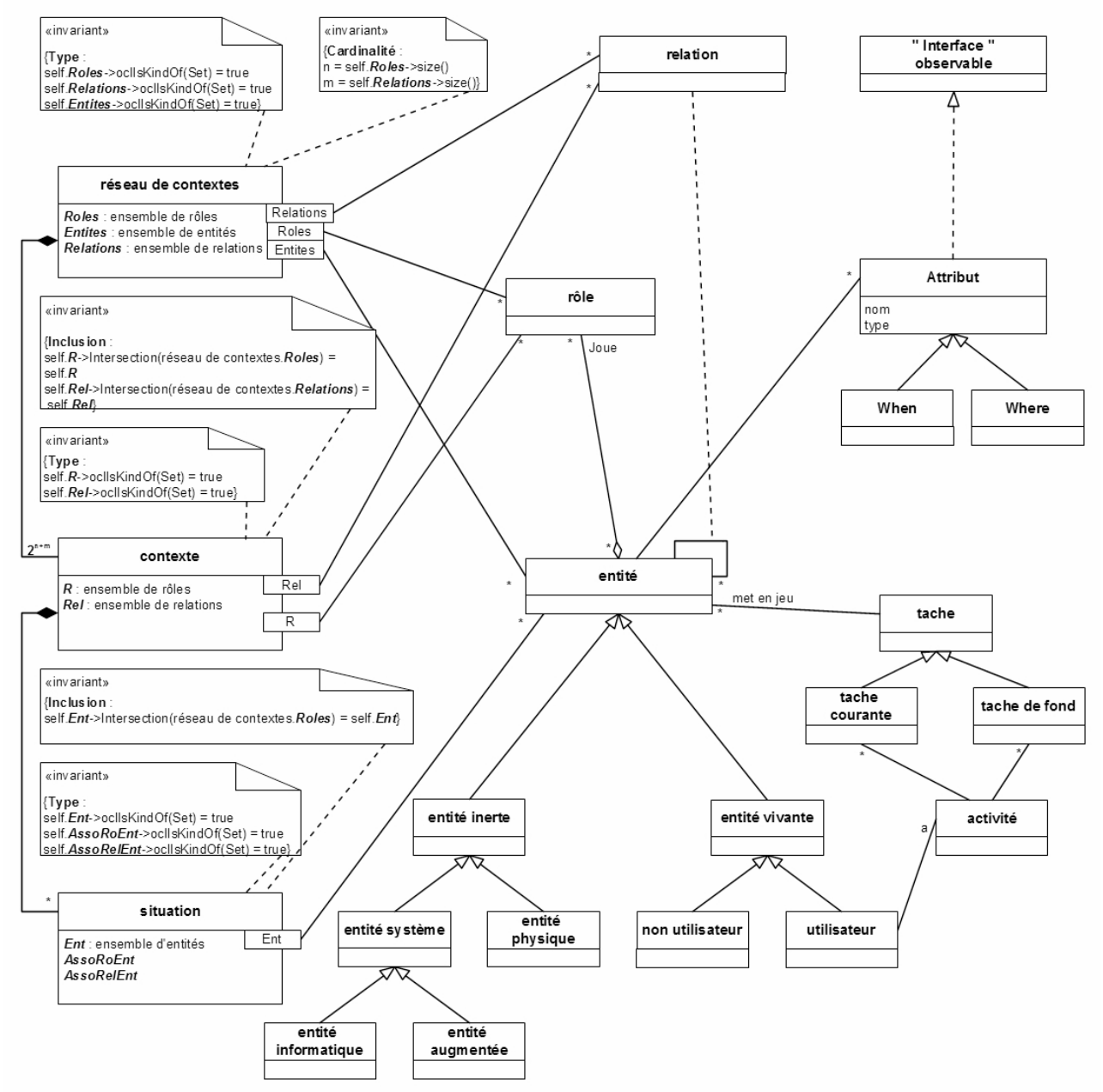


Figure 1
Diagramme de classes correspondant à ma modélisation du monde.

des entités"), nous reviendrons sur ces différentes classes d'entité. Toute entité est un regroupement d'observables. C'est à partir de la mesure d'observables que le système détient la capacité de détecter la présence d'entités du monde physique, de les reconnaître et de les suivre, de même, déterminer la valeur de leurs attributs et de leurs relations.

Une entité peut remplir plusieurs fonctions dites, rôles. Inversement, un rôle peut être joué simultanément par plusieurs entités. La figure 1 montre comment représenter le concept de rôle selon le patron UML de rôle

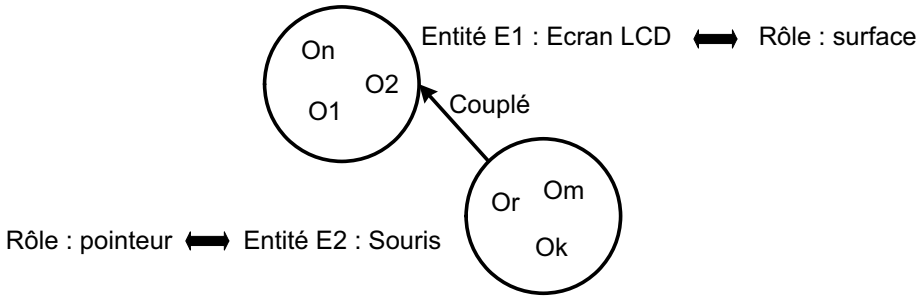


Figure 2
Représentations des entités, des rôles joués par ces entités ainsi que des relations entre les entités.
Ici, l'entité *Ecran LCD*, représentée par le groupement d'observables (O_1,O_2,O_n), joue le rôle de *surface* et l'entité *souris*, représentée par le groupement d'observables (O_r,O_m,O_k), joue le rôle de *pointeur*. Les deux entités entretiennent une relation : la souris est *couplé* à l'écran.

[Baümer 97]. Cette notion de rôle couvre le rôle d'interaction introduit par Lachenal [Lachenal 04] où un rôle d'interaction correspond à un rôle de dispositif d'entrée ou de dispositif de sortie. Dans l'exemple de la figure 2, l'entité écran joue le rôle de surface, sous-classe du rôle dispositif de sortie. Avec l'exemple de la section 3.3 ("Gestionnaire de lumière :"), nous verrons d'autres exemples de rôle : celui d'occupant de bureau ou de plante verte.

Les entités peuvent entretenir des relations : relations spatiales, temporelles, ou autres. L'exemple de la figure 2 illustre une relation de couplage entre un écran et une souris [Barralon 04]. Ce couplage permet l'utilisation de la souris comme dispositif de pointage (rôle de la souris) sur la surface (rôle de l'écran) à laquelle elle est couplée.

Comme le montre le diagramme de la figure 1, les relations, les rôles et les entités constituent les ensembles de définition d'un réseau de contextes et c'est dans un réseau de contextes qu'ont lieu les activités humaines.

3.2. RÉSEAU DE CONTEXTES ET CONTEXTE

Un réseau de contextes *Rc*, est défini sur trois ensembles (*Roles*, *Relations*, *Entites*) et un prédicat (*joueRole*) :

- *Roles* est l'ensemble des rôles considérés par le concepteur du système interactif. $Roles = \{r_1, r_2, \ldots, r_n\}$ où r_i est un rôle,
- *Relations* correspond à l'ensemble des relations entre les entités considérées par le concepteur du système interactif. $Relations = \{rel_1, rel_2, \ldots, rel_n\}$ où rel_i est une relation,
- *Entites* désigne l'ensemble des entités considérées par le concepteur du système interactif. $Entites = \{e_1, e_2, \ldots, e_n\}$ où e_i est une entité.
- *joueRole(e,r)* est un prédicat qui se vérifie si et seulement si l'entité *e* joue le rôle *r* avec $e \in Entites$ et $r \in Roles$.

Chaque noeud du réseau Rc correspond à un contexte C_i.

Un contexte C_i est défini par le couple (R_i,Rel_i) où :

- R_i est l'ensemble des rôles joués par les entités, avec $R_i \subseteq Roles$. De plus, quel que soit $r \in R_i$, il existe une entité $e \in Entites$ tel que e joue le rôle r (propriété 1).

$$\forall(r \in R_i), \exists(e \in Entites) / joueRole\ (e,r) \qquad (1)$$

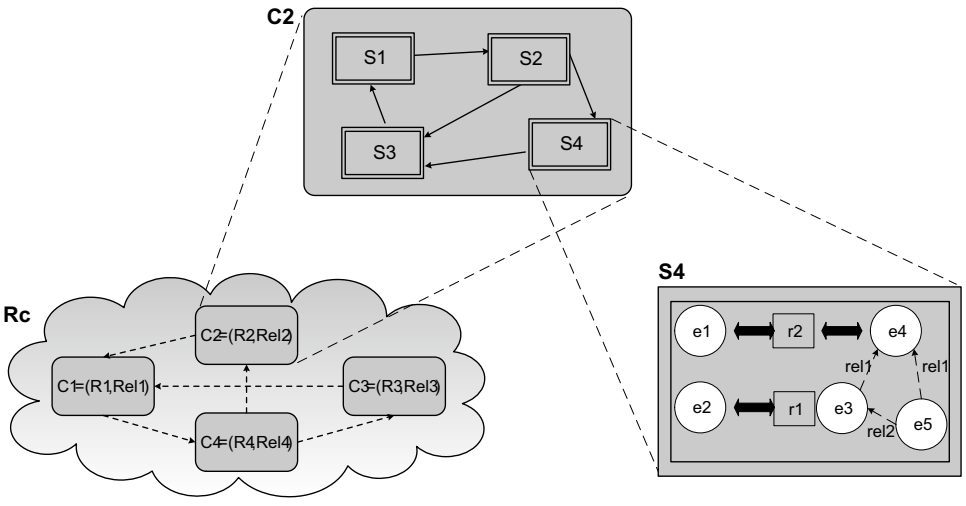
- Rel_i est l'ensemble des relations (entre entités), avec $Rel_i \subseteq Relations$. De plus, quel que soit la relation rel (d'arité j) $\in Rel_i$, il existe j entités e_1 à $e_j \in Entites$ tel que les entités e_1 à e_j entretiennent la relation rel (propriété 2).

$$\forall((rel \in Rel_i) \wedge (arity(rel) = j)),\ \exists(e_1 \ldots e_j \in Entites) / rel(e_1 \ldots e_j) \qquad (2)$$

De ces définitions et propriétés, on déduit qu'il y a changement de contexte lorsque l'une des conditions suivantes devient vraie :

- l'ensemble R_i des rôles remplis change : apparition de rôles et/ou disparition de rôles,
- l'ensemble Rel_i des relations entretenues change : apparition de nouvelles relations et/ou disparition de relations.

Figure 3
Décomposition d'un réseau de contextes en contextes et situations.
Le nuage représente le réseau de contextes Rc composé des quatre contextes (C_1 à C_4). C_2 comprend quatre situations (S_1 à S_4). S_4 est composée de 5 entités (e_1 à e_5), de deux rôles (r_1 et r_2) et de deux relations (rel_1 et rel_2). Les entités e_1 et e_4 jouent le rôle r_2 tandis que l'entité e_2 joue le rôle r_1. Les paires d'entités (e_3 et e_4) et (e_4 et e_5) entretiennent la relation rel_1 tandis que (e_3 et e_5) entretiennent rel_2.



Ce qui s'écrit formellement :

$$\forall((C_1 = (R_1, Rel_1)) \in Rc), \forall((C_2 = (R_2, Rel_2)) \in Rc)/ \\ (C_1 = C_2) \Leftrightarrow ((R_1 = R_2) \wedge (Rel_1 = Rel_2)) \quad (3)$$

Le contexte C_i, nœud du réseau de contexte Rc, respecte les conditions suivantes :

$$\forall(C_i \in Rc), \exists R_i, Rel_i/ \\ (C = (R_i, Rel_i)) \Leftrightarrow (R_i \in \wp(Roles)) \wedge (Rel_i \in \wp(Relations)) \quad (4)$$

où $\wp(Roles)$ et $\wp(Relations)$ sont, respectivement, l'ensemble des parties de *Roles* et l'ensemble des parties de *Relations*.

La cardinalité de Rc est le produit des cardinalités de $\wp(Roles)$ et de $\wp(Relations)$. Soit n la cardinalité de *Roles* et m la cardinalité de *Relations*, les formules suivantes détaillent le calcul de la cardinalité de Rc :

$$Card(\wp(Roles)) = \sum_{k=0}^{n} C_n^k = \sum_{k=0}^{n} \frac{n(n-1)\ldots(n-k+1)}{k!} = 2^n \quad (5)$$

$$Card(\wp(Relations)) = \sum_{k=0}^{m} C_m^k = 2^m \quad (6)$$

$$Card(Rc) = Card(\wp(Roles)) \times Card(\wp(Relations)) \\ Card(Rc) = 2^n \times 2^m = 2^{n+m} \quad (7)$$

Cette information, facilement calculable, permet aux concepteurs de systèmes interactifs de détecter dans leur analyse, l'oubli de contextes, oublis pouvant entraîner des comportements anormaux de la part du système.

Comme le montre la figure 3, chaque contexte est à son tour, un réseau de situations. Pour l'instant, on s'en tient au réseau de contextes que nous illustrons au moyen d'un exemple : le gestionnaire de lumière. Ce gestionnaire, on le verra, justifiera la notion de situation.

3.3. GESTIONNAIRE DE LUMIÈRE :

Nous devons réaliser un système de gestion automatique de l'éclairage d'un bureau. Pour modifier la lumière du bureau, le programme dispose de deux moyens : la lampe du plafond et les rideaux électriques des fenêtres. Après une étude des besoins, nous relevons trois points importants : pour des raisons d'économie et de confort, les occupants préfèrent utiliser la lumière naturelle plutôt que la lampe. Les rideaux servent aussi à sécuriser le bâtiment. Ils doivent donc être fermés le soir et pendant les

jours de congé. Enfin, il est courant que des utilisateurs possèdent des plantes vertes dans leur bureau. Une plante verte a besoin d'un peu de lumière naturelle pour survivre.

Pour ce problème, les entités mises en jeu sont les suivantes :
- le personnel de bureau,
- les plantes vertes, fleurs et autres végétaux chlorophylliens,
- le plafonnier,
- les rideaux motorisés,
- le bureau,
- et l'environnement extérieur.

Dans cette liste, nous pourrons ignorer deux entités, la lampe du plafond et les rideaux, qui sont deux effecteurs et qui n'interviendront pas dans notre analyse.

Nous obtenons l'ensemble *Entites* = {*personnes*, *plantes vertes*, *environnement extérieur*, *bureau*}.

Il nous faut maintenant définir les contextes utiles au problème. Pour cela, nous commençons par définir (et/ou identifier) les rôles et les relations que nous voudrions observer puisque le contexte est défini en fonction de ces ensembles. En l'occurrence, deux rôles et une relation nous intéressent :
- Nous voulons savoir s'il y a quelqu'un dans le bureau, c'est-à-dire, s'il existe au moins une entité jouant le rôle *d'occupant* de bureau.

Figure 4
Enumération des 8 cas identifiés pour l'exemple de la gestion automatique de la lumière.

Relation / **Rôles**	La relation *illumine* n'est pas assurée		La relation *illumine* est assurée	
	Le rôle *végétaux* est assuré	Le rôle *végétaux* n'est pas assuré	Le rôle *végétaux* est assuré	Le rôle *végétaux* n'est pas assuré
Le rôle *occupant* est assuré	C1	C2	C5	C6
Le rôle *occupant* n'est pas assuré	C3	C4	C7	C8

- Nous voulons savoir si le bureau contient des plantes vertes, des fleurs ou autres végétaux chlorophylliens. Ces végétaux ont besoin de soleil pour transformer le CO_2 de l'air en sucre (énergie utilisable) via le processus de photosynthèse. Par conséquent, nous voulons savoir s'il existe au moins une entité jouant le rôle de *végétaux*.

Nous obtenons : $Roles = \{occupant, végétaux\}$.

Avec ces deux rôles, nous pouvons définir quatre contextes correspondant aux combinaisons suivantes (se reporter au tableau de la figure 4) :
- Présence d'occupants et de végétaux (C1), $R = \{occupant, végétaux\}$;
- Présence uniquement d'occupants (C2), $R = \{occupant\}$;
- Présence uniquement de végétaux (C3), $R = \{végétaux\}$;
- Aucune présence (C4), $R = \varnothing$.

Ayant défini les rôles, il convient de définir les relations. Dans notre cas, la relation *illumine*. Cette relation indique si l'entité environnement extérieur fournit de la lumière aux entités du bureau. Alors : $Relations = \{illumine\}$;

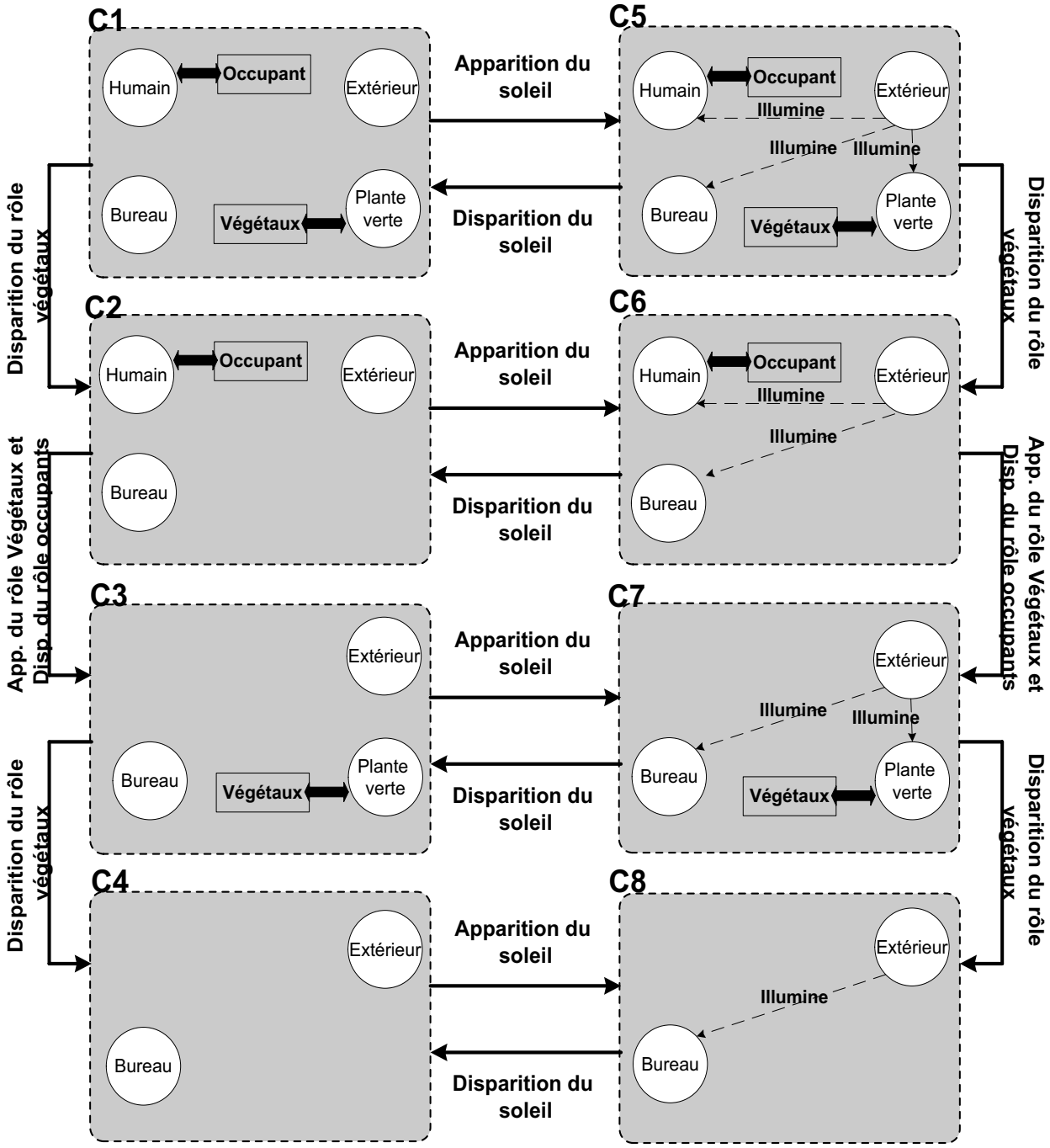
Chacun des quatre contextes, identifiés jusqu'ici sans tenir compte de Relations, se dédouble au final en deux contextes selon que la relation illumine existe ou pas. Comme le montre la figure 4.
- Dans les contextes notés C1 à C4, il ne fait pas soleil. Il n'y a pas de relation *illumine* entre l'entité *environnement extérieur* et les autres entités. Il convient de noter que cette modélisation ne fait pas de différence entre la nuit ou un jour nuageux : $Rel = \varnothing$
- Dans les contextes C5 à C8, il fait soleil : l'entité *environnement extérieur* entretient la relation *illumine* avec les autres entités du bureau : $Rel = \{illumine\}$;

Les contextes du réseau ayant été identifiés, calculons la cardinalité de Rc en appliquant l'équation (7) : Card (Rc) = 8 puique Card($Roles$) = 2 et Card($Relations$) = 1. Passons en revue chacun des huit contextes obtenus.

La figure 5 exprime de manière graphique l'ensemble des contextes possibles avec la notation suivante : un rectangle tracé en pointillé à fond gris, aux bords arrondis et étiqueté d'une chaîne de caractères, représente un contexte. La chaîne de caractères est un nom unique de contexte. Un cercle à fond blanc désigne une instance d'entité repérée par son nom. Un rôle est dénoté par une étiquette entourée d'un rectangle relié par une double flèche à l'entité (ou aux entités) jouant ce rôle. Les relations sont représentées par des flèches en pointillé décorées d'un nom de relation. Les relations symétriques sont représentées par des doubles flèches. La figure 6 montre les enchaînements entre les contextes de notre exemple.

Les contextes C1 et C5 correspondent à la présence d'utilisateur(s) et de végétaux chlorophylliens. Ils diffèrent par l'existence de relations *illumine* entre l'*environnement extérieur* et les autres entités (*bureau*, *humain* et *plante verte*). Dans C1 et C5, les *humains* jouent le rôle *d'occupants* et les

Figure 5
Représentation du réseau de contextes de l'exemple du gestionnaire de lumière.
Ce réseaux de contextes est composé de huit contextes différents (C1 à C8). Le schéma représente aussi les entités, les rôles et les relations entre les entités pour chacun des huit contextes.



plantes vertes jouent le rôle de *végétaux*. Les entités *bureau* et *environnement extérieur* sont présentes sans jouer de rôle. C5 est identique à C1 à ceci près que *l'environnement extérieur illumine* les autres entités du contexte. Lorsque, dans le contexte C1, le système détecte la présence de soleil, le système provoquera l'ouverture des rideaux et C5 deviendra le contexte courant. En l'absence de soleil, le système allumera le plafonnier en sorte que les *occupants* puissent travailler dans de bonnes conditions.

Dans les contextes C2 et C6, il n'existe pas de *plantes vertes*. Le rôle *végétaux* n'est donc pas assuré. Comme précédemment, si dans le contexte C2, le système détecte la présence de soleil, le système provoquera l'ouverture des rideaux et C6 deviendra le contexte courant. En l'absence de soleil, on reste dans C2, mais le plafonnier sera allumé.

Les contextes C3 et C7 correspondent aux cas où le bureau ne contient que des *plantes vertes*. Si, en C3, le système de gestion détecte la présence de soleil, il provoquera l'ouverture des rideaux, mais en l'absence de soleil, il maintiendra la lampe éteinte et les rideaux fermés.

En C4 et C8, il n'y a ni *occupant*, ni *plante verte*. En C4, si le système détecte la présence de soleil, les rideaux seront maintenus fermés (par sécurité) et la lampe sera éteinte (par économie d'énergie). En C8, les rideaux seront fermés et la lumière éteinte. Le contexte courant deviendra C4.

L'existence de C4 et C8 tient à la modélisation de l'entité bureau. Si l'on considère que cette entité n'est pas pertinente pour l'application, elle peut être supprimée. Alors, C4 et C8 sont confondus et nous obtenons un réseau à 7 nœuds.

Ces huit contextes permettent de gérer convenablement les conditions lumineuses du bureau. Considérons maintenant des exceptions. Par exemple, un arbre, qui est un élément extérieur, peut faire de l'ombre sur une partie du bureau. Supposons que le contexte courant soit C5, C6 ou C7 (les entités du bureau sont éclairées par le soleil). Le système détecte la présence d'ombre dans le bureau :

- Soit l'ombre couvre l'ensemble des entités du contexte : il y a changement de contexte car la relation *illumine* disparaît.
- Soit l'ombre ne touche aucune entité du contexte : il n'y a pas de changement de contexte.
- Soit l'ombre couvre un sous-ensemble des entités du contexte : par exemple en C5, l'utilisateur rentre dans l'ombre, mais la plante reste au soleil. Les rôles restent assurés, la relation *illumine* n'est plus instanciée entre l'extérieur et l'humain, mais elle existe entre l'extérieur et les autres entités du bureau. D'après notre définition, le contexte reste C5. Et pourtant, les conditions lumineuses dans C5 ne sont plus les mêmes. La notion de situation permet d'exprimer cette différence.

3.4. CONTEXTE ET SITUATION :

Un contexte $C = (R,Rel)$ est un réseau de situations tel que toutes les situations de C partagent les mêmes ensembles de rôles R et de relations Rel où $R \subseteq Roles$ et $Rel \subseteq Relations$.

Au sein d'un contexte $C = (R,Rel)$, une situation S est définie sur trois ensemble (Ent, $AssoRoEnt$, $AssoRelEnt$) et un prédicat $estPresent$:

- Ent est l'ensemble des entités présentes dans S, avec $Ent \subseteq Entites$,

$$\forall(e \in Ent) \,/\, ((e \in Entite) \wedge estPresent(e)) \qquad (8)$$

où le prédicat $estPresent(e)$ est vrai si et seulement si l'entité e est présente dans S.

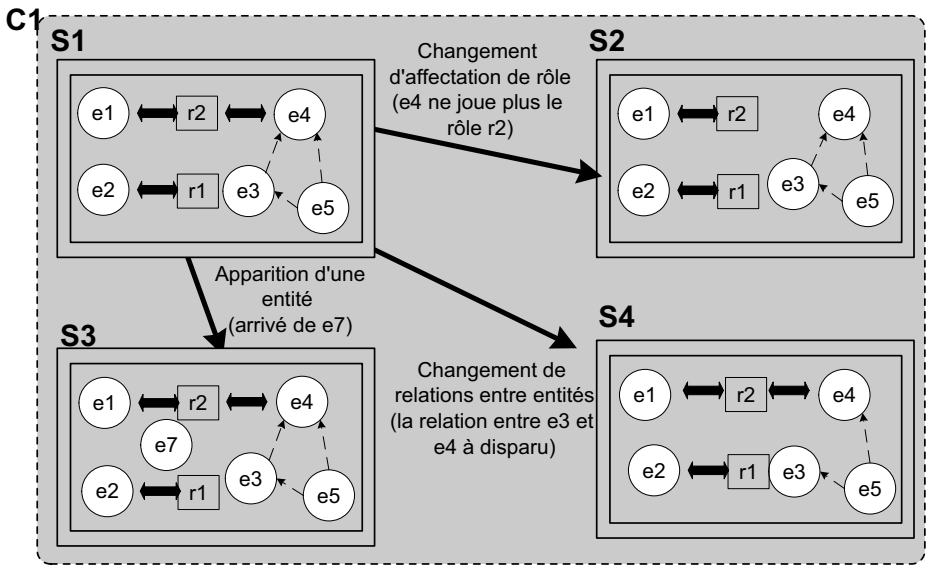
- $AssoRoEnt$ est l'ensemble des associations entre les rôles de R et les entités de Ent. Rappelons que chaque rôle r_i est joué par au moins une entité et que chaque entité joue zéro, un ou plusieurs rôles,

$$\forall(a \in AssoRoEnt), \exists(e \in Ent \wedge r \in R) \,/\, a = (e,r) \wedge joueRôle\ (e,r) \qquad (9)$$

- $AssoRelEnt$ est l'ensemble des associations entre les relations de Rel et les entités de Ent.

$$\forall(b \in AssoRelEnt), \exists(e_1 \ldots e_j \in Ent \wedge rel \in Rel \wedge arity(rel) = j\ ) \,/\, b = (rel, e_1, \ldots, e_j) \wedge rel(e_1, \ldots, e_j) \qquad (10)$$

Figure 6
Changements de situation.
Le contexte C1 comprend quatre situations (S1 à S4). Toutes les situations ont en commun les même ensemble de rôles et de relations.



Au sein d'un contexte $C = (R,Rel)$, il y a changement de situation (figure 6) si l'une au moins des conditions suivantes est remplie :

- l'ensemble *Ent* des entités change : apparition et/ou disparition d'entité(s),
- l'ensemble *AssoRoEnt* des associations entre les rôles et les entités change : apparition et/ou disparition d'association(s) rôle-entité,
- l'ensemble *AssoRelEnt* des associations entre les relations et les entités change : apparition et/ou disparition d'association(s) relation-entités.

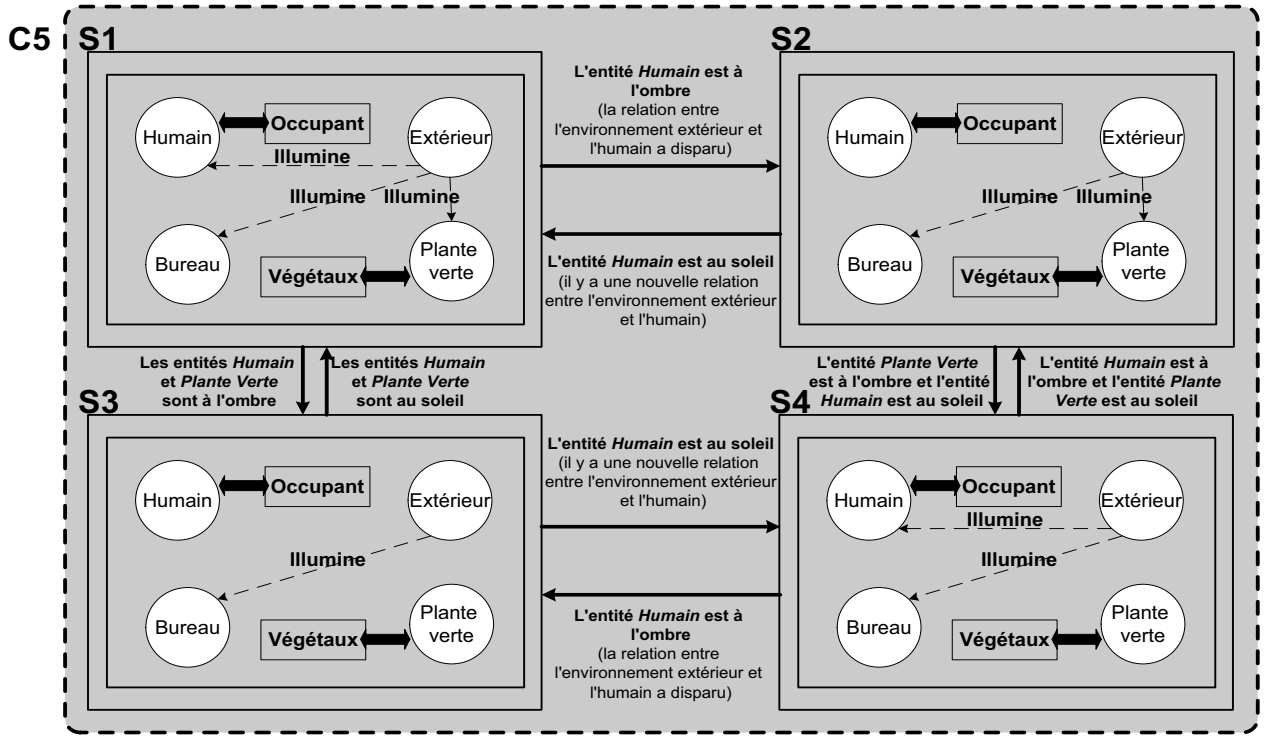
La figure 6 illustre graphiquement les transitions entre situations. Une situation est représentée par un double rectangle étiqueté avec le nom de la situation.

De manière plus formelle, une situation S se définit comme suit :

Soit $C = (R,Rel)$, $R \subseteq Roles$, $Rel \subseteq Relations$ alors :

$$\forall (S \in C = (R, Rel)), \exists Ent, AssoRoEnt, AssoRelEnt \,/\, (S = Ent, AssoRoEnt, AssoRelEnt) \Leftrightarrow (Ent \in \wp(Entites)) \wedge (AssoRoEnt = f(Ent, R)) \wedge (AssoRelEnt = g(Ent, Rel)) \quad (11)$$

Figure 7
Les situations du contexte C5 du gestionnaire de lumière.
Le contexte C5 comprend quatre situations (S1 à S4). Dans ce contexte, les situations se différencient en fonction des associations relations - entités. Par exemple, pour la situation S3, *AssoRelEnt* = {(*illumine*,*extérieur*,*bureau*)} alors que pour la situation S4, *AssoRelEnt* = {(*illumine*,*extérieur*,*bureau*),(*illumine*,*extérieur*,*humain*)}.



où $\wp(Entites)$ est l'ensemble des parties de *Entites*, *f* est la fonction d'association entre une entité *e* et un rôle *r* telle que «e joue r» et *g* la fonction d'association entre une ou plusieurs entités $e_1 ... e_j$ et une relation *rel* telle que $rel(e_1, ..., e_j)$ est vrai.

De plus, la propriété suivante est vérifiée quelle que soit la situation *S* :

$$\begin{gathered}\forall((S_1 = Ent_1, AssoRoEnt_1, AssoRelEnt_1) \in C),\\ \forall((S_2 = Ent_2, AssoRoEnt_2, AssoRelEnt_2) \in C) /\\ (S_1 = S_2) \Leftrightarrow (Ent_1 = Ent_2) \wedge (AssoRoEnt_1 = AssoRoEnt_2)\\ \wedge (AssoRelEnt_1 = AssoRelEnt_2)\end{gathered} \quad (12)$$

Pour revenir à l'exemple du gestionnaire de lumière, la notion de situation permet de différencier le fait que tout ou partie du bureau est sous le soleil. La figure 7 montre, pour le contexte C5, différentes situations possibles.

Jusqu'ici, nous avons appréhendé les ensembles *Roles*, *Relations* et *Entites* d'une manière abstraite et générique. Dans la section qui suit, typologie des contextes, nous optons pour une perspective plus pragmatique.

4. Typologie des Contextes

Cette typologie distingue trois points de vue combinés aux deux étapes essentielles du processus de développement d'un système : la conception et l'exécution.

4.1. TROIS POINTS DE VUE SUR LES CONTEXTES

L'activité utilisateur, on le rappelle, se déroule dans un réseau de contextes défini sur les ensembles *Roles*, *Relations* et *Entites*. Ces ensembles s'instancient différemment selon la perspective adoptée. En IHM, toute notion s'analyse selon deux perspectives : le point de vue système et le point de vue utilisateur, l'un et l'autre s'inscrivant dans une perspective globale du possible. Appliquant le raisonnement au réseau de contextes, on obtient les contextes brut, système et utilisateur :

- Le contexte brut (global) est défini sur les ensembles *RolesBrut*, *RelationsBrut* et *EntitesBrut* qui désignent respectivement tous les rôles, relations et entités possibles pour les activités considérées d'un utilisateur donné.
- Le contexte système correspond au réseau de contextes définis sur les ensembles *RolesSysteme*, *RelationsSysteme* et *EntitesSysteme*

qui représentent respectivement les rôles, relations et entités observées par le système pour les activités de l'utilisateur en question.

- Le contexte utilisateur est le réseau de contextes définis sur les ensembles *RolesUtilisateur*, *RelationsUtilisateur* et *EntitesUtilisateur* qui recouvrent les rôles, relations et entités que l'utilisateur met en jeu dans ses activités.

La figure 8 traduit les relations entre ces trois types de contextes (brut, système et utilisateur). Par définition, les contextes système et utilisateur sont inclus dans le contexte brut, soit :

$$\forall RolesUtilisateur \wedge \forall RolesSysteme, \quad RolesUtilisateur \subseteq RolesBrut \wedge \quad RolesSysteme \subseteq RolesBrut \tag{13}$$

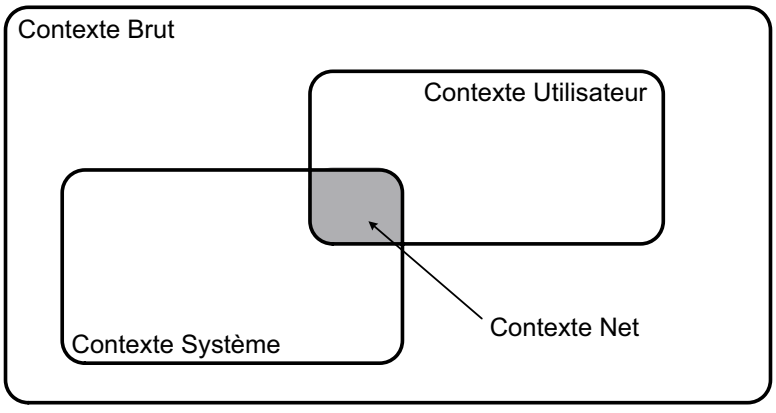
$$\forall RelationsUtilisateur \wedge \forall RelationsSysteme, \quad RelationsUtilisateur \subseteq RelationsBrut \wedge \quad RelationsSysteme \subseteq RelationsBrut \tag{14}$$

$$\forall EntitesUtilisateur \wedge \forall EntitesSysteme, \quad EntitesUtilisateur \subseteq EntitesBrut \wedge \quad EntitesSysteme \subseteq EntitesBrut \tag{15}$$

Mais les contextes système et utilisateur ne se recouvrent pas nécessairement : l'utilisateur a un vécu (les variables qu'il modélise mentalement) que le système ignore (à tort ou à raison). Inversement, le système dispose de variables dont l'utilisateur n'est pas averti.

Le contexte net correspond à l'intersection des contextes système et utilisateur. Si l'on se réfère au modèle formel du contexte, il désigne le

Figure 8
Points de vue du contexte.



sous-réseau de contextes que les acteurs système et utilisateur partagent. Nous verrons plus loin l'intérêt du contexte net.

$$\forall(RolesUtilisateur \wedge RolesSysteme), \quad RolesNet = (RolesUtilisateur \cap RolesSysteme) \tag{16}$$

$$\forall(RelationsUtilisateur \wedge RelationsSysteme), \quad RelationsNet = (RelationsUtilisateur \cap RelationsSysteme) \tag{17}$$

$$\forall(EntitesUtilisateur \wedge EntitesSysteme), \quad EntitesNet = (EntitesUtilisateur \cap EntitesSysteme) \tag{18}$$

Le *Contexte d'Interaction* de l'utilisateur U est l'union des contextes nets pour cet utilisateur U quelles que soient ses activités :

$$RolesInteraction = \bigcup_{Actitites_u} RolesNet \tag{19}$$

$$RelationsInteraction = \bigcup_{Actitites_u} RelationsNet \tag{20}$$

$$EntitesInteraction = \bigcup_{Actitites_u} ENtitesNet \tag{21}$$

4.2. DEUX ÉTAPES DE DEVELOPPEMENT 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*. À l'exécution, les *effectifs* ne sont pas nécessairement conformes aux *souhaitables* : cas des pannes, cas de détournement de la technologie par les utilisateurs, cas de non conformité entre les spécifications et la mise en œuvre,etc.

Pour cette raison, il convient de distinguer :

- *Pour le contexte système* : le contexte captable et le contexte capté. Le contexte captable dénote le contexte système tel qu'il a été défini à l'étape d'analyse/conception pour l'activité de l'utilisateur. 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 ou d'autres capteurs non prévus peuvent être disponibles.
- *Pour le contexte utilisateur* : le contexte utilisable et le contexte utilisé. En analyse amont, les concepteurs ont défini le contexte utilisable comme l'ensemble des facteurs auxquels l'utilisateur risque de faire appel dans la conduite de son activité. Dans les faits, il est possible que certains de ces facteurs n'aient pas eu d'impact sur l'activité ou que les concepteurs aient oublié des facteurs pertinents : le contexte utilisé peut donc être différent du contexte utilisable.

- *À 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 captables et utilisables, et des contextes captés et utilisés.

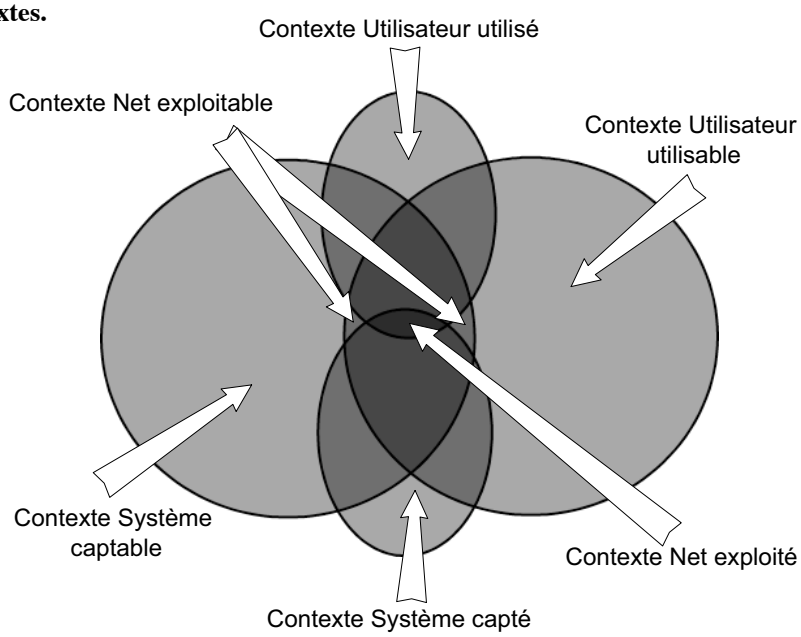
4.3. UTILITÉ DE LA TYPOLOGIE DES CONTEXTES

La figure 9 synthétise la typologie des contextes.

La distinction entre les contextes système et utilisateur est utile pour raisonner sur la tolérance des écarts entre ce que le système fera et ce que l'utilisateur croira que le système fera (si l'on raisonne pendant la conception) ou, si l'on considère l'exécution, ce que le système fait et ce que l'utilisateur croit que le système fait. Si le contexte net est vide, c'est que le système et l'utilisateur ne partageront (ou ne partagent) aucune information de nature contextuelle. Si ce constat s'impose à l'étape de conception, mieux vaut revoir l'analyse. Si ce constat survient à l'exécution, il est clair que le système ne sera d'aucune aide pour l'utilisateur en tant que système ambiant. L'évolution du contexte net à l'exécution constitue également un élément d'analyse intéressant sur l'apport du système au regard des attentes de l'utilisateur.

La distinction entre les contextes en -é et les contextes en -able permet de comparer les contextes entre ce qu'ils sont et ce qu'ils sont censés être. En particulier, le taux de recouvrement entre un contexte souhaitable (qu'il s'agisse de contexte système, utilisateur, ou net) et son correspondant effectif est un indicateur de conformité entre l'étape de conception et la mise en œuvre, voire la robustesse de cette mise en œuvre.

Figure 9
Représentation de la typologie des contextes.



Cette typologie des contextes qui envisage les ensembles *Roles*, *Relations*, et *Entites* sous un angle plus pragmatique est maintenant complétée par une typologie des entités et leurs attributs spatio-temporels, la localisation et le temps étant centraux en informatique ambiante.

5. *Classification des entités*

La diversité des entités susceptibles d'intervenir dans l'expression du contexte est une cause de difficulté pour le concepteur. Une classification devrait faciliter sa tâche d'analyse. Cette section est illustrée en reprenant l'exemple du gestionnaire de lumière.

5.1. TYPOLOGIE DES ENTITES

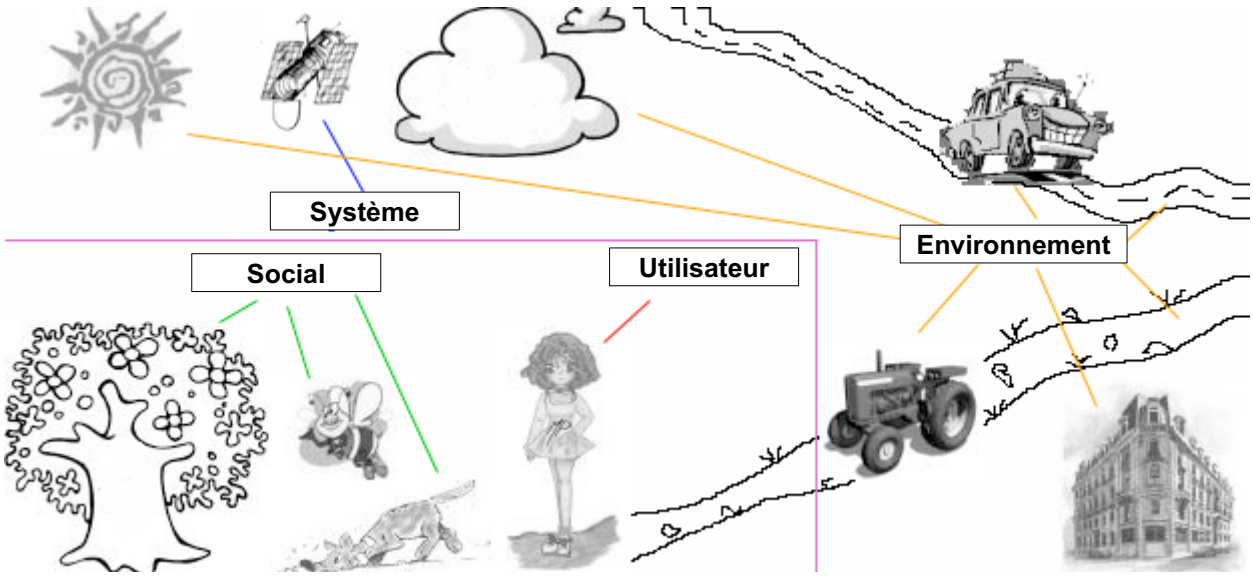
Comme le montre le diagramme UML de la figure 1, les entités d'un espace contextuel se répartissent en deux catégories :

- Les êtres vivants (les humains, mais aussi les animaux et les végétaux) qui, à leur tour, forment deux sous-groupes : les utilisateurs et l'environnement social.
- Les entités inertes (objets, lieux...) qui constituent également deux sous-groupes : le système et l'environnement physique.

La figure 10 illustre cette typologie à gros grain :

- Le groupe *Utilisateur* désigne l'ensemble des être animés considérés comme utilisateurs du système.

Figure 10
Classification des entités.



- Le groupe *Social* contient l'ensemble des êtres vivants autres que l'utilisateur.
- Le groupe *Physique* est l'ensemble des objets n'ayant aucune capacité informatique.
- Le groupe *Système* couvre l'ensemble des entités inertes ayant des capacités de traitement ou de stockage de données informatiques (numériques ou analogiques). Dans ce groupe, on distingue les entités informatiques (technologiques) des entités augmentées. Les entités informatiques ont des capacités de calcul, de stockage, de communication endogènes (c'est-à-dire qui leur sont propres). Les ordinateurs, les écrans, les baladeurs numériques et les clefs USB sont des exemples d'entités informatiques du groupe Système. Les entités augmentées [Mackay 96] sont définies ici comme des entités du groupe physique couplées (au sens de [Barralon 04]) à des entités du groupe Système (que celles-ci soient des entités informatiques ou des entités augmentées).

L'ancrage des entités avec le monde réel s'effectue grâce à la notion d'observable : une entité est un regroupement d'observables (voir la définition de la section 3.1).

Par la mesure d'observables et en les regroupant, le système en déduit les attributs d'entités tels que l'identification, la taille, le poids, la forme, la couleur, l'activité (lorsqu'il s'agit d'entité utilisateur). Parmi l'ensemble des attributs d'entité, les concepts d'espace et de temps, communs à la plupart des modèles de l'état de l'art, méritent une attention particulière.

5.2. L'ESPACE

Comme nous le voyons sur la figure 11, la notion d'espace est représentée par le concept de *Where*. Un *Where* correspond à la classe mère unifiant les concepts de *SymbolicLocation*, *Region* et *PhysicalLocation* [Coutaz 02b].

Une *SymbolicLocation* est une entité, fixe ou mobile, qui peut contenir d'autres entités telles que des personnes, des objets ou d'autres *SymbolicLocations*. Par exemple, le *bureau B204 du clips* et la *voiture de Gaëtan* sont des *SymbolicLocations*. Chaque *SymbolicLocation* couvre une région de l'espace.

Une *Region* est une zone fixe de l'espace limitée par des frontières (*SpatialBounds*). Une *Region* peut contenir ou couper d'autres *Regions*. Chaque *Region* possède au moins une *PhysicalLocation*. Souvent le centre d'une région est associé à sa *PhysicalLocation*. La notion de centre est assez floue. il s'agit souvent du barycentre de la zone considérée mais ce n'est pas toujours le cas. Par exemple, la *SymbolicLocation* France couvre une zone géographique délimitée par les frontières du pays et la ville de Paris est souvent considérée comme le point central de la France.

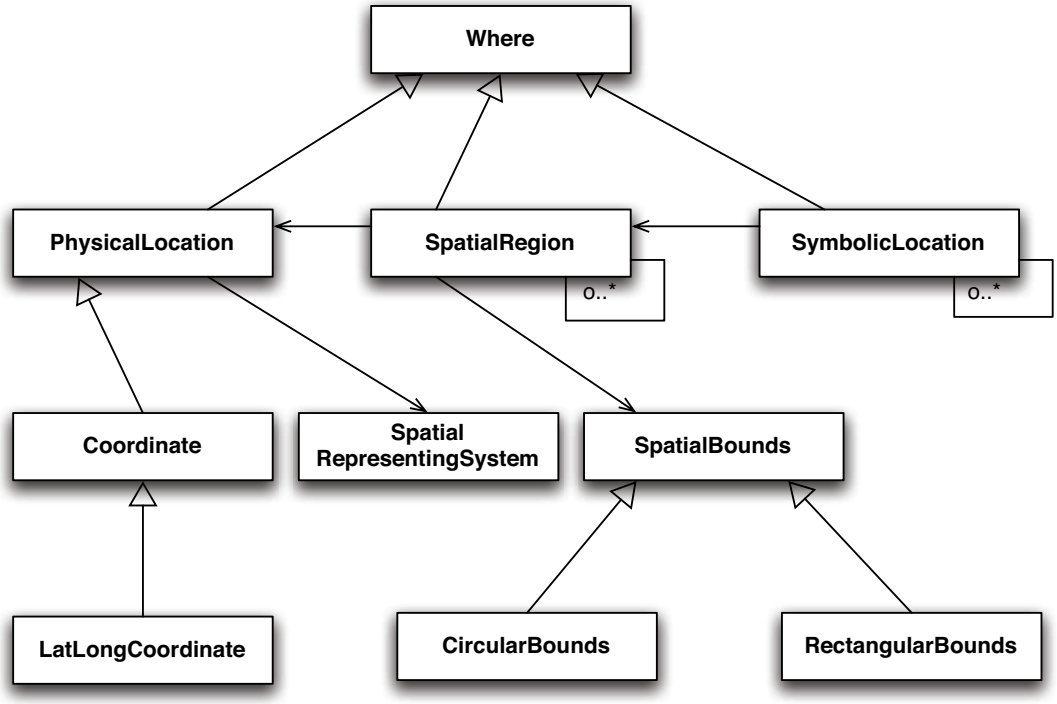


Figure 11
Diagramme de classes représentant la notion d'espace.

Une *SpatialBounds* est un contour en 2 ou 3 dimensions délimitant une zone géographique (*Region*). Par exemple, une *CircularBounds* est une frontière circulaire en 2 dimensions et une *RectangularBounds* est une frontière rectangulaire toujours en 2 dimensions.

Une *PhysicalLocation* est un point représenté dans un *SpatialRepresentingSystem*. Par exemple, une coordonnée est une spécialisation d'une *PhysicalLocation* représentant un point dans un système de coordonnées géographiques en deux ou trois dimensions.

Un exemple de système de coordonnées géographiques en deux dimensions, est le système de coordonnées terrestres qui découpe la terre à l'aide des latitudes et longitudes. Une coordonnée dans ce système est une *LatLongCoordinate*.

5.3. LE TEMPS

La figure 11 représente la notion de temps selon les mêmes principes de modélisation que pour la localisation. Le concept de temps est représenté par la classe mère *When* qui unifie les concepts de *SymbolicTime*, *TemporalRegion* et de *Time*.

Un *SymbolicTime* est un instant, une période ou un ensemble de périodes temporelles. Un *SymbolicTime* peut être composé par plusieurs autres *SymbolicTimes* et chaque *SymbolicTime* possède une *TemporalRegion*. Par exemple dans la phrase *Il est l'heure de manger*, la référence

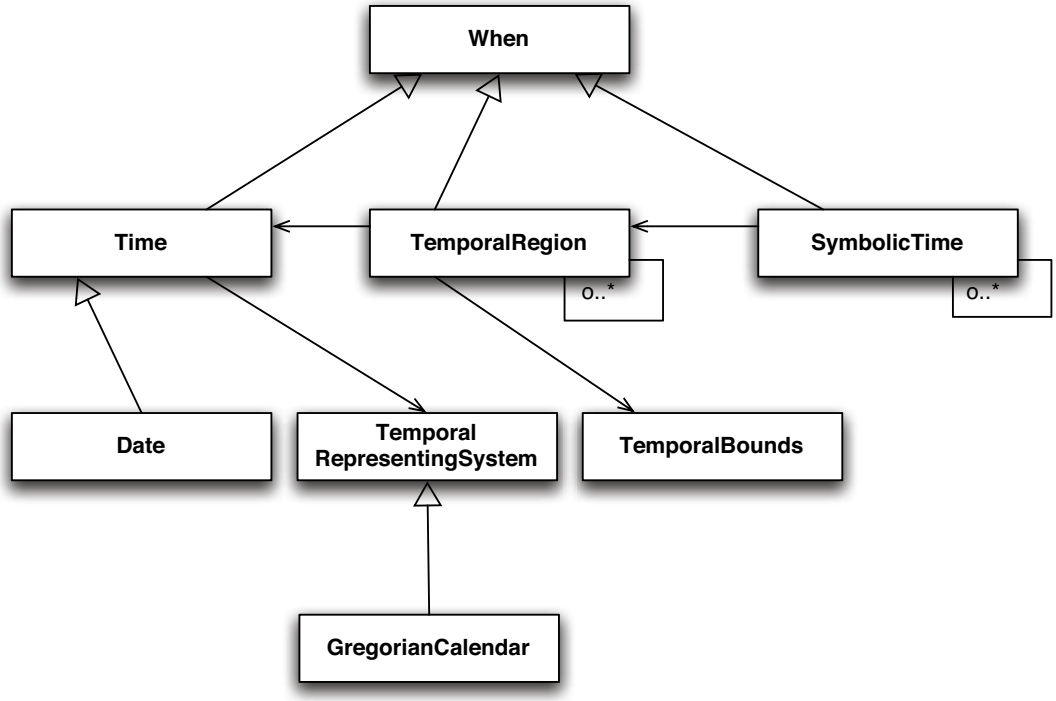


Figure 12
Diagramme de classes représentant la notion

heure de manger est un *SymbolicTime*. Ce *SymbolicTime* couvre une *TemporalRegion* allant de 11h30 à 13h30.

Une *TemporalRegion* est une période continue de temps délimitée par une frontière (*TemporalBounds*). Par exemple, un jour est un *SymbolicTime*, possédant une *TemporalRegion* de vingt quatre heures correspondant à la révolution complète de la terre sur son axe.

Un *Time* est un point dans le temps exprimé à l'aide d'un système de représentation particulier (*TemporalRepresentingSystem*). Le plus célèbre des systèmes de représentation du temps est le calendrier grégorien (*GregorianCalendar*). Par exemple le 19 juillet 1977 est une *Date*, c'est-à-dire un *Time* représenté à l'aide du *GregorianCalendar*.

5.4. ILLUSTRATION : LE GESTIONNAIRE DE LUMIÈRE

Reprenons notre gestionnaire de lumière. La figure 13 présente les 4 entités de l'ensemble *Entites* avec les attributs (observables) pertinents pour notre problème. Chaque observable est qualifié d'un type. Celui-ci correspond au type de la valeur de l'observable, c'est-à-dire à l'ensemble des valeurs que peut prendre l'attribut observé.

Pour *l'humain*, il est nécessaire de savoir s'il est dans la pièce, de même que pour la *plante verte*. De plus, la *plante verte* est caractérisée par une durée d'ensoleillement reçu au cours de la journée. En ce qui concerne le *bureau* la liste des entités qui y sont présentes permettra de définir le

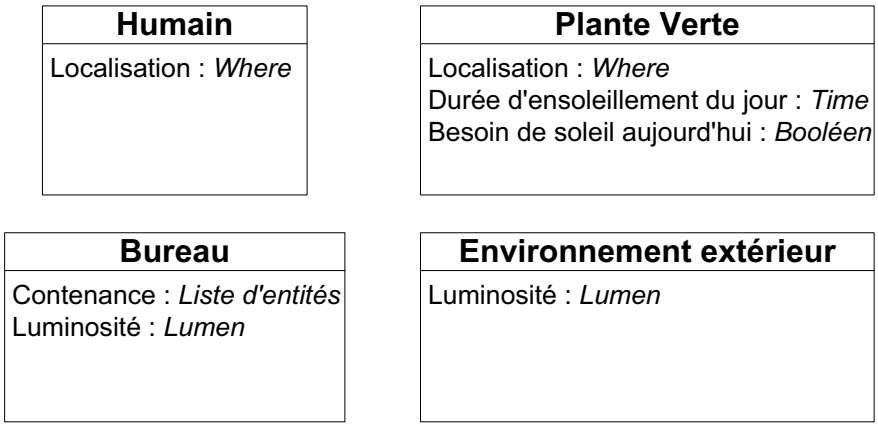


Figure 13
Les entités du gestionnaire de lumière avec leurs attributs.

contexte approprié. Enfin, un capteur de lumière indiquera s'il y a du soleil dans *l'environnement extérieur*.

On notera que la *présence* d'entités dans un bureau est modélisée de deux manières : D'une part, par l'attribut *contenance* de l'entité *bureau* et d'autre part, avec attributs *localisation* des entités *humain* et *plante verte*. Cette redondance peut s'avérer utile : en phase de modélisation, aucun choix technologique n'est, en principe, connu. Dans notre cas, le concepteur ignore si les bureaux seront équipés de caméra et/ou si les entités *humain* (et *plante verte*) seront équipées d'un système personnel de localisation.

Le gestionnaire d'ambiance lumineuse a servi de cas d'école pour illustrer les concepts du cadre conceptuel. Avec la section qui suit, on montre comment s'en servir.

6. Mode d'emploi du modèle

Dans cette section, on trouvera comment construire un réseau de contextes et de situations à partir d'une description informelle des finalités d'un système interactif. Un système réel, ContextNotePad, illustre le processus.

6.1. FINALITÉ DE CONTEXTNOTEPAD

ContextNotePad offre les services usuels de bloc note et mémo, mais adapte son Interface Homme-Machine en fonction du contexte physique [Schmidt 00]. Le système, qui fonctionne sur Palm Pilot, est sensible aux mouvements de faible amplitude, au déplacement et à la lumière.

Figure 14
Différents états de l'application ContextNotePad.
a) normal, b) texte de grande taille, c) rétro-éclairage, d) confidentialité.

La figure 14 montre les quatre cas d'adaptation :

- *Normal (figure 14a)* : l'écran s'allume dès que l'utilisateur le prend dans la main. Inversement, il s'éteint lorsqu'il est posé sur une surface immobile.
- *Texte de grande Taille (figure 14b)* : Lorsque l'utilisateur tient le PDA en marchant, la taille des caractères est augmentée afin de faciliter la lecture. Le système rétablit la taille normale de la police dès qu'il détecte la disparition du mouvement.
- *Rétro-éclairage (figure 14c)* : Si la lumière ambiante baisse de manière significative, l'écran passe automatiquement en mode rétro-éclairé facilitant ainsi la lecture dans un milieu sombre.
- *Confidentialité (figure 14d)* : Si le dispositif n'est pas utilisé et qu'il détecte la présence d'étrangers à proximité, l'écran est masqué pour assurer la confidentialité.

6.2. LE PROCESSUS DE MODÉLISATION DU CONTEXTE

Le processus de modélisation du contexte procède selon les étapes suivantes :

1. Instancier les trois ensembles de définition Roles, Relations, Entites pertinents pour le système cible. Pour l'ensemble Entites, structurer l'espace selon la typologie de la section 4. Identifier et analyser les redondances.
2. Calculer le nombre de nœuds (ou contextes) du réseau (formule 7 de la section 3.2).
3. Réduire la combinatoire du réseau par factorisation ou élimination de contextes. On procède par factorisation lorsque plusieurs contextes

impliquent le même comportement système. Il y a élimination des contextes pour lesquels le système ne change pas d'état.

4 Affinement des contextes en réseau de situations et de même, factorisation et suppression éventielles de situations.

6.3. APPLICATION AU CAS DE CONTEXTNOTEPAD

1. Instanciation de Roles, Relations, Entites. On propose de considérer :

- Les entités humain, pda et environnement extérieur puisque le système met en jeu des utilisateurs ainsi qu'un dispositif d'interaction sensible à l'environnement : le PDA. Alors *Entites* = {*humain*, *pda*, *envExt*} ;
- Le rôle de propriétaire pour tenir compte des conditions de confidentialité : *Roles* = {*propriétaire*} ;
- Les relations illumine, enMouvement et estTenu sachant que l'environnement illumine le PDA, que le PDA peut être en mouvement et qu'un humain peut prendre le PDA dans sa main : (*Relations* = {*illumine*, *enMouvement*, *estTenu*}.

2. Cardinalité du réseau de contextes. Défini sur 1 rôle et 3 relations, le réseau de contextes de ContextNotePad comporte, en théorie, 16 nœuds : 8 nœuds correspondant aux contextes dans lesquels la relation estTenu n'est pas assurée, et 8 contextes où, au contraire, cette relation est assurée.

3. Réduction de la combinatoire. Ignorons les 8 contextes pour lesquels l'état du système est invariable. Il s'agit des contextes dans lesquels le PDA doit rester éteint. Il reste donc 8 contextes qui correspondent au cas où la relation estTenu est assurée (se reporter à la figure 15).

Les contextes C1 à C4 correspondent au cas où le rôle de propriétaire n'est pas joué alors que le PDA est tenu. Dans ces contextes, le système doit activer le mode confidentialité que l'environnement extérieur illumine ou non le PDA.

Les contextes C5 à C8 couvrent les cas où le rôle de propriétaire est assuré alors que le PDA estTenu dans la main. Ces contextes doivent être affinés en un réseau de situations.

4. Affinement des contextes en réseau de situations

Les contextes C5 à C8 sont structurés en un réseau de deux situations pour gérer la confidentialité :

- Dans la situation S1, un humain joue le rôle du propriétaire du PDA, mais le PDA n'est pas tenu par son propriétaire.
- Dans la situation S2, l'humain joue le rôle de propriétaire et tient le PDA.

Les conditions de passage entre S1 et S2 tiennent au changement d'association rôle-entité : quels que soient les contextes C5 à C8, on passe de S1 à S2 lorsque le rôle de propriétaire passe de l'humain qui ne tient pas le PDA à l'humain qui tient le PDA. Inversement, on passe de S2 à S1

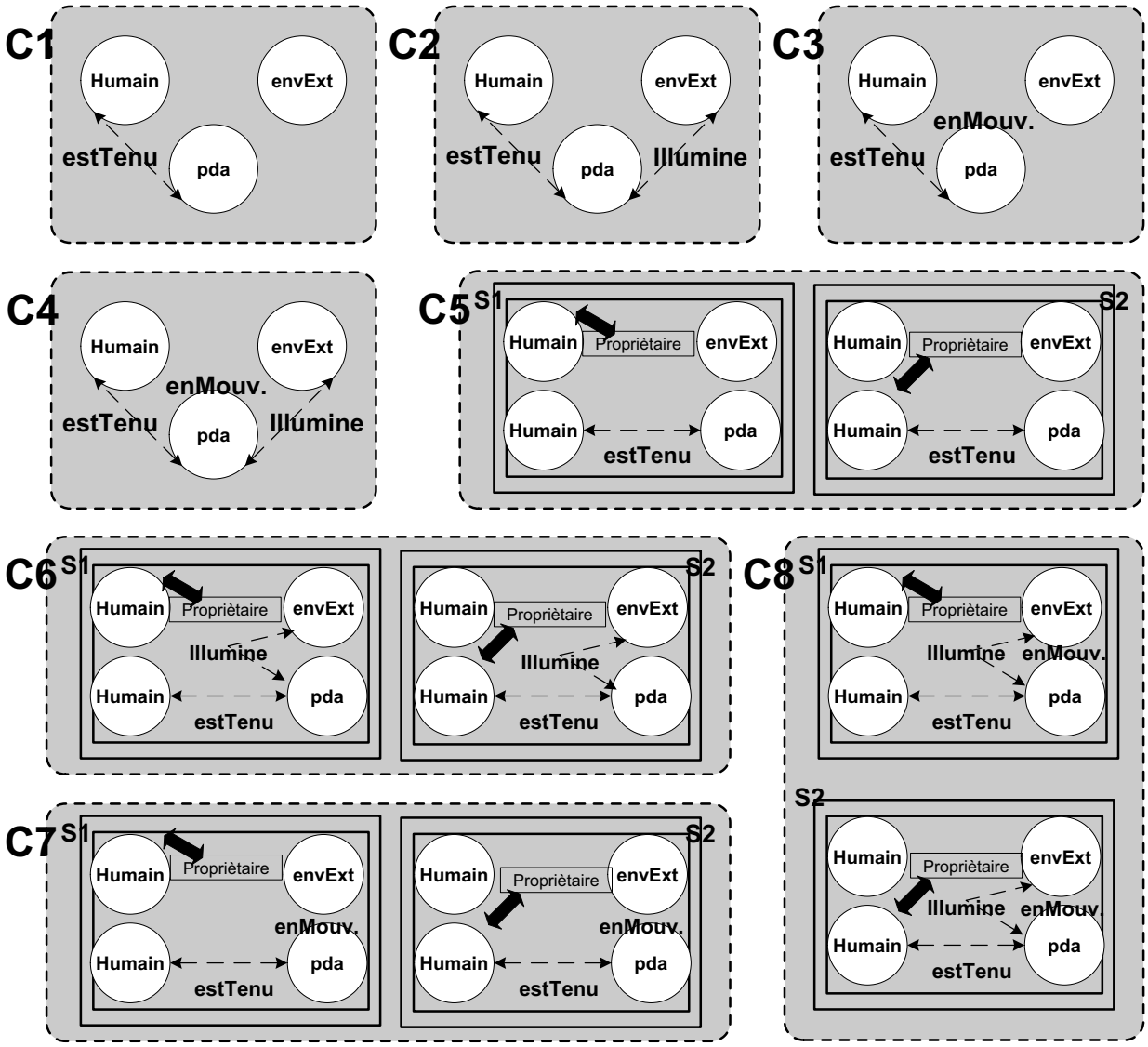


Figure 15
Représentation des contextes et des situations significatifs pour le ContextNotePad.
Les contextes C1 à C4 ne sont pas composés de situations car celles ci n'auraient pas apportées d'informations suplémentaires. Les situations S1 et S2 des contextes C5 à C8 se différencient par les différentes associations du rôle Propriétaire et avec les entités Humain.

lorsque le rôle de propriétaire passe de l'humain qui tient le PDA à l'humain qui ne tient pas le PDA. En S1, le PDA doit passer en mode confidentialité. En S2, le comportement dépend du contexte :

- En C5.S2, le PDA est tenu par son propriétaire, mais l'environnement ne l'illumine pas : l'écran passe en mode rétro-éclairé.
- En C6.S2, le mode normal est activé : l'environnement extérieur illumine le PDA.
- En C7.S2, le PDA est tenu par son propriétaire. Il est en mouvement, mais n'est pas illuminé par l'environnement : la police de caractères est agrandie et l'écran doit être rétro-éclairé.

- En C8.S2, le PDA est tenu par son propriétaire. Il est en mouvement et se trouve illuminé par l'environnement : la police de caractères est agrandie.

Le processus préconisé ici conduit à une analyse exhaustive du problème. Il n'y a pas d'oubli possible. Toutefois, la combinatoire du nombre de contextes et de situations peut être prohibitive. Or, tout réseau de contextes et de situations peut se décomposer en sous-réseaux pour lesquels le système doit observer le même comportement. Ces sous-réseau peuvent être factorisés ou éliminés. Dans l'exemple de Context-NotePad, le système conserve le même état (il reste éteint) pour tous les contextes pour lesquels la relation *estTenu* n'est pas assurée. Ces contextes n'ont donc pas été considérés dans la modélisation.

Il nous reste à montrer une autre utilité du cadre conceptuel en tant que moyen de classification des systèmes interactifs en informatique ambiante.

7. Typologie des systèmes interactifs en informatique ambiante

Le contexte se voit véhiculé dans de nombreux systèmes interactifs de l'informatique ambiante. Certains, comme les systèmes d'aide à la navigation à base de GPS, sont disponibles sur le marché, mais la plupart relèvent du démonstrateur de concepts. Cette section présente une analyse comparative synthétique des nombreuses réalisations de l'état de l'art à partir des éléments du cadre conceptuel.

7.1. ELÉMENTS TAXINOMIQUES

La classification proposée repose sur les deux dimensions suivantes :

- La finalité du contexte système.
- Les classes d'entités observées qui permettent, par interprétation de leur état d'atteindre la finalité en question.

Les finalités possibles sont les suivantes (inspirées de [Dey 01]):

- *Présentation d'information et de services* : le système présentent des informations contextuelles ou utilise le contexte pour proposer à l'utilisateur un ensemble d'actions, d'activités ou de choix appropriés.
- *Exécution automatique de services pour le compte de l'utilisateur* : il s'agit principalement de migration vers le système de tâches jugées peu gratifiantes par l'utilisateur.
- *Auto-adaptation* : le système se reconfigure pour tenir compte du changement de contexte.

- *Enrichissement d'information par des méta-informations de nature contextuelle* : par exemple la date et le nom de l'endroit associés à une photo. L'idée est ici de construire des applications de type aide-mémoire.

Systèmes interactifs	Présentation	Automatisation	Adaptation	Enrichissement
ContextNotePad			X	
In / Out Board	X			
Context-Aware Office Assistant	X			
DUMMBO Meeting Board		X		X
Cyberguide		X		
CyberDesk		X		
myCampus	X	X		
WatchMe	X			X

Table 1 : Classifications des applications représentatives en fonction de leur finalité.

Comme nous l'avons vu dans la section 4, les entités observées se répartissent dans les classes :
- Utilisateur
- Social
- Système
- Physique

Systèmes interactifs	utilisateur	social	système	physique
ContextNotePad	X		X	X
In / Out Board	X			X
Context-Aware Office Assistant	X	X		
DUMMBO Meeting Board	X			X
Cyberguide	X			
CyberDesk			X	
myCampus	X	X	X	X
WatchMe	X	X		X

Table 2 : Classification des systèmes interactifs présentés en début de chapitre en fonction des catégories d'entités utilisées lors de la modélisation du contexte.

7.2. ILLUSTRATION

Les tableaux 1 et 2 résument de manière synthétique une analyse comparative de quelques exemples de systèmes interactifs en informatique ambiante.

In/Out Board

Comme le montre la figure 16, In/Out Board se présente sous la forme d'un tableau qui indique, pour chaque membre d'une entreprise, sa présence (lumière verte) ou son absence (lumière rouge) avec les heures d'arrivée et de départ [Salber 99]. Ce système relève de la classe présentation d'information et de services. Il observe des entités de classe Utilisateurs et physique (les bureaux).

Context-Aware Office Assistant

Context-Aware Office Assistant est un assistant de secrétaire de bureau [Yan 00] : il annonce les nouveaux arrivants, les fait patienter, leur propose d'autres rendez-vous en consultant leur agenda. Ce système rentre dans les classes de systèmes Présentation d'information contextuelle (annonce des nouveaux arrivants) et exécution automatique de services (prise de rendez-vous). Il observe des entités utilisateur, social (collaboration entre utilisateurs) et système (les agendas électroniques).

DUMMBO Meeting Board

DUMMBO Meeting Board est, au sens du chapitre précédent, une entité augmentée : il est le résultat du couplage d'une entité physique (un tableau blanc) et d'entités informatiques (des services d'enregistrement) [Salber 99] : enregistrement des inscriptions tracées sur le tableau, de même enregistrement des scènes audio-vidéo. La capture des scènes démarre automatiquement dès que deux personnes au moins se trouvent à proximité du tableau. C'est un système de type exécution automatique de services (les enregistrements) avec observation d'entités utilisateur et physiques (le tableau).

Figure 16
Interface graphique de l'appication In/Out Board
Le point à coté des noms des utilisateurs est vert si l'utilisateur est présent ou rouge s'il est absent.

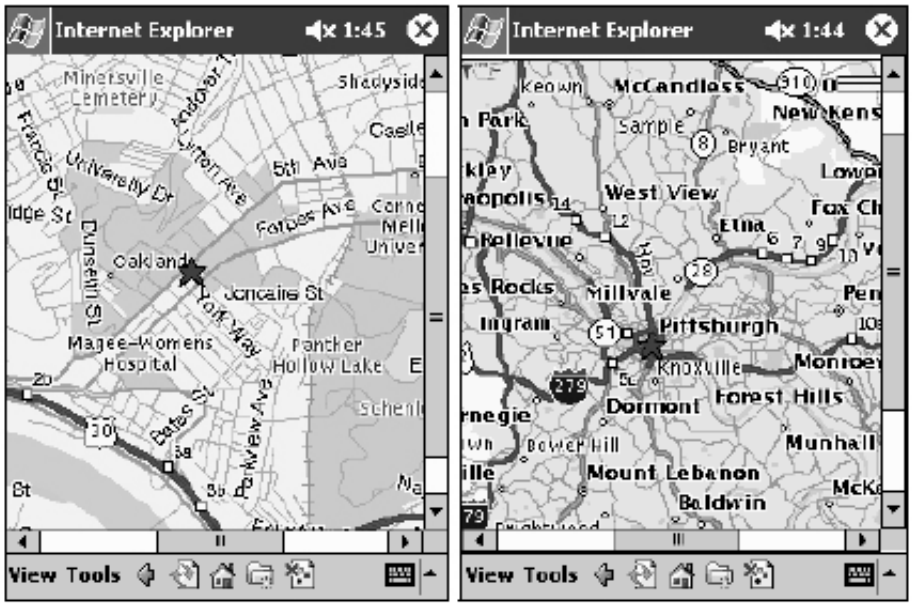
Cyberguide Cyberguide est un guide de navigation électronique qui fonctionne sur PDA. Semblable dans l'esprit au système GUIDE [Cheverst 01], Cyberguide fournit à l'utilisateur sa position et le représente sur une carte augmentée d'informations sur les alentours [Abowd 98]. Il guide le voyageur dans la ville, peut traduire les kanji japonnais filmé dans la rue... Cyberguide est de type exécution automatique de service et observe des entités utilisateur.

CyberDesk CyberDesk utilise le contexte pour faciliter l'intégration de services de bureau comme le courriel et le Web [Abowd 98]. Par exemple, si un message électronique en cours de lecture fait référence à une URL, CyberDesk ouvre automatiquement cette page, mais, pour éviter toute disruption, il place cette page en tâche de fond. Si le message en question est suivi de la lecture d'un autre message, CyberDesk considère que le lien ouvert ne présente pas d'intérêt pour l'utilisateur : la page est refermée. CyberDesk est de type exécution automatique de service.

myCampus myCampus cherche à faciliter la vie quotidienne des étudiants du Campus de Carnegie Mellon [Gandon 04]. Il s'agit d'un système multi-agents où chaque agent assure un service particulier :

L'agent *concierge* indique où prendre son repas en tenant compte des préférences " gastronomiques " de l'utilisateur, de sa position actuelle et des conditions météorologiques.

Figure 17
Exemple de présentation pour le service de cartographie de myCampus.



L'agent *messager* filtre les messages reçus en tenant compte des centres d'intérêt et de la disponibilité du destinataire. Par exemple, "quand je suis occupé, retarder les messages jusqu'à ce que mon activité soit terminée".

Des *post-its virtuels* peuvent être déposés en des lieux physiques réels. Une personne équipée d'un PDA, passant à proximité d'un Post-It dont le contenu est conforme à ses centres d'intérêt pourra récupérer le Post-It. L'utilisateur pourra consulter ultérieurement l'ensemble des Post-Its ainsi récupérés. Ce concept est similaire à la notion de Hearsay [Munro 01] et de GeoNote [Espinoza 01].

Cinéma et météorologie. Ces deux services proposent respectivement une liste des films projetés dans les environs de l'utilisateur, et les prévisions météorologiques.

L'agent *cartographie* permet de localiser des personnes amies à différents niveaux de précision. La figure 17 montre deux exemples de résultats pour un utilisateur cherchant à localiser Marie et Jim. À gauche, le résultat obtenu pour Marie qui autorise d'être localisée avec précision avec l'adresse du lieu où elle se trouve. À droite, le résultat obtenu pour Jim qui limite la précision à la ville la plus proche.

L'agent *assistant de réunion* permet de trouver un créneau libre dans les agendas de deux personnes. Ce service est assez similaire au Context-Aware Office Assistant présenté plus haut.

L'agent *projection de présentation* permet de choisir une présentation PowerPoint pour la projeter sur le vidéoprojecteur le plus proche. Le vidéoprojecteur est alors télé-opérable au moyen du PDA.

WatchMe

WatchMe [Marmasse 04] est un communicateur personnel sensible au contexte de la taille d'un téléphone portable (sans clavier) qui permet rester en contact avec ses amis et sa famille. WatchMe extrait des données de nombreux capteurs (GPS, accelerомètre, microphone ...) pour fournir ces informations aux autres utilisateurs.

8. Synthèse

En synthèse, la notion de contexte ici proposée s'appuie sur les concepts *d'Entites*, de *Relations* entre entités, et de *Roles*. Entités, relations et rôles sont instanciables dans le domaine d'application (ou finalité) choisie. En ce sens, le fondement conceptuel se veut générique.

L'information contextuelle est structurée en réseaux de contextes et de situations dont les frontières sont facilement identifiables au moyen de

règles systématiques sur les ensembles de définition *Roles*, *Relations*, *Entites* : deux contextes diffèrent si les entités n'y jouent pas le même ensemble de rôles et/ou n'assurent pas le même ensemble de relations. Les situations d'un contexte portent toutes sur la même partition de *Roles* et la même partition de *Relations*. Mais deux situations d'un contexte diffèrent si les instances d'entités diffèrent et/ou si l'instanciation des rôles et/ou des relations par ces entités diffèrent.

La structure en réseau répond au constat initial : (a) l'information contextuelle est structurée et évolutive. (b) absence de concepts. Les nœuds représentent des contextes (et au sein d'un contexte, les nœuds dénotent des situations). Les arcs expriment l'évolution. Le réseau, au sens mathématique, exploité comme métamodèle, fournit un fondement conceptuel générique.

Le modèle conceptuel est complété d'une méthode qui explicite le processus de construction d'un réseau de contextes et de situations et d'une notation graphique dont la définition est rappelée en annexe A. Le processus préconisé permet une analyse exhaustive du problème. Pour limiter la combinatoire du nombre de contextes et de situations, les contextes pour lesquels le système observe le même comportement sont factorisés sous un même chapeau, et si le comportement est la fonction Identité (pas de changement d'état), ces contextes sont supprimés.

Enfin, le cadre conceptuel offre des éléments de classification des systèmes interactifs de l'informatique ambiante en termes de finalités et d'entités observées. Cette vue synthétique de l'existant permet d'entrevoir les requis fonctionnels des infrastructures logicielles utiles à la mise en œuvre. Cet aspect est traité au chapitre suivant avec la présentation de l'état de l'art en la matière.

Chapitre II *Les infrastructures de capture du contexte*

Par ordre d'entrée en scène : des particules infimes, dans un désordre indescriptible. Puis, résultats de leurs accouplements, les premiers atomes qui tentent, eux aussi, des liaisons explosives au cœur d'astres brûlants.

1996, La plus belle histoire du monde. Acte 1 - Scène 2 : L'univers s'organise, p37

Comme démontré au chapitre I, nous disposons d'un outil conceptuel de modélisation du contexte : un réseau de contextes et de situations défini sur des ensembles d'entités, de relations et de rôles, le tout identifié au moyen d'observables. Il convient maintenant de s'interroger sur la réalisation technique. Etant donné un modèle conçu pour une finalité donnée, par exemple le réseau de contextes de ContextNotePad du chapitre précédent, le développeur a le choix, pour l'implémentation, entre une solution ad hoc et la réutilisation d'outils généraux. L'approche ad hoc a souvent l'avantage d'une exécution efficace au prix d'un surcoût de développement et de maintenance. Les outils généraux comme les infrastructures logicielles d'exécution et les boîtes à outils[1], offrent des services d'utilité publique prêts à l'emploi. De facto, ces outils réduisent les coûts de développement et améliorent la cohérence entre les applications au risque d'un surcoût d'apprentissage pour le développeur et d'une augmentation sensible des temps de réponse et des besoins en ressources de calcul. Ces appréciations générales sur les infrastructures logicielles et les boîtes à outils s'appliquent-elles aux outils logiciels pour le contexte ?

Ce chapitre propose une revue analytique des infrastructures logicielles pour le contexte. Après quelques constats sur l'utilisation de ces infrastructures, je propose dans la section 2, une décomposition fonctionnelle de référence qui nous permettra de comparer les infrastructures pour le contexte en termes de couverture de services, comparaison que l'on affinera ensuite en critères implémentationnels et architecturaux (section 3). L'espace de classification ainsi obtenu fera l'objet de la section 3. En 4, des infrastructures de contexte seront présentées en détail avant de conclure par une synthèse sur l'offre actuelle en matière d'infrastructures de contexte.

1. Constats

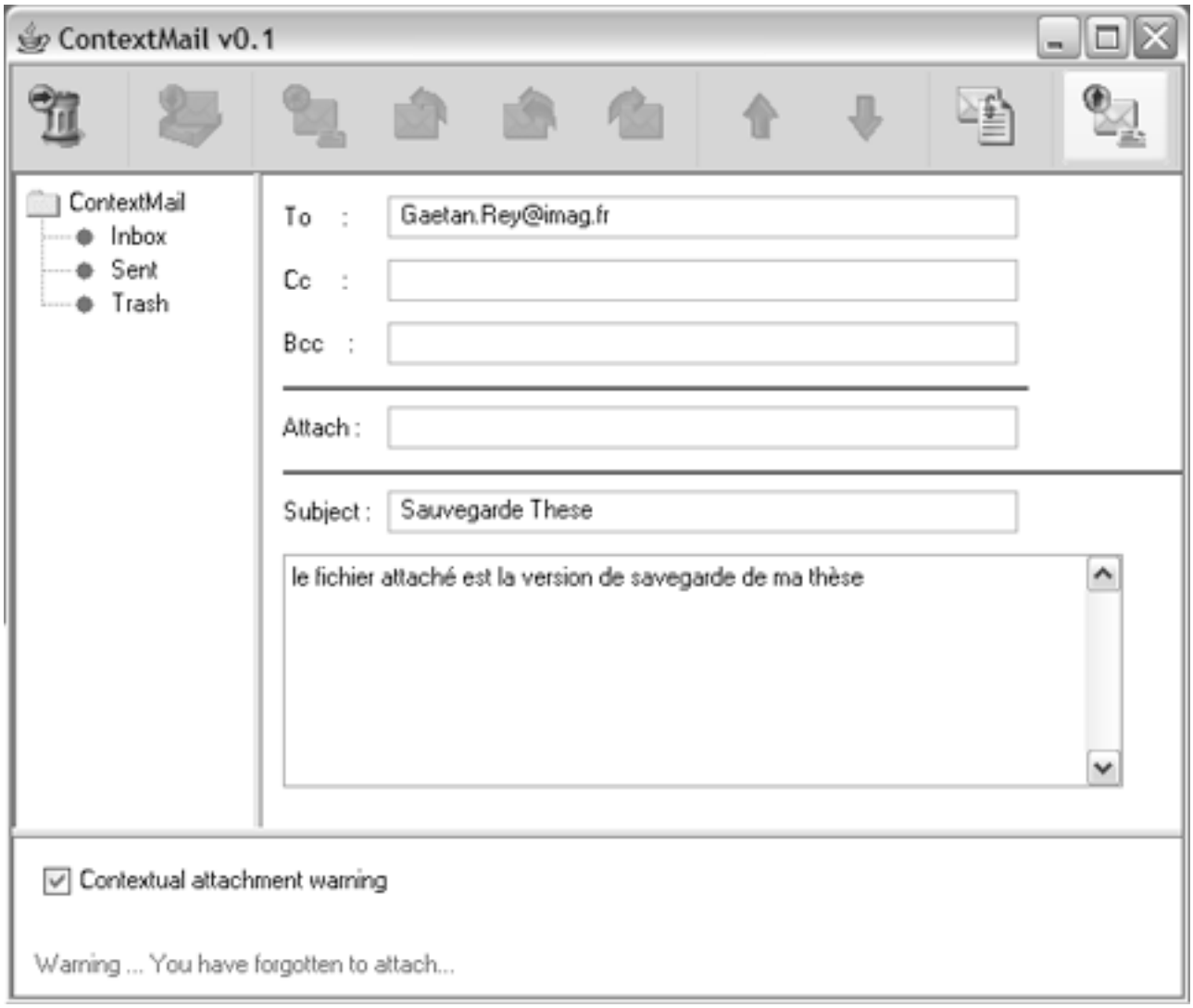
En Interaction Homme-Machine, nous l'avons vu dans le chapitre d'introduction, l'analyse du contexte fait partie de la pratique méthodologique du processus de conception [Beyer 98]. Or, ces connaissances sur le contexte se diluent progressivement dans les étapes d'implémentation au point de ne plus exister de manière explicite dans les applications finales. Une première explication à cette perte d'information est l'absence

1. On entend par infrastructure logicielle d'exécution, ou intergiciel, un ensemble cohérent de services d'utilité publique, disponible en permanence et soutenant l'exécution ou le développement de services visibles. Parmi ces services visibles, certains, eux aussi d'utilité publique, sont fournis par des boîtes à outils.

de pont méthodologique entre les pratiques des spécialistes en facteurs humains et les méthodes et outils du Génie Logiciel. Le processus présenté au chapitre précédent (section 6), vise à maîtriser ce passage. La seconde raison tient à la nouveauté des infrastructures pour le contexte. Bien qu'existant en grand nombre, aucune ne s'est imposée comme véritable outil de développement. Toutes relèvent encore de l'expérimentation de laboratoire. Dans ces conditions, la tentation est grande, pour le développeur, d'adopter une approche exploratoire, au cas par cas.

Le choix d'une solution ad hoc est raisonnable pour un démonstrateur de concept dont la durée de vie est de quelques mois. Il se justifie également lorsque la finalité du contexte est spécifique à l'extrême. Dans ce cas, les infrastructures sont, par définition, de peu d'utilité. À cet égard, je propose de distinguer les applications dominées par des *own contextual*

Figure 1
Représentation de l'interface de ContextMail.



information, des applications qui se suffisent de *public contextual information*.

- ***Own Contextual Information*** désigne une information contextuelle qui n'a d'intérêt, ou de sens, que pour une application. Il s'agit d'information contextuelle spécifique.
- ***Public Contextual Information*** qualifie une information ayant un intérêt pour plusieurs applications.

À titre d'exemple, considérons *ContextMail*, un courriel capable de détecter les oublis de pièce jointe. La figure 1 en montre l'IHM. Il s'agit d'une application de messagerie électronique à laquelle j'ai ajouté une fonction d'analyse du contenu des messages. Lorsqu'elle est activée, cette fonction examine les messages avant leur envoi au serveur SMTP, à la recherche d'indices sur l'attachement de fichiers. Si, dans le corps du message à expédier, il est fait mention d'un attachement et qu'aucun fichier n'est joint, l'envoi du message est suspendu. Averti de son oubli, l'utilisateur a la possibilité de corriger son erreur ou de forcer l'envoi du courrier (en cas d'oubli volontaire ou de fausse alarme).

L'information concernant l'attachement du fichier est contextuelle, mais n'a d'intérêt que pour le logiciel de messagerie électronique. Il s'agit d'information *own contextual*. Dans le cas de *ContextMail*, qui n'utilise qu'une information de type *own*, on comprend qu'il n'est pas nécessaire d'utiliser une infrastructure de gestion du contexte. Toutefois, comme le montrent les exemples du chapitre précédent (section 7), peu d'applications se suffisent de *own contextual information*. Les infrastructures générales pour le contexte ont alors toute leur utilité. Le choix d'une infrastructure particulière est guidé par ses capacités à satisfaire les requis du contexte à mettre en œuvre. La première de ces capacités concerne la couverture fonctionnelle. Dans la section qui suit, je propose un modèle d'architecture conceptuelle qui organise les fonctions d'une infrastructure pour le contexte à différents niveaux d'abstraction

2. *Modèle d'architecture conceptuelle pour infrastructure de contexte*

La figure 2 montre le positionnement et le rôle d'une infrastructure de contexte. Située entre les capteurs physiques et les applications, elle fournit à ses applications clientes et gère pour leur compte des public contextual information au bon niveau d'abstraction [Hong 01].

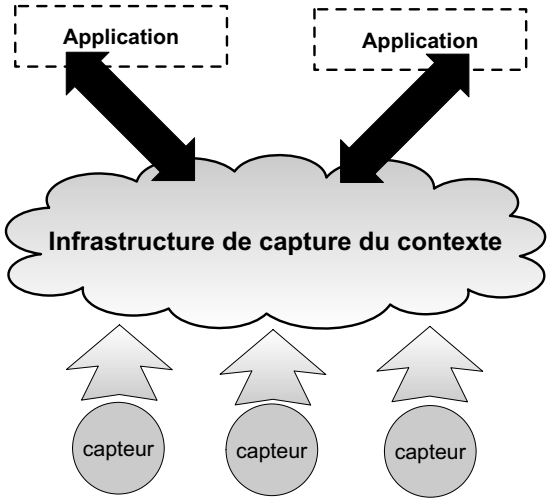


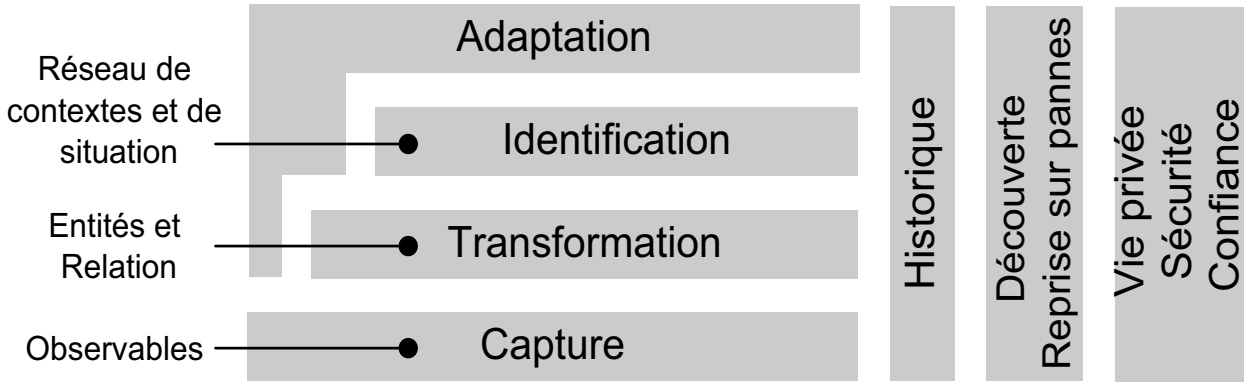
Figure 2
Infrastructure de contexte.
Intergiciel se situant entre les capteurs physiques et les applications utilisants des informations contextuelles .

Je propose de structurer les services contextuels en une pyramide à quatre niveaux d'abstraction (capture, transformation, identification et adaptation) tissés de mécanismes pour servir l'historique, la découverte de ressources, la reprise sur panne, la sécurité, la protection de l'espace privé ou la confiance. D'ùo le nom de ce modèle : *pyramide du contexte*.

2.1. LES NIVEAUX D'ABSTRACTION DE LA PYRAMIDE DE CONTEXTE

Comme le montre la figure 3, au plus bas niveau d'abstraction, la couche de *capture* interface les capteurs physiques : elle en masque la diversité et encapsule les données fournies par les capteurs sous forme d'observables. À ce niveau d'abstraction, les observables sont des grandeurs physiques rendues accessibles au monde numérique. Une image captée par une caméra, éventuellement seuillée pour éliminer le bruit, est un *observable numérique*.

Figure 3
La Pyramide du contexte.
Le modèle, ses services transversaux et ses liens avec les termes de l'ontologie du chapitre I.



La couche *transformation* s'appuie sur les observables numériques pour produire des *observables symboliques* et de là, regrouper des observables pour détecter la présence d'entités, les identifier et les suivre, de même déterminer leurs relations et leurs rôles. Les observables symboliques correspondent aux observables du modèle de conception du chapitre I. Par exemple, à partir de plusieurs images du niveau capture (observables numériques), le niveau de transformation détermine le nombre d'individus en présence (observable symbolique) et découvre que deux d'entre eux sont à proximité du tableau (relation spatiale) en train de parler (identification d'activité).

La couche *identification* est chargée de détecter les changements de contexte et de situation et de parcourir les réseaux de contextes et de situations pour identifier le contexte et la situation actuels. À partir de là, ce niveau est capable de répondre à des requêtes de haut niveau d'abstraction, par exemple, " quelle est la situation actuelle ? " ou encore "le bureau B204 du laboratoire CLIPS de Grenoble est-il occupé?".

Au plus haut niveau de la pyramide, la couche *adaptation*. Par analogie avec l'adaptateur de noyau fonctionnel des modèles Seeheim [Seeheim 85], Arch [Arch 92] et PAC-Amodeus [Nigay 94], ce niveau permet de régler toutes les discordances entre l'infrastructure et les applications clientes : adaptation des structures de données, adaptation du protocole de communication, adaptation des formalismes et notamment transformation d'une requête applicative dans les termes attendus par l'infrastructure, au niveau d'abstraction souhaité. À l'interface entre applications et infrastructure, cette couche peut tout aussi bien faire partie de l'application.

Plus important, par souci d'efficacité, les applications peuvent utiliser directement le niveau d'abstraction désiré sans devoir passer par des couches intermédiaires inutiles. On obtient ainsi une pyramide de machines abstraites qui constitue le pendant logiciel du cadre conceptuel du chapitre I : observables de phénomènes physiques (couche capture), observables symboliques et regroupement d'observables pour identifier les entités, les relations, les rôles et les attributs pertinents (couche transformation), parcours du réseau (couche identification), puis interfaçage avec les applications au bon niveau d'abstraction (couche adaptation). Ces services de base sont complétés de services transversaux.

2.2. LES SERVICES TRANSVERSAUX

Tous les niveaux de la pyramide sont concernés par les aspects suivants : historique, découverte de services et disponibilité, sécurité et protection de la sphère privée, confiance. Ces aspects, qui se retrouvent à chaque niveau d'abstraction, sont répartis et implémentés dans les termes de chaque niveau d'abstraction pour former un tout cohérent. Par exemple, le service historique du niveau capture alimente le service historique du

niveau transformation sur lequel s'appuie le niveau identification pour maintenir un historique des contextes et situations. Il en va de même pour les autres services transversaux.

Comme son nom l'indique, le service historique mémorise les valeurs que prend une information au cours du temps. Cette sauvegarde présente un double intérêt :

- pour l'infrastructure elle-même comme base d'informations pour les reprises sur panne,
- pour tous les développeurs de systèmes interactifs en informatique ambiante. Si l'on se réfère à la typologie du chapitre I (section 7), pour les systèmes de type "Présentation d'information", l'historique sur une information permet, en plus de la valeur instantanée de cette information, de calculer des tendances. Nourrissant l'apprentissage, l'historique sert aussi les systèmes à exécution automatique de services ou à l'auto-adaptation qui, mieux renseignés, choisissent un comportement plus adapté. Enfin, tous les systèmes de type aide-mémoire s'appuient de facto sur des historiques.

La *découverte de services* et la *reprise sur panne* sont nécessaires dans un milieu ambiant évolutif, instable, imprévisible. La découverte de services doit offrir la possibilité d'ajouter dynamiquement des services ou d'en retirer si ceux-ci ne sont plus accessibles. Les fonctions de reprise sur panne permettent aux applications de retrouver les services qu'elles utilisaient si leur communication avec ces services devait être interrompue. Elles permettent aussi aux applications de retrouver des services équivalents aux services qu'elles utilisaient si ces derniers devenaient inaccessibles.

La troisième catégorie de services regroupe les fonctions de *sécurité* et, en corollaire, la protection de la *sphère privée* et la *confiance (trust)*. La sécurité concerne la protection contre les accès non autorisés. Elle sera analysée plus avant en section 3.1.2. La protection de l'espace privé est un sujet sensible en IHM depuis le déploiement des collecticiels et notamment, des mediaspaces [Coutaz 98]. Avec l'informatique ambiante, le problème est exacerbé par la disparition de l'ordinateur au profit de capteurs invisibles, de connexions dynamiques de services aux frontières impalpables et aux comportements non observables. Dans ces conditions, il est difficile de satisfaire les principes connus du respect de la sphère privée comme la réciprocité (si tu me vois, je te vois). De là, comment entretenir le sentiment de confiance ?

La Pyramide du contexte est à rapprocher des propositions de Salber et de Hong. Le modèle de Salber [Salber 01] n'inclut pas les services transversaux ni n'explicite la couche d'identification. Chez Hong, l'accent est mis sur la couche identification, au détriment de la couche transfor-

mation dont les fonctionnalités de cette dernière sont dispersées dans les deux autres couches (identification et capture). [Hong 01].

La pyramide du contexte est une représentation architecturale fonctionnelle exprimée dans le style machine abstraite. En conception logicielle, elle permet d'organiser à gros grain la couverture fonctionnelle d'une infrastructure de contexte. Elle peut servir aussi à évaluer et comparer la couverture fonctionnelle d'infrastructures existantes. La section qui suit présente des critères d'analyse complémentaires.

3. Critères d'analyse

Les facteurs, critères et métriques définis pour le processus de qualité logicielle par les organismes de normalisation comme l'AFNOR, IEEE et bien d'autres, fournissent un bon cadre de référence à aux besoins de notre analyse. Appliquant les facteurs de McCall [McCall 77] aux infrastructures de contexte, nous retenons :

- La flexibilité pour caractériser les facultés d'évolution face aux changements du milieu ambiant,
- La fiabilité et l'efficacité pour qualifier les aptitudes opératives : par définition, une infrastructure doit être disponible en permanence : fiabilité requise. Comme nous l'avons vu en introduction, les temps de réponse et la consommation des ressources (mémoire, CPU, réseau) sont à surveiller de près : efficacité requise.

Si l'efficacité est importante, ce facteur ne sera pas retenu comme élément de comparaison : les mesures de latence et de consommation des ressources des infrastructures de contexte ne sont pas publiées. Au mieux, sont-elles exprimées de manière approximative. Alors, pour une analyse comparative crédible, il conviendrait de mener une campagne de tests, dans des conditions (quasi) identiques, de l'efficacité des infrastructures les plus représentatives. La mesure de performance de systèmes répartis relève d'une compétence spécifique, en pratique difficile à acquérir dans le temps imparti à cette thèse. Mais surtout, les infrastructures de contexte actuelles sont des prototypes de recherche qui n'ont pas toujours fait l'objet d'optimisations. A priori, les infrastructures les plus anciennes seraient alors avantagées au détriment des solutions plus récentes, aux innovations intéressantes, mais sans la maturité technique de leurs aînées.

La figure 4 traduit les critères d'analyse retenus pour assurer les facteurs de flexibilité et de fiabilité :

- Les techniques d'autorégulation,
- Les styles architecturaux.

Les autres critères de la figure 4 correspondent aux services visibles pour le programmeur d'application :

- Les niveaux d'abstraction de la pyramide et les services transversaux,
- Les modes de demande et d'obtention de ces services,
- La couverture réseau que l'infrastructure assure.

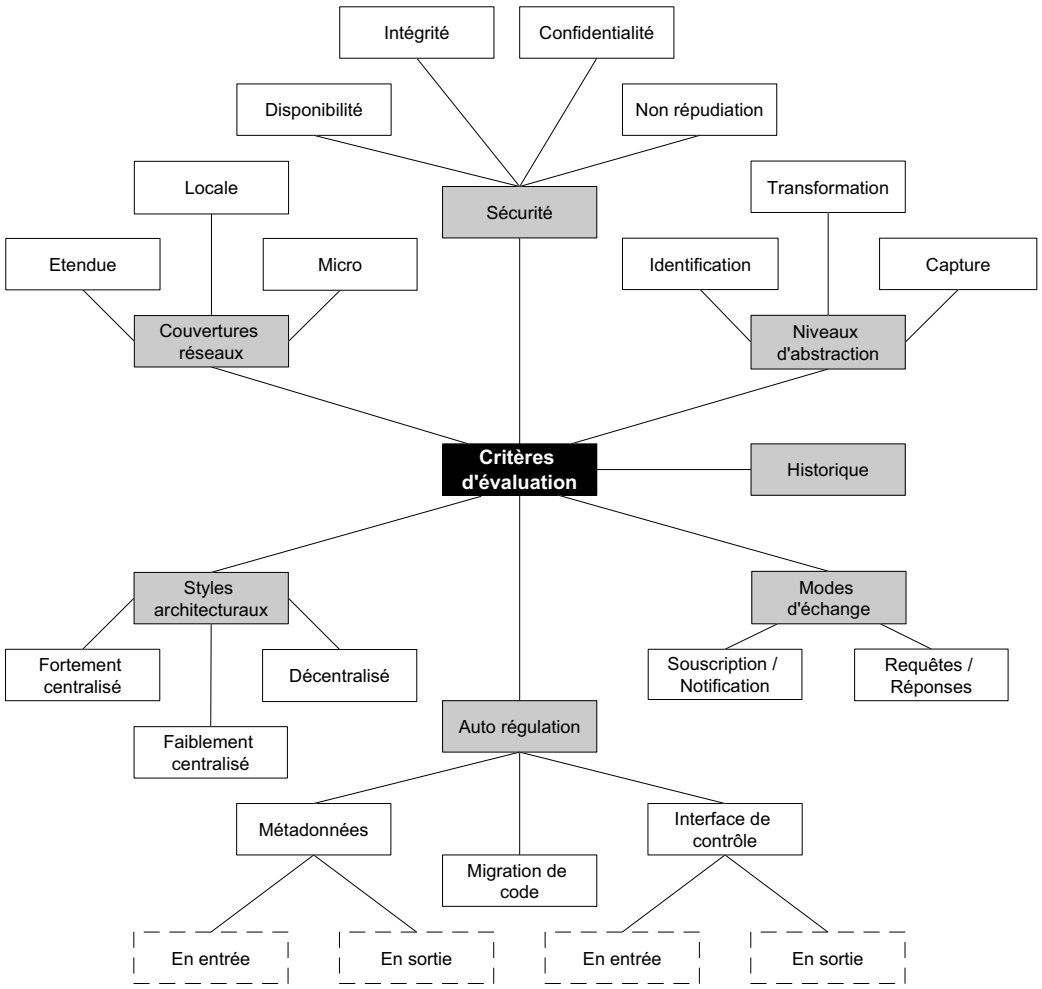
Développons l'analyse en commençant par ces aspects fonctionnels. Nous verrons ensuite les aspects non fonctionnels des infrastructures de contexte avec les aspects architecture et l'autorégulation.

3.1. ASPECTS FONCTIONNELS

Niveaux d'abstraction

Les niveaux d'abstraction du modèle en pyramide (figure 3) constituent une bonne caractéristique des capacités fonctionnelles fondamentales d'une infrastructure de contexte. Plus elle couvre de niveaux, plus l'infras-

Figure 4
Liste détaillée des critères retenus pour l'évaluation des infrastructures contextuelles.



tructure facilite et simplifie les tâches de conception et de mise en œuvre du développeur. Toutefois, une bonne couverture de services peut aussi signifier une perte en efficacité si, pour exploiter un service de bas niveau d'abstraction, l'application cliente doit " acheter " tous les niveaux. Ici, le style d'architecture associé à des mécanismes de composition dynamiques a toute son importance.

Etudions maintenant les services transversaux de la pyramide.

Sécurité

En informatique conventionnelle, la sécurité consiste à assurer que les ressources matérielles et/ou logicielles d'une organisation sont utilisées dans le cadre prévu. En informatique ambiante, elle sert de fondement à la protection de la sphère privée. Pour une infrastructure de contexte, la sécurité consiste à garantir l'intégrité, la confidentialité, la disponibilité et la non répudiation [Florin 97] :

- *L'intégrité* garantit que les informations ne sont pas modifiées ou corrompues. Les techniques de water-marking relèvent de cette catégorie de services [Cox 99].
- La *confidentialité* assure l'accès à des ressources et informations données aux seules entités autorisées (utilisateurs et/ou services logiciels les représentant). Les techniques par login et mot de passe, de même le cryptage, relèvent de la confidentialité.
- La *disponibilité* définit l'accessibilité temporelle aux ressources et informations.
- La *non répudiation* consiste à assurer qu'une personne (ou un service logiciel) ne puisse nier avoir participé à un échange.

De manière idéale, ces quatre aspects devraient être assurés à tous les niveaux d'abstraction de la pyramide. Nous verrons dans la section de synthèse qu'il n'en est rien.

Historique

L'historique doit inclure des services de sauvegarde et de recherche d'information, de suppression et de compilation. Sauvegarde et recherche sont fondamentales : la première, pour répondre à la définition de la notion d'historique, la seconde puisqu'il ne sert à rien de mémoriser s'il n'existe pas de moyen pour retrouver ce qui a été sauvegardé.

La sauvegarde et la recherche consistent respectivement à mémoriser et à retrouver sur critêres, les valeurs successives d'observables donnés (pour les couches basses de la pyramide) et, pour les couches hautes de la pyramide, les états successifs d'entités particulières, de rôles et de relations, mais aussi de transitions spécifiques dans les réseaux de contextes et de situations. La difficulté pour le concepteur est de définir a priori les informations utiles pour le futur. En informatique ambiante, le temps se déroule bien au-delà de la session de l'informatique conventionnelle : l'utilisation du système se veut perpétuelle.

La suppression d'informations répond au problème de l'accroissement sans fin de l'historique. Par conséquent, il est vital que le service d'historique limite le nombre d'informations sauvegardée et/ou utilise des méthodes de sauvegarde avancées comme les sauvegardes incrémentales ou autre Z-scheme présentées dans [Kurmas 00]. Une autre approche consisterait à "oublier" certaines informations ou à les encapsuler par compilation. (Chez l'Homme, l'oubli est souvent perçu comme une fatalité. Il est aussi source de bien-être mental)

Par analogie avec la notion de compilation introduite par Anderson dans le modèle ACT* [Anderson 92], des informations pourraient être assemblées par apprentissage pour former une nouvelle unité de connaissance. Le service historique pourrait ainsi remplacer un ensemble d'informations par un autre de cardinalité moindre tout en étant sémantiquement plus riche.

Couverture réseau

La couverture réseau désigne l'étendue spatiale des services de l'infrastructure, qu'il s'agisse des services de la pyramide ou des services transversaux. Plus grande est la couverture réseau, plus loin sont les applications susceptibles de découvrir les services de l'infrastructure. Toutefois, l'éloignement signifie un surcoût en communication auquel il convient de veiller.

Je propose trois niveaux de couverture d'un service d'infrastructure de contexte :

- Couverture micro lorsque le service est limité à la machine sur laquelle il s'exécute.
- Couverture locale : le service est accessible sur le sous-réseau local.
- Couverture étendue : le service est accessible au-delà du sous-réseau local, éventuellement au niveau planétaire.

Echanges des données

Le dernier critère retenu pour l'évaluation des infrastructures de contexte du point de vue fonctionnel concerne les modes d'échanges de données avec les applications clients. Les deux modes les plus répandus sont :

- ***Souscription / Notification*** : Le client s'abonne à un service particulier auprès de l'infrastructure. Celle-ci envoie les données en fonction du contrat négocié. L'envoi a lieu par exemple, toutes les n millisecondes (cas le plus courant), ou encore chaque fois que la donnée change de valeur.
- ***Requêtes / Réponses*** : Comme dans le cas précédent, le client s'abonne à un service, mais l'infrastructure n'envoie des données que sur demande explicite de l'abonné.

Le mode Souscription / Notification économise les échanges sur le réseau et se présente comme un mécanisme à événements. Les concepteurs d'interfaces graphiques sont familiers avec ce type de communication

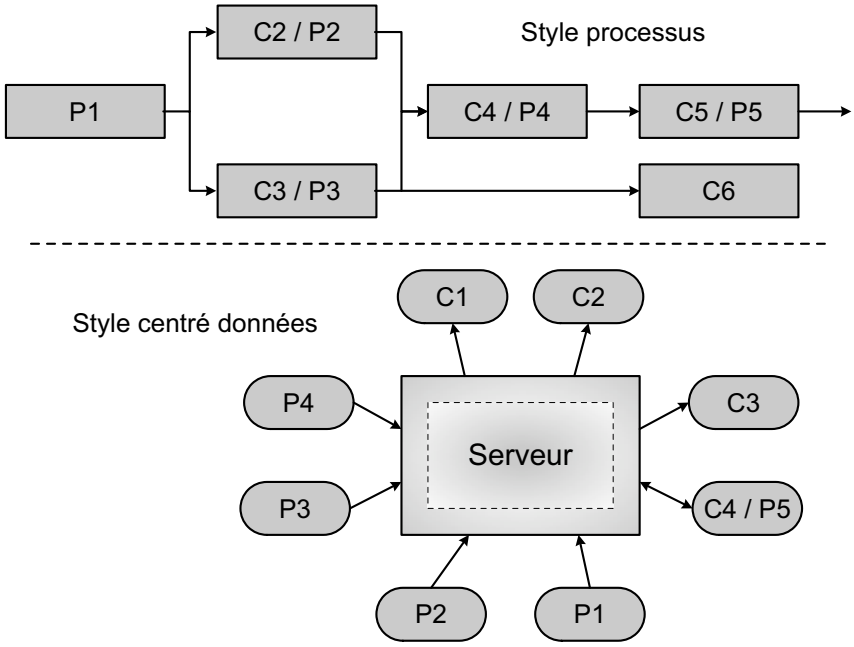
asynchrone. Toutefois, pour ne pas perdre de données, ils doivent être attentifs à la synchronisation dans l'écriture des applications clientes. Le mode Requêtes / Réponses a l'avantage d'être proche du paradigme de programmation synchrone et élimine l'expression explicite de la synchronisation dans les applications clientes. Ce mode étant le plus souvent implémenté par appels de procédure à distance, il est transparent aux développeurs d'applications.

Au bilan, le choix d'une infrastructure pour son mode d'échanges des données ne tient pas aux avantages techniques de l'un par rapport à l'autre, mais plutôt de l'expérience du développeur en programmation procédurale versus programmation événementielle. À l'évidence, les infrastructures offrant les deux modes d'échanges de données ont l'avantage sur les autres.

3.2. ASPECTS NON FONCTIONNELS

Notre constat général sur la nature du contexte fait référence à un ensemble d'informations … qui sert une finalité (Section 2.2 du chapitre I). Autrement dit, le contexte est un état (ensemble d'informations) qui s'inscrit dans un processus (il sert une finalité) [Coutaz 05]. Cette dualité, état - processus se traduit, dans les infrastructures de contexte actuelles, par des styles architecturaux différents selon l'importance accordée à la dimension état ou à la dimension processus du contexte.

Figure 5
Illustrations des deux styles d'architectures.
Pour chaque style, sont représentés les producteurs (Pi), les consommateurs (Ci) et les flux de données (les flêches).



Style et classe d'architecture

Un style d'architecture se définit selon trois volets [Garlan 93], [Kolski 97] : (1) un vocabulaire d'éléments (par exemple, les concepts de filtre et de tuyau, ou celui de machine abstraite) ; (2) des règles de configuration des éléments du vocabulaire (par exemple, un tuyau doit relier deux filtres) ; (3) une sémantique qui donne un sens à la représentation des architectures exprimées dans ce vocabulaire et selon ces règles.

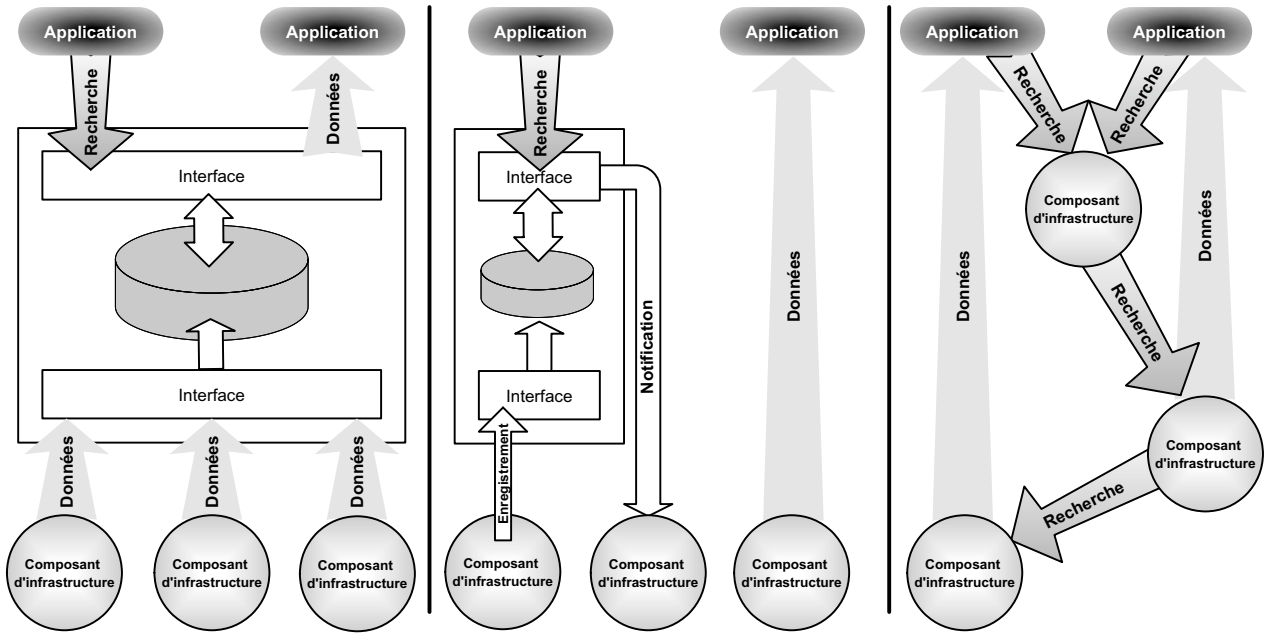
La figure 5 montre l'application de deux styles d'architecture pour la mise en œuvre d'infrastructures de contexte, selon la priorité accordée à la perspective état ou à celle de processus.

- Dans un style centré sur les données, l'infrastructure est organisée autour d'une mémoire partagée détentrice des informations contextuelles. C'est l'approche adoptée par Winograd [Winograd 01].
- Dans un style centré sur le processus, l'infrastructure se présente sous la forme d'une composition de processus de traitement. Les informations contextuelles sont alors réparties entre les processus sous forme de flots de données. ContextToolkit fonctionne selon ce style.

Un style n'est pas une architecture, mais toute architecture repose sur un style. En cela, un style peut être compris comme une classe d'architectures. Les styles centrés données et processus de la figure 5 conduisent aux trois classes d'architectures de la figure 6 :

- ***Fortement centralisée*** : un composant central, le serveur, prend en charge l'ensemble des services de l'infrastructure de contexte. Les

Figure 6
Style architecturaux.
De gauche à droite : fortement centralisé, faiblement centralisé, pleinement décentralisé.



systèmes fortement centralisés correspondent au style centré données de type base de données, tuple-spaces et blackboard. Ils sont structurés en deux types de composants (voir figure 5) : une structure centrale, le serveur, détient les informations contextuelles et une collection de composants indépendants (les clients et les producteurs) opèrent sur la structure centrale. Ici, centralisation signifie que le serveur est le point de passage obligé de toute transaction entre les producteurs d'information (tels les capteurs) et les clients. Cette centralisation n'exclut pas qu'au niveau de la structure physique, le serveur soit réparti sur plusieurs machines.

- ***Faiblement centralisée*** : un composant central prend en charge la tâche de découverte de services. Il met en relation le client (application) et la source du service (par exemple un composant de capture). Les autres fonctions de l'infrastructure de contexte sont attribuées à d'autres composants spécifiques répertoriés dans le composant central. Contrairement au cas précédent, ici le serveur n'est qu'un annuaire de services. Il n'est plus le point de passage obligatoire de toutes les transactions entre les producteurs et les clients mais seulement des messages d'enregistrements et de découvertes. Une infrastructure faiblement centralisée est une combinaison de style centrée données (pour l'annuaire) et de style centré processus (pour le traitement proprement dit du contexte).
- ***Décentralisée*** : absence de composant central. Les services sont alors répartis dans un ensemble de composants. Lors d'une recherche, les messages transitent de composant en composant jusqu'à atteindre leurs destinataires. Une fois le fournisseur du service trouvé, la communication fournisseur-client peut alors se faire directement. Comme pour les infrastructures faiblement centralisées, les systèmes décentralisés utilisent principalement un modèle en flots de données, où les entrées des composants clients sont les sorties des composants producteurs.

Chacune de ces trois classes d'architecture a ses avantages et inconvénients. Bien que les infrastructures fortement centralisées soient plus facilement administrables, elles nécessitent l'utilisation de machines puissantes, capables d'assurer le rôle du serveur. Elles sont vulnérables et peuvent être entièrement paralysées en cas de panne du serveur. Pour sa part, le modèle faiblement centralisé limite les fonctions du serveur ce qui a pour effets de diminuer la puissance nécessaire de la machine serveur et de faciliter la mise en place de capacité de reconfiguration. Inversement, l'administration de l'infrastructure est plus complexe. Contrairement aux approches fortement et faiblement centralisées, le modèle décentralisé dilue les fonctions de l'infrastructure dans chacun de ses composants. La vulnérabilité due au composant central disparaît en même temps que la nécessité de machines performantes. Une machine peut exécuter autant de

composants que sa puissance le lui permet : peu sur un PDA, beaucoup sur une station de travail haute performance. Cependant, l'administration de telles architectures est souvent très complexe.

Autorégulation

De manière à prendre en compte l'évolution constante du contexte et la variabilité du milieu ambiant, une infrastructure doit être capable de réguler de manière autonome la consommation de ressources nécessaire à son bon fonctionnement. Pour cela, je propose comme sous-critères, l'existence de métadonnées, d'interface de contrôle et la migration de code.

- ***Metadonnées.*** Les informations contextuelles sont élaborées à partir d'observables acquis au moyen de capteurs. Or, nous le verrons en détail au chapitre suivant, les capteurs ne sont pas précis, ils sont instables, et tombent en panne. Dans ces conditions, il convient que l'infrastructure auto-évalue le service rendu. Le résultat de ces évaluations est consigné dans des métadonnées qui, attachées aux informations contextuelles, qualifient ces informations. La nature des métadonnées dépend du niveau d'abstraction de l'information contextuelle, mais latence et facteur de confiance concernent tous les niveaux d'abstraction. Un aspect important de la gestion des métadonnées est la synchronisation entre la donnée et les métadonnées qui la qualifient. L'architecture doit mettre en avant cette synchronisation sans quoi elle s'expose à des erreurs d'interprétation.
- ***Interface de contrôle***. De manière à ce que les applications clientes puissent réagir à tout instant et efficacement aux changements du milieu ambiant, une infrastructure de contexte doit proposer une interface de contrôle. Une interface de contrôle permet aux clients de modifier dynamiquement leur demande de service. Par exemple, changer le mode d'échange des données, demander l'amélioration d'une métadonnée (réduire la latence ou améliorer la précision), ou encore arrêter temporairement l'utilisation d'un sous-ensemble de l'infrastructure tant que celle-ci n'a pas rétabli un niveau satisfaisant de qualité. En résumé, l'interface de contrôle permet à chaque application d'affiner le réglage des échanges entre elle et l'infrastructure durant son exécution. En plus de l'amélioration de la qualité des services, l'interface de contrôle devrait permettre d'économiser des ressources. En effet, il est inutile de calculer et d'échanger des données erronées. Mieux vaut réduire la fréquence de calcul et d'échange de ces données jusqu'au rétablissement d'un niveau de qualité satisfaisant. Les ressources ainsi économisées peuvent utilement servir à améliorer d'autres données.
- ***Migration de code.*** De manière à rendre optimale l'utilisation des ressources et ceci quel que soit le lieu d'utilisation des applications clientes, les infrastructures de contexte devraient être capables de répartir leur exécution sur les différentes machines à leur disposition.

Le déplacement de code est une solution à ce problème d'utilisation de ressources. Par exemple, en déplaçant le code au plus près de l'application cliente, on allège l'utilisation du réseau et les latences qui en découle. Une autre exploitation de la migration est l'économie d'énergie en déplaçant le code de dispositifs mobiles vers des dispositifs fixes.

Les critères de la figure 4 nous servent maintenant à l'analyse de quelques infrastructures de contexte.

4. Exemples d'infrastructures

Parmi les infrastructures de contexte de l'état de l'art, quatre réalisations retiennent l'attention :

- la *Context-toolkit* [Dey 99], l'infrastructure de référence du domaine,
- l'architecture de Schmidt [Schmidt 99], une infrastructure proche des capteurs à fondements mathématiques,
- le *Context-handling component* de Salber [Salber 01], l'une des toutes premières infrastructures à métadonnées, extension de Arch, modèle d'architecture de référence en IHM [Arch 92],
- la *StratClyde Context Infrastructure* [Glassey 03] inspirée de mes travaux de Master [Rey 01],

4.1. LE CONTEXT-TOOLKIT

Context-toolkit tient son nom par analogie avec les bibliothèques graphiques de widgets. Comme pour les boîtes à outils graphiques, qui offrent un modèle computationnel et des widgets d'utilité publique (menu, formulaire, bouton), l'objectif de la *Contex-Toolkit* est d'offrir un modèle d'exécution et des composants de base réutilisables.

Description

Comme le montre la figure 7, la Context Toolkit est une infrastructure centrée processus, de type flot de données. Elle comprend cinq classes de composants logiciels et deux classes de composants matériels.

Un *context widget*, ou interacteur de contexte, est un composant autonome qui sert de lien entre un capteur physique et les applications. Son rôle est de communiquer à un (ou plusieurs) *aggregator*(s) ou *interpreter*(s) les données en provenance de capteurs. Il est également capable de constituer un historique qui lui est propre. La fonction historique peut être désactivée à la demande.

Les interpréteurs (*interpreter*) donnent une sémantique aux signaux transmis par les *contexts widgets* et ceci au bon niveau d'abstraction. Par

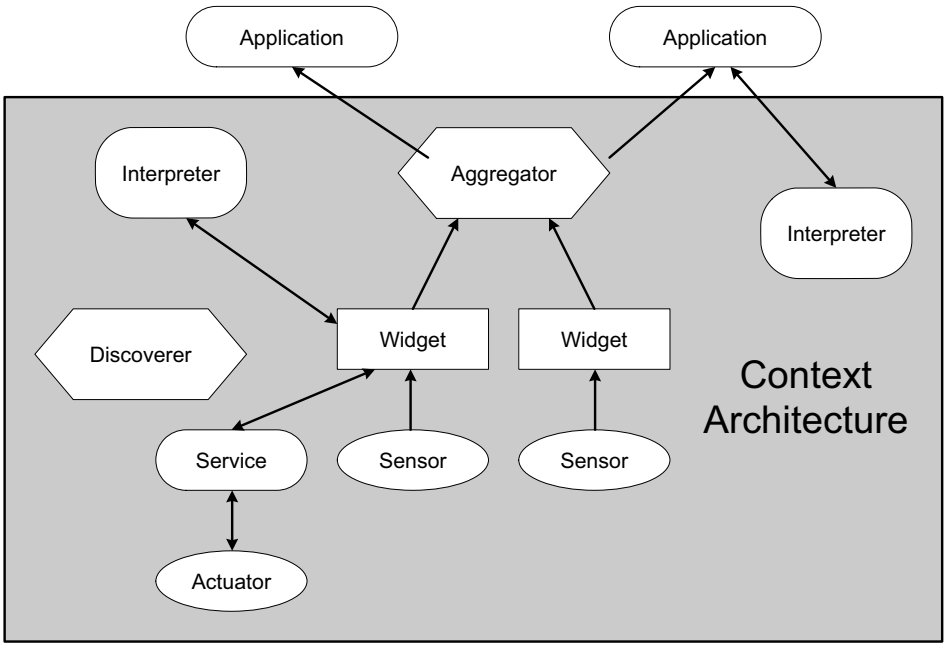


Figure 7
Architecture de la Context Toolkit extraite de [Dey 01].

exemple, un interpréteur de localisation de personne peut fournir comme informations :

- Personne A dans la salle B204
- Personne A dans le Bâtiment B de l'IMAG
- Personne A dans le Campus de Grenoble

Les agrégateurs (*aggregator*) collectent l'ensemble des signaux émis par les autres composants et fait le lien entre les applications et les *context-widgets*. Les agrégateurs ont également la charge de synthétiser les informations en informations de niveau d'abstraction supérieur. Cependant le mécanisme de synthèse n'est pas fourni et revient au programmeur.

Les connexions entre les *Context-Widgets*, les *Aggregators* et les applications ont lieu par l'intermédiaire du *Discoverer*. Le *Discoverer* maintient la liste des composants disponibles. Il peut être interrogé par les applications cherchant des composants et/ou des services spécifiques.

Par le biais des *Actuators*, les *Services* exécutent des actions pour le compte des applications offrant ainsi des capacités d'action en symétrie avec la perception par le biais des capteurs.

Sur le plan technique, le système de communication entre les *context-widgets* et l'application se fait principalement par souscriptions et la réaction aux souscriptions s'effectue sous forme de call-back (réaction).

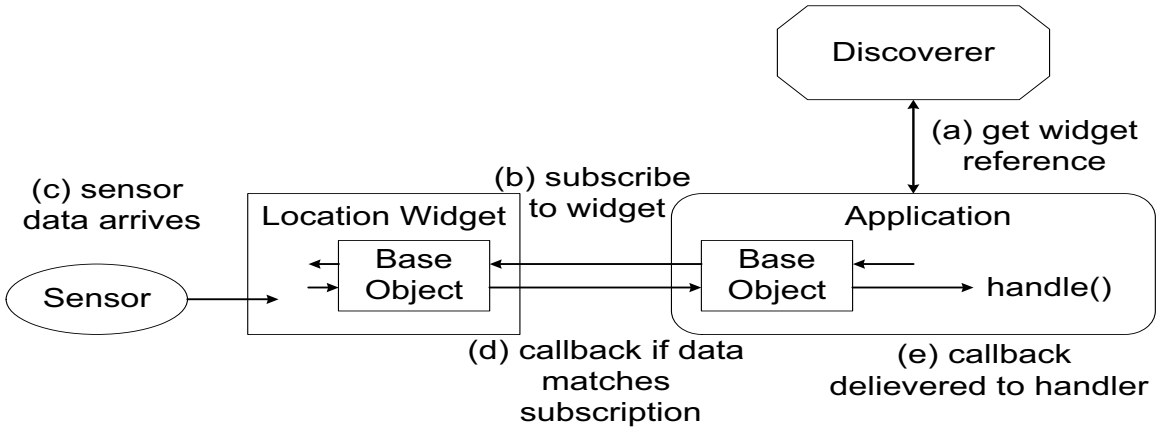


Figure 8
Exemple d'interaction entre une application et un widget de contexte, d'après [Dey 01].

La figure 8 montre un exemple d'interaction entre l'application et un widget de contexte. Un mode requêtes/réponses est également disponible.

Analyse

Les *context-widgets* offrent un accès simple aux capteurs. Les interpréteurs donnent un sens aux données capturées et les agrégateurs font le lien entre les *context-widgets* et les applications sensibles au contexte. Les *context-widgets* ne sont que des pilotes logiciels de capteurs. Ils ne fournissent pas de sens aux informations nécessitant l'existence d'interpréteurs. Les agrégateurs regroupent le gros du travail puisqu'ils synthétisent de nouvelles informations à partir des informations fournies par les *context-widgets*.

Au bilan, la context toolkit est une infrastructure simple à comprendre, construite autour d'un modèle centré processus faiblement centralisé. Elle couvre les trois niveaux d'abstraction de la pyramide : les *context-widgets* pour le niveau de capture, les interpréteurs pour le niveau interprétation et les agrégateurs pour le niveau identification. Elle offre en plus un service d'actions basé sur les actuators.

Le service d'historique est disponible au niveau des *context-widgets* seulement, la découverte se fait par le biais du discoverer, la sécurité et la protection de la sphère privée ne sont pas traitées. Le modèle n'inclut pas de mesures de la qualité (métadonnées). L'application doit donc faire conconfiance ou pratiquer ces mesures elle-même.

4.2. L'ARCHITECTURE DE SCHMIDT

Comme le montre la figure 9, l'architecture de Schmidt s'appuie sur un modèle en couches. La couche du niveau le plus bas est composée des capteurs nécessaires à l'application. Chaque capteur i est modélisé par une

fonction Si ayant pour unique variable le temps t. Le résultat est un signal Xi appartenant à l'ensemble des signaux pouvant être émis par i, soit :

$$S_i : t \rightarrow X_i \qquad (1)$$

Le niveau intermédiaire est composé d'entités appelées indices (cues). Les indices sont des abstractions des capteurs physiques de la couche inférieure. À un capteur peuvent correspondre plusieurs indices. Nous retrouvons ici la notion d'interpréteur du Context toolkit. Chaque indice j est modélisé par une fonction Cj ayant pour paramètre l'ensemble des signaux Si émis entre l'instant t-n et l'instant t par le capteur i associé à j. Cette fonction renvoie un résultat Yij qui exprime une sémantique donnée. Soit :

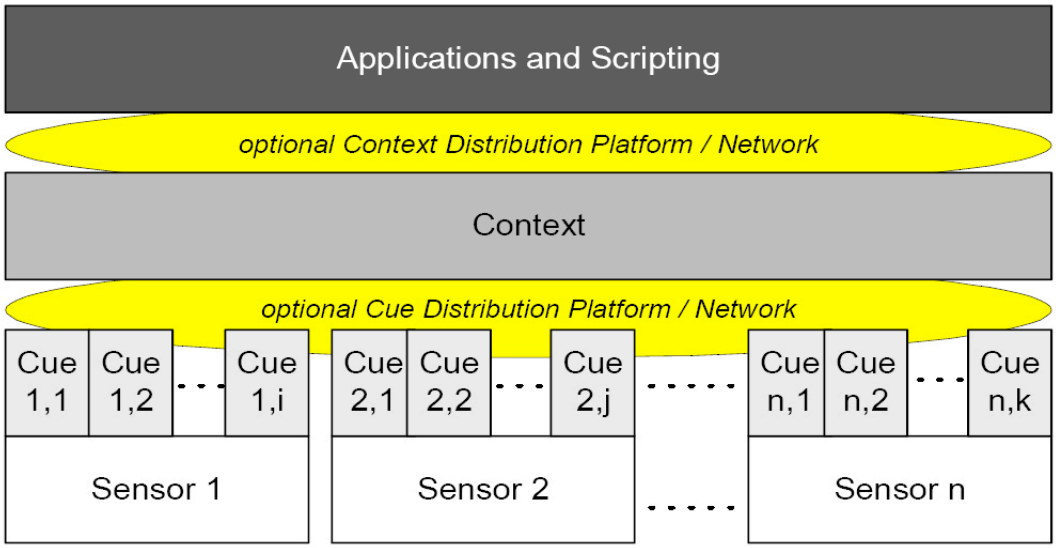
$$C_j : S_i(t) \times S_i(t-1) \times \ldots \times S_i(t-n) \rightarrow Y_{ij} \qquad (2)$$

La strate de niveau supérieur regroupe l'ensemble des indices à partir desquels elle détermine le contexte actuel et en notifie l'application cliente. Cette strate se modélise aussi par une fonction mathématique T ayant pour paramètre l'ensemble des indices efficients à l'instant t. Le résultat fourni est un vecteur h(v,p) composé d'une valeur v symbolique d'une situation et d'une valeur p indiquant la valeur de confiance associée à v. Soit :

$$T : C_0(S_0,t) \times C_1(S_0,t) \times \ldots \times C_k(S_0,t) \ldots C_0(S_i,t) \times C_1(C_1,t) \times \ldots \times C_m(S_i,t) \rightarrow h \qquad (3)$$

Le vecteur h définit une surface, fonction du temps, qui est comparée à des surfaces de situations de référence (voir la figure 10) : trois axes

Figure 9
Architecture en couches de Schmidt d'après [Schmidt 02] page 79.



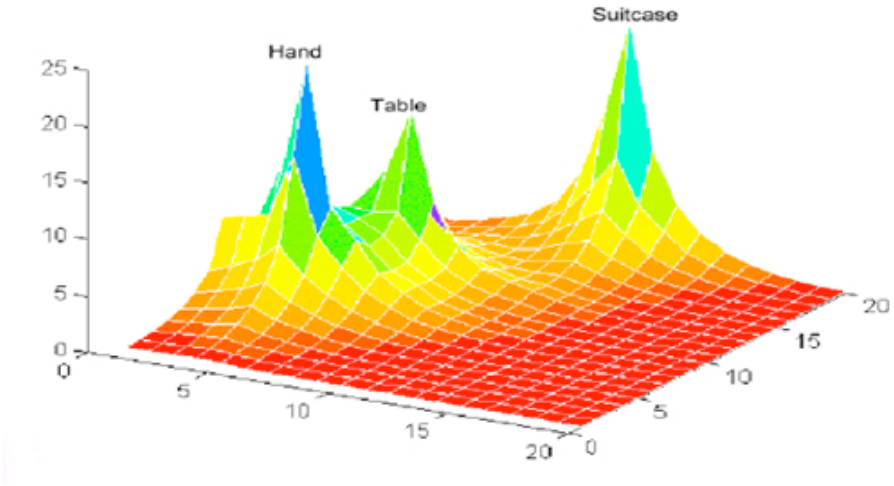


Figure 10
Fonction T dans l'architecture de Schmidt.
Les pics représentent les entités identifiées. Ici, une table, une main et une valise.

représentent respectivement la valeur de v, symbole de l'environnement, un axe p graduant une échelle de confiance et un axe t représentant le temps.

Analyse

Comme pour la context-toolkit, la proposition de Schmidt couvre les 3 niveaux d'abstraction de la pyramide :

- le niveau capteur correspondant à la couche capture de la pyramide,
- le niveau indice pour la couche transformation
- le niveau contexte pour la couche d'identification.

Contrairement à la context-toolkit, Schmidt s'intéresse à la qualité des données avec des valeurs de confiance.

Inversement,

- La force du modèle (son fondement mathématique) pourra être perçue comme une faiblesse car éloigné du concepteur logiciel qui s'attend à un modèle plus proche de ses préoccupations de mise en œuvre.
- On regrettera également que les informations sur la qualité des données fassent leur apparition au niveau identification seulement. C'est que l'empilement des machines abstraites est étanche : le programmeur n'a pas accès aux couches inférieures.
- L'identification du contexte nécessite la définition préalable de surfaces de situation de référence. Cela semble assez compliqué pour des cas simples comme par exemple, savoir s'il fait chaud.

Au bilan, le modèle de Schmidt :

- présente un fondement mathématique fort,
- introduit la notion de qualité de service,

- structure le problème en trois niveaux d'abstraction étanches et fortement couplés,
- mais peut sembler complexe et lourd pour le traitement de cas simples.

4.3. LE CONTEXT-HANDLING COMPONENT

Sur le plan architectural, ce modèle se positionne d'emblée comme une extension du modèle Arch, l'un des modèles d'architecture de référence en Interaction Homme-Machine (voir figure 11).

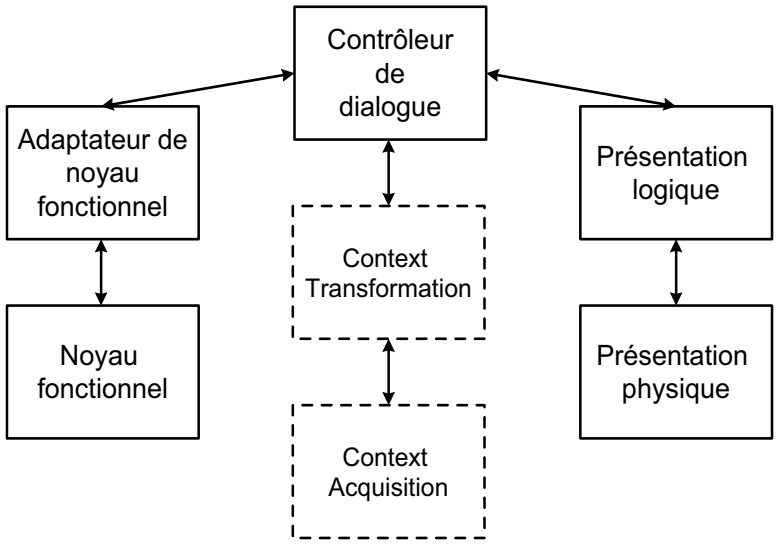
Pour rappel, Arch propose, pour tout système interactif, la décomposition fonctionnelle suivante: aux deux pieds de l'arche, le *noyau fonctionnel* (ou application) et la *présentation physique* des concepts et services réalisés dans le *noyau fonctionnel* (interface graphique). En clé de voûte, le *contrôleur de dialogue*, responsable de l'enchaînement des tâches utilisateur. De chaque côté, un adaptateur (l'*adaptateur de noyau fonctionnel* et le composant de *présentation logique*) qui minimise les dépendances du *contrôleur de dialogue* avec le *noyau fonctionnel* et la *présentation physique*.

Comme le montre la figure 11, le modèle de Salber vient se brancher sur le contrôleur de dialogue qui instruit à son tour la branche présentation et/ou la branche noyau fonctionnel de l'arche.

Le modèle de Salber est structuré en deux niveaux d'abstraction reprenant ainsi les fondements de Arch (découplage du physique et du logique) :

- Le niveau *Context Acquisition* qui prend en charge la capture de l'information dans le monde physique à l'aide de capteurs.

Figure 11
Extension du modèle Arch avec, en pointillé, le Context-handling Component.



- Le niveau *Context Transformation* qui extrait les informations significatives en provenance du Context Acquisition et les transmet au Contrôleur de Dialogue au niveau d'abstration idoine.

A leur tour, ces deux composants sont constitués d'un ensemble de composants appelés *Context handling* component dont l'interface de communication, représentée dans la figure 12, comprend 3 flots d'entrée et 3 flots de sortie.

- Les flots d'entrée incluent un flot de donnée (*Data In*), un flot de métadonnées (*Meta Data In*) et un flot de contrôle (*Control*).
- Les flots de sortie sont composés d'un flot de donnée (*Data Out*), d'un flot de métadonnées (*Meta Data Out*) et d'un flot d'action (*Action*).

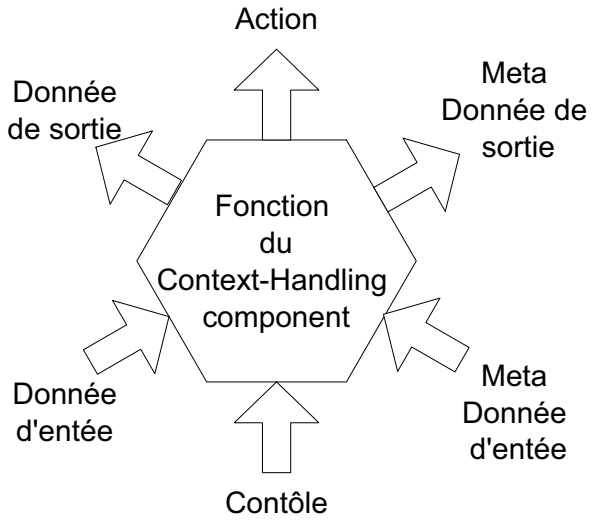
Les flots *Data* (In ou Out) correspondent aux informations reçues ou émises par le composant tandis que les flots *Meta-Data* (In ou Out) livrent les mesures de qualité sur les informations reçues ou émises par le composant. Le flot *Control* permet à d'autres composants de modifier le comportement interne du composant (supprimer l'activité de ce composant reconnu non fiable, changer les paramètres de fonctionnement, etc.). Réciproquement, le flot *Action* sert à produire une action sur l'environnement (changer la vitesse d'un moteur, éclairer la lumière …).

Le modèle précise les règles de composition des *Context-handling* components : les flots *Data* (et *Meta-data*) sont connectables entre eux, c'est-à-dire un flot *Out* est relié à un flot *In*. Il en va de même pour un flot Action qui vient alimenter un flot *Control*.

Analyse Ce modèle présente des avantages sur les modèles analysés jusqu'ici :

- Il s'inscrit dans une architecture de référence en Interaction Homme-Machine. Il est donc facile d'accès aux développeurs de systèmes interactifs,

Figure 12
Interface d'un «Context-handling Component»



- Il véhicule la mesure de qualité des données à tous les niveaux d'abstraction (*MetaData*).

Inversement,

- Le modèle regroupe en deux niveaux d'abstraction les trois niveaux de la pyramide sans indiquer clairement la nature des frontières entre ces deux niveaux,
- Les *Meta-data* et les *Data* transitent dans des flots distincts qui risquent donc d'être désynchronisés. Or, une donnée et sa métadonnée sont fortement couplées. Elles doivent donc être communiquées ensemble.
- Aucun service transversal n'est proposé.

4.4. LA STRATHCLYDE CONTEXT INFRASTRUCTURE

La *Strathclyde Context Infrastructure* [Glassey 03] est organisé en deux niveaux distinct.

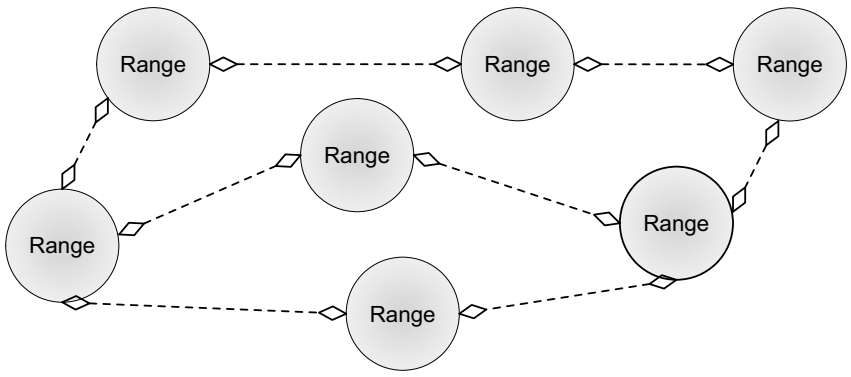
- Au niveau le plus haut, un recouvrement de réseau de noeuds partiellement reliés appelé SCINET (voir la figure 13).
- Au niveau inférieur, le contenu de chaque nœud appelé range. Un nœud produit, contrôle et utilise de l'information contextuelle.

SCINET prend en charge la gestion des interactions entre les ranges en sorte de fournir des informations contextuelles au-delà d'un range : des composants d'un range sont susceptibles d'être intéressés par des informations contextuelles provenant d'autres ranges.

La technique du recouvrement de réseau qu'adopte SCINET favorise le passage à l'échelle (scalability) [Wang 04], de même améliore la robustesse qui n'auraient pas été possibles avec un arrangement hiérarchique des noeuds.

Un range correspond à un espace qui peut être décrit en termes d'informations physiques (un ensemble de salles adjacentes, un étage de

Figure 13
SCINET : StrathClyde Contexte Infrastructure NETwork.



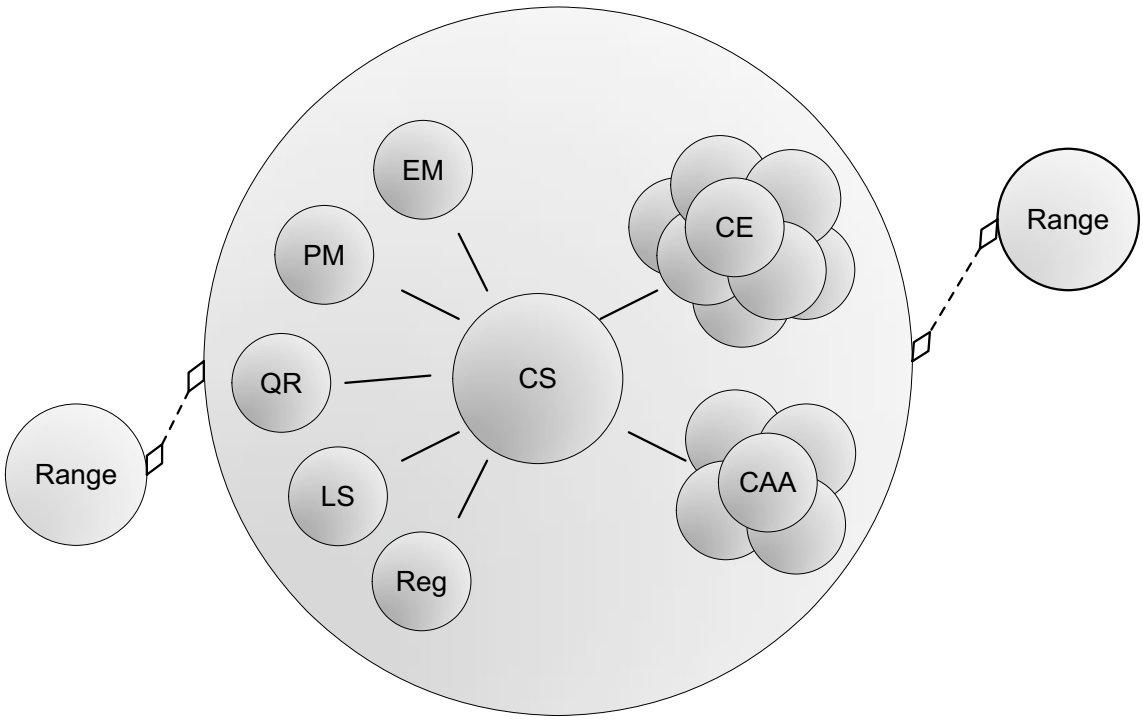
bâtiment) et/ou en termes logiques par la couverture de fonctionnement efficace d'un type particulier de réseau (par exemple un réseau sans fil).

Comme le montre la figure 14 chaque range s'autogouverne autour d'un serveur individuel de contexte (CS), centre fonctionnel pour les autres composants du range. Le serveur de contexte contrôle les composants du range et fournit les moyens de communiquer avec les autres ranges du SCINET. Du point de vue des applications clientes, il constitue le point d'accès à l'infrastructure.

Une entité de contexte (CE) est un composant logiciel représentant une entité du range. Il maintient un profil de cette entité ainsi que les services que cette entité peut fournir à d'autres entités. Toutes les CE sont enregistrées dans le CS quand elles arrivent dans le range et effacées à leur départ. Par exemple, pour l'application CAPA décrite dans [Glassey 03], les CE correspondent aux différentes imprimantes. L'application CAPA, d'impression localisée, peut, une fois informé de la position de l'utilisateur, choisir à quelle CE, elle doit envoyer le document à imprimer.

Une application sensible aux contextes (CAA) est une application qui a la capacité de tirer parti de l'information contextuelle proposée par l'infrastructure. Lorsqu'elle rejoint le range, la CAA interroge le CS qui analyse sa requête. Une fois la requête analysée, le CS met en place les CE

Figure 14
Détail de l'intérieur d'un *range*.



nécessaires à la résolution du service demandé. Les CE étant déployées, il ne reste plus qu'à prévenir la CAA et lui indiquer où trouver ce qu'elle recherche.

Les utilitaires de contexte (CU) sont des services spécialisés qui aident le CS dans la gestion d'un range. Par exemple :

- ***Range Service*** : Responsable de la détection de l'arrivée et du départ des entités d'un range.
- ***Query Resolver*** : Transforme une requête de haut niveau d'abstraction en une liste de services que les CE du range peuvent rendre.
- ***Location Service*** : Manipule les tâches liées à un range.
- ***Profil Manager*** : Permet l'accès et la mise à jour des profils d'entités maintenus par les CE.
- ***Event Mediator*** : Contrôle l'établissement, l'entretien et la suppression des liens entre les CE et les CAA. Ces liens correspondent aux demandes de souscription de services émises par les CAA.
- ***Registrar*** : Maintient une vue précise de toutes les entités du range.

Analyse

Au regard des critères retenus, la Strathclyde Context Infrastructure présente les avantages suivants :

- elle propose une couverture réseau au niveau global via SCINET et au niveau local pour chaque range. Ces derniers sont architecturés autour d'un modèle faiblement centralisé (une grande partie des charges du CS est répartie dans les CU).
- Par définition, les CU sont de bons véhicules pour étendre les services transversaux.
- Les CE, architecturées autour d'un modèle en flot de données, fournissent aux CAA des données après que ces dernières aient souscrit à leurs services. Par conséquent, les CAA ne sont dépendante du CS que pour la découverte de service. Une panne du CS ne bloquera pas les CE ni les CAA déjà en fonctionnement.

Toutefois, la Strathclyde Context Infrastructure a aussi quelques lacunes :

- Ici, les métadonnées décrivent les interfaces des CE. Elles ne servent pas à caractériser la qualité des données fournies par les CE.
- Les niveaux d'abstraction Capture et Transformation de la pyramide sont ici fusionnés. Or une distinction entre ces niveaux est essentielle pour une bonne répartition des tâches de conception : la conception d'un pilote matériel/logiciel permettant l'interfaçage avec un capteur demande des compétences différentes que celles nécessaires aux calculs de nouvelles informations à partir de données numériques.

5. Synthèse

Les infrastructures de contexte doivent être fiables alors qu'elles interfacent des capteurs aux comportements imprévisibles et qu'elles s'appuient sur des technologies réseau dont la disponibilité n'est pas toujours garantie. Ces infrastructures doivent être flexibles pour répondre aux variations du milieu ambiant : arrivée et départ opportunistes de ressources de toutes sortes, couvertures réseau de local à global, extensibilité des données et des services et notamment ceux liés à l'historique, la sécurité et la protection de la sphère privée.

L'analyse de l'état de l'art que synthétise le tableau 1, appelle les constats suivants :

- Les niveaux d'abstraction de la pyramide et ses services transversaux constituent une représentation (presque) canonique de la couverture fonctionnelle des infrastructures de contexte : les niveaux d'abstraction sont pour la plupart du temps bien couverts par les infrastructures actuelles (colonne de gauche du tableau). On assiste presque systématiquement à la distinction entre le contexte "capturé", les transformations des observables du contexte par interprétation et inférence, et l'expression des demandes de services exprimées dans un langage de haut niveau comme dans Liquid [Hong 01], ou en logique du première ordre comme dans Gaia Context [Hess 03]. A contrario, les services transversaux comme la sécurité et l'historique sont largement négligés ou traités à un seul niveau d'abstraction (colonne de droite du tableau). Par exemple, la *Strathclyde Context Infrastructure* ne fournit aucun de ces deux services alors que la *Context-Toolkit* ne propose un système d'historique qu'au niveau de ces *Context-Widget* (Niveau Capture).
- Les infrastructures de contexte se répartissent en deux courants selon que le contexte est vu d'abord comme un espace d'informations ou d'abord comme un processus. Ces deux visions se traduisent par des classes d'architecture allant du fortement centralisé au fortement décentralisé (colonne 5 du tableau). La première répond mal au requis de robustesse, mais est plus facile à déployer que les solutions fortement décentralisée. Aussi assiste-t-on à des solutions hybrides comme la Strathclyde Context Infrastructure qui observe une architecture localement centralisée avec la notion de range couplée à une couverture réseau globale par la technique de recouvrement de réseaux. De manière générale, il est raisonnable de penser qu'à chaque niveau d'abstraction de la pyramide peut correspondre un style spécifique. Il semblerait que les couches basses de capture soient mieux servies avec une vision processus et composants, alors que les couches hautes du raisonnement sur le contexte soient plus faciles à appréhender par une approche centrée donnée de type t-uple spaces.

	(1) Niveaux d'abstraction	(2) Couverture réseau	(3) Echanges des données	(4) Méta-données / méta-interfaces	(5) Classe architecturale (Style)	(6) Services transversaux
CTK [Dey 01]	Capture Tranformation Identification	Locale	Souscription Requête	Non / Meta Interface (planifié)	Faiblement centralisé (processus)	Historique
Schmidt [Schmidt 99]	Capture Tranformation Identification	-	-	Oui / Non	-	-
Salber [Salber 01]	Capture Tranformation	-	-	Oui / Meta Interface	Décentralisé (machine abstraite)	aucun
SCI [Glassey 03]	Capture Identification	Locale & Globale	Souscription	Non / Meta Interface	Faiblement centralisé (processus)	Autres
Irisnet [Gibbons 03]	Capture Identification	Globale	Requête	Non / Meta Interface	Fortement centralisé	Historique Sécurité
Confab [Hong 01]	Capture Identification	Locale & Globale	Requête	Faible (uniquement un facteur de confiance) / Meta Interface	Fortement centralisé (centré donnée)	Historique Sécurité
Aura (CIS) [Steenkiste 03]	Identification	Locale & Globale	Requête	Oui / Meta Interface	Fortement centralisé (centré donnée)	Historique
BT node [Beutel 03]	Capture	Globale	-	Non / Non	Décentralisé	Migration
EgoSpace [Julien 04]	Capture Identification	Locale & Globale	Souscription	Non / Non	Décentralisé	Sécurité Migration
CAABWS [Chaari 05]	Capture Tranformation Identification	Globale	Souscription	Oui / Meta Interface	Faiblement centralisé	Historique
Gaia Context [Hess 03]	Capture Tranformation Identification	Locale	Souscription Requête	Non (planifié) / Non	Fortement centralisé (centré donnée)	Autres

Table 1 : Récapitulatif des infrastructures en fonction des critères d'évaluations.

- La plupart des infrastructures ont une couverture réseau locale. La Strathclyde Context Infrastructure et sa technique de recouvrement réseaux SCINET, de même IrisNet font exception. Par exemple, Irisnet est basée sur un système de bases de données réparties simulant un système fortement centralisé.

- La découverte de services contextuels s'appuie le plus souvent sur l'existence d'un service d'enregistrement. C'est le cas du *discoverer* de la ContextToolkit ou les serveurs *infospace* de Confab [Hong 01]. Ce procédé est simple, mais l'existence d'un serveur fragilise la disponibilité de l'infrastructure ou encore limite le passage à l'échelle (colonne 2). La *Strathclyde Context Infrastructure*, qui dispose de context servers interconnectés sur la Planète, propose une solution intéressante à ce problème : un service non disponible dans le range peut être fourni par un range distant. Nous verrons au chapitre III, une autre approche résolument pair à pair qui élimine le serveur local.
- Les facultés d'autorégulation et de reconfiguration participent à la fois à flexibilité et à la fiabilité. À cet égard, l'approche par composants constitue une réponse satisfaisante à la flexibilité sous réserve que service et communication soient implémentés de manière indépendante. Sans cette séparation des aspects, le logiciel n'est pas reconfigurable. Si, de plus, les composants sont dotés d'interfaces de contrôle, alors ceux-ci deviennent contrôlables et inspectables [Kiczales 92]. La flexibilité, mais aussi la fiabilité peuvent s'en trouver améliorées. Appliquées aux infrastructures de contexte, les interfaces de contrôle permettent d'exprimer la qualité de service en termes de métadonnées, de commander la migration de code, ou de remplacer un service par un autre. Par exemple, les applications nécessitant un service de localisation comme GeoCast [Dürr 03], se fondent sur un modèle explicite de l'espace géographique. En revanche, le couplage de ressources d'interactions présenté dans [Barralon 05] n'utilise que la notion de proximité. Ces deux systèmes mettent en jeu des *Where*, mais emploient différentes échelles et modèles pour cette notion. Sans flexibilité, l'infrastructure ne peut fournir cette diversité de besoins pour un même concept (dans notre exemple, celui de *Where*). Or, très peu d'infrastructures, hormis l'infrastructure CAABSW [Chaari 05] et les Contextual Information Providers (CIS) d'Aura proposent à la fois les métadonnées et les méta-interfaces (colonne 4). IrisNet ne gère pas de métadonnées mais permet aux applications d'exprimer une tolérance sur les données, ou, avec leur notion de senselet, de définir des filtres sur les données brutes (par exemple, transformer une image captée par une caméra de façon à protéger la sphère privée) [Gibbons 03].
- Le contexte, nous l'avons vu, est en perpétuelle évolution. Le service d'historique est une façon de mémoriser l'évolution. L'adjonction de nouveaux services en est une autre. Mais cet apport dynamique ne doit pas pour autant perturber le fonctionnement de l'infrastructure. L'approche composant-connecteur permet cette extension, encore faut-il que les données échangées soient auto-descriptives. On constate que, sur ce point, toutes les infrastructures du tableau sont satisfaisantes.

Toutes utilisent un langage extensible façon SOAP [SOAP] de description du format des données échangées.

- Peu d'infrastructures offrent plusieurs modes d'échange des données (colonne 3). Le mode *Requêtes/Réponses* correspond à un besoin d'information précise alors que le mode *Souscriptions/Notifications* satisfait les cas où les applications cherchent à acquérir des informations aux occurrences hypothétiques ou inconnues.
- Les services transversaux restent les parents pauvres des infrastructures actuelles. Trop peu d'historique et de sécurité, notamment. Si ces services sont présents, ils ne sont pas pensés sur tous les niveaux d'abstraction. Il faut reconnaître qu'en ce domaine, les travaux sont balbutiants. Par conséquent, les requis techniques ne sont pas définis. La technique de publication des informations personnelles filtrées, initiée pour les collecticiels [Salber 95], est reprise par certains systèmes ambiants. Par exemple, dans myCampus, Marie autorise l'utilisateur de la localiser par une adresse précise alors que Jim n'exporte à cet utilisateur que le nom d'une ville voisine (section 7 du chapitre I). A contrario, si trop peu d'informations personnelles sont publiées, le système ambiant peut-il encore rendre un service utile ? Le problème est posé [Etzioni 99], [Garfinkel 01], [Lahlou 05]. Des guides de recommandations sont produits sans pour autant proposer des solutions techniques satisfaisantes [Lahlou 03].

Cette revue de l'état de l'art appelle de nouvelles solutions. Je propose le contexteur. Cette abstraction logicielle reflète les fondements conceptuels de l'ontologie du chapitre I. Sa mise en œuvre vise le respect des critères d'analyse identifiés dans ce chapitre, avec comme couverture fonctionnelle, les niveaux capture et transformation de la pyramide. Le reste de cette thèse est dédiée à cette contribution.

Chapitre III *Le Contexteur : Modèle conceptuel*

La vie est capable de se reproduire, d'utiliser l'énergie, d'évoluer, de mourir... La matière, elle, est inerte, immobile, incapable de se reproduire. En regardant, d'un coté, le monde vivant et, de l'autre, le monde minèral, on ne pouvait s'empêcher de les considérer comme opposés. Mais jadis, on ne savait pas que les molécules étaient faites d'atomes, ni que les cellules étaient faites de molécules. Alors, on expliquait que la vit était apparue sur terre par la volonté des dieux ou par un hasard extraordinaire. C'était en fait une manière de cacher son ignorance.

De Rosnay, 1996, La plus belle histoire du monde. Acte 2 - Scène 1 : La soupe primitive, p67

La pyramide du chapitre II explicite les niveaux de traitement du contexte : capture, transformation, identification et adaptation. Chaque niveau, nous l'avons vu, correspond à la mise en œuvre des concepts clefs de l'ontologie du chapitre I : observables numériques (niveau capture), observables symboliques, entités, relations et rôles (niveau transformation), et réseaux de contextes et de situations (niveau identification). De manière orthogonale à ces couches fonctionnelles, le contexte, qui peut s'envisager comme un état et/ou comme un processus[1], donne lieu à des mises en œuvre aux styles architecturaux, soit centrés données, soit centrées processus. Dans la synthèse du chapitre II, nous évoquions la possibilité de styles hétérogènes qui allient ces deux vues. Nous relevions également le peu d'attention accordée jusqu'ici au problème du passage à l'échelle.

En réponse, je propose le contexteur et son extension, le répéteur. Le contexteur est une abstraction logicielle orientée processus qui vise la couverture des niveaux capture et transformation de la pyramide. Le répéteur étend le service d'une famille de contexteurs au-delà de sa portée locale. L'orientation processus tient à la nécessité de gérer les fluctuations du réseau et des capteurs par une architecture flexible capable de passer à l'échelle. La vue orientée données, si elle doit se révéler utile, est laissée au niveau identification de l'infrastructure de contexte.

Ce chapitre est structuré selon les unités architecturales de l'infrastructure proposée : en section 1, le transducteur et le capteur qui alimentent les contexteurs en informations contextuelles d'origine physique. Puis, le contexteur est présenté en section 2 suivi en 3, par la description du répéteur qui assure le passage à l'échelle. La section 4 présente les limites de cette proposition.

1. Transducteur et Capteur

Les transducteurs et les capteurs forment la base de la chaîne contextuelle de l'infrastructure de contexteurs. Ils prennent en charge la mesure des propriétés physiques (observables) du monde réel. Une fois ces propriétés mesurées, ils les expriment dans une unité connue de manière à ce qu'elles puissent être exploitées par toute entité numérique, et notamment les contexteurs.

Commençons par la description du cœur d'une unité de capture : le transducteur, puis étendons cette description au reste du capteur, pour

1. Context as data vs context as process [Coutaz 05]

finir par les propriétés opérationnelles d'un capteur et sa couverture fonctionnelle.

1.1. TRANSDUCTEUR

Définition

Le Comité du langage scientifique définit le transducteur ainsi :

> *Tout système ou dispositif intermédiaire qui, recevant de l'énergie en provenance de plusieurs systèmes ou milieux, fournit à plusieurs autres systèmes ou milieux une énergie de même nature ou de nature différente de l'énergie reçue, en correspondance avec celle-ci" [Hachette Multimédia].*

De cette définition, il convient de distinguer les transducteurs d'entrée qui reçoivent de l'énergie du milieu pour la fournir à un système et les transducteurs de sortie qui font l'opération inverse, du système vers le milieu. Dans cette étude, l'accent est mis sur les transducteurs d'entrée avec la définition suivante :

> *Un transducteur est un dispositif matériel capable de détecter un phénomène physique en vue de le représenter sous la forme d'un signal analogique.*

Les transducteurs sont largement utilisés en électricité, en électronique et leurs applications avec notamment la famille des cristaux piézoélectriques (dont les quartz font partie) qui transforment un flux d'énergie mécanique en énergie électrique (transducteur électromécanique). On distingue deux grandes catégories de transducteurs : les transducteurs passifs et les transducteurs actifs.

Transducteurs passifs et actifs

Un transducteur est *passif* si l'énergie qu'il délivre provient exclusivement de l'énergie qu'il reçoit. Ainsi, un filtre électrique formé uniquement de résistances et de condensateurs est un transducteur passif : il délivre une fraction plus ou moins grande de l'énergie électrique qui lui est fournie par le générateur placé entre ses bornes d'entrée.

Un transducteur est *actif* si l'énergie qu'il délivre provient au moins en partie de sources autres que celles qui lui fournissent de l'énergie. Un amplificateur est un transducteur actif car il reçoit de l'énergie d'un dispositif relié à ses bornes d'entrée (générateur, capteur, antenne radioélectrique) et de plus, il restitue une énergie beaucoup plus grande entre ses bornes de sortie. Ces énergies sont transportées par des courants de même fréquence. L'amplificateur ne produit pas l'énergie qu'il cède, mais il est conçu pour permettre la fourniture de cette énergie par une alimentation incorporée dans l'appareil. C'est la constitution de l'amplificateur qui fait qu'il y a correspondance entre l'énergie de sortie et l'énergie d'entrée.

Cependant la détection d'un phénomène physique ne suffit pas aux systèmes numériques. Les systèmes sensibles au contexte doivent

construire une représentation numérique de l'univers. Aussi, j'introduis la notion d'unité de capture que je mets en correspondance avec la notion de capteur.

1.2. UNITÉ DE CAPTURE ET CAPTEUR

Nous trouvons ci-dessous la définition d'une unité de capture suivie de notre acception du terme capteur. Ensuite viennent les propriétés permettant de caractériser les qualités fonctionnelles et opérationnelles d'un capteur.

Unité de capture définition

La figure 1 montre la structure d'une unité de capture. Autrement dit :

Une unité de capture est l'association d'un transducteur et d'une fonction de calcul qui permet de quantifier l'énergie reçue du transducteur sous forme numérique en fonction d'un étalonnage.

Capteur définition

Dans son acception courante, un capteur désigne un transducteur d'entrée:

Ce dispositif est capable de fournir des grandeurs physiques comme la lumière, le son, la température, la force, la pression, etc., ou chimiques, ainsi que leurs variations [Webencyclo].

Pour chaque type de grandeur (mécanique, électrique, magnétique, thermique, radiative ou chimique), le signal capté peut comporter plusieurs variables, par exemple, pour la lumière, la phase, l'amplitude, la longueur d'onde, la polarisation, etc.

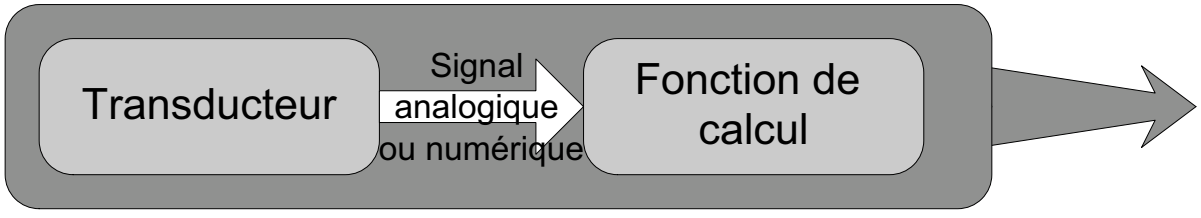
Un système sensible au contexte, qui doit mesurer les phénomènes physiques de l'univers, doit comporter des capteurs mais ces capteurs doivent être encapsulés par une unité de traitement.

Aussi, pour les besoins de cette étude :

Un capteur est une unité de capture dont la fonction de calcul est un driver ou composant logiciel qui interface le transducteur d'entrée avec la plate-forme d'accueil.

Le choix d'un capteur et son intégration dans un système dépend de ses propriétés opérationnelles et de sa couverture fonctionnelle.

Figure 1
Structure d'une unité de capture.



Propriétés opérationnelles d'un capteur

Une propriété opérationnelle caractérise la performance d'une entité sous forme, si possible, de métrique. Bérard dans [Bérard 99] présente les qualités requises d'un système de capture fondé sur la vision par ordinateur. Je reprends à mon compte ces travaux que je complète.

Il convient de distinguer les caractéristiques suivantes d'un capteur :

- La *résolution* est la plus petite variation du phénomène physique à laquelle le capteur est sensible. Si la variation du phénomène est inférieure à la résolution du capteur, celle-ci ne sera pas relevée.
- La *latence* est le temps écoulé entre la présentation d'un stimulus au capteur et le retour d'information qui résulte du traitement de ce stimulus.
- Le *pas d'échantillonnage* est le temps minimal entre deux captures. En conséquence, toute variation du phénomène physique, quelle que soit son importance, survenant dans cet intervalle n'est pas relevée par le capteur.
- La *stabilité* est la capacité du capteur à ne pas indiquer de variation si le phénomène physique ne varie pas.
- La *portée* est la distance maximale à laquelle le phénomène physique est détectable par le capteur.
- Le *champ d'action* est l'angle (centré sur le capteur) à l'intérieur duquel le phénomène physique peut être observé.
- La *taille* désigne de manière simplifiée à la fois le volume, le poids et la forme du capteur. Cette caractéristique est déterminante lorsque le capteur doit être embarqué sur un dispositif mobile de petite taille.
- L'*autonomie* est la durée de fonctionnement du capteur sans alimentation électrique. Tout capteur qui nécessite une alimentation électrique a une autonomie nulle. Plus rarement, les capteurs pour lesquels le signal du phénomène physique suffit à leur fonctionnement, ont une autonomie infinie.
- L'*orientabilité* est la capacité du capteur à changer son champ d'action. Elle peut être manuelle, motorisée ou automatique.
- Le *calibrage* désigne le réglage des paramètres du capteur qui, à leur tour, déterminent les conditions du fonctionnement du capteur. Par exemple, pour un capteur de son, le paramètre "niveau de son" détermine le niveau sonore ambiant minimal auquel le capteur doit être sensible. Pour une caméra, le focus ou l'ouverture du diaphragme sont des paramètres de calibrage. Le calibrage peut être automatique. Dans ce cas, le délai de calibrage est une propriété pertinente du capteur. Le calibrage peut être manuel, c'est-à-dire nécessite une intervention extérieure, ou inexistant. Le calibrage peut aussi être semi-automatique: il n'est pas déclenché automatiquement, mais une fois l'ordre donné, le calibrage est automatique.
- La *durée de vie* est la durée (estimée) de fonctionnement du capteur avant que celui-ci ne tombe en panne. Une fois la durée de vie

dépassée, le capteur peut fonctionner, mais les caractéristiques opérationnelles du capteur ne sont plus garanties. Par exemple, l'autonomie peut être réduite ou alors la résolution peut varier.

Ces propriétés permettent d'alimenter les métadonnées d'un contexteur.

Couverture fonctionnelle d'un capteur

La couverture fonctionnelle d'une entité désigne l'ensemble des services qu'elle est capable de fournir. Concernant les capteurs, je propose de distinguer les fonctions suivantes (la liste n'est pas exhaustive) :

- La *détection* est la capacité du capteur à repérer l'existence d'un type de phénomène physique. Par exemple, la présence d'une entité mobile dans la pièce.
- L'*identification* est la capacité du capteur à reconnaître une instance d'une classe de phénomènes physiques. Par exemple, l'entité mobile en question est un chat.
- Le *suivi* est la capacité du capteur à suivre à tout instant le phénomène physique détecté ou identifié.

Ces fonctions pourront servir à décrire le service contextuel fournis par une infrastructure. Par exemple un service de localisation à base de détecteurs infrarouge permettra de faire de la détection de personnes alors qu'un système similaire basé sur des capteurs RFID permettra, en plus de la détection, de faire de l'identification.

Ayant introduit le composant de base, le capteur et ses propriétés, voyons comment, avec la notion de contexteur, nous pouvons généraliser le processus de capture du contexte par un système numérique.

2. *Contexteur*

Un contexteur est une abstraction logicielle qui fournit une valeur d'observable. Cette section en présente la définition, puis indique comment composer les contexteurs en nouvelles unités fonctionnelles.

2.1. DESCRIPTION

Comme le montre figure 2, un contexteur comprend trois classes d'éléments : des entrées, des sorties, et un noyau fonctionnel.

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

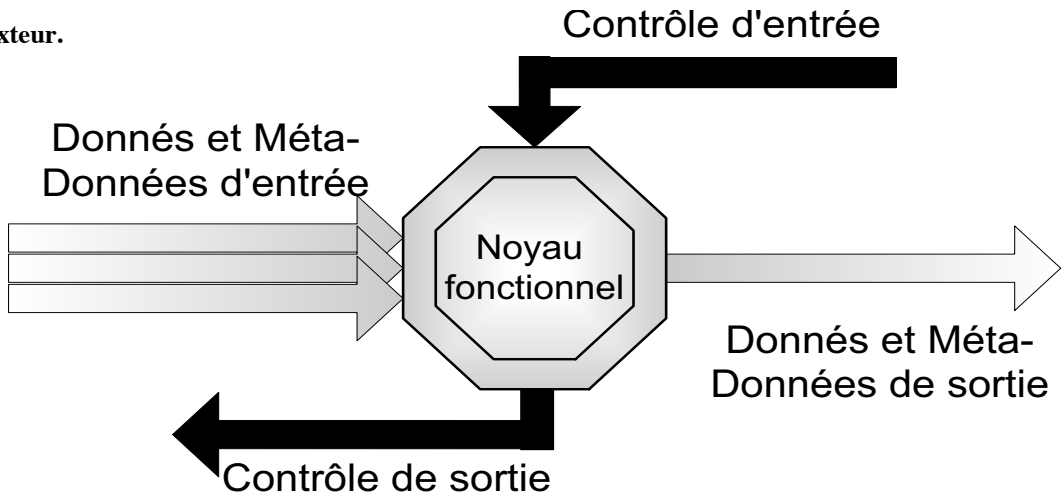
caractéristiques des capteurs. Ce dernier point est important : alors qu'en IHM conventionnelle, les données sont certaines (un événement souris ne peut faire de doute), il en va tout autrement avec le monde des capteurs et du raisonnement par inférence. Les données d'entrée sont multiples. En effet, les entrées comportent plusieurs classes de données d'entrée et chaque classe peut être représentée par plusieurs instances. Le fait d'avoir plusieurs classes d'entrée permet aux contexteurs de fusionner des données. De même, les différentes instances de chaque classe permettent d'améliorer les métadonnées par fusion de métadonnées.

- 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 sa Qualité de Service (QoS).

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, ou la couche identification). Chaque donnée de sortie est assortie de métadonnées qui en expriment la qualité. Il n'y a qu'une classe de données de sortie par contexteur correspondant au service qu'il fournit. Plusieurs instances de cette classe peuvent néanmoins co-exister, chaque instance correspondant à un client du contexteur. Les instances peuvent différer en fonction des modes de transmission des données entre le contexteur et son client (plus de détails sont présentés sur ces modes de transmission dans la suite du document), ou en fonction de l'unité de représentation de la donnée. Par exemple, un contexteur de température pour fournir cette information en degrés Celsius à un client, en degrés Fahrenheit à un second client et en Kelvin à un troisième.

Figure 2
Modèle du contexteur.



- 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 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.

Le corps fonctionnel désigne la fonction que le contexteur remplit. Nous trouvons dans [Rey 01] une classification des contexteurs qui étend la proposition de Dey [Dey 01] : contexteur élémentaire (qui encapsule un capteur physique et 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'entrées 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). Comme nous le verrons dans la section 2.3, ces classes peuvent être étendues dynamiquement.

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

2.2. PROPRIÉTES

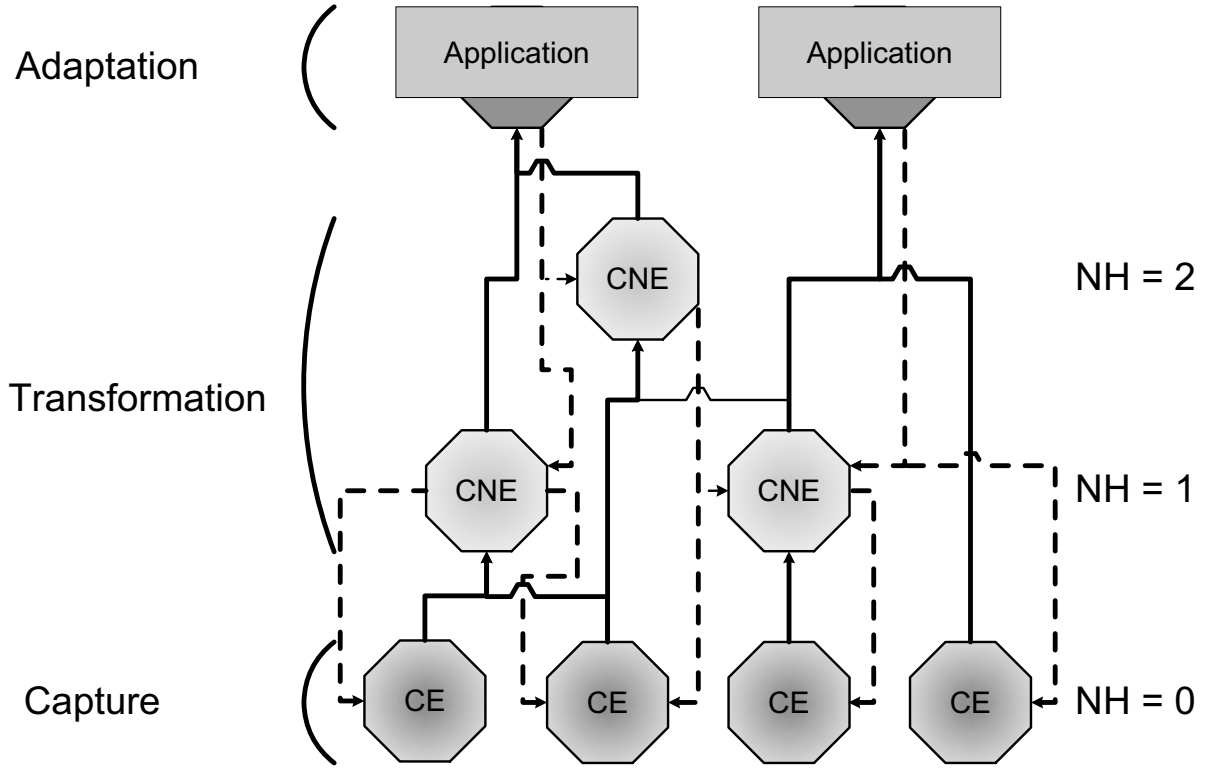
Les propriétés des contexteurs sont motivées par un environnement d'exécution évolutif et incertain : un capteur peut tomber en panne, des ressources d'interaction peuvent être ajoutées dynamiquement par l'utilisateur ou disparaître, etc. Pour ces raisons, un contexteur est réflexif et peut répondre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

- Un contexteur est réflexif : il est capable de s'autodécrire, c'est-à-dire fournir les informations suivantes : son nom, sa classe, ses interfaces d'entrée, ses interfaces de sortie, la fonction 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.
- Un contexteur est autonome. Une fois démarré et tant que le système sur lequel il fonctionne (ou l'administrateur de ce système) ne lui ordonne pas de s'arrêter, le contexteur est capable de fonctionner seul. En particulier, il sait initialiser le capteur qu'il pilote (s'il s'agit d'un contexteur du niveau capture), il sait trouver les données (et

métadonnées) nécessaires à son fonctionnement pour produire ses données de sortie (et métadonnées), il sait fournir ses services aux applications qui en ont besoin et répondre et émettre des ordres de contrôle pour minimiser les ressources qu'il consomme en fonction de son rendement.

- Un contexteur peut exister en plusieurs instances simultanément. De manière à être facilement exploitable, le contexteur doit se présenter comme un véritable composant. De ce fait, de multiple exécutions d'un même contexteur peuvent créer des instances de ce contexteur qui sont à la fois identiques fonctionnellement (capables de fournir le même service) mais distinguables (au moyen d'un identifiant unique).
- Un contexteur n'est pas dépendant de sa plate-forme d'accueil, ni du langage dans lequel il a été écrit. Pour améliorer la diffusion des contexteurs, il est nécessaire que ceux-ci fonctionnent sur un maximum de plate-forme (Windows, Linux, Mac-OS, Pocket-PC, ...). De plus, pour qu'un maximum de développeurs utilise ce modèle dans leurs applications, il faut qu'il soit interopérable avec un maximum de langage de programmation. Pour cela, je préconise l'utilisation d'un protocole de communication basé sur les couches basses (TCP et / ou UDP) qui sont supportées par un maximum de langages pour ne pas dire tous. Pour renforcer cette interopérabilité je propose l'utilisation

Figure 3
Exemple de hierarchisation d'un ensenble de contexteurs.



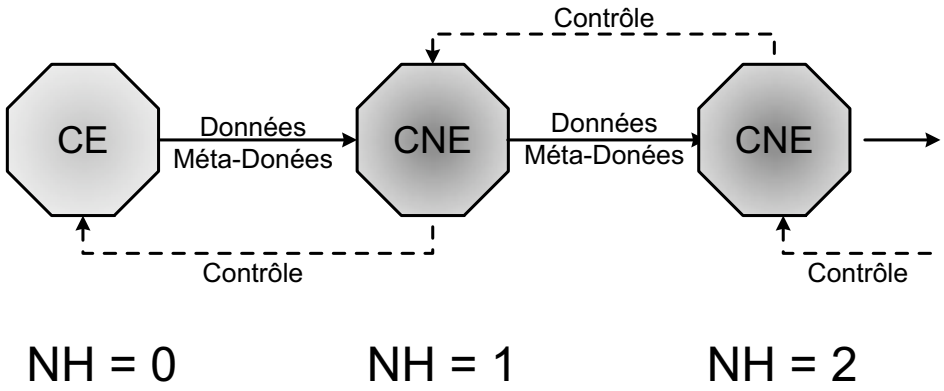


Figure 4
Schéma montrant la symétrie entre les flots de données et de contrôle entre trois contexteurs

de messages au format [XML] pour les échanges de données. Nous normalisons ces messages via des [DTD] ou des schéma XML [XML Schema].

2.3. COMPOSITION

Jusqu'ici, nous avons étudié le contexteur en tant qu'unité de représentation ou de calcul d'un observable. Voyons maintenant comment les contexteurs peuvent se composer pour représenter d'autres observables et de là, des entités, des relations et des rôles. Je propose deux mécanismes complémentaires de composition : l'assemblage hiérarchique qui s'effectue à l'exécution et l'encapsulation décidée par le concepteur de logiciel.

Composition dynamique : la hiérarchisation

Comme le montre la figure 3, 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.

La figure 4 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. Cela permet à un contexteur de modifier le rapport qu'il entretient avec ses (contexteurs) sources.

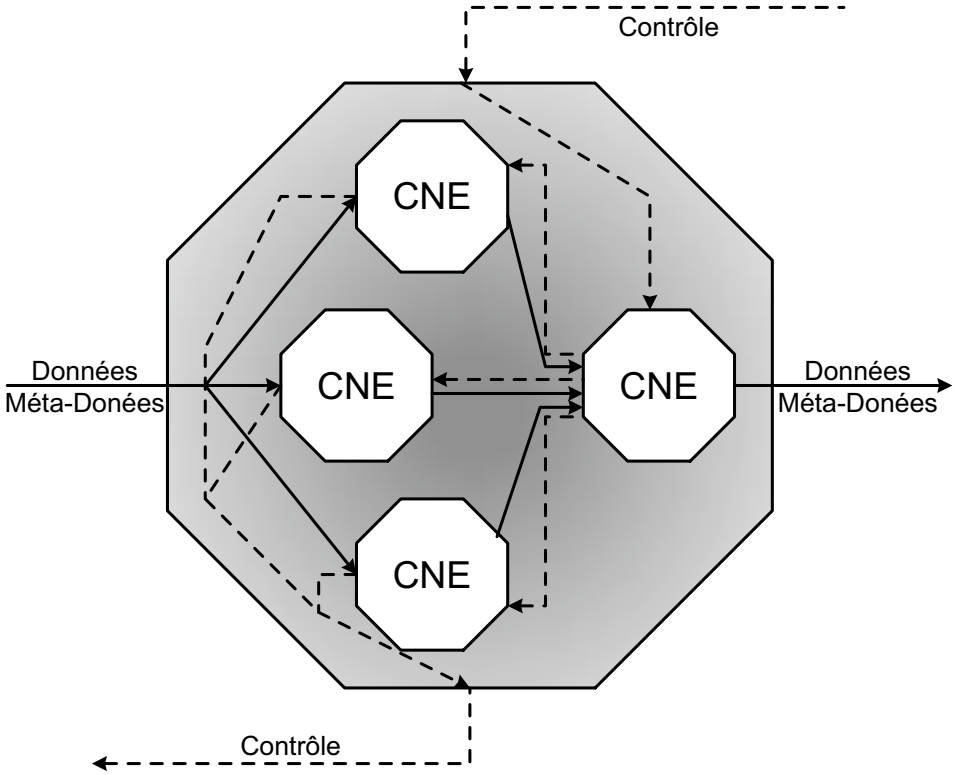


Figure 5
Exemple de contexteur obtenu par encapsulation

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 l'encapsulation.

Composition statique : l'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. 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, représentées sur la figure 5, sont des liens statiques qui, du point de vue de la mise en œuvre, peuvent correspondre à des appels de méthodes (pour la version encapsulée) alors que les liaisons dans la composition hiérarchique de la figure 3 correspondent, au niveau mise en œuvre, à des connexions réseau créées dynamiquement.

La figure 5 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 reprise

de [Crowley 00] (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 d'images. 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). L'encapsulation permet de réduire le niveau de hiérarchie des contexteurs et donc le degré de dépendance.

2.4. CONTEXTEURS ET PYRAMIDE DU CONTEXTE

La correspondance entre les contexteurs et les niveaux d'abstraction de la pyramide de contexte du chapitre II, tient de leur localisation dans une chaîne hiérarchique (se référer à la figure 3):

Au bas de la hiérarchie de contexteurs, les *contexteurs élémentaires*. Ces composants ont un niveau hiérarchique égal à zéro car ils n'ont pas de données (et métadonnées) d'entrée au sens du modèle des contexteurs. Leur rôle est de servir de liaison entre les mondes physique et numérique. Ils reçoivent des données mais seulement de(s) l'unité(s) de capture qu'ils encapsulent. Les contexteurs élémentaires peuplent la couche de capture de la pyramide.

Au centre de la hiérarchie, les *contexteurs non élémentaires* : ils ont des données (et métadonnées) d'entrée et de sortie. Ces composants utilisent les données (et métadonnées) provenant d'autres contexteurs pour créer des informations d'un niveau d'abstraction supérieure (observables symboliques), ou pour améliorer la qualité des données en multipliant le nombre de sources. Ces contexteurs ont un niveau hiérarchique supérieur ou égal à un. Ils correspondent à la couche Transformation de la pyramide.

Au sommet de la hiérarchie, les *contexteurs d'adaptation* (ou adaptateurs de contexte). Il s'agit de composants chargés du lien entre les applications et le monde des contexteurs. Ces adaptateurs sont des modules logiciels intégrables dans l'application qui souhaite utiliser les contexteurs. Ils prennent en charge la recherche et les échanges de données entre les contexteurs sollicités et l'application sensible au contexte. À l'aide de ces composants, les développeurs d'applications peuvent se connecter à l'infrastructure de contexteurs sans devoir en connaître le fonctionnement. Les contexteurs d'adaptation appartiennent au niveau adaptation de la pyramide. Grâce aux contexteurs d'adaptation, l'application peut utiliser directement les contexteurs élémentaires du niveau capture sans devoir passer par les contexteurs non élémentaires du niveau transformation...

2.5. CONTEXTEUR ET ÉTAT DE L'ART

Le contexteur rappelle le principe des *context-widgets* du Context-toolkit de Dey, du *context handling component* de Salber et des *context-entities* de l'architecture SCI de Strathclyde. Il les améliore en plusieurs points :

Par rapport au modèle de Dey, le contexteur fournit des informations qui font sens (inutile donc d'introduire la notion d'interpréteur). En outre un contexteur, comme un *context handling component*, dispose de contrôles et décore ses retours d'information de métadonnées qui expriment la qualité de sa production.

Par rapport au modèle de Salber,

- Le contexteur associe, sur un même flot, les données et les métadonnées qui sont, par nature et fonction, fortement couplées.
- Le modèle est affiné en classes de contexteurs que n'offre pas Salber.
- Le modèle introduit deux mécanismes complémentaires de composition de contexteurs : la hiérarchisation et l'encapsulation. Cette dernière offre une vision récursive du modèle (un assemblage de contexteurs constitue un contexteur) que nous ne retrouvons pas chez Salber.
- Enfin, nous exprimons des requis et des propriétés de contexteurs qui permettent de diriger les solutions de mise en œuvre qui seront traités au chapitre suivant.

Par rapport à la SCI, les contexteurs apportent surtout le support des métadonnées qui n'est pas présent dans les *Context Entities*. De plus, les contexteurs d'adaptation permettent d'intégrer facilement le monde des contexteurs dans des applications, alors que les développeurs des *Context Aware Applications* doivent faire un effort supplémentaire pour s'inscrire dans le modèle de la SCI et pour pouvoir communiquer avec l'ensemble de ses composants. Enfin, l'infrastructure apporte un mode de composition par encapsulation qui n'est pas présent dans SCI. Toutefois, comme SCI, nous visons le passage à l'échelle avec la notion de répéteur.

3. Le Répéteur

De manière à rendre à une application donnée le service qu'elle demande, les contexteurs d'adaptation ont besoin de trouver à la volée les contexteurs non élémentaires et/ou élémentaires capables de fournir les données nécessaires au service requis. Comment assurer cette découverte dynamique tout en répondant au passage à l'échelle et au requis de disponibilité de service ?

Plusieurs approches sont envisageables : la technique usuelle des annuaires comme dans eDonkey [eDonkey] ou l'approche plus évoluée des Context Server de la SCI, ou encore l'approche radicale des architectures pair à pair qui élimine la notion de serveur au prix d'un protocole de recherche par inondation. Les contexteurs adoptent une approche complètement décentralisée de type égal à égal. L'avantage de cette méthode est de ne pas avoir besoin d'annuaire et donc de pouvoir fonctionner sur des «ilots de connectivité». En contre partie, la recherche d'un service est légèrement plus compliquée.

3.1. LE PROTOCOLE DE RECHERCHE

Le protocole de découverte de contexteurs est le suivant :

- le contexteur crée un message contenant la description des données dont il a besoin. Ces données ont un nom, un groupe d'appartenance (correspondant aux catégories des entité définie dans le chapitre I section 5.1 ("Typologie des entites")), ainsi qu'une liste de métadonnées que le distributeur du service recherché doit impérativement fournir.
- Une fois le message créé, le contexteur y ajoute une brève description de lui-même : son identifiant ainsi que des informations permettant aux autres contexteurs d'entrer en contact avec lui.
- Le message est ensuite diffusé sur le réseau. Il sera reçu par tous les contexteurs opérationnels.
- À la réception d'un message de demande de service, tout contexteur opérationnel en examine le contenu. S'il peut fournir les informations correspondant à une (ou plusieurs) classes de données recherchées, il entre en contact avec le demandeur (à l'aide des coordonnées incluses dans le message) et lui propose ses services.

L'avantage de ce protocole est l'absence d'annuaire. Les contexteurs n'ont pas à connaître de serveur et peuvent facilement être déployés sur n'importe quelle plate-forme d'accueil disposant d'une couche transport (TCP/UDP). Cette approche permet d'utiliser les contexteurs sur des machines connectées sur des réseaux ad hoc comme ceux créés via des portables équipés de cartes WiFi (802.11a/b/g). Inversement, cette découverte par inondation n'est pas sans poser quelques problèmes.

3.2. PROBLÈMES D'INONDATION

Le premier problème de l'inondation est une augmentation de la charge réseau. Plus le nombre de contexteurs non élémentaires et de contexteurs d'adaptation sera élevé, plus l'occupation réseau pour la recherche des composants nécessaires au fonctionnement de ces contexteurs sera importante.

Rappelons cependant que les contexteurs élémentaires n'émettent pas de requête de recherche puisqu'ils ne possèdent pas de donnée d'entrée et que par conséquent ils n'ont pas besoin de les alimenter.

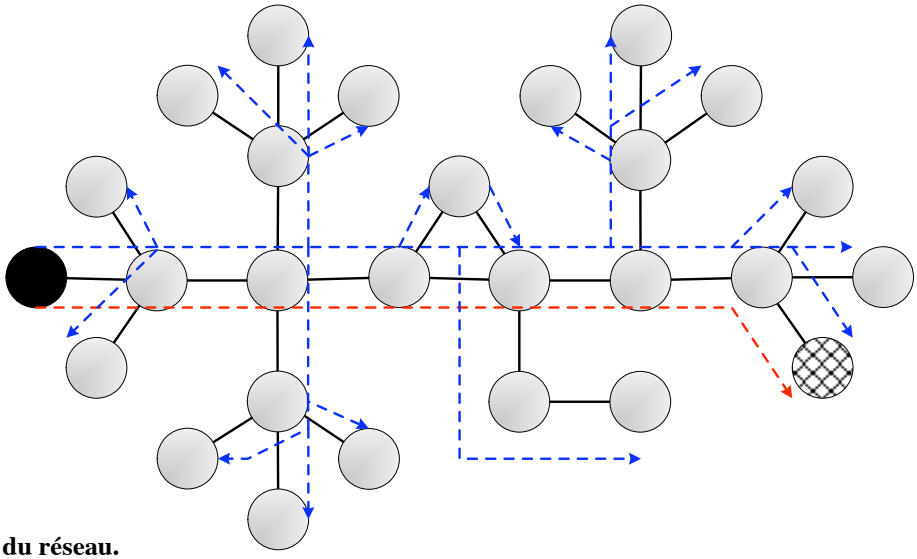


Figure 6
Schéma illustrant l'inondation du réseau.
En rouge, chemin correspondant à l'envoie souhaité. En bleu, Innondation provoqué par la recherche de la cible verte.

Cependant, une surcharge réseau due à la recherche n'a lieu que si tous les contexteurs (ou du moins un grand nombre d'entre eux) effectuent leur recherche en même temps. Or, il y a peu de chance que cela arrive. De plus, une fois la connexion établie, les messages de recherche ne sont plus émis.

Le deuxième problème concerne la propagation des messages sur le réseau. Vu que le protocole de recherche fonctionne par inondation, si un contexteur (ou une application) atteint un service (par exemple, la température qu'il fait chez une personne à New York) éloigné du lieu d'exécution (par exemple, depuis Grenoble), il y aura inondation d'un grand nombre de machines. Comme le montre la figure 6, si un contexteur fonctionnant sur la machine représentée par le disque noir, veut trouver le service fourni par un contexteur situé sur la machine hachurée, il doit émettre une requête inondant toutes les machines présentes sur ces réseaux. D'où l'introduction de la notion de répéteur.

3.3. LE RÉPÉTEUR

Pour éviter des inondations massives des réseaux et sous-réseaux situés entre l'application cherchant des informations contextuelles et les contexteurs fournisseurs ces informations, j'introduis un nouveau type de composant : le répéteur. Cette proposition s'appuie sur l'hypothèse que les applications et/ou les contexteurs ont le plus souvent besoin d'informations fournies par des contexteurs locaux. C'est-à-dire, des contexteurs situés à proximité et ayant de forte chance de se trouver sur le même réseau. Dans ce cas, la diffusion par inondation n'est pas un problème puisqu'elle est limitée au réseau local. L'encombrement et les perturbations qu'elle génère restent faibles.

Pour atteindre des contexteurs non locaux, cas plus rare, je propose le déploiement d'une architecture secondaire. À l'envoi d'une requête de recherche (nous cherchons ici un contexteur très loin de nous), celle-ci est envoyée localement mais elle est capturée (de manière transparente) par le répéteur local. Les répéteurs analysent toutes les requêtes qu'ils capturent.

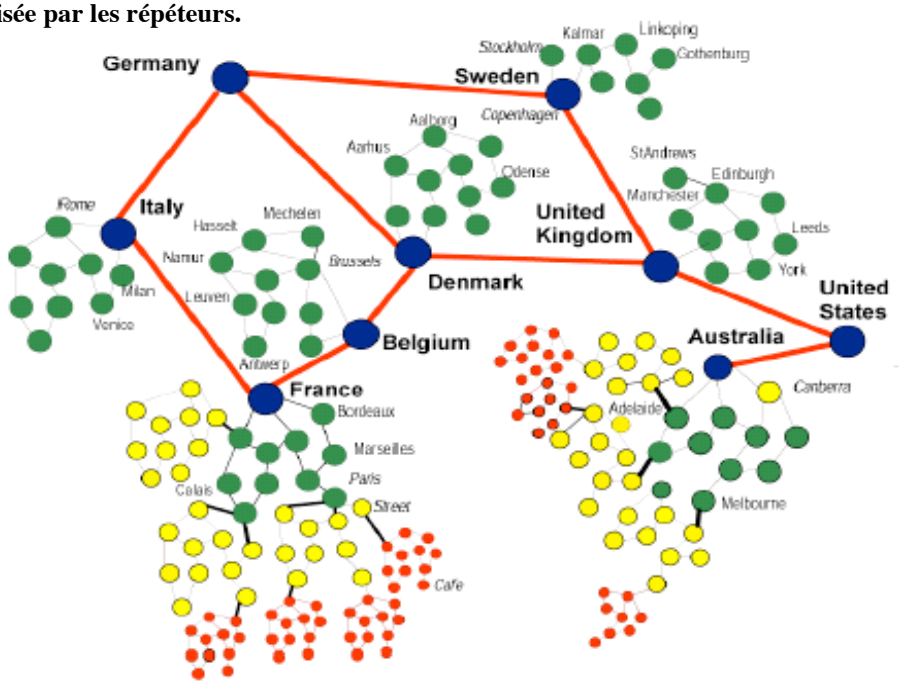
Comme décrit précédemment, une requête de recherche précise la ou les classes de contexteurs recherchées : pour chaque classe de contexteur, la requête contient le groupe, le type des données de sortie ainsi que les métadonnées associées à ces données et la localisation du dit contexteur.

L'information de localisation permet au répéteur de savoir si la requête est destinée au réseau local ou si elle est destinée à un réseau distant. Si la requête est émise à destination de contexteurs locaux, le répéteur ne fait rien, sinon il effectue le routage de cette requête vers les répéteurs idoines qui, à leur tour, diffusent la requête sur leur réseau local.

Pour se transmettre les requêtes, les répéteurs sont organisés selon le modèle hybride de la figure 7 et décrit dans [Kirby 02]. Ce modèle se présente comme une composition de deux modèles existants.

- D'une part le modèle hiérarchique classique où les noeuds (dans notre cas des répéteurs) se voient attribuer un rang dans la hiérarchie et où chaque noeud connaît son père et un ensemble de fils. Ici l'ordre hiérarchique s'appuie sur la localisation géographique des répéteurs.
- D'autre part, un modèle entièrement P2P où chaque noeud connaît un ensemble de voisins, c'est-à-dire les contexteurs proches de lui.

Figure 7
Architecture P2P hybride utilisée par les répéteurs.



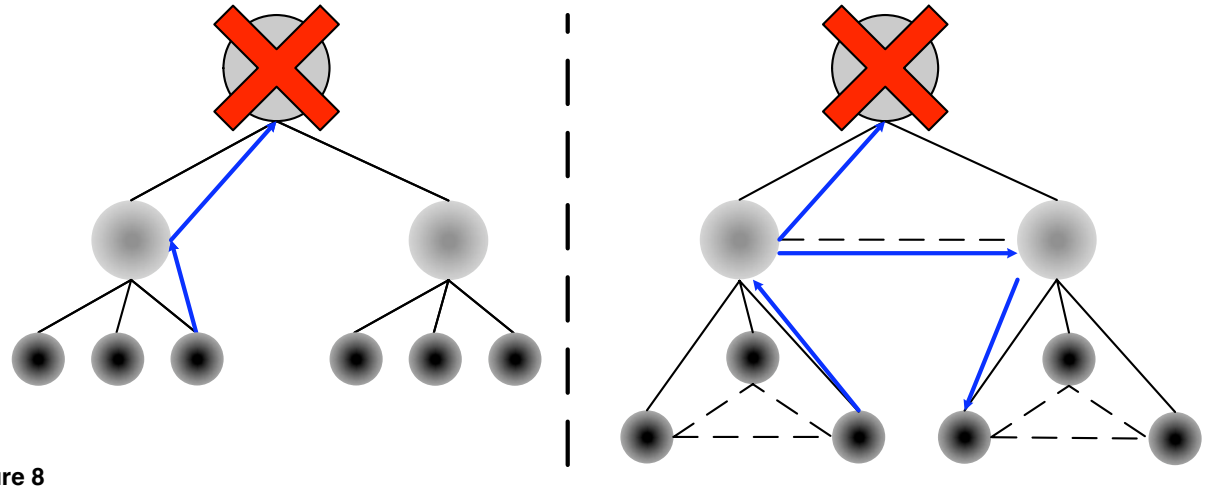


Figure 8
Exemple de l'avantage du modèle hybride sur le modèle hiérarchique.

L'assemblage résultant de ces deux modèles correspond à un modèle hiérarchique où les noeuds ayant le même père sont connectés par un réseau de type égal à égal.

Sans entrer dans les détails, l'avantage d'un tel modèle est sa robustesse proche des réseaux P2P (pratiquement tous les messages arrivent à destination car le réseau est fortement connecté) en gardant une efficacité similaire aux réseaux hiérarchiques (pas d'inondation et gestion des connexions père - fils simplifiées).

Par exemple, si nous examinons un réseau hiérarchique composé de trois niveaux, l'arrêt du noeud de plus haut niveau isole complètement les sous-réseaux alors que pour le modèle hybride, l'arrêt du même noeud ne gêne pas la communication entre l'ensemble des fils du nœud hors service ou de leurs descendants.

Si nous reprenons l'exemple de la figure 6, l'infrastructure de répéteurs permet de limiter grandement l'inondation que génèrent les contexteurs comme représenté sur le schéma de la figure 9. Le répéteur capture la requête émise par le contexteur noir, la transmet à son répéteur-père (de niveau hiérarchique supérieur) en fonction de la localisation des données de la requête. Une fois le bon répéteur trouvé, c'est-à-dire celui qui gère la localisation recherchée, le message de recherche est relâché sur le réseau. Les contexteurs s'y trouvant, peuvent alors le recevoir comme si le message avait été envoyé depuis un contexteur proche d'eux.

En résumé, les contexteurs fonctionnent de manière autonome sur des réseaux locaux (ad hoc ou non). Ils sont complétés par une infrastructure composée de répéteurs. Ces répéteurs sont connectés entre eux suivant un modèle hybride (P2P-hiérarchique). Ils s'occupent du routage des requêtes longues distances des contexteurs. Pour ce faire, chaque répéteur a en

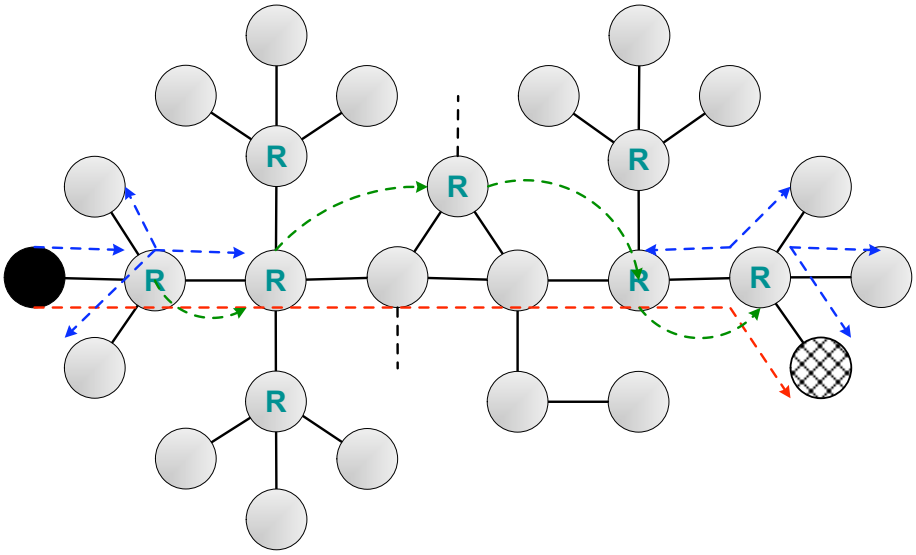


Figure 9
Schéma illustrant l'utilisation des répéteurs.

charge une région (au sens de la définition donnée dans le chapitre I) géographique pour laquelle il est responsable de deux activités :

- Analyser les requêtes émises par les contexteurs dans cette zone et vérifier si elles sont destinées à d'autres zones géographiques. Si c'est le cas, le répéteur doit transmettre ces requêtes aux répéteurs gérant les régions géographiques auxquelles elles sont destinées.
- Diffuser sur le réseau P2P des contexteurs les requêtes qu'il reçoit des répéteurs auxquels il est connecté. Bien entendu, les requêtes ne sont émises sur le réseau P2P couvert par le répéteur que si elles y sont destinées. Dans le cas contraire, la requête est dirigée vers le bon répéteur.

L'avantage de cette deuxième infrastructure est sa complémentarité avec l'infrastructure des contexteurs. Plus important que cette complémentarité, c'est sa transparence d'utilisation vis-à-vis des contexteurs : bien qu'il utilise ses services de routage, le contexteur n'a pas conscience de l'existence des répéteurs.

4. *Limites du modèle*

Malgré les avantages énoncés, mon modèle théorique présente trois limitations.

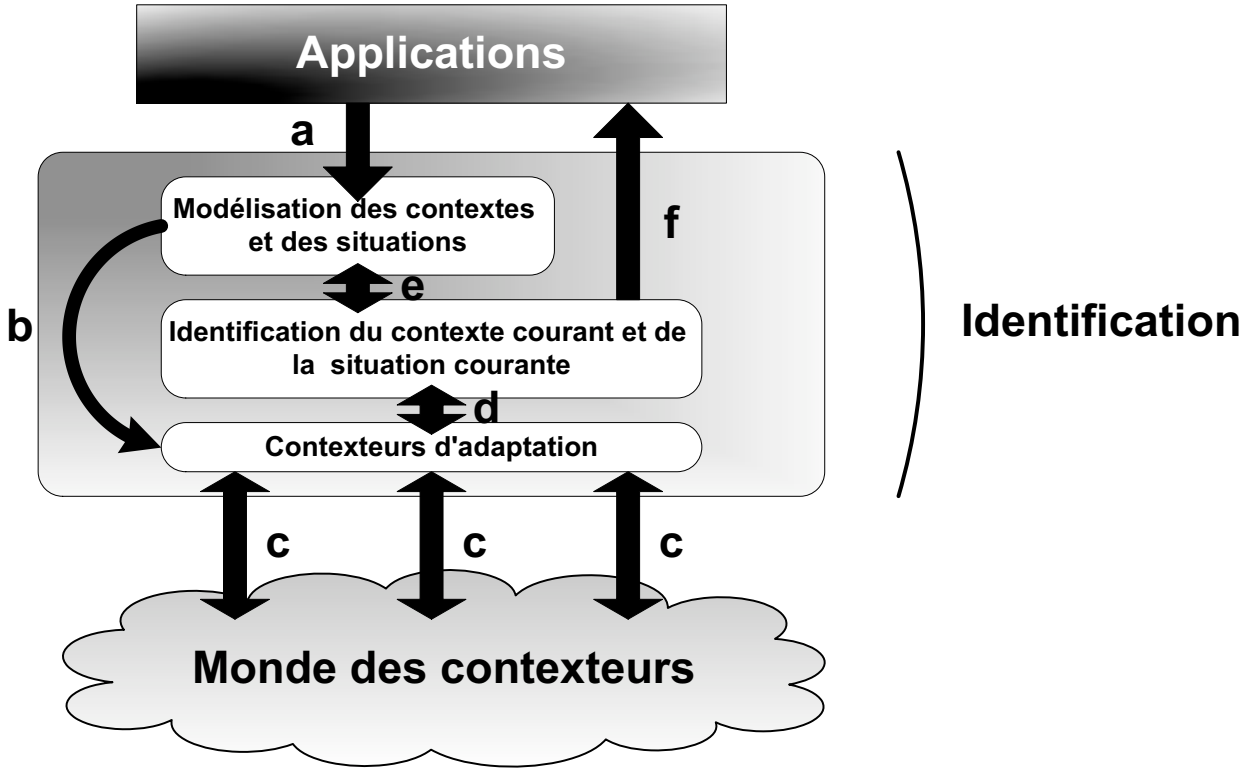
- La première correspond à l'absence de la couche Identification de la pyramide de référence.
- La deuxième correspond à l'absence d'une partie des services transversaux.
- La troisième correspond à l'utilisation de l'infrastructure de répéteurs qui vient rompre avec le modèle égal à égal des contexteurs.

4.1. LA COUCHE IDENTIFICATION

Concernant la couche *Identification*, mes travaux sont moins avancés que pour le reste de l'infrastructure. L'approche actuelle concernant cette couche est basée sur une architecture centrée données qui va à l'encontre de l'architecture des contexteurs. Est ce que ce choix est judicieux ? Est-il intéressant de mélanger les styles architecturaux ? Pouvons-nous utiliser l'infrastructure des répéteurs pour supporter les fonctions de la couche *Identification* ? Voilà certaines questions qui restent en suspend au moment de la rédaction de ce manuscrit.

Ce qui est sûr, c'est que chaque style architectural a des avantages et qu'il est important de tirer parti de chacun d'eux. Regardons en détail mes propositions concernant la couche Identification avant de parler de ses évolutions futures.

Figure 10
Détail de la couche identification.



Comme le montre la figure 10, la couche *Identification* se décompose en trois modules logiciels.

Le premier de ces modules est le composant de *modélisation des contextes et des situations*. Il a pour objectif la création d'une représentation interne des contextes et situations tels que les lui décrivent les applications clientes (en **a** sur la figure 10). La description des souhaits des applications pourrait être exprimée à l'aide d'un langage de haut niveau de type XML ou à l'aide d'autres langage. Dans [Reignier 05], Reignier utilise les réseaux de Pétri pour décrire le réseau d'un contexte donnée.

Une fois cette modélisation représentée, ce module génère (**b**) les contexteurs d'adaptation nécessaires à la capture de ces informations. Cette Capture (**c**) est effectuée par les contexteurs peuplant les couches *Transformation* et *Capture*.

Utilisant les données des contexteurs d'adaptation (**d**), le module *Identification du contexte courant et de la situation courante*, calcule quel est l'état (le contexte et la situation) actuel pour le réseau de contextes représenté par le composant *modélisation des contextes et des situations* (**e**). Une fois le contexte et la situation courante identifiés, il ne reste plus qu'à notifier l'application de état du contexte (**f**).

La poursuite de l'étude et du développement de cette couche architecturale rejoint un projet de l'équipe IIHM connu sous le nom de Ethylene [Balme 04]. Ce projet fera l'objet d'une présentation détaillée lors du chapitre *Illustrations*. J'espère bien que ce projet permettra de poursuivre l'étude de la couche *Identification* et d'en proposer une version cohérente avec le reste du modèle.

4.2. LES RÉPÉTEURS

Le deuxième reproche que nous pouvons émettre à l'encontre de mon infrastructure concerne l'utilité des répéteurs. Alors que les contexteurs ont été conçus comme un ensemble de composants autonomes s'assemblant pour former une architecture en flot de données, le modèle des répéteurs vient rompre cette harmonie. Toutefois, cette rupture ne modifie en rien l'architecture autoconfigurable des contexteurs ni leur utilisation sur les réseaux ad hoc.

Bien que l'infrastructure des répéteurs collabore avec les contexteurs, cette collaboration est transparente pour ces derniers. Aucune gêne n'est induite par les répéteurs. Ceux-ci servent à transporter les messages de recherche émis par les contexteurs dans le sous-réseau adéquat. L'architecture reste autoconfigurable et continue à fonctionner sur des réseaux ad hoc. Si ces derniers sont connectés à des réseaux supportant une infrastructure de répéteurs, ceux-ci pourront l'exploiter. S'ils sont isolés, utiliser des répéteurs ne leur seraient d'aucune utilité puisqu'ils ne

pourraient joindre les calculateurs des autres réseaux du fait de leur isolement.

Malgré cela, il peut paraître fastidieux d'installer un répéteur sur le réseau pour permettre aux applications d'avoir accès aux contexteurs de l'internet. Pour cela, il faudrait pouvoir se passer des répéteurs et par conséquent pouvoir contacter des contexteurs éloignés sans les connaître ni savoir si ils existent vraiment.

Pour ce faire, nous pouvons utiliser les travaux sur les nouvelles technologies P2P comme ceux de [Maymounkov 02] concernant le protocole Kademlia. Basé sur quatre procédures (PING, STORE, FIND_NODE, FINDVALUE), un nouvel algorithme de calcul de distance (XOR) et un identifiant unique (comme dans [Stoica 01]), le protocole Kademlia est le premier système P2P à fournir un routage des paquets avec une faible latence et de bonnes performances. Le point fort de Kademlia est son calcul de distance qui n'est pas basé sur la situation géographique des participant mais sur une distance modélisée dans l'identifiant de chaque participant. De ce fait, si le nombre de participants du réseau double, l'utilisateur (système interrogeant le réseau) ne devra interroger qu'un seul nœud de plus. L'autre particularité intéressante de Kademlia est sa prise en charge la disparition de noeuds du réseau, ce qui est un élément important pour l'architecture des contexteurs.

Par conséquent, il serait intéressant d'adapter ce protocole aux besoins des contexteurs de manière à permettre la suppression de l'infrastructure des répéteurs tout en gardant un encombrement réseau peu pénalisant.

Chapitre IV Le Contexteur : Mise en œuvre

Jusqu'à présent, notre histoire ressemble à un jeu de Lego : les assemblages sont de plus en plus complexes et forment maintenant des chaînes de molécules géeantes. Mais c'est toujours de la matière. Par quel coup de baguette magique la vie va-t-elle surgir ?

Simonnet, 1996, La plus belle histoire du monde. Acte 2 - Scène 2 : La vie s'organise, p93

Ce chapitre propose une réalisation logicielle du concept de contexteur. Cette mise en œuvre, dirigée par les requis de flexibilité et de fiabilité justifiés au chapitre II (section 3), s'appuie sur les principes suivants :

- Séparation des aspects " service " des aspects " communication " afin d'assurer la (re)configuration dynamique de l'infrastructure de contexteurs. L'approche composant-connecteur répond à ce principe de séparation.
- Élimination des serveurs en sorte d'assurer la disponibilité des services contextuels. Le style architectural pair à pair répond à ce principe.

L'infrastructure de contexteurs ici proposée se présente donc comme un ensemble de composants autonomiques pair à pair qui s'autoconfigurent dynamiquement pour répondre aux demandes de services contextuels d'applications clientes. Cette infrastructure, réalisée en Java 1.4, est disponible sur toute plate-forme d'accueil (Windows, MacOS, etc.) dotée des protocoles TCP-IP et UDP.

Ce chapitre est structuré comme suit : Une présentation générale du cycle de vie d'un contexteur (section 1) est suivie, en section 2, par la description détaillée de la mise en œuvre du contexteur. Nous verrons notamment comment s'établissement les connexions entre contexteurs pour constituer un tissu reconfigurable adapté aux requêtes des applications clientes. Dans la section 3, nous traiterons le problème particulier des déconnexions entre contexteurs et en 4, la mise en œuvre des répéteurs pour une couverture réseau globale. On conclut ce chapitre par un bref rapport sur les performances de l'infrastructure de contexteurs.

1. Cycle de vie du contexteur

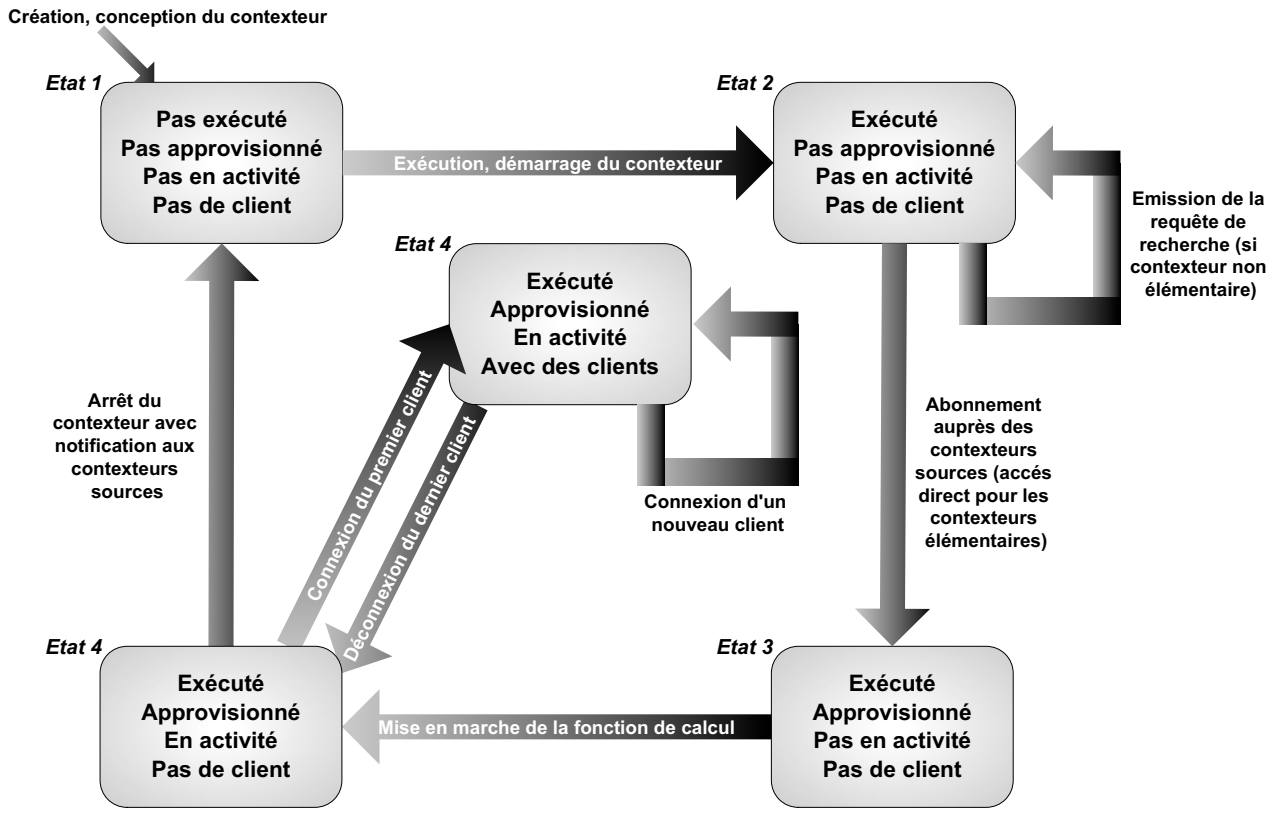
Le contexteur répond à un cycle qui rythme son existence en cinq états. La figure 1 en donne une représentation sous forme d'automate à états fini.

Dans **l'état 1**, le contexteur existe sous forme d'un fichier exécutable stocké sur disque. Cet état est caractérisé ainsi : *pas exécuté, pas approvisionné* (le contexteur n'est pas encore connecté aux contexteurs sources dont il a besoin pour fonctionner), *pas en activité* (il ne calcule pas de données de sortie) et *pas de client*. L'ordre d'exécution du contexteur se traduit par le chargement de son exécutable en mémoire et du fichier de configuration qui lui est associé. Si ces chargements s'effectuent avec succès, le contexteur prend l'état 2.

L'état 2 correspond à la phase d'initialisation et d'abonnement du contexteur auprès des contexteurs dont il a besoin pour approvisionner ses entrées (contexteurs sources). L'état est caractérisé par: *exécuté* (le contexteur est maintenant vivant), *pas approvisionné* (le contexteur est encore isolé), *pas en activité* (il ne calcule pas de données de sortie) et *pas de client*. Le contexteur initialise l'ensemble de ses paramètres à l'aide du fichier de configuration. Il s'auto-configure (nom, groupe, nombre de clients, ...) et génère son identifiant unique. S'il est un contexteur élémentaire, il configure les capteurs qu'il encapsule et entre dans l'état 3. Si le contexteur n'est pas élémentaire, il cherche à entrer en contact avec les contexteurs qui lui sont nécessaires en entrée. Pour chaque contexteur source trouvé, suit une phase de négociation dans laquelle le contexteur indique la manière dont il veut que les données (du contexteur source) lui soient transmises. (Nous reviendrons plus loin sur la nature de ces modes.) Une fois la connexion établie avec tous les contexteurs sources nécessaires, le contexteur entre dans l'état 3.

L'état 3 désigne la phase de mise en marche de la fonction de calcul du contexteur (son noyau fonctionnel). Le contexteur est *exécuté* et *approvisionné*, mais il n'est *pas en activité* (il n'a pas encore fourni de données de

Figure 1
Cycle de vie d'un contexteurs



sortie), ni *n'a de client*. Pour produire une donnée de sortie, un contexteur élémentaire doit recevoir des données en provenance du capteur qu'il encapsule. Pour un contexteur non élémentaire, il lui faut recevoir un exemplaire de chaque donnée d'entrée en provenance des contexteurs sources. Une fois la première donnée de sortie calculée, le contexteur entre dans l'état 4. Le calcul de la donnée de sortie tient compte des métadonnées associées à chaque donnée. De mauvaises métadonnées pouvant entraîner le calcul de mauvaises données (valeurs fausses, imprécises, ...), le contexteur peut envoyer des requêtes de contrôle (via ses ports de contrôle de sortie) aux contexteurs sources auxquels il est connecté. Ces requêtes peuvent demander l'arrêt de la connexion avec un contexteur source (valeurs erronées provenant de ce contexteur), la mise en pause de cette connexion jusqu'à rétablissement d'une certaine qualité, la modification du mode de transmission, le changement de l'unité de la donnée envoyée, etc. Nous reviendrons sur ce point dans la section 2.3.

L'état 4 correspond à la phase d'attente de clients. 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, qualité de service, …) avec le demandeur (contexteur client), puis passe dans l'état 5. L'état 4 est aussi l'état qui correspond à un arrêt normal du contexteur. Si un tel arrêt est sollicité (soit par l'administrateur de la machine, soit par le système d'exploitation, ou toute autre raison), le contexteur indique à chacun des contexteurs sources auquel il est connecté qu'il n'a plus besoin de ses services. Une fois les déconnexions toutes terminées, le contexteur s'arrête et retourne dans l'état 1.

L'état 5 est la phase de service du contexteur : il est *exécuté, approvisionné, en activité* et *il a des clients*. À chacun de ses clients, il fournit les données calculées selon le contrat négocié avec eux en phase de connexion. Il continue à accepter de nouveaux clients si le nombre maximum de clients spécifiés au moment de la configuration du contexteur dans l'état 2 n'est pas atteint. Si un nouveau contexteur (et/ou application) le sollicite, il négocie avec lui (ou elle) un contrat de service. Le contexteur écoute également les requêtes sur ses ports de contrôle d'entrée. Une requête de contrôle peut notifier le contexteur d'une demande de changement de mode de transmission de données, d'une demande de modification du type de données transmises (conversion de l'unité de la donnée dans une autre unité), d'une demande de modification de la qualité de service (augmentation de la précision, ...), ou d'une demande d'arrêt. Ces requêtes ont pour effet de modifier les requis de la fonction de calcul. Elles ont éventuellement des répercussions sur les contexteurs sources. En cas de demande d'arrêt, le contexteur informe ses clients de l'arrêt de son service avant de retourner dans l'état 4 pour suivre la procédure normale d'arrêt. S'il n'a pas reçu de demande d'arrêt, il retourne dans l'état 4 lorsque son dernier client se déconnecte.

Notons que la figure 1 exprime le cycle de vie idéal du contexteur. Les cas de déconnexions intempestives analysés en section 3, ne sont pas représentés dans le schéma.

Jusqu'ici, nous avons traité le contexteur comme une unité d'abstraction. Voyons maintenant sa structure interne.

2. *Réalisation détaillée*

Le diagramme de classe UML de la figure 3 définit la structure interne d'un contexteur tandis que le diagramme de séquences de la figure 2 synthétise le fonctionnement de ses constituants. La section 2.1 explique ce fonctionnement. En 2.2, un exemple de fichier de configuration est présenté suivi en 2.3 de la description des messages échangés entre contexteurs.

2.1. DÉTAILS DE FONCTIONNEMENT

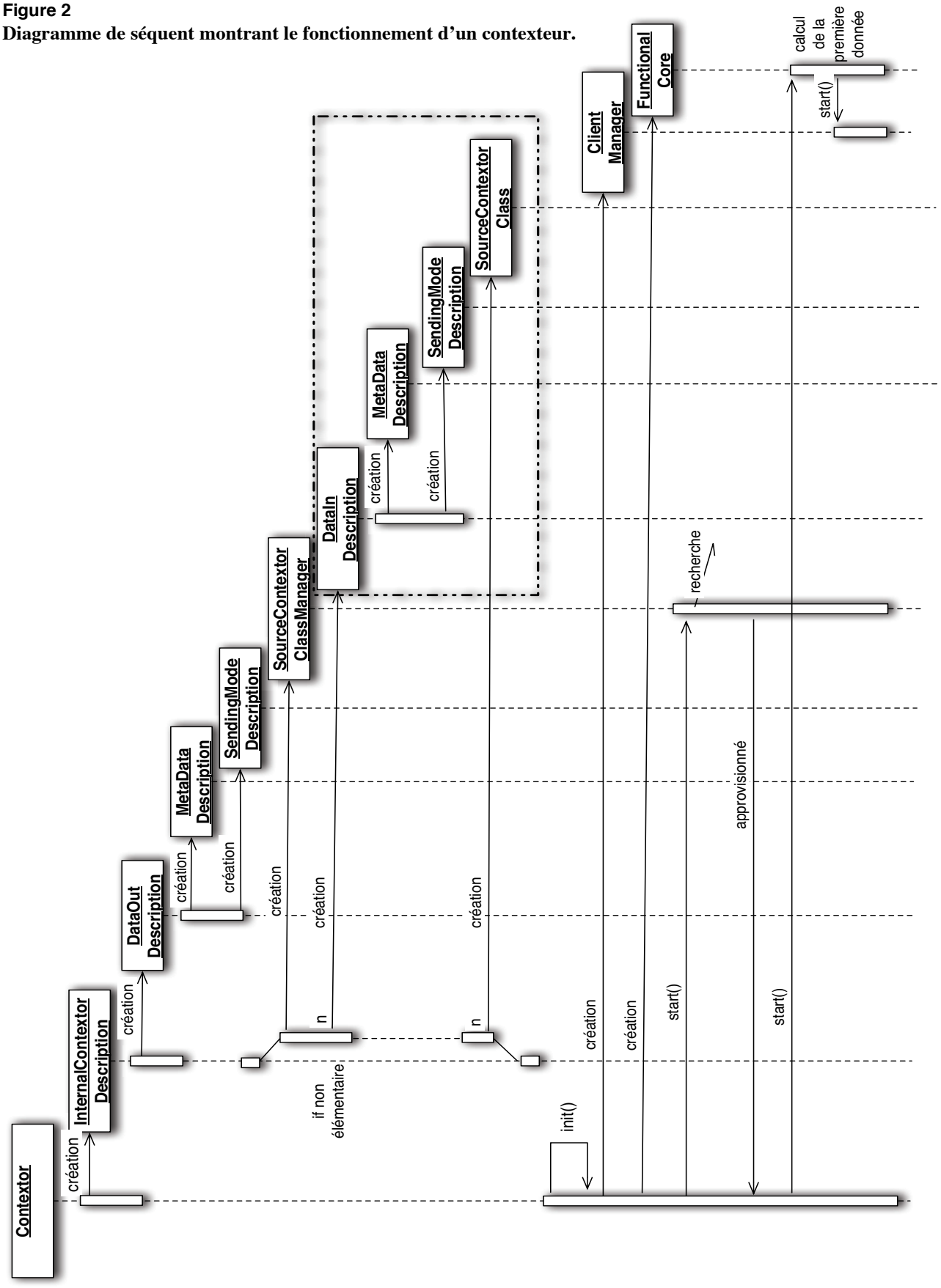
Au démarrage (transition entre l'état 1 et l'état 2 de la figure 1), le contexteur fait appel à son fichier de configuration.

Ce fichier, au format XML, contient des éléments permettant l'initialisation standard du contexteur : nom, groupe, localisation, nombre maximum de clients acceptables, description des données d'entrée et de la donnée de sortie. Le rôle de chaque élément de ce fichier sera précisé au fur et à mesure des besoins.

Le fichier de configuration est analysé par l'objet *InternalContextorDescription* du contexteur. À partir des informations du fichier de configuration, *InternalContextorDescription* crée et initialise les objets suivants :

- Un objet *DataOutDescription* pour la donnée de sortie auquel est associé un objet *MetaDataDescription* qui prend en charge les métadonnées de sortie, et un objet *SendingModeDescription* qui définit les modes d'échange de la donnée de sortie que le contexteur permet.
- Pour chaque classe de données d'entrée, un objet *DataInDescription* associé à un objet *MetaDataDescription* et un objet *SendingModeDescription* qui décrivent respectivement les métadonnées et les modes de transmission souhaités pour cette classe de données d'entrée. Ces objets (*DataInDescription*, *MetaDataDescription* et *SendingModeDescription*) n'existent que pour un contexteur non élémentaire (*NonElementaryContextor*).
- Des *SourceContextorClass* pour chaque classe de données d'entrée.

Figure 2
Diagramme de séquent montrant le fonctionnement d'un contexteur.



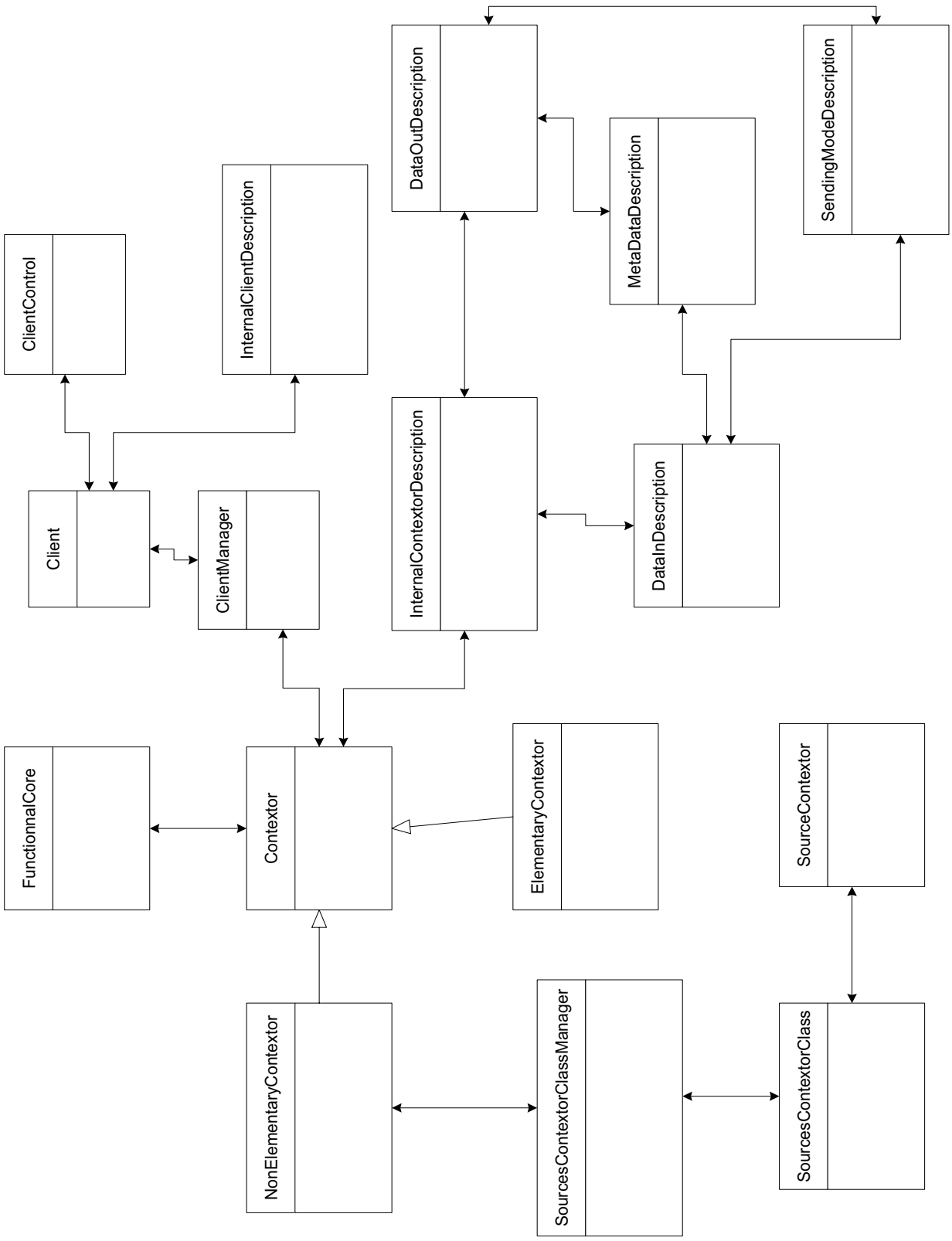


Figure 3
Diagramme de classes d'un contexteur.

Une fois sa configuration interne achevée, le contexteur exécute la méthode init(). Cette méthode abstraite doit être instanciée par le concepteur du contexteur pour compléter les initialisations standard avec les initialisations spécifiques à chaque contexteur, par exemple pour un contexteur élémentaire, établir sa liaison avec un capteur. L'initialisation du contexteur est maintenant achevée (il est dans l'état 2).

S'il n'est pas élémentaire, le contexteur part à la recherche de contexteurs sources. Pour cela, il établit une connexion virtuelle avec le réseau P2P [1] de contexteurs. Cette connexion est dite virtuelle car l'implémentation P2P actuelle s'appuie sur le protocole Multicast UDP. Par conséquent, il n'y a pas de vraie liaison établie entre les différents contexteurs du réseau P2P (mode déconnecté). Le réseau P2P de contexteurs est un réseau virtuel. Une fois membre du réseau virtuel, le contexteur peut émettre des demandes d'approvisionnement via son *SourcesClassManager*. Cet aspect est détaillé un peu plus loin dans cette section.

Une fois approvisionné ou relié à son capteur (état 3 du cycle de vie), le contexteur démarre son noyau fonctionnel FunctionnalCore. Il commence alors à produire des données de sortie (état 4). Le contexteur est fonctionnel, mais n'a pas encore de clients. Il démarre un gestionnaire de clients (*ClientManager*). Cela a pour effet de créer un certain nombre (défini dans le fichier de configuration) d'objets Client possédant leur propre flot d'exécution (Thread). Chaque Objet Client ainsi créé est mis en attente dans une liste de clients disponibles. Une fois ces opérations terminées, le *ClientManager* écoute le réseau virtuel, en attente de requêtes de service.

À la réception d'une requête de service émise par un contexteur client potentiel (étape 1 de la figure 3), le *ClientManager* du contexteur transfère la requête à un objet *Client* disponible et réveille le thread de ce client. Le *Client* crée un objet *InternalClientDescription* qui analyse la requête et vérifie que le contexteur peut fournir une des données spécifiées dans la requête (étape 2 de la figure 3). Si ce n'est pas le cas, il l'indique à l'objet *Client*. Celui-ci réinitialise son état (libération des objets créés entre temps, initialisation des variables globales ...) et redevient disponible. Inversement, si le contexteur source peut subvenir à la demande, une connexion directe (TCP) est établie avec le client (étape 3 de la figure 3). Un objet *ClientControl* est créé pour prendre en charge les requêtes de contrôle du client.

Une fois la connexion établie avec le client, un message de connexion lui est envoyé. Ce message contient le nom de la classe de données à laquelle répond le contexteur ainsi que le mode de transmission sélectionné parmi

1. P2P : un raccourci d'écriture pour Peer to Peer (pair à pair).

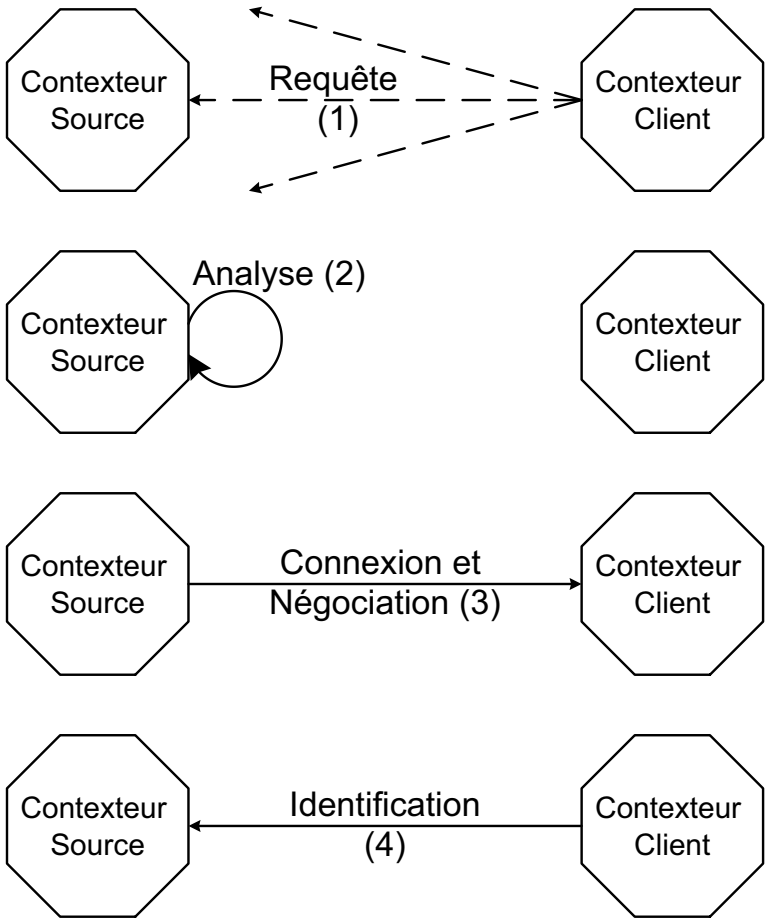


Figure 4
Découverte et connexion à un contexteur source

les modes permis par le contexteur en fonction de la liste de souhaits du client. Une fois le message envoyé, le contexteur source attend un message d'identification du contexteur client indiquant que ce client a les autorisations nécessaires (étape 4 de la figure 3). S'il n'a pas l'autorisation, la liaison est rompue et l'objet *Client* du contexteur source redevient disponible après avoir réinitialisé son état.

Si le client a l'autorisation, le contexteur transmet les données en fonction du mode d'échange choisi. Actuellement 5 modes de transmission sont assurés par les contexteurs :

- **Once** : envoi unique des données puis ferme la connexion avec le contexteur client.
- **OnChange** : envoi des données chaque fois que leur valeur change (plus une première fois à la connexion).
- **OnRequest** : envoi des données sur demande du contexteur client.
- **OnCompute** : envoi des données à chaque calcul de ces données par le noyau fonctionnel du contexteur.

- **AfterDelay** : envoi des données après un délai d'attente et cela de manière cyclique.

Alors que la classe *Client* transmet les données, la classe *ClientControl* écoute les requêtes du contexteur client. Actuellement, les contexteurs gèrent 4 types de requête :

- **Close** : message de fermeture de la connexion.
- **Ident** : message d'identification décrit précédemment.
- **Request** : message demandant l'envoi de donnée.
- **Control** : message demandant une modification du contexteur (par exemple : changement de mode d'envoi).

Nous venons de voir le fonctionnement d'un contexteur du point de vue de ses sorties. Revenons sur la phase d'approvisionnement. Cette phase comprend plusieurs étapes.

- Le contexteur crée un *SourceContextorClassManager* qui prend en charge la réception des connexions directes (TCP) (étape 1 de la figure 3) expliquée précédemment.
- Le *SourceContextorClassManager* crée et émet une requête de recherche contenant le nom du contexteur, comment se connecter à lui, et la description de toutes les classes de données qui lui sont nécessaires en entrée (étape 1 de la figure 3). Il se met à l'écoute du réseau.
- À la réception d'une réponse, le *SourceContextorClassManager* analyse le début du message pour trouver à quelle classe de données d'entrée le message correspond. Le message est transmis au *SourceContextorClass* approprié. Il existe un objet *SourceContextorClass* créé par classe de données d'entrée. Ces objets sont créés lors de la phase de configuration du contexteur.
- Un *SourceContextorClass* qui reçoit un message de connexion, crée un objet *SourceContextor* chargé de la connexion avec le fournisseur de données. Pour chaque classe de données d'entrée, c'est-à-dire pour chaque objet *SourceContextorClass*, il peut y avoir plusieurs objets SourceContextor. Le nombre d'objets SourceContextor est défini par les champs *NumberMin* et *NumberMax* de l'objet *DataInDescription*. *NumberMin* indique le nombre minimum de sources pour que la classe soit approvisionnée. *NumberMax* indique le nombre maximum de sources acceptées pour cette classe de données d'entrée. Un contexteur est approvisionné lorsque toutes ses classes de données d'entrée sont approvisionnées. Autrement dit, chaque objet *SourceContextorClass* a établi (via ses objets *SourceContextor*) un nombre de connexions comprises entre son champ *NumberMin* et son champ *NumberMax*.
- À la déconnexion d'un fournisseur de données, le *SourceContextor* associé est détruit.
- De plus, les *SourceContextorClass* prennent chacun en charge la réémission de requêtes de recherche relevant de leur classe. Alors que

la première émission est effectuée par le *SourceContextor-ClassManager*, les suivantes ne le seront pas. En effet, il serait beaucoup trop compliqué de centraliser les réémissions des requêtes de recherche car chaque *SourceContextorClass* a des requis différents tel que la vitesse de réémission ou le nombre de *SourceContextor* recherchés ... Cela explique pourquoi, une fois la première requête émise, chaque *SourceContextorClass* prend en charge ses propres recherches.

Nous l'avons vu, la configuration d'un contexteur s'appuie sur l'existence d'un fichier de configuration. Voyons comment en détail au moyen d'un exemple.

2.2. LE FICHIER DE CONFIGURATION.

La structure d'un fichier de configuration d'un contexteur dépend de sa classe : contexteur non élémentaire, contexteur élémentaire et contexteur d'adaptation. Un fichier de configuration, écrit en [XML], répond aux schémas XML ([XML Schema]) d'un contexteur. Les fichiers de configuration peuvent être produits au moyen d'un assistant qui évite au développeur de contexteurs de manipuler le format XML.

Fichier de configuration de contexteur non élémentaire

Prenons le cas du fichier de configuration du contexteur d'activité tiré de l'*Observatoire d'activité* présenté au chapitre suivant.

Tout fichier de configuration de contexteur commence par le tag CONTEXTOR_DESCRIPTION puis renseigne le nom du contexteur, son groupe d'appartenance et la localisation de la machine sur laquelle il est censé s'exécuter (un where).

```
<CONTEXTOR_DESCRIPTION>
   <Name>ActivityContextor</Name>
   <ContextGroup>System/Application/Observatoire</ContextGroup>
   <Location>
      World/Europe/France/RhoneAlpes/Isere/Grenoble/
   </Location>
...
```

Le groupe d'un contexteur dénote la nature de ses informations de sortie. Il correspond à la typologie des entités et observables présentée au chapitre I (section 5). Le contexteur relève du groupe :

- système (System) lorsque sa donnée de sortie correspond à des informations logicielles et/ou matérielles sur sa plate-forme d'exécution.
- utilisateur (User) lorsque sa donnée de sortie correspond à des informations sur les utilisateurs de sa plate-forme.
- social (Social) lorsque sa donnée de sortie correspond à des informations sur les êtres vivants non - utilisateurs de la plate-forme.
- physique (Physique) lorsque sa donnée de sortie correspond à des informations sur l'environnement physique (objets, bâtiments, …).

La localisation (qui peut évoluer au cours du temps) est organisée sous la forme d'un modèle hiérarchique [Flury 03], c'est-à-dire d'une liste de régions géographiques de plus en plus précises. La région de rang n dans la liste contient la région de rang n+1 et est contenue dans la région de rang n-1, selon le découpage suivant :

World / *Continent* / *Country* / *State* / *Province* / *City* / *District* / *Street* / *Building* / *Floor* / *Office*

Ce découpage est inspiré des travaux sur la localisation dans le projet Aura [Jiang 02]. Aura utilise un modèle de localisation hybride qui combine les principes des modèles hiérarchiques et des modèles à base de coordonnées. Dans un modèle hiérarchique, la localisation des entités physiques est définie de manière symbolique, descriptive ou topologique. Cette description correspond aux deux plus hautes couches du modèle de Flury. Dans un modèle à base de coordonnées, la localisation des entités physiques est définie de manière géométrique ou métrique. Par exemple, la localisation ALI (Aura Location Identifier) *ali://cmu/wean-hall/floor3#{(1,0),(-1.5,0.5),(0,3),(2,3.5),(3,1.5)}-(1,5)}* désigne une localisation correspondant au troisième étage du bâtiment Wean Hall à CMU (Carnegie Mellon University). La liste de points qui suit le symbole # décrit un polygone fermé parallèle au plan contenant le Wean Hall et (1,5) spécifie l'étendue de la hauteur de la zone. Si l'on réfère à notre définition de *Where* (Chapitre I, section 5.2, figure 11), le Wean Hall est une *SymbolicLocation* dont la *SpatialRegion* est définie à l'aide de cinq *PhysicalLocation* dont les coordonnées sont décrites dans un *SpatialRepresentatingSystem* particulier.

Ensuite, le fichier continue avec les paramètres d'identification du contexteur. L'identifiant et le mot de passe servent de contrôle d'accès au données fournies par le contexteur lors de la phase d'identification (étape 3 de la figure 4).

```
…
<Identification>
   <Login>myLogin</Login>
   <Password>myPass</Password>
</Identification>
…
```

La version actuelle des contexteurs ne traite pas l'encodage des mots de passe. Ceux-ci sont sauvegardés en clair dans le fichier de configuration. Le cryptage de ces données, comme des autres données sensibles, améliorerait grandement l'aspect sécurité des contexteurs.

Puis, le fichier décrit la configuration du réseau virtuel (P2P) du contexteur. Basé sur le protocole [UDP Multicast], le réseau virtuel des contexteurs est décrit à l'aide de 3 variables : le groupe (une adresse IP dans la plage réservée par le protocole UDP Multicast c'est-à-dire une

adresse de classe D comprises entre 224.0.0.0 et 239.255.255.255), un numéro de port et la valeur du TTL (Time to Live). Le TTL permet de régler le niveau de propagation des paquets UDP. Plus il est élevé, plus il inonde le réseau. Je conseille les trois valeurs suivantes en fonction des besoins du contexteur :

- Pour une recherche locale à la machine d'exécution : TTL = 0
- Pour une recherche dans sous réseau local : TTL = 1
- Pour une recherche étendue : TTL = 2

```
…
<P2PDescription>
    <P2PGroup>239.0.0.0</P2PGroup>
    <P2PPort>19077</P2PPort>
    <P2PTTL>0</P2PTTL>
</P2PDescription>
…
```

Ces valeurs modélisent les trois niveaux de couverture réseau introduits au chapitre II.

- TTL = 0 correspond au niveau micro,
- TTL = 1 correspond au niveau local et
- TTL = 2 correspond au niveau étendu.

Le dernier paramètre correspond au nombre maximum de clients qu'accepte le contexteur.

```
…
<MaxClientNumber>50</MaxClientNumber>
…
```

Ensuite, le fichier contient la section décrivant les données, délimitée par le tag DataDescription.

```
…
<DataDescription>
…
```

Cette section est composée de la description des données de sortie. Cette description contient le type de la donnée (entier, booléen, …) et son unité de mesure (centimètre, newton, kelvin …). Les sous-sections MetaData et *DataSendingMode* décrivent respectivement les listes des métadonnées et des modes de transmission (*Once*, *OnChange*, *OnRequest*, *OnCompute*, *AfterDelay*) acceptés par le contexteur. Les métadonnées incluent l'ensemble des propriétés opérationnelles d'un capteur (résolution, latence, ...), et / ou des mesures de la qualité de la donnée (confiance, incertitude, ...), des dates.

```
…
<DataOutDescription Type="xml" Unit="none">
    <MetaData>
        <Confidence> true </Confidence>
        <InitialDate> true </InitialDate>
        <CurrentDate> true </CurrentDate>
```

```
        </MetaData>
        <DataSendingMode>
            <OnChange>true</OnChange>
            <OnCompute>true</OnCompute>
            <Once>true< Once >
            <OnRequest>true</OnRequest >
            <AfterDelay> true</AfterDelay>
        </DataSendingMode>
    </DataOutDescription>
    …
```

Les métadonnées de date incluent :

- **CurrentDate** : elle correspond à la date de capture (calcule) de la donnée à laquelle elle est associée.
- **InitialDate** : elle correspond à la date de la donnée la plus ancienne ayant servi à calculer cette donnée.

Les sections suivantes correspondent aux données d'entrée. Un contexteur peut avoir plusieurs données d'entrée donc plusieurs sections peuvent être présentes en de multiples exemplaires. Pour chaque *DataInDescription*, c'est-à-dire pour chaque classe de données d'entrée, le fichier décrit le type de la donnée, son unité, la liste des métadonnées et des modes de transmission souhaités.

En plus des paramètres déjà introduits, les données d'entrée présentent quelques particularités :

- Chaque classe de données d'entrée doit être représentée par une clef. Cette clef sert d'identifiant unique pour un contexteur donné. Elle permet aux développeurs d'accéder facilement à la classe de données d'entrée souhaitée.
- Puisqu'il d'agit d'une classe de données (non pas d'une donnée unique), il est nécessaire de spécifier le nombre minimum (*NumberMin*) et maximum (*NumberMax*) de source de données que doit chercher un contexteur. Ces valeurs sont représentées par des entiers allant de 1 à n (n étant une valeur finie) pour *NumberMin* et de *NumberMin* à k (k étant une valeur finie) pour *NumberMax*. Si *NumberMax* vaut -1, alors le contexteur cherche toutes les sources correspondant à cette classe de donnée d'entrée (sans valeur maximum).
- Enfin, de manière à limiter l'envoi de requêtes de recherche sur le réseau, le champ *Delay* permet de régler l'intervalle de temps qui s'écoule entre deux requêtes.

```
    …
    <DataInDescription Key="key" Delay="1000" NumberMin="1"
NumberMax="-1" Type="int" Unit="none">
        <Name>KeyboardActivityContextor</Name>
        <ContextGroup> System/Keyboard/ </ContextGroup>
        <Location>
            World/Europe/France/RhoneAlpes/Isere/Grenoble/
        </Location>
```

```
<Identificafion>
   <Login>Login1</Login>
   <Password>Pass1</Password>
</Identificafion>
<MetaData>
   <AbsoluteUncertainty> true </AbsoluteUncertainty>
   <InitialDate> true </InitialDate>
   <CurrentDate> true </CurrentDate>
</MetaData>
<DataSendingMode>
   <AfterDelay Delay="100">true</AfterDelay>
</DataSendingMode>
</DataInDescription>
…[...]...
```

Le fichier se termine par les tags DataDescription et CONTEXTOR_DESCRIPTION qui ferment les deux sections précédemment ouvertes.

```
   …
   </DataDescription>
</CONTEXTOR_DESCRIPTION>
```

Fichiers de configuration de contexteur élémentaire et de contexteur d'adaptation

La structure d'un fichier de configuration d'un contexteur élémentaire est identique à celle d'un contexteur non élémentaire à ceci près qu'elle ne contient pas de description des données d'entrée. On le rappelle, un contexteur élémentaire ne fait qu'encapsuler un capteur. Il n'a pas de données d'entrée.

A contrario, le fichier de configuration d'un contexteur d'adaptation comprend la description des données d'entrée mais est dépourvu de donnée de sortie. De plus, parcequ'il n'est pas destiné à être découvert, sa description d'un adaptateur est grandement simplifiée : le contexteur d'adaptation est un composant logiciel inclus dans une application cliente. Son rôle est de prendre en charge la liaison entre cette application et l'infrastructure des contexteurs.

L'assistant de configuration

Du point de vue du programmeur, créer un nouveau contexteur revient à :

- créer une classe java qui étend l'une des trois classes abstraites caractérisant le type de contexteurs choisi (non élémentaire, élémentaire ou adaptateur).
- produire un fichier de configuration.

L'assistant de configuration permet de créer ce fichier, évitant au programmeur une tâche fastidieuse.

L'assistant de configuration :

- génère les fichiers de configuration des différents composants de l'infrastructure,

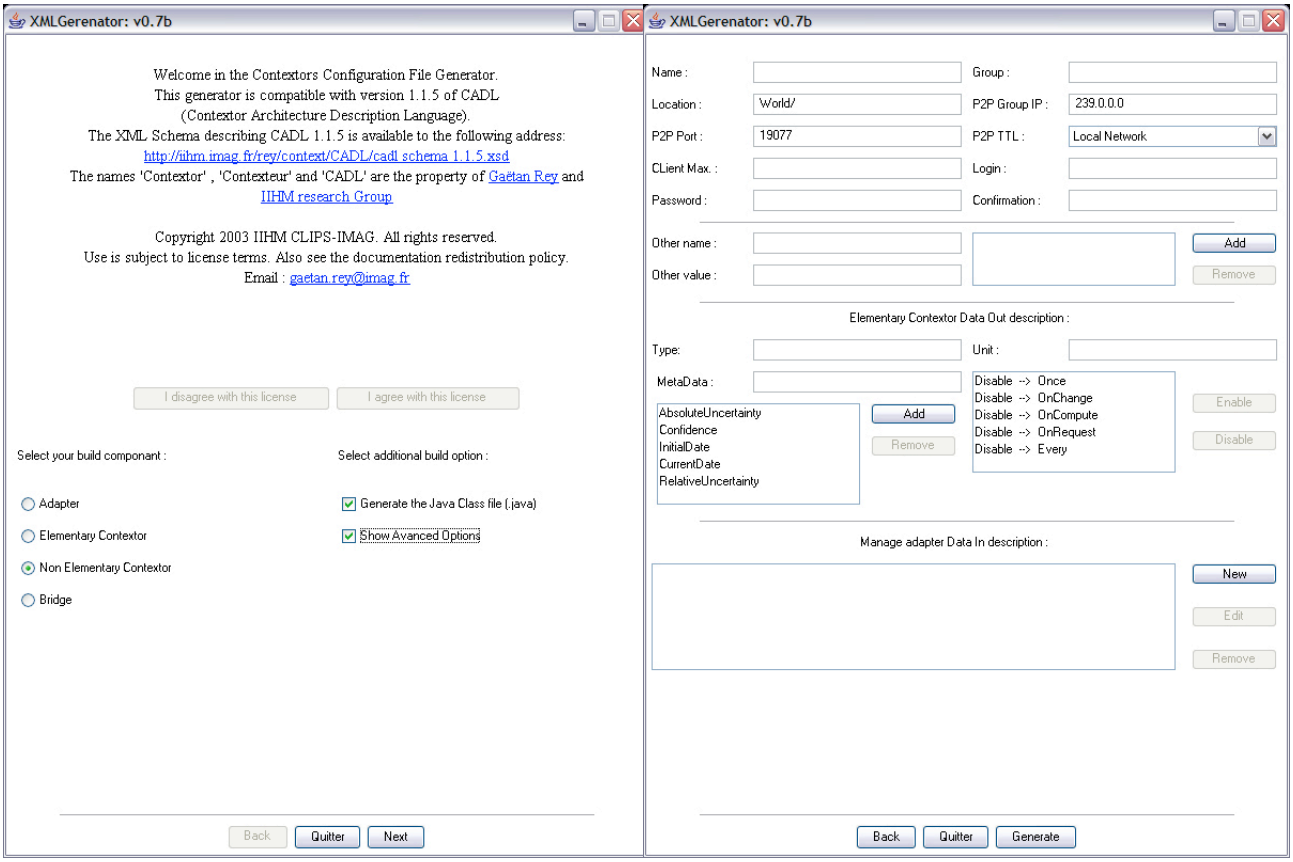


Figure 5
Interface du générateur de fichiers de configurations pour les contexteurs.

- garantit la conformité des fichiers générés avec les schémas XML de description,
- simplifie l'écriture des fichiers de configuration et
- a la portabilité de Java 1.4.

Comme le montrent les captures d'écran de la figure 5, l'utilisation de l'assistant consiste à remplir les champs d'un formulaire. Au lancement, l'utilisateur indique la classe de contexteurs à laquelle il s'intéresse (non élémentaire, élémentaire ou adaptateur). Cette information permet au configurateur d'adapter l'interface pour ne proposer que les options utiles à cette catégorie de contexteur. Une fois le formulaire rempli et si ce dernier est valide (une vérification des valeurs saisies est effectuée pour avertir l'utilisateur de la validité de ses choix), le générateur produit le fichier de configuration conformément aux schéma XML définis.

De plus, deux options s'offrent à l'utilisateur dans le but de simplifier sa tâche :

- **Génération de code** : si cette option est activée, l'assistant génère, en plus du fichier de configuration, un squelette de fichier java correspondant au fichier écrit.

- **Options avancées** : si cette option n'est pas activée, l'interface de saisie du formulaire cache certains points de l'écriture du fichier de configuration comme les aspects réseaux. Les choix par défaut seront alors appliqués. Si l'utilisateur se considère comme expert, il active cette option pour modifier les valeurs par défaut.

Une nouvelle version d'assistant, en cours de développement, permettra la génération des fichiers de configuration des répéteurs.

2.3. COMMUNICATIONS ET MESSAGES

Les messages échangés entre les contexteurs sont codés en XML. L'interopérabilité entre contexteurs et l'extensibilité des données motivent ce choix. L'interopérabilité va de pair avec le requis de configuration dynamique. L'extensibilité des descriptions est nécessaire dans un domaine en pleine évolution comme l'informatique ambiante. En particulier, personne n'a encore cerné le périmètre des métadonnées. Les données et les métadonnées sont encodées à l'aide du protocole [Base64]. Ce protocole augmente la taille des messages, de même la consommation des ressources pour l'encodage et le décodage des messages. Inversement, il facilite et accélère l'analyse des messages en évitant l'utilisation de caractères de balisage XML à l'intérieur des données. Les messages entre contexteurs sont de quatre types : recherche, contrôle, connexion et données.

Les requêtes de recherche

Les requêtes de recherche permettent la découverte des contexteurs sources nécessaires aux contexteurs qui les émettent. Il s'agit ici de messages envoyés sur le réseau virtuel des contexteurs. Cette émission, on l'a vu, se fait en utilisant le protocole multicast UDP. L'exemple ci-dessous présente un message de recherche typique. Encadré par deux balises SEARCH, le message comprend deux sections : la description des cibles (les contexteurs sources recherchés), suivie de l'identité de l'émetteur de la requête.

```
<SEARCH>
   <Target>
      <Key>Temperature</Key>
      <Name>*</Name>
      <ContextGroup>System</ContextGroup>
      <Location>
         World/Europe/France/RhoneAlpes/Isere/
      </Location>
      <Data type="float" unit="Celsius" />
      <DataSendMode>OnChange,OnRequest</DataSendMode>
   </Target>
```

La balise Target désigne la cible recherchée : elle est identifiée par la clé *Temperature*, elle doit être située en *Isère*, appartenir au groupe *System* et cela quel que soit son nom (le joker * indique que le champ peut prendre n'importe quelle valeur). De plus, la requête spécifie que le contexteur

recherché doit être capable de fournir ses données sous la forme d'un réel (*float*) correspondant à une température en degrés Celsius et qu'il doit assurer les modes d'échange *OnChange* et / ou *OnRequest*. Pour cette cible, aucune précision n'est spécifiée sur les métadonnées.

```
<Target>
   <Key>Meteo</Key>
   <Name>MeteorologyContextor</Name>
   <ContextGroup>*</ContextGroup>
   <Location>
      World/Europe/France/
   </Location>
   <Data type="*" unit="*" />
   <DataSendMode>AfterDelay=1000,OnChange</DataSendMode>
   <MetaData>initialDate,confidence</MetaData>
</Target>
```

La seconde cible décrite dans la requête de recherche ci-dessus est identifiée par la clé *Meteo*. Les contexteurs répondant aux critères de cette cible sont les contexteurs ayant pour nom *MeteorologyContextor* situés en France. Aucune information n'est spécifiée sur leur groupe ni sur le type et l'unité des données. Par contre, la cible doit autoriser deux modes d'échanges de données : *AfterDelay* et *OnChange* et deux méta données (initialDate et confidence). Le mode *AfterDelay* est réglé sur une seconde (1000 millisecondes).

```
   <Identity>
      <Id>...</Id>
      <DirectHostName>...</DirectHostName>
      <DirectPort>...</DirectPort>
   </Identity>
</SEARCH>
```

Une fois décrites les cibles recherchées (il n'y a pas de limite sur le nombre de cibles), le message de recherche décrit son expéditeur. L'identité de l'expéditeur comprend trois paramètres :

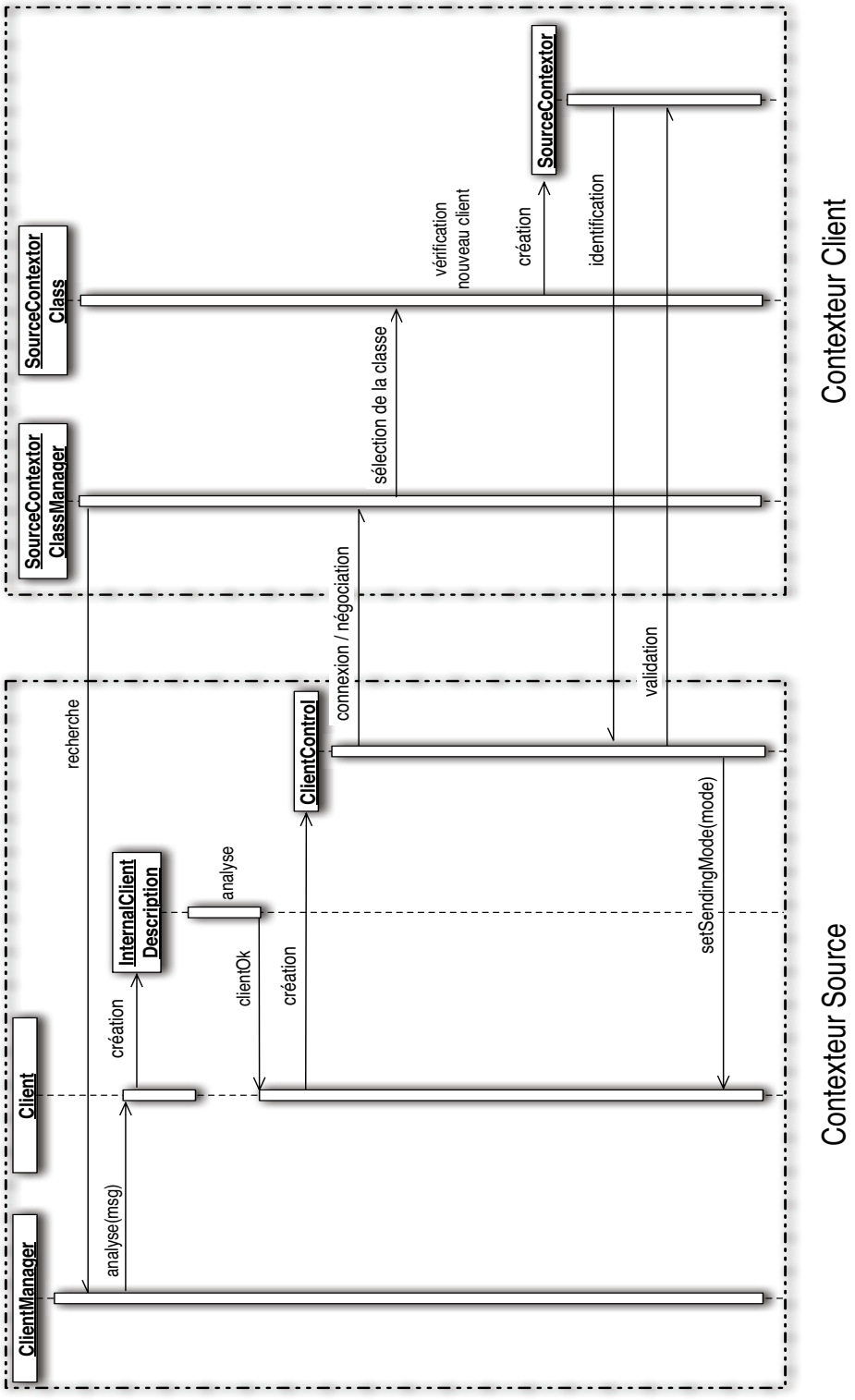
- L'identifiant unique (Id) qui caractérise le contexteur émetteur du message de recherche. Cet identifiant, on le rappelle, est généré par le contexteur dans l'état 2 de son cycle de vie. L'identifiant permet au contexteur source de ne pas répondre à une requête de recherche provenant d'un contexteur client auquel il fournit déjà des données.
- L'adresse IP ou le nom de domaine (DirectHostName) de la machine hébergeant le contexteur émetteur de la recherche.
- Le numéro de port (DirectPort) qu'écoute le *ClientManager* du contexteur émetteur.

Les deux derniers paramètres permettent au contexteur source d'établir une connexion directe (TCP) avec son futur client comme cela est présenté à l'étape 3 de la figure 4. Une fois la connexion établie, le contexteur envoie un message de connexion.

Les messages de connexion et d'identification

Sous ce nom, on regroupe trois types de messages servant à établir la connexion entre un contexteur source et un contexteur client.

Figure 6
Diagramme de séquence représentant les phases de connexion et d'identification des contexteurs.



Le premier message est le message de connexion. Il est envoyé par la source vers le client en réponse à la requête du client (message détaillé dans la partie précédente). Ce message a la forme suivante :

```
<CONTEXTOR_CONNECTION>
   <Key>Meteo</Key>
   <DataSendingMode>OnChange</DataSendingMode>
</CONTEXTOR_CONNECTION>
```

Comme décrit précédemment, le message de connexion comporte deux champs :

- *Key* indique à quelle cible du message de recherche le contexteur source doit fournir des données. Ici la cible *Meteo* de l'exemple précédent.
- *DataSendingMode* spécifie le mode choisi par le contexteur source parmi la liste des modes proposés dans le message de recherche du contexteur client. Ici le mode *OnChange* a été sélectionné.

En réponse au message de connexion, le contexteur client émet le message d'identification suivant :

```
<CONTEXTOR_IDENTIFICATION>
   <Login> monLoginDeTest </Login>
   <Password> monPasswordDeTest </Password>
</CONTEXTOR_IDENTIFICATION>
```

A la réception de ce message, le contexteur source vérifie que le contexteur client a les autorisations pour utiliser ses services. Si ce n'est pas le cas, il indique au contexteur client que l'identification a échoué à l'aide du message suivant avant de fermer la connexion entre eux.

```
<CONTEXTOR_VALIDATION>
   Failed
</CONTEXTOR_VALIDATION>
```

Par contre, si l'identification réussit, le contexteur source envoie au contexteur client le message suivant :

```
<CONTEXTOR_VALIDATION>
   Succeed
</CONTEXTOR_VALIDATION>
```

Maintenant que nos deux contexteurs sont connectés, ils vont pouvoir échanger des données.

Les messages d'échange de données

Les messages d'échange de données correspondent à l'envoi par un contexteur d'une donnée et de ses métadonnées vers un client. Les messages de données utilisent la connexion directe (TCP) établie lors de la phase de connexion et d'identification. Relativement simple, ces messages ont la forme suivante :

```
<CONTEXTOR_DATA>
   <Id>8053<Id>
   <Data> les données encodé en base64 </Data>
   <MetaData>
```

```
        <initialDate>une meta donnée encodé en base64 </initialDate>
    <MetaData>
</CONTEXTOR_DATA>
```

En plus des données et des métadonnées encodées en Base64, le message contient un identifiant correspondant au numéro du message envoyé. Cet identifiant est incrémenté de un en un après chaque envoi de message. Il permet au contexteur client d'ignorer d'anciens messages qui se seraient perdus sur le réseau.

Les requêtes de contrôle

Les requêtes de contrôle permettent de modifier les relations entre un contexteur client et un contexteur source. Pour ce message, le protocole TCP est utilisé via la connexion établie pendant la phase "connexion et identification". Il existe plusieurs types de requêtes de contrôle. En voici quelques exemples :

Requête de fermeture :

```
<CONTEXTOR_REQUEST>
    <Close> true </Close>
</CONTEXTOR_REQUEST>
```

Requête demandant l'envoie de donnée :

```
<CONTEXTOR_REQUEST>
    <Request> true </Request>
</CONTEXTOR_REQUEST>
```

Requête demandant l'arrêt des échanges pendant une minute :

```
<CONTEXTOR_REQUEST>
    <Control> wait=«60000» </Control>
</CONTEXTOR_REQUEST>
```

Requête demandant la modification du délai d'envoie du mode *AfterDelay* :

```
<CONTEXTOR_REQUEST>
    <Control> modifyAfterDelay=«50» </Control>
</CONTEXTOR_REQUEST>
```

Requête demandant le remplacement du mode d'envoi actuel par le mode *OnChange* :

```
<CONTEXTOR_REQUEST>
    <Control> modifySendingMode=«Onchange» </Control>
</CONTEXTOR_REQUEST>
```

Jusqu'ici, nous avons présenté le fonctionnement " nominal " des contexteurs. La section qui suit analyse plus avant les problèmes de déconnexion.

3. Les déconnexions

Les problèmes que nous évoquions lors de la présentation du cycle de vie du contexteur, concernent les déconnections des contexteurs sources en fonction de l'état dans lequel se trouve le contexteur.

3.1. GESTION DES DÉCONNEXIONS :

Soit un contexteur C. Considérons la déconnexion d'au moins un contexteur source. Celle-ci a trois effets selon l'état de C:

Cas 1 : C est connecté à plus de contexteurs sources que son besoin minimal. C est soit dans l'état 5. Soit dans l'état 2 (toutes les classes de sources ne sont pas nécessairement approvisionnées). Alors, il convient de distinguer trois sous-cas possibles :

- Le contexteur recherche un nombre maximal infini de sources : il continue sa recherche et ne change pas d'état.
- Le contexteur cherche un nombre maximal fini de sources et il n'a pas encore atteint ce nombre : il continue sa recherche et ne change pas d'état.
- Le contexteur cherche un nombre maximal fini de sources et il a atteint ce nombre : il faut qu'il reprenne la recherche d'une nouvelle source sans pour autant changer d'état.

Cas 2 : C est connecté à moins de contexteurs sources que son besoin minimal. Le contexteur est encore dans l'état 2, il cherche les contexteurs sources dont il a besoin pour fonctionner. Il continue sa recherche et reste dans l'état 2.

Cas 3 : le contexteur est connecté à exactement son nombre minimal de contexteurs sources nécessaires et la perte d'une source le fait descendre en-dessous de ses besoins de fonctionnement. Là encore trois sous cas sont possibles :

- Le contexteur est dans l'état 2. Il reste dans l'état 2 en attendant qu'il rétablisse ses besoins. Le contexteur doit relancer une recherche sur la classe de données d'entrée correspondante à la classe du contexteur source perdu si et seulement si son nombre de sources minimal est égale à son nombre de source maximale. Dans le cas contraire une recherche est déjà en cours.
- Le contexteur est dans l'état 3 ou dans l'état 4. Il stoppe sa fonction de calcul (et retourne dans l'état 3) puis retourne dans l'état 2 en attendant qu'il rétablisse ses besoins. De la même manière que dans le cas 2, une recherche de sources est relancée si nécessaire.
- Le contexteur est dans l'état 5. Le manque de sources oblige le contexteur à arrêter sa fonction de calcul. Or, cette fonction alimente actuellement les clients du contexteur. La version développée actuellement informe les clients de son arrêt, se déconnecte des clients (retourne dans l'état 4), puis gère le problème comme s'il était

initialement dans l'état 4 (cas précédent). Cependant cette solution, qui a le mérite d'être simple et efficace, a un effet de bord qui peut être problématique.

3.2. AMÉLIORATION DE LA SOLUTION ACTUELLE :

La déconnexion d'un contexteur qui a des clients peut provoquer des déconnexions en chaîne. En effet, le client peut ne plus être approvisionné. Il doit alors arrêter sa fonction de calcul et ce faisant ne plus approvionner un client, et ainsi de suite. Cette chaîne de déconnexions, non seulement paralyse une partie de l'infrastructure de contexteurs, mais aussi provoque un flux de requêtes de recherche qui à leur tour risquent d'écrouler le réseau et donc d'aggraver la paralysie du service contextuel.

De manière à relativiser un peu le problème, il faut penser que :

- Tous les contexteurs n'ont pas de dépendance (directe ou non). Une telle situation paralysera seulement la partie liée à la première déconnexion.
- Tous les contexteurs ne fournissent pas un service de manière unique. Il devrait être fréquent de trouver plus de contexteurs sources que le nombre minimal souhaité.

Cependant, de manière à résoudre ce problème, une gestion améliorée des déconnexions est en cours de développement. Cette amélioration apportera les modifications suivantes :

- Notification de mise en attente des clients avec possibilité de négociation (au moment de la connexion et / ou au moment de la mise en attente) de la durée maximale de cette mise en attente. Cette solution permet d'économiser de la bande passante en évitant la réémission de message de recherche et permet d'accélérer la reconstitution de chaîne en simplifiant les phases de négociation et d'identification (celles-ci sont inutile puisque les contexteurs restent connectés).
- Prédiction de déconnexions à l'aide de détecteur de défaillances [Marchand 03] avec recherche préventive de nouvelles sources. L'objectif de cette méthode est de détecter de manière préventive les possibles pannes sur les contexteurs (congestion réseau, contexteur instable ...) de manière à lancer une recherche et de trouver de nouvelles sources avant que les défaillances soient réelles. Cette solution permet d'éviter qu'un contexteur ne puisse rendre sont service à cause d'un manque de source et donc améliore la disponibilité des contexteurs.

Jusqu'ici, nous avons envisagé l'infrastructure de contexteurs indépendamment de sa couverture réseau. Comme annoncé au chapitre précédant, la globalité est traitée au moyen de répéteurs.

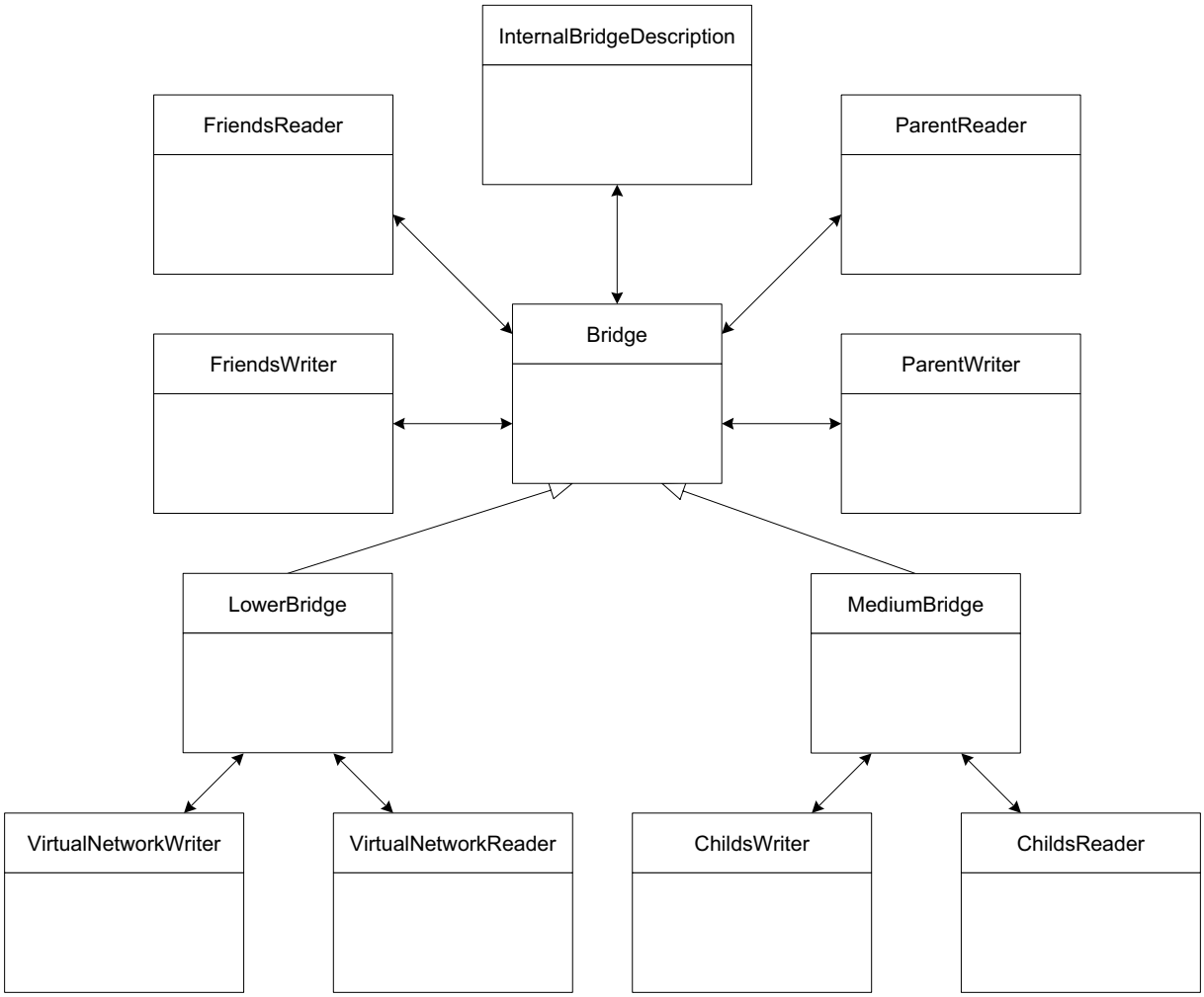


Figure 7
Diagramme de classes de l'infrastructure des répéteurs.

4. Les répéteurs

Les répéteurs, on le rappelle, forment une architecture annexe à celle des contexteurs. Leur mission est le routage des messages de recherche sur de grandes distances et ceci de manière transparente pour le programmeur de contexteurs. Ce routage a pour but d'éviter l'inondation excessive des réseaux. Les répéteurs forment une sorte de squelette qui relie les îlots dans lesquels évoluent les contexteurs. Enfin, on rappelle que les répéteurs sont organisés selon un modèle hybride (mi-hiérarchique, mi-pair à pair) fondé sur une répartition géographique.

4.1. DIAGRAMME DE CLASSES

L'infrastructure des répéteurs est réalisée selon les douze classes présentées dans la figure 7. Voyons en détails comment fonctionne actuellement les répéteurs.

Bridge est la classe abstraite qui prend en charge les fonctions principales des répéteurs. Elle crée l'objet *InternalBridgeDescription* à l'aide d'un fichier de configuration initiale. Puis elle lance les modules de communication entre les répéteurs. Cette classe étant abstraite, elle doit être étendue. Les classes *LowerBridge* et *MediumBridge* en sont des spécialisations.

- La classe *LowerBridge* représente les répéteurs de bas niveau. Il s'agit des répéteurs qui interceptent et réémettent les messages de recherche des contexteurs. Pour cela, le *LowerBridge* utilise un *VirtualNetworkReader* et un *VirtualNetworkWriter.*
- La classe *MediumBridge* représente les répéteurs de niveau supérieur. Ils ne sont pas en contact avec le réseau virtuel des contexteurs.

Enfin, trois classes *Reader* et trois classes *Writer* prennent en charge la communication entre répéteurs. Les *ParentWriter* et *ParentReader* permettent de communiquer avec le répéteur de niveau géographique directement supérieur, les *ChildsWriter* et *ChildsReader* permettent de communiquer avec le répéteur de niveau géographique directement inférieur et les *FriendsWriter* et *FriendsReader* servent à la communication entre les répéteurs de niveau géographique équivalents et de même parent géographique.

Prenons quelques exemples illustrant le fonctionnement des répéteurs.

4.2. EXEMPLE DE FONCTIONNEMENT

Dans les exemples qui suivent, la description des zones géographiques a été simplifiée de manière à ce qu'elles soient rapidement compréhensibles (les notions administratives de départements et de régions variant d'un pays à l'autre).

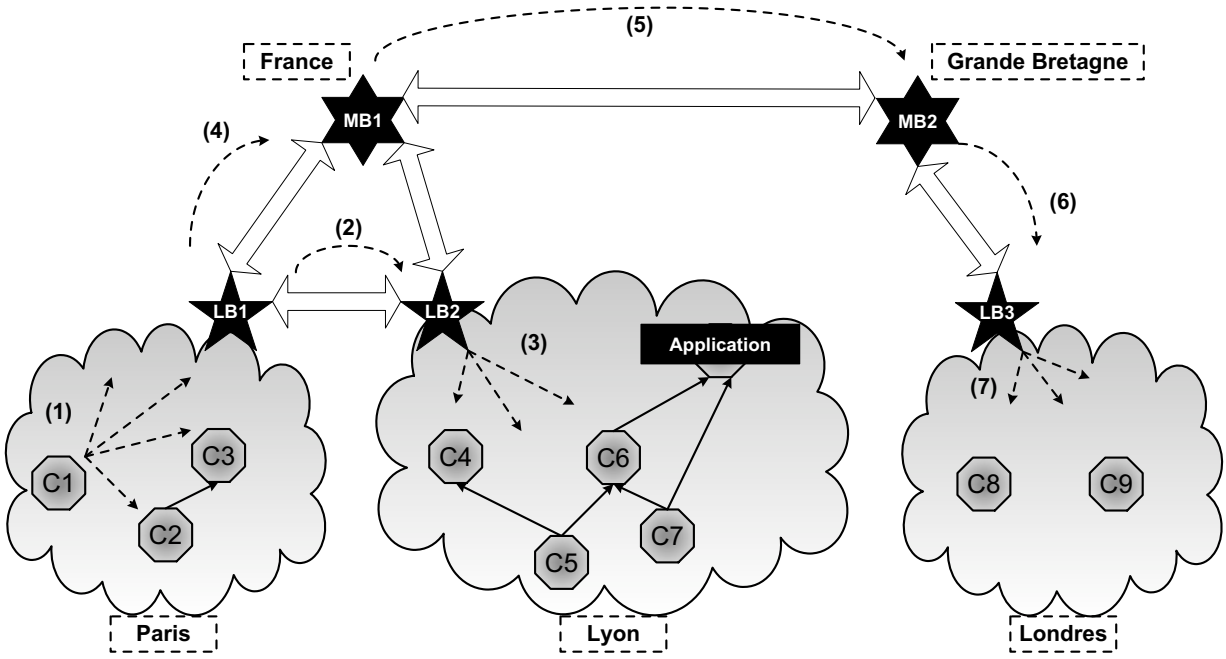
Dans l'exemple de la figure 8, le contexteur C1 a besoin, pour fonctionner, de deux données différentes. Il envoie une requête de recherche (étape 1 de la figure 8) indiquant qu'il a besoin d'un contexteur C4 situé dans la zone géographique de Lyon et un contexteur C9 qui se trouve dans la zone de Londres. Les contexteurs C2 et C3 analysant le message de recherche ne répondent pas car ils ne correspondent pas à ce qui est demandé. De son coté, le répéteur LB1, de type *LowerBridge*, analyse aussi le message de recherche et en déduit qu'il est destiné à d'autres zones géographiques. Connaissant le répéteur gestionnaire de l'une de ces zones, il recompose un message de recherche ne contenant que la cible concernant le contexteur C4 de Lyon et transmet ce message à son "ami", le répéteur LB2 (étape 2 sur la figure 8). LB2 diffuse ce message sur son réseau comme l'aurait fait un contexteur de ce réseau (étape 3 de la figure 8). C4 reçoit le message et initie une procédure de connexion (directe) avec C1.

Revenons sur LB1 et sur la deuxième cible du message qu'il a intercepté. Ne connaissant pas le répéteur gérant la zone géographique de Londres, LB1 transfère le message (étape 4 de la figure 8) à son répéteur parent (MB1). MB1, de type *MediumBridge* et gérant de la zone géographique France, ne connaît pas la zone de Londres mais il sait que le répéteur MB2 gère la zone englobante. Il lui transfère le message comme l'illustre la figure 8 (étape 5). Les zones géographiques étant définies suivant le formalisme décrit dans ce chapitre (section 2.2), il est relativement facile de connaître les zones englobantes d'une zone géographique donnée. MB2 est en mesure de transférer le message au répéteur LB3 chargé de Londres (étape 6 de la figure 8). À son tour, LB3 peut émettre la requête sur le réseau virtuel des contexteurs dont il a la charge (étape 7 de la figure 8).

Dans l'exemple de la figure 9, notre contexteur C1 recherche tous les contexteurs C6 se trouvant en Grande Bretagne. Pour les trouver, il émet une requête de recherche sur le réseau virtuel local (étape 1). Comme dans l'exemple précédant, le message est reçu par les contexteurs C2 et C3 qui ne donnent pas suite. De plus, la requête est aussi reçue par le répéteur LB1 qui, ne connaissant pas la zone Grande Bretagne, transmet le message à son parent (étape 2 de la figure 9). MB1 connaissant quel répéteur gère la zone géographique de la Grande Bretagne, lui transmet le message (étape 3 de la figure 9).

Alors que dans le premier exemple le gros du travail était assuré par le répéteur LB1 (découpage de la requête initiale en deux requêtes plus

Figure 8
Illustration du fonctionnement des répéteurs bas niveau (LB) et intermédiaires (MB) : exemple 1.



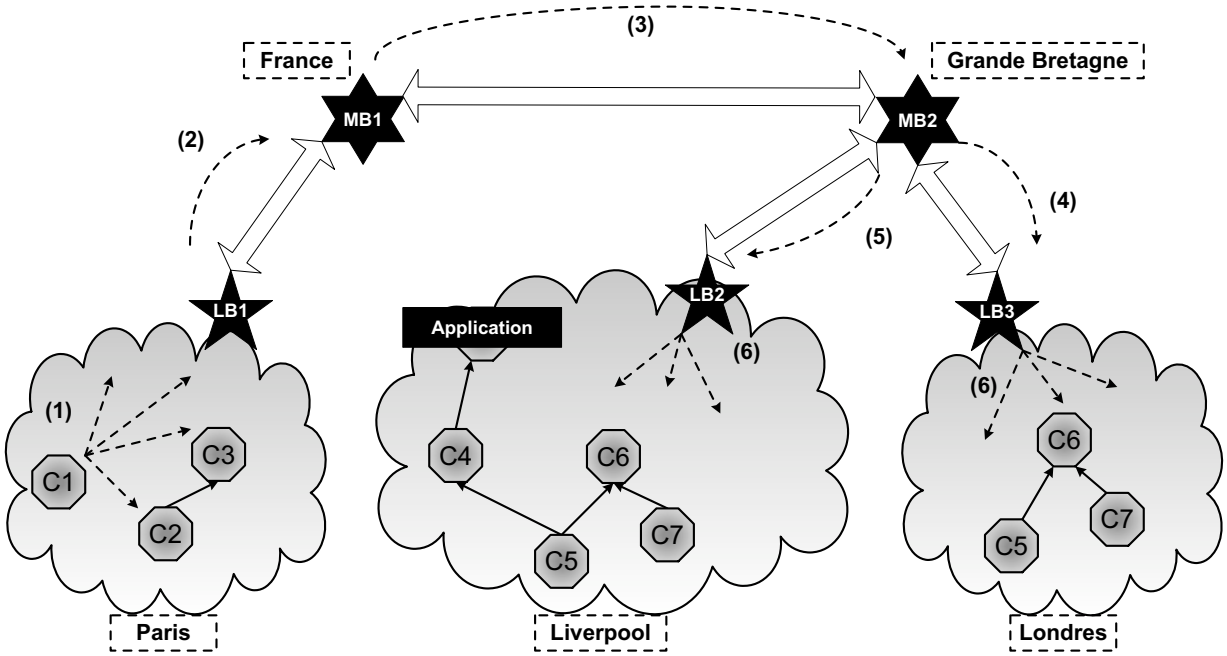


Figure 9
Illustration du fonctionnement des répéteurs bas niveau (LB) et intermédiaires (MB) : exemple 2.

simples), ici, MB2 gère les difficultés. En effet, lors de la réception du message, MB2 constatant qu'il est destiné à toute sa zone géographique, doit émettre des requêtes vers ses répéteurs fils. Or, il doit aussi modifier la zone de couverture de chacune des requêtes pour éviter que ses fils ne lui retournent le message ou ne se l'échangent entre eux. Le message transmis au répéteur LB3 (étape 4 de la figure 9) a pour zone de couverture Londres alors que le message transmis au répéteur LB2 (étape 5 de la figure 9) a pour zone de couverture Liverpool. Les deux répéteurs LB2 et LB3 se limitent à transmettre dans leur réseau virtuel respectif le message de recherche en provenance de MB2.

5. *Evalutation préliminaire de l'infrastructure de contexteurs*

La mise en œuvre ici présentée a pour but essentiel de démontrer la faisabilité technique du modèle conceptuel et son contrat en termes de flexibilité et de fiabilité. Cet aspect est étudié au chapitre suivant avec la présentation d'applications utilisant l'infrastructure de contexteurs.

Au chapitre II sur les infrastructures logicielles de contexte, nous nous interrogions sur les coûts induits par ces outils sur la consommation de

ressources et la latence. Les mesures ici présentées apportent quelques éléments de réponse, bien que partiels, il faut bien le dire.

Les tests de mesure du surcoût des contexteurs, dans leur implémentation actuelle, ont été réalisés sur la plate-forme suivante :

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

5.1. CONSOMMATION DE RESSOURCES MÉMOIRE

L'objectif ici est de mesurer les ressources mémoire consommées par un contexteur. Pour cela, on a utilisé un contexteur élémentaire dont l'unique fonction est la transmission d'un texte.

À l'initialisation, une fois configuré, le contexteur de test charge un fichier contenant un extrait de Candide de Voltaire dont la taille est de 25Ko. Ensuite il attend la connexion de clients auxquels il transmet le texte toutes les secondes (mode Every avec Delay=1000). De manière à obtenir une valeur significative de la consommation de ressource mémoire, 5000 exemplaires de ce contexteur a été exécuté dans la même machine virtuelle sur une durée d'un mois. L'observation a été réalisée à l'aide d'un processus Windows externe qui a mémorisé la taille du processus Java (JVM + 5000 Contexteurs) toutes les minutes pendant la période de test.

Le dépouillement du fichier de log indique que la taille mémoire utilisée par un contexteur est comprise entre 90 Ko et 60 Ko. À cette taille, il faut soustraire la taille des données transmises par le contexteur (soit 25Ko) ainsi que 1/5000éme de la taille mémoire utilisées par la machine virtuelle (celle-ci étant répartie entre chacun des contexteurs).

On constate qu'un contexteur ne consomme que peu de mémoire (entre 40 et 70 Ko). Par conséquent, un déploiement sur de petits dispositifs comme les PDA et les téléphones portables est envisageable.

5.2. RÉACTIVITÉ / LATENCE

Mesurer la latence permet de savoir si les contexteurs peuvent être utilisés dans des tâches d'interaction à fortes contraintes temporelles. C'est le cas notamment de l'Interaction Fortement Couplée qui exige une latence système inférieure à 50 ms [Bérard 99]. Cette mesure de latence exprime également la pénalité qu'introduit chaque segment d'une chaîne de contexteurs.

Le protocole expérimental reprend le contexteur élémentaire du test précédent servant de source à un contexteur non élémentaire particulier. Ce contexteur particulier a pour rôle de mesurer le temps de transfert des données en provenance d'une seule source et à destination d'un seul client.

Après un mois de test, et cela sur un millier de contexteurs en parallèle, la latence intrinsèque introduite par un contexteur est de l'ordre de 20 ms. Ce

temps correspond au temps que mettent les données pour traverser un contexteur sans que celui n'effectue de traitement.

Cette valeur indique que les contexteurs, tels qu'ils sont implémentés actuellement, ne permettent pas de faire de l'interaction fortement couplée si la chaîne de contexteurs comprend plus de deux segments. Cependant, bien que la fonction de calcul du contexteur non élémentaire soit vide (temps de calcul égal à zéro), les données transmises sont de taille importante. Or, dans la majorité des cas, les éléments capturés et les données transmises par les capteurs et donc par les contexteurs, auront une taille beaucoup plus petite. Et puisque l'encodage et le décodage des données en Base64 correspond à la majorité du temps de calcul des contexteurs, la latence de chaque contexteur est fortement liée à la taille de ses données.

Par conséquent, avant de sacrifier les contexteurs du fait de l'importante latence observée par ce test, il conviendrait d'évaluer avec précision l'impact de la taille des données sur la latence.

5.3. BILAN ET PERSPECTIVES

Ces résultats sont encourageants pour plusieurs raisons :

- Les contexteurs consomment peu de mémoire et peuvent donc être utilisés sur de petites machines.
- En Interaction Fortement Couplée, les contexteurs sont envisageables à condition de limiter la chaîne de coopération. Le mécanisme d'encapsulation introduit au chapitre III permet de réduire la chaîne.
- La réalisation actuelle, bien que prototype de recherche, est capable de fonctionner sans panne pendant un mois (au moins).

Cependant, d'autres tests sont nécessaires pour confirmer ces résultats. Sont envisagés à court terme :

- L'évaluation de la consommation de ressources CPU pour confirmer ou infirmer l'utilisation possible des contexteurs sur de petits dispositifs.
- L'évaluation de l'impact de la taille des données sur la latence ainsi l'impact de la longueur des chaînes de contexteurs.
- Des "stress tests" pour vérifier la stabilité des contexteurs lors de connexions / déconnexions massives de client et / ou de sources.

D'autres expérimentations sont actuellement en cours. Elles concernent notamment les tests sur l'architecture hybride (avec les répéteurs) et la mobilité de l'utilisateur.

Chapitre V *Illustrations*

Les cellules, trop longtemps solitaires, se retrouvent solidaires. Un monde haut en couleur s'épanouit : les espèces naissent, meurent, se diversifient. La vie croît et se multiplie.

1996, La plus belle histoire du monde. Acte 2 - Scène 3 : L'explosion des espèces, p93

Nous venons de voir la mise en œuvre technique du cœur d'une infrastructure de contexteurs. Ce chapitre offre un point de vue complémentaire : l'exploitation de l'infrastructure par le programmeur et sa validation par des applications clientes. Nous les présentons par ordre de complexité croissante : un observatoire d'activités de personnes, une Machine d'Interaction Abstraite qui permet le couplage dynamique de ressources d'interaction, et Ethylene, un intergiciel pour la plasticité des Interfaces Homme-Machine. Chacune de ces applications est ici présentée à partir du cadre méthodologique du chapitre I, expression abstraite traduite ensuite en termes de contexteurs qui, à leur tour, trouvent leur réalisation technique grâce à l'infrastructure de contexteurs.

1. Observatoire d'activité

Inspiré du In/Out board présenté au chapitre II, l'observatoire d'activité mesure l'utilisation des machines de l'équipe IIHM du laboratoire CLIPS. Comme le montre la figure 1, le résultat de ces mesures est affiché dans une page Web sous forme d'un tableau à raison d'une case par machine. Chaque case contient :

- un titre qui regroupe l'adresse IP et le nom de la machine observée,
- le nom de l'utilisateur identifié sur la machine,
- les dates des dernières activités clavier et souris sur cette machine, et
- une icône représentant le niveau d'activité (rouge pour beaucoup d'activité, orange pour une activité moyenne et jaune pour peu d'activité) complétée par un entier compris entre 0 et 100 qui précise ce niveau d'activité.

Ainsi, Laurence, connectée à la machine mobylet, est actuellement très active. Inversement, Planet et Secret n'ont pas d'utilisateurs. Nicolas, sur Grimpet, n'est plus très actif. Voyons comment, du point de vue du développeur, l'observatoire est conçu puis implémenté. En accord avec les principes méthodologiques du chapitre I, la finalité du système doit d'abord être précisée dans les termes du cadre conceptuel : Entites, Relations et Roles.

1.1. MONDELISATION

L'observatoire met en jeu les entités (humain et ordinateur) et observe l'activité d'utilisateurs sur leur ordinateur. On obtient :

- *Roles* = {∅}
- *Relations* = {*utilise*}
- *Entites* = {*humain1*, ..., *humainN, ordinateur1*, *ordinateurN*}

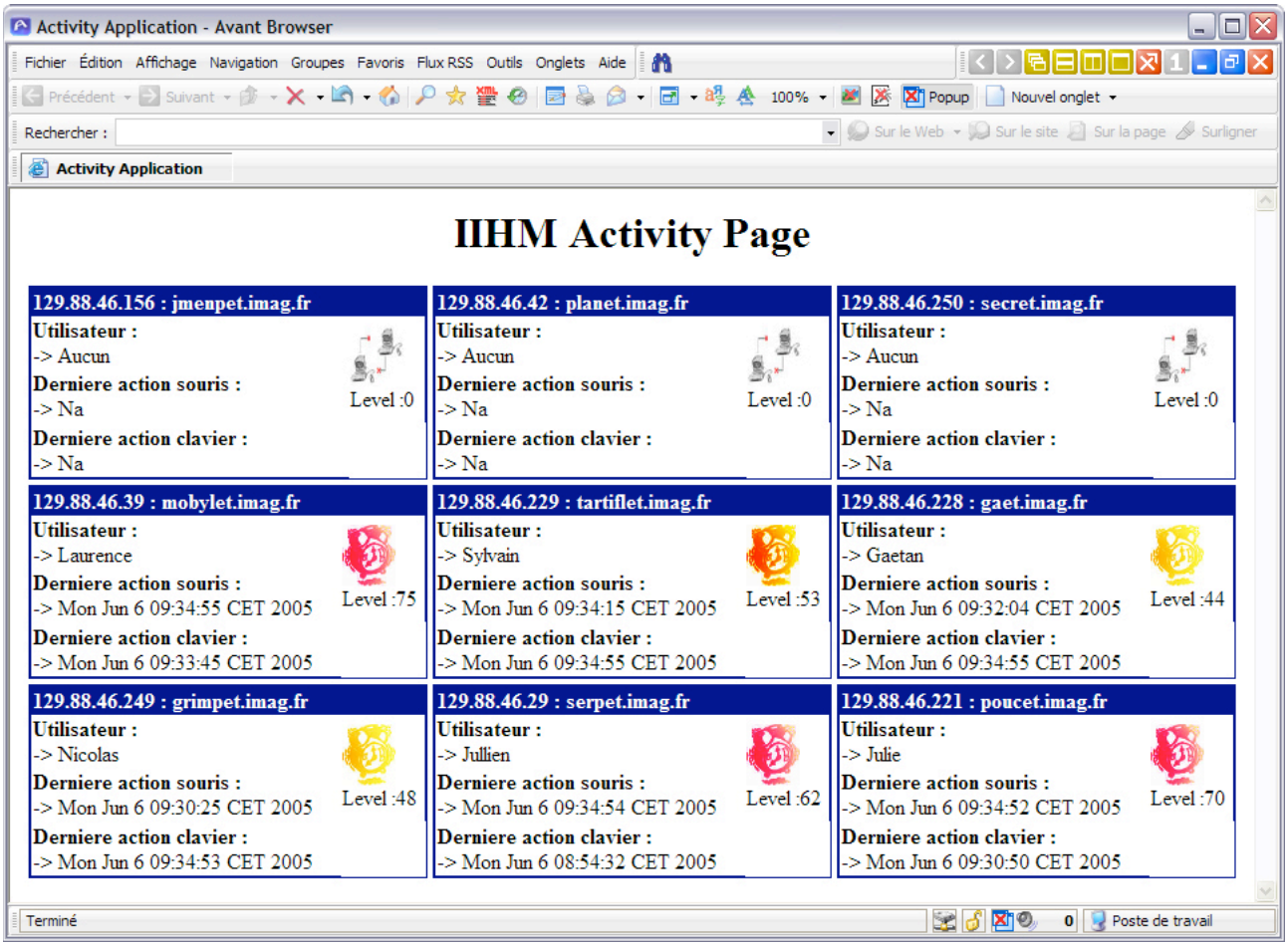


Figure 1
Exemple de page web affichant le résultat de l'observatoire d'activité

Comme l'indique la figure 2, nous obtenons deux contextes : l'ensemble des rôles est vide et la cardinalité de Relations vaut 1. La figure représente l'ensemble des situations considérées pour un seul humain utilisant au plus un seul ordinateur : dans le cas de cette application, la modélisation complète avec tous les utilisateurs et ordinateurs possibles n'apporte aucune information supplémentaire.

Analysons plus avant les cinq situations des deux contextes C1 et C2 :

Regardons en détail les cinq situations de nos deux contextes C1 et C2.

- La situation S1 du contexte C2 correspond au cas où l'utilisateur exploite son ordinateur. La mesure de cette activité consiste à mesurer les flux des périphériques d'entrée de l'ordinateur, sachant que l'utilisateur est identifié (loggé).
- La situation S1 du contexte C1 correspond au cas où il n'y a aucune entité.

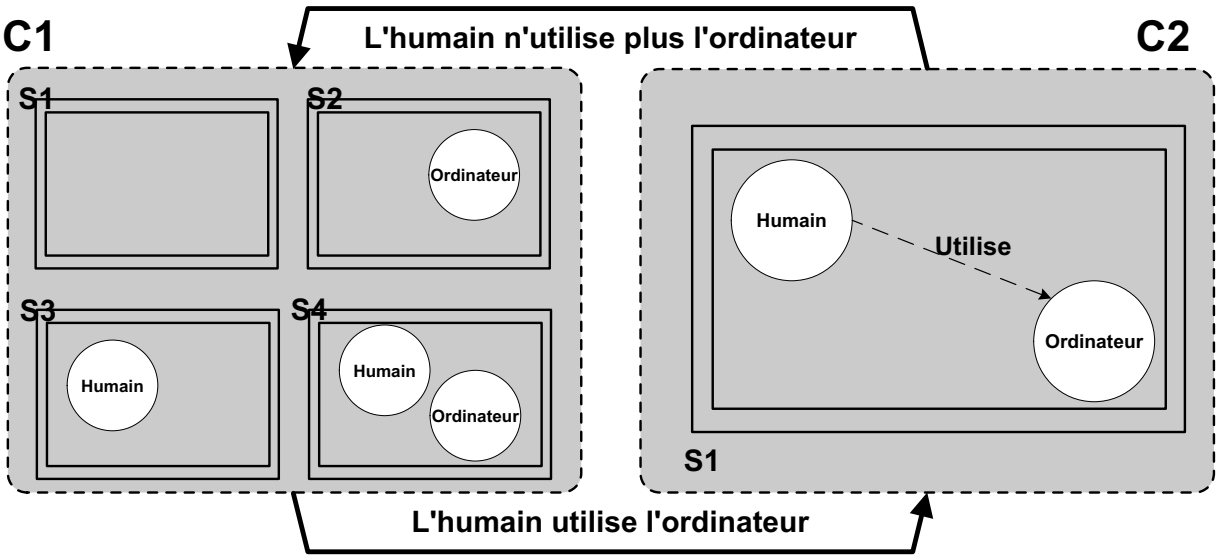


Figure 2
Modélisation du réseaux de contextes pour l'observatoire d'activité.

- Dans la situation S2 de C1, l'utilisateur est absent : les flux des périphériques d'entrée sont nuls et l'utilisateur n'est pas identifié sur la machine.
- Dans la situation S3 du contexte C1, l'humain se retrouve seul sans machine. Du point de vue technique, cette situation sera confondue avec la situation S1 de C1.
- Dans la situation S4 du contexte C1, l'utilisateur est présent, mais n'utilise pas son ordinateur : l'utilisateur est identifié, mais les flux des périphériques d'entrée sont nuls.

En résumé, l'observatoire d'activité est intéressé par les informations suivantes :

- Identité des machines observées,
- Identité des humains utilisant ces machines,
- Activité des flux des périphériques d'entrée sur les machines observées.

Il nous faut maintenant définir les contexteurs capables de fournir ces observables et de les synthétiser sous forme d'un niveau d'activité.

1.2. LES CONTEXTEURS

Au vu de la nature des observables requis, il nous faut :

- des contexteurs élémentaires pour capter les flux d'information clavier et souris, de même l'identité des machines et des utilisateurs. Les contexteurs élémentaires sont installés et lancés sur chaque machine à observer en attente de demande de service.

- au moins un contexteur non élémentaire qui synthétise au bon niveau d'abstraction et sur chaque machine, les observables fournis par les contexteurs élémentaires,
- un adaptateur qui sert de pont entre l'infrastructure et l'application: il exprime les services requis par l'observatoire d'activités. L'interprétation des requêtes de l'adaptateur donne lieu à la construction d'un réseau de contexteurs. Comme énoncé au chapitre III, l'adaptateur fait partie de l'application. (ici, une servlet).

L'adaptateur d'activité

L'adaptateur d'activité permet au programmeur d'application d'exprimer le service requis. Sans reprendre l'ensemble des paramètres du fichier de configuration décrit au chapitre IV, examinons quelques-uns de ses attributs :

```
<P2PDescription>
   <P2PGroup>239.0.0.0</P2PGroup>
   <P2PPort>19077</P2PPort>
   <P2PTTL>1</P2PTTL>
</P2PDescription>
```

Ici, le paramètre *P2PTTL* est positionné à 1 : la recherche des contexteurs couvre uniquement le réseau local. Les deux autres paramètres correspondent à des valeurs par défaut. Ces paramètres déterminent le réseau virtuel des contexteurs. Si nous changeons leurs valeurs, l'adaptateur d'activité recherchera des contexteurs sur un autre réseau virtuel. L'adaptateur possède une seule classe de données d'entrée décrite comme suit ;

```
<DataInDescription Key="activite" Delay="1000" NumberMin="1"
NumberMax="-1" Type="xml" Unit="none">
      <Name>LocalActivityContextor</Name>
      <ContextGroup> * </ContextGroup>
      <Location>
         World/Europe/France/RhoneAlpes/Isere/
      </Location>
      ...
      <MetaData>
         <InitialDate> true </InitialDate>
         <CurrentDate> true </CurrentDate>
      </MetaData>
      <DataSendingMode>
         <AfterDelay Delay="500">true</AfterDelay>
      </DataSendingMode>
   </DataInDescription>
```

Cette description exprime les requis suivants :

- L'adaptateur recherche au moins un contexteur source (*NumberMin="1"*), il n'a pas de limite haute (*NumberMax="-1"*), et la recherche sera effectuée toutes les secondes (*Delay="1000"*) jusqu'à obtention des contexteurs sources..

- Les données fournies par les contexteurs sources doivent être exprimées en XML (*Type="xml"*).
- Les contexteurs sources recherchés se nomment *LocalActivityContextor*. Ils doivent être situés en Isère (*World/Europe/France/RhoneAlpes/Isere/*) mais peuvent appartenir à n'importe quel groupe (**<***ContextGroup***> * </***ContextGroup***>**). doivent envoyer leurs données selon la méthode *AfterDelay* toutes les 500 millisecondes. Les métadonnées associées à ces données doivent inclure les dates de capture (*InitialDate*) et de fusion de l'information (*CurrentDate*).

Cette requête est reçue par un contexteur d'activité locale.

Le contexteur d'activité locale

Le contexteur d'activité locale répond aux requis de l'adaptateur. Il est automatiquement relié à l'adaptateur d'activité, s'il est lui-même prêt à fournir des données (c'est-à-dire s'il est dans l'état 4 présenté au chapitre IV). Ce contexteur fusionne les données provenant de différentes sources pour fournir à l'adaptateur un seul interlocuteur par machine observée.

Après son lancement et son initialisation (état 2), ses contexteurs sources sont recherchés. Ces sources sont au nombre de deux :

- Un contexteur d'information locale, représenté sur la figure 3, et
- Au moins, un contexteur d'activité : contexteur d'activité clavier et/ou un contexteur d'activité souris.

La recherche de ces contexteurs se fait au niveau micro : elle est limitée à la machine où est exécuté le contexteur d'activité locale. Une fois la connexion avec les contexteurs sources établie et les premières données reçues, le contexteur calcule sa première donnée. Il crée un message au format XML contenant l'ensemble des données recueillies (nom de l'utilisateur identifié, activité clavier, activité souris ...) qui est envoyé selon le protocole *AfterDelay* spécifié par l'adaptateur d'activité.

Les Contexteurs d'activité souris et clavier

Les contexteurs d'activité souris et clavier sont élémentaires : ils n'ont pas de données d'entrée, mais ils encapsulent un capteur. Ici, le capteur est un compteur sur la file d'événements système correspondant respectivement à la souris et au clavier.

Après s'être initialisé, c'est-à-dire avoir établi un lien avec le processus système de gestion des événements souris (respectivement clavier), le contexteur d'activité attend ses clients potentiels (dans notre cas, le contexteur d'information locale décrit ci-dessus). Une fois la connexion établie, le contexteur transmet la valeur de l'intensité de l'activité de l'utilisateur. Cette valeur correspond aux nombres d'événements souris (respectivement clavier) ayant lieu dans un intervalle de temps de 1 seconde.

Nous conviendrons que cette donnée est arbitraire et ne permet pas, à elle seule, de représenter l'activité de l'utilisateur devant sa machine. Mais c'est suffisant pour démontrer l'intérêt technique des contexteurs.

Le contexteur d'information locale

Le contexteur d'information locale est élémentaire. Son rôle est de fournir des informations sur la machine où il s'exécute, à savoir :

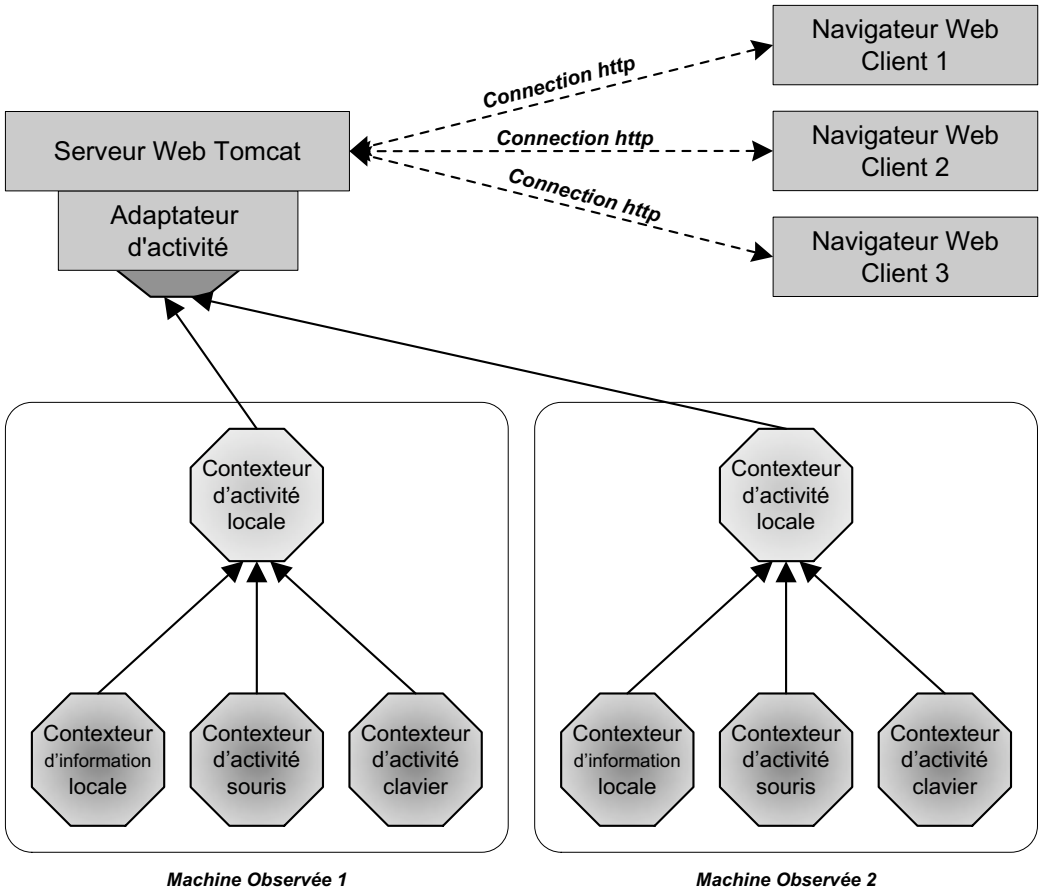
- le nom de la machine (hostname),
- le nom de l'utilisateur identifié sur cette machine
- un ensemble d'informations sur le système d'exploitation utilisé sur cette machine telles que le nom et la version du système, ...

On obtient l'architecture globale suivante.

1.3. ARCHITECTURE GÉNÉRALE DE L'OBSERVATEUR D'ACTIVITÉ

L'observatoire d'activité est une servlet exécutée par un serveur Tomcat 4.1. Cette servlet a des clients (des navigateurs web) où sont affichés les niveaux d'activité. Par son adaptateur d'activité, la servlet requiert et reçoit au bon niveau d'abstraction, des informations sur le niveau d'activité des machines à observer. Ces machines exécutent

Figure 3
Vue générale du fonctionnement de l'observateur d'activité.



chacune un réseau de contexteurs qui se construit dynamiquement à la demande de l'adaptateur. La construction ne peut avoir lieu que si les contexteurs élémentaires sources exigés sont lancés (c.-à.d dans l'état 4). Sur chaque machine, le réseau comprend :

- Un contexteur d'activité clavier qui fournit le nombre d'événements clavier par seconde sur la machine où il est exécuté ; et/ou un contexteur d'activité souris qui fournit le nombre d'événements souris par seconde sur la machine où il est exécuté.
- Un contexteur d'information locale qui fournit le nom de l'utilisateur identifié sur la machine, le nom de la machine (ainsi que d'autres informations non pertinentes pour cet exemple).
- Un contexteur d'activité locale qui fusionne les données de ses contexteurs sources élémentaires.

Grâce aux données fournies par l'ensemble des réseaux de contexteurs, la servlet navigue dans le réseau de contextes et de situations de la figure 2. Elle affiche les résultats en conséquence. Pour une machine donnée :

- On est dans la situation S1 du contexte C2 lorsque l'utilisateur et la machine sont identifiés et que le niveau d'activité n'est pas nul (dans l'IHM : icône orangée, étiquette différente de zéro).
- On est dans la situation S4 du contexte C1 si l'utilisateur est encore identifié, mais avec un flux d'activité nul (dans l'IHM : icône en jaune, étiquette égale à zéro).
- En situation S2 du contexte C1, l'utilisateur n'est pas identifié (dans l'IHM : absence de nom d'utilisateur).
- Etc.

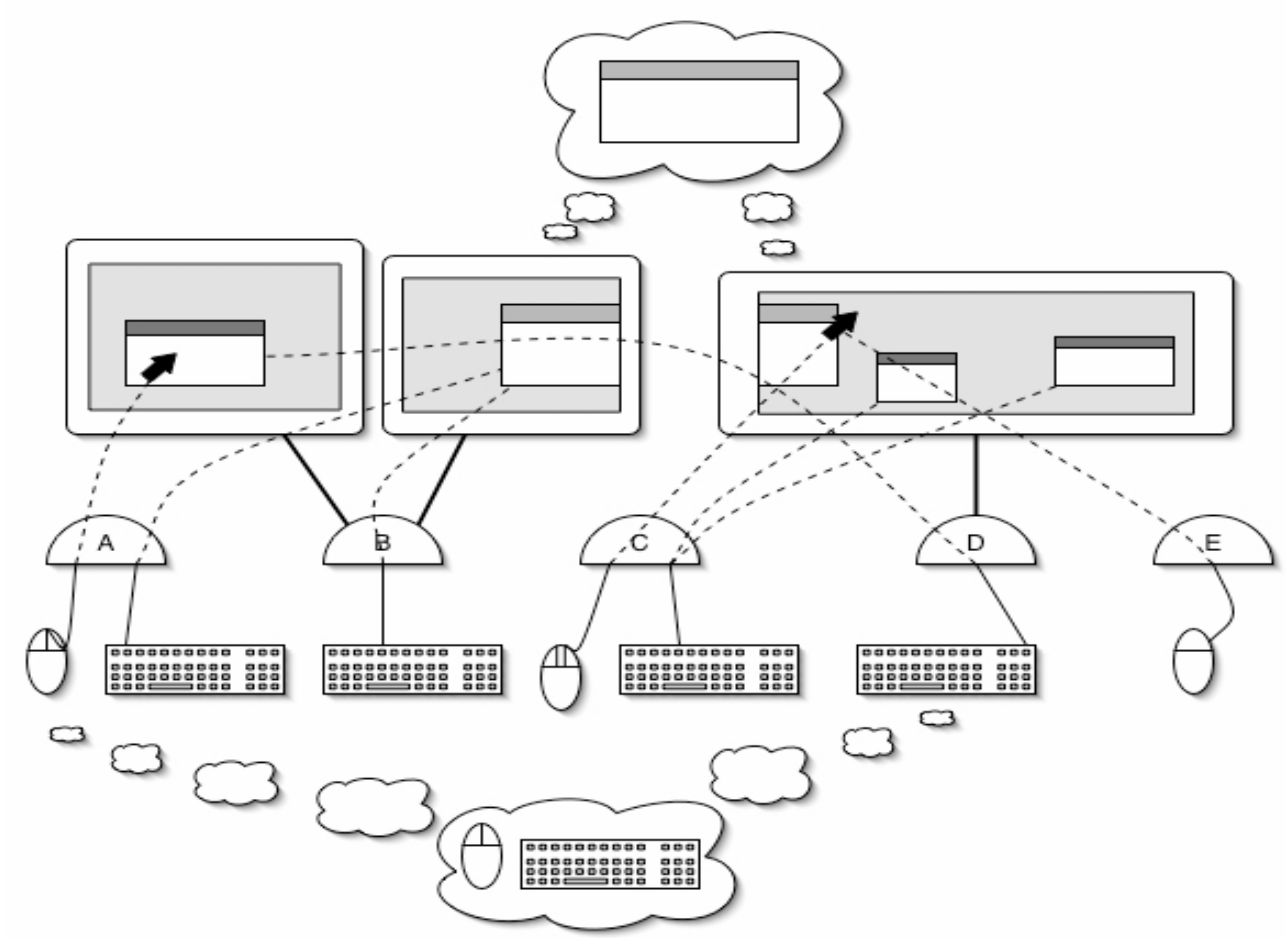
La figure 1 indique au moyen d'une icône particulière, que trois machines, jmenpet, planet et secret, ne sont pas présentes (situation S1 et S3 du contexte C1). On notera que l'observateur d'activité est capable d'indiquer l'absence de machines seulement si celles-ci lui ont déjà communiqué des informations via un contexteur d'activité local. L'observatoire d'activité n'a pas connaissance des machines observées, mais il mémorise les machines qui lui ont déjà fourni des informations contextuelles. Par conséquent, seulement une partie des situations S1 et S3 du contexte C1 est réellement représentée.

L'observatoire d'activité est une application jouet dont l'objectif était de tester l'infrastructure de contexteurs. Dans I-AM, l'infrastructure des contexteurs est exploitée au sein d'un intergiciel pour la gestion dynamique de ressources d'interaction.

2. *I-AM*

I-AM (Interaction Abstract Machine) est une machine du même niveau d'abstraction que les gestionnaires graphiques de base comme Xwindows [Lachenal 04]. Ces gestionnaires gèrent les ressources d'interaction d'une station de travail dont ils assurent le partage entre les applications clientes. Comme ces gestionnaires, I-AM est un intergiciel orienté Interaction Homme-Machine, mais I-AM en étend la couverture fonctionnelle pour tenir compte des requis de l'informatique ambiante. I-AM permet la construction dynamique d'espaces interactifs par couplage de ressources d'interaction gérées par des stations de travail distinctes qui exécutent des systèmes d'exploitation éventuellement différents. I-AM donne l'illusion, au programmeur comme à l'utilisateur, que ces ressources sont gérées de manière uniforme par une seule station de travail. La figure 4 en illustre les principes :

Figure 4
Exemple de configuration pour I-AM.
Dans cet exemple, I-AM unifie les ressources d'intéractions de cinq machines différentes (A à E).



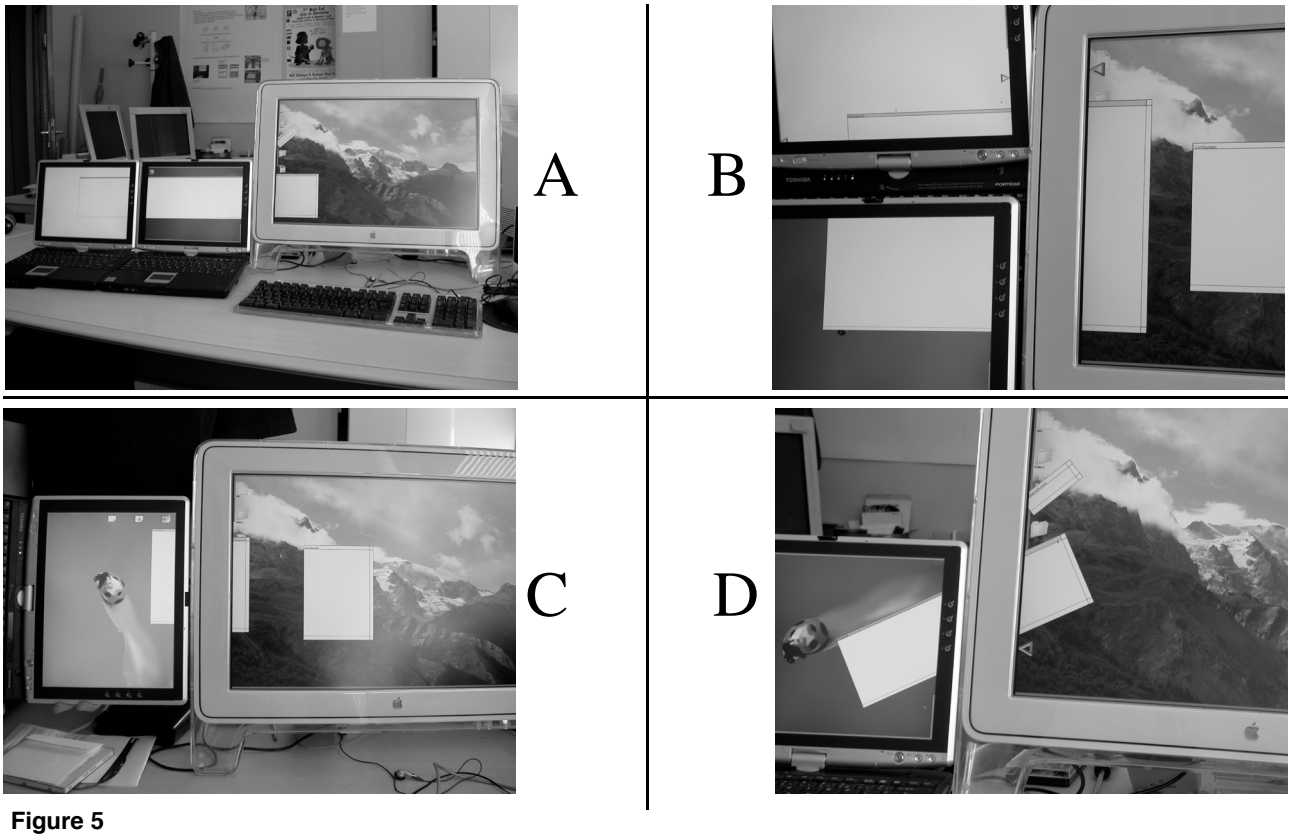


Figure 5
Photographies montrant les différentes utilisation de I-AM.
A) Couplage de trois écrans cote à cote. B) Couplage de trois écrans de manière non habituelle. C) Couplage de deux écrans avec rotation de l'écran de gauche. D) Couplage de deux écrans cote à cote et rotation de la fenettre à cheval.

- Au bas de la figure, la vue de l'utilisateur : dans cet exemple, l'IHM graphique est distribuée sur trois surfaces gérées chacune par une machine exécutant des systèmes d'exploitation distincts (MacOS, Windows XP, Windows NT). Les fenêtres peuvent être déplacées dans tout l'espace d'affichage que forment les trois surfaces au moyen de n'importe quel instrument. Par exemple, en utilisant la souris reliée à la machine de droite, l'utilisateur peut manipuler une fenêtre affichée sur la surface reliée à la machine de gauche.
- Au sommet de la figure, la vue du développeur. Ici, les fenêtres sont créées et gérées dans un espace logique uniforme. I-AM maintient la correspondance entre l'espace numérique logique et l'espace physique d'affichage. Dans l'illustration, la fenêtre la plus à gauche de l'espace logique tient sur une seule surface. D'autres, comme la fenêtre la plus à droite, sont réparties sur deux surfaces. Dans ce cas, l'interacteur logique de la vue du développeur est projeté sur chaque surface en deux interacteurs effectifs. Ces derniers sont affichés à la bonne échelle et à la bonne position en sorte que l'utilisateur ait l'illusion d'un interacteur unique. Ainsi, lorsque l'utilisateur manipule l'un des interacteurs effectifs, l'interacteur effectif "jumeau" réagit en

conformité avec les réactions du premier. D'une manière générale, le nombre d'interacteurs effectifs correspondant au rendu d'un interacteur logique dépend de la position de l'interacteur logique dans l'espace logique, des relations spatiales des surfaces et de la fonction de projection de l'espace logique sur l'espace physique.

La figure 5 illustre le couplage d'écrans connectés à des stations de travail Windows et MacOS. Ces écrans, de taille et de résolution différentes, sont assemblés par proximité selon différentes orientations et positionnement relatifs. Ainsi, les fenêtres peuvent migrer d'écran en écran comme si ces écrans étaient reliés à une même carte graphique, I-AM s'occupant d'appliquer la bonne transformation affine (translation, rotation, mise à l'échelle). En 5A, les écrans sont assemblés côte à côte de gauche à droite, les bords du bas étant alignés. En 5B, 3 écrans forment une mosaïque, une fenêtre se trouvant à cheval sur ces trois écrans. En 5C, l'écran de gauche a subi une rotation de 90 degrés. Comme le montre la figure 5D, I-AM assure une bonne continuité visuelle.

En résumé, I-AM :
- Assure la découverte dynamique des ressources d'interaction,
- Modélise les relations spatiales entre les ressources d'interaction et leur couplage,
- Masque l'hétérogénéité du matériel et des systèmes d'exploitation sous-jacents au bon niveau d'abstraction,
- Permet la distribution et la migration des IHM graphiques au niveau du pixel.

La découverte des ressources, la détection du couplage de surfaces par proximité et les relations spatiales s'appuient sur l'existence de contexteurs. Pour identifier ces contexteurs, procédons d'abord à la modélisation.

2.1. LA MODÉLISATION

Avant de nous lancer dans la modélisation de notre problème, rappelons les définitions des notions de ressource d'interaction et de couplage.
- Une ressource d'interaction, telle que la définie Lachenal dans sa thèse [Lachenal 04] à la page 16 est *une entité physique intervenant dans un canal d'interaction*. Dans notre cas, nous limiterons notre modélisation aux cas des écrans.
- La notion de couplage, introduite par Barralon dans [Barralon 04], est *l'action de lier deux entités de manière à ce qu'elles opèrent conjointement pour fournir de nouvelles fonctions*. Du fait de notre limitation à l'étude des écrans, nous nous intéresserons seulement aux couplages de ceux-ci.

Ces éléments nous amène à considérer les trois ensembles suivants :

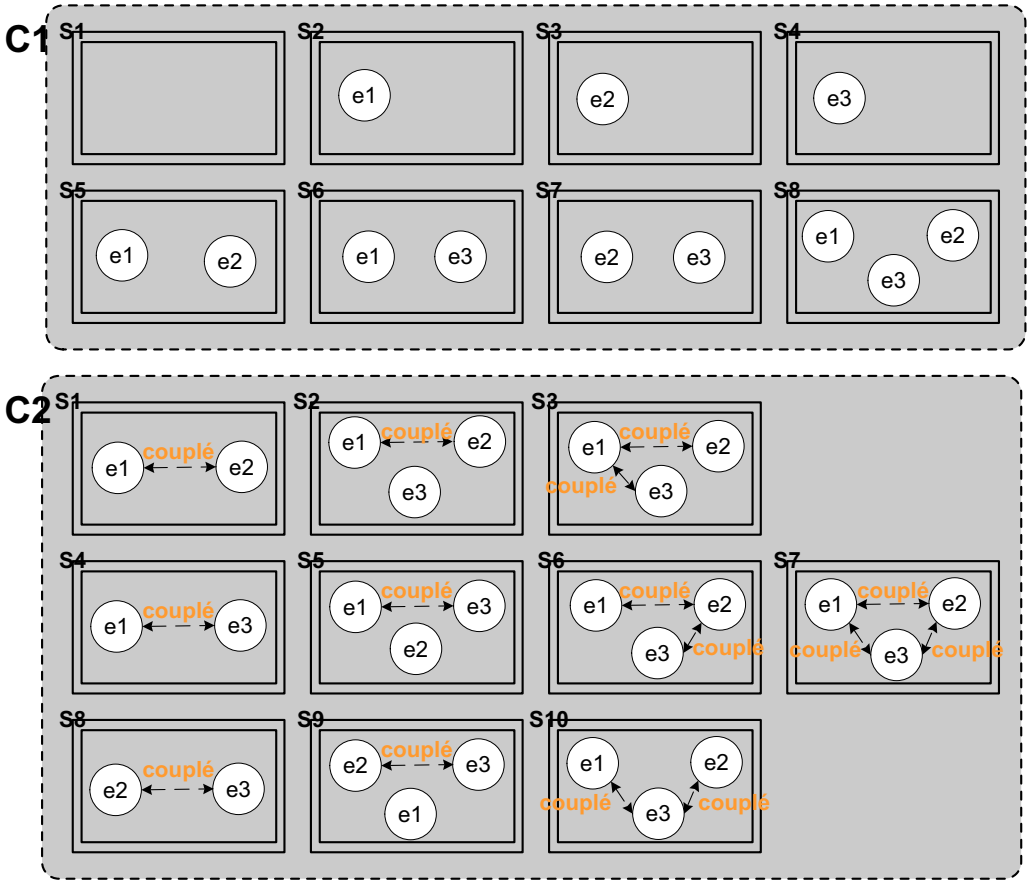
- *Entites* = {*e1*,*e2*,*e3*}. L'ensemble des entités considérées sont des écrans. Pour simplifier le raisonnement, nous nous limitons à trois écrans (e1, e2, e3).
- *Roles* = ∅. Tous les écrans sont supposés assurer en permanence le rôle de surface d'affichage. Ce rôle étant constant, cela revient à considérer qu'il est vide.
- *Relations* = {*couplé*}. Le couplage s'exprime par la relation *couplé(e1,e2)* vrai si les écrans e1 et e2 sont effectivement couplés, c'est-à-dire si les écrans sont assez proches pour être utilisés comme un unique écran. Dans I-AM, cette relation est symétrique (1), non réflexive (2) et non transitive (3).

$$(\forall e1 \in Entites \wedge \forall e2 \in Entites), \quad couplé(e1,e2) \equiv couplé(e2,e1) \tag{1}$$

$$(\forall e1 \in Entites), \neg couplé(e1,e1) \tag{2}$$

$$\forall e1 \in Entites \wedge \forall e2 \in Entites \wedge \forall e3 \in Entites, \quad \neg(couplé(e1,e2) \wedge couplé(e2,e3) \Rightarrow couplé(e1,e3)) \tag{3}$$

Figure 6
Modélisation des contextes et de situations pour l'exemple de la détection du couplage.



D'après la formule (7) du chapitre I, zéro rôle et une seule relation conduisent à considérer deux contextes distincts que décrivent les schémas de la figure 6. Le contexte C1=(R_1,Rel_1) correspond au cas où il n'y a ni rôle ni relation ($R_1 = \varnothing$ et $Rel_1 = \varnothing$). Le contexte C2=(R_2,Rel_2) correspond au cas où il n'y a aucun rôle et une seule relation ($R_2 = \varnothing$ et $Rel_2 = \{couplé\}$).

C1 est composé de huit situations (S1 à S8) différenciées par la présence des entités (ici, les écrans e1, e2 et e3). Les ensemble R1 et Rel1 étant vides, quelle que soit la situation de C1, il ne peut pas y avoir d'association rôle - entité. Par conséquent, toutes les situations composant le contexte C1 auront leurs ensembles *AssoRoEnt* et *AssoRelEnt* vide. Elles seront différenciées par la valeur de leur ensemble *Ent*. *Ent* étant une partie de *Entites* et ce dernier ayant une cardinalité de trois, nous avons bien huit situations possibles dans le contexte C1. Chacune de ces huit situations se distingue par la présence des différents écrans. On identifie ici l'une des deux fonctions des contexteurs dans I-AM : la détection des ressources d'interaction.

Dans le contexte C2, les dix situations possèdent toutes la relation *couplé :* $Rel = \{couplé\} \wedge AssoRelEnt \neq \varnothing$. Elles se différencient en fonction :

- des entités présentes (comme pour le contexte C1) ce qui correspond à la détection des ressources d'interactions par les contexteurs.
- des associations entités-relation (*AssoRelEnt* peut prendre différentes valeurs), ce qui correspond à la détection du couplage des ressources d'interactions par les contexteurs.

2.2. LES CONTEXTEURS

Pour détecter l'arrivée et le départ des entités écrans et leurs attributs (les observables), nous avons déployé les contexteurs suivants :

- le *DisplayContextor* qui décrit les écrans,
- le ProximityContexteur qui detecte la proximité entre deux écrans et leurs positions relatives.

Le Display-Contextor

Le *DisplayContextor* est un contexteur élémentaire dont la mission est de fournir la description des écrans connectés à l'ordinateur sur lequel il est exécuté. Ces informations incluent :

- Les attributs habituels : définition en pixel, fréquence de rafraîchissement horizontale, le nombre de couleurs (bit depth).
- Les attributs physiques externes : taille de l'écran (en millimètre), mais aussi épaisseur de chacun des bords (en millimètre), type (LCD ou CRT), modèle et fabricant.

Dans la mise en œuvre actuelle, certains attributs (résolution, bit depth ...) sont capturés dynamiquement par le contexteur. Les attributs physiques externes sont lus dans un fichier de description. De manière à rendre le

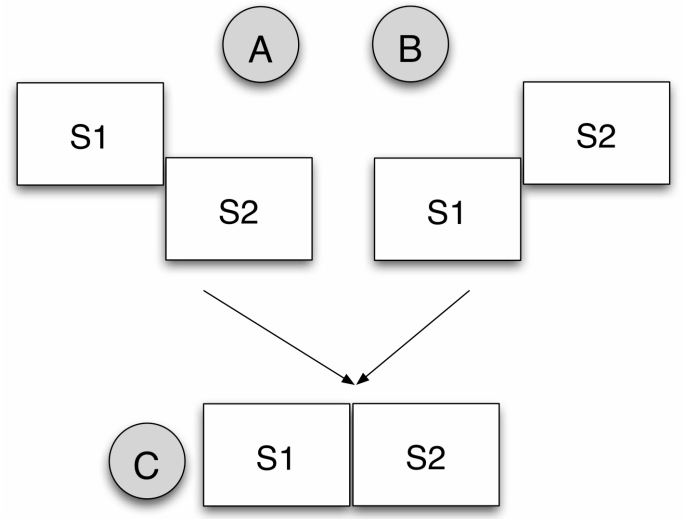


Figure 7
Couplage par gestes synchronisés et interprétation.

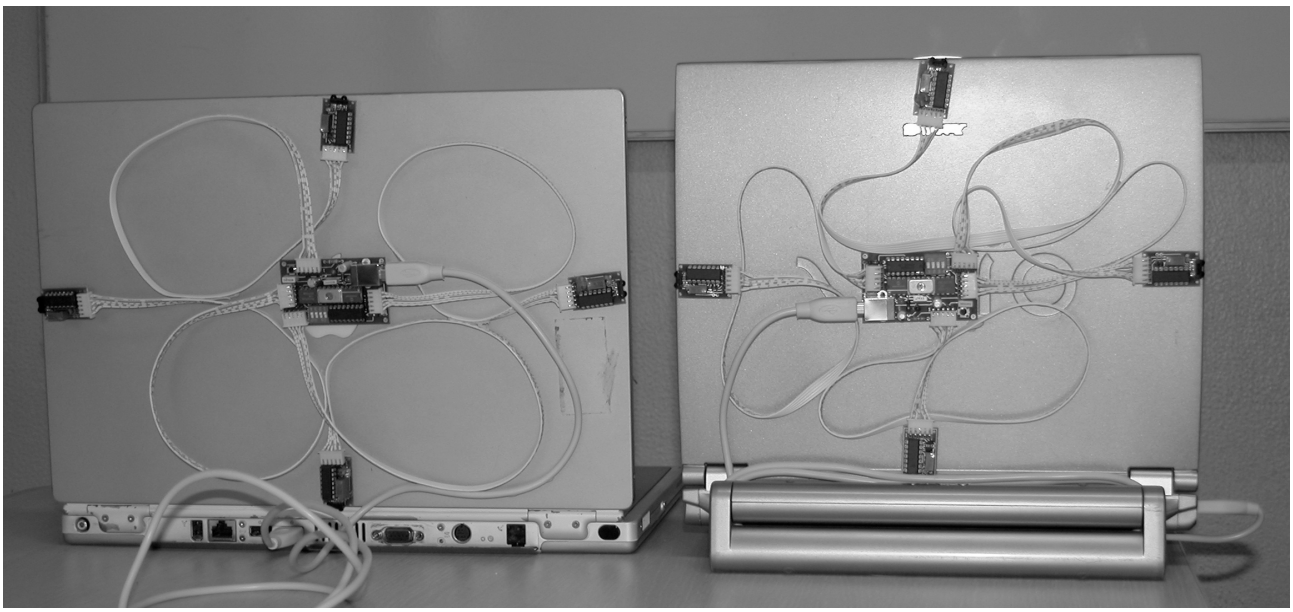
contexteur plus autonome, une solution sans fichier de description est à l'étude. Cette solution pourrait utiliser une base de données distante pour les informations physiques et le protocole de description des moniteurs pour les autres informations.

Proximity-Contexteur

Le couplage de deux écrans se fait par proximité. Le *Proximity-Contextor* encapsule un capteur de proximité que nous avons réalisé.

Après avoir étudié différentes solutions technologiques de la littérature (celles utilisées pour les ConnecTables [Tandler 01] ou par Hinckley [Hinckley 03]), nous avons jugé préférable de développer notre propre

Figure 8
Portables équipés de notre capteur de proximité.



capteur de proximité. Les raisons qui justifient ce choix sont les suivantes :

- La solution RFID des ConnecTables nécessite l'utilisation d'antennes dont la consommation électrique est pénalisante en situation mobile.
- Les accéléromètres utilisés par Hinckley permettent de détecter les côtés mis en jeu par le couplage de surface, mais pas leurs positions relatives. Ainsi des écrans mis en contact comme le cas A ou le cas B de la figure 7 produit l'interprétation C. Cette technique ne permet pas de réaliser la configuration B de la figure 5 où les positions relatives des bords sont significatives.

Par conséquent, nous avons développé un capteur à faible consommation pour une utilisation en mobilité, et pouvant fournir plus de précision sur les relations spatiales entre les surfaces [Barralon 05].

Comme le montre la figure 8, le capteur de proximité comprend un module principal, le PIC 16C745, quatre modules satellites d'émission/ réception infrarouge et un connecteur USB pour la connexion à un ordinateur. Les modules satellites sont fixés sur les bords de chaque écran (voir figure 8). Il y a détection de proximité lorsque deux satellites appartenant à des écrans différents " se voient ".

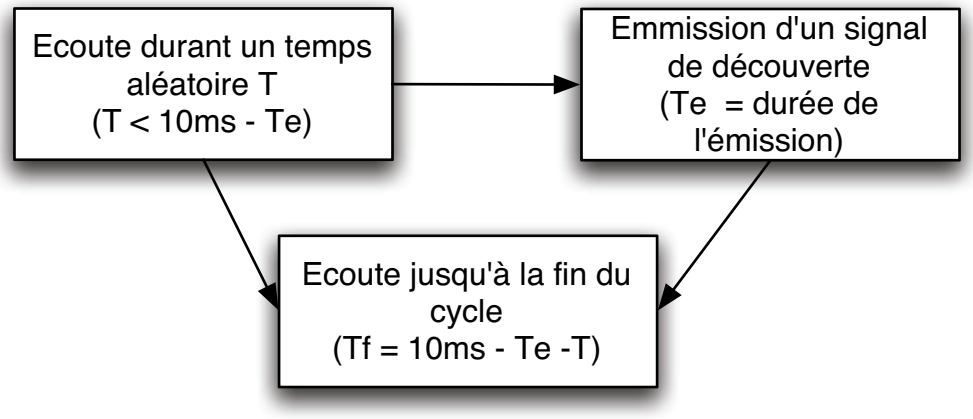
Le module principal

Le module principal PIC 16C745 assure les fonctions suivantes :

- Contrôle des modules satellites d'Emission/Réception (E/R),
- Coordination des travaux, processus d'acquisition et de traitements des données,
- Communication avec la machine hôte à laquelle il est connecté.

La fonction de contrôle correspond au cadencement des émissions de signaux de découverte. Un signal de découverte est émis régulièrement par chaque module satellite. Comme le montre le cycle de la figure 9, le

Figure 9
Cycle d'émission/réception du module principal.



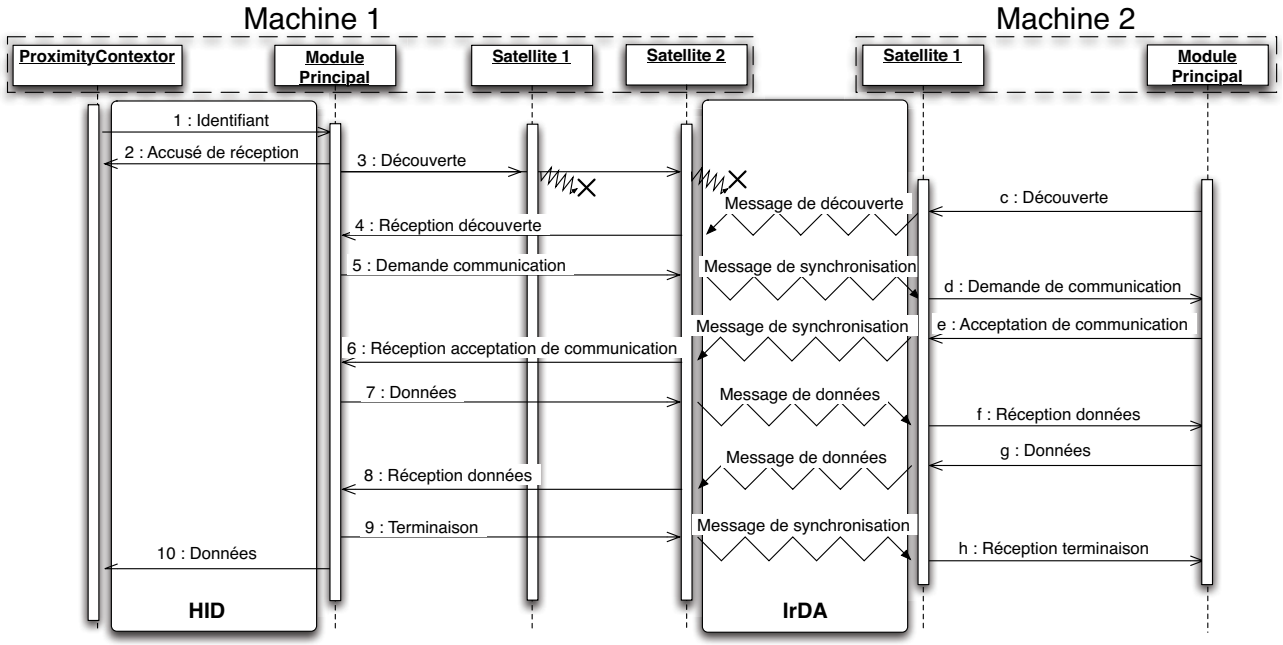


Figure10
Diagramme de séquence de la détection d'un couplage entre deux machines.
Pour plus de clarté, deux modules satellites ont été supprimé pour la machine 1 et seul le module principal et un satellite sont représentés sur la machine 2.

module principal commence par écouter pendant un temps aléatoire T (T < 10 ms - temps d'émission d'un signal de découverte) puis il émet le signal de découverte à destination d'un hypothétique module voisin (message 3 de la figure 10). Enfin, s'il reste du temps avant la fin du cycle (chaque cycle dure 10 ms), le module principal écoute à nouveau. Ce procédé permet :

- d'augmenter la probabilité de découverte (ou de disparition),
- d'être indépendant du démarrage des différentes machines (pas de synchronisation entre les machines),
- de permettre à deux entités physiques de se détecter plus rapidement par proximité.

Comme le montre la figure 10 (messages 4 à 9), quand deux entités physiques se détectent, elles échangent des données via les modules satellites. Les données comprennent :

- Un identifiant d'entité. Dans notre cas, l'adresse IP de la machine hôte de l'entité physique.
- Un identifiant de face. Un code correspondant à la face sur laquelle se trouve le module satellite (haut, bas, droite ou gauche).
- Des codes de détection d'erreur permettant la suppression des messages incomplets pour éviter la découverte de voisin fantôme.

La transmission des données entre deux modules satellites d'ordinateurs voisins s'effectue à l'aide du protocole [IrDA] garantissant une liaison fiable.

La fonction de coordination des travaux assure que la fréquence de détection de chaque module E/R est d'au moins 25 Hz tout en régulant la consommation. Pour économiser de l'énergie, quand un module satellite E/R a détecté la proximité d'un autre module satellite, le module principal lui ordonne de ne plus émettre le signal de découverte pendant un certain temps. Pendant ce temps, les autres modules E/R continuent à fonctionner normalement.

La fonction de communication du module principal avec l'hôte est réalisée par une liaison sur le bus USB. L'utilisation de ce mode de connexion offre les avantages suivants :
- un faible coût de l'interface,
- une alimentation via le câble,
- une indépendance vis-à-vis des machines hôtes,
- Hot Plug & Play,
- une liaison fiable et sécurisée,
- un mode veille lorsque l'on n'utilise pas le dispositif.

Les modules satellites

Les modules satellites ne sont que des transmetteurs/récepteurs infrarouges. Comme le montre la figure 10, ils ont pour fonction :
- l'émission des données transmises par le module principal vers l'extérieur (par exemple le message 9 de la figure 10).
- la réception de données externes qui sont transmises vers le module principal (par exemple le message 8 de la figure 10).

Le ProximityContextor

Du point de vue du *ProximityContextor*, le module principal entre dans la catégorie des périphériques [HID]. Cette catégorie est utilisée et est reconnue par une grande variété de systèmes d'exploitation (Windows, MacOS, …), ce qui garantit une meilleure portabilité du capteur de proximité. Le *ProximityContextor* pilote le capteur de proximité. Il informe le capteur de la machine sur laquelle il fonctionne de son identifiant d'entité (messages 1 et 2 de la figure 10) et réceptionne les données de couplage du capteur (message 10 de la figure 10). Quant aux données reçues du *DisplayContextor* elles servent à connaitre l'identifiant de l'écran auquel est rattaché le capteur de proximité.

2.3. L'INTÉGRATION DANS I-AM

De la décomposition fonctionnelle d'I-AM, deux composants nous intéressent ici : *I-AMPlatformManager* et *I-AMPhysicalTopology-Manager*.

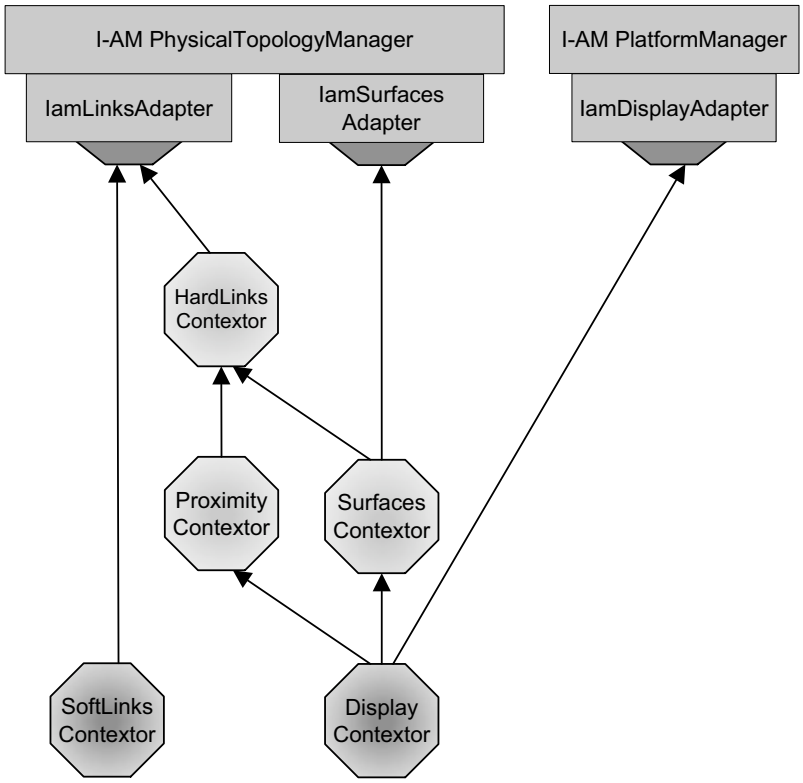


Figure 11
Schéma illustrant les cinq contexteurs et les trois adaptateurs de contexte utilisés par I-AM.

- *I-AMPlatformManager*, répliqué sur chaque machine de l'espace interactif, assure la gestion des ressources d'interaction locales à la machine et publie l'existence de ces ressources aux autres *I-AMPlatformManagers* de l'espace interactif. Ainsi, I-AM connaît, à tout instant, l'ensemble des ressources d'interaction disponibles dans l'espace interactif.
- *I-AMPhysicalTopologyManager*, répliqué sur chaque machine de l'espace interactif, gère la topologie physique des écrans locaux à cette machine en sorte qu'I-AM puisse projeter les espaces logiques des applications sur les surfaces physiques de chaque machine.

Conformément à notre modèle architectural, les deux composants, *I-AMPlatformManager* et *I-AMPhysicalTopologyManager* demandent les services des contexteurs par le biais d'adaptateurs de contexte: *IamDisplayAdapter*, *IamSurfacesAdapter*, *IamLinkAdapter*. Au bas de la hiérarchie des contexteurs, les *DisplayContextors* et les *SoftLinkContextors*. Le schéma de la figure 11 révèle la présence de contexteurs non élémentaires. Nous en justifions la présence dans la discussion qui suit.

2.4. LA DETECTION DE RESSOURCES D'INTERACTION

La détection des ressources d'interaction est mise en œuvre par deux contexteurs : le *DisplayContextor* déjà décrit, le *SurfacesContextor* et deux adaptateurs de contexte.

le Surfaces-Contextor

Alimenté par le *DisplayContextor*, le *SurfacesContextor* collecte les caractéristiques de toutes les surfaces (écrans) locales à la machine sur laquelle il s'exécute, en publie l'existence et tous les changements d'état. Une surface désigne une zone d'affichage [Lachenal 04]. Dans la majorité des cas, l'association surface-écran est très simple et la description de la surface correspond à la description de l'écran à laquelle le *SurfacesContextor* ajoute un identifiant unique de surface. Cet identifiant est de la forme: IP:D:S où IP est l'adresse IP de la machine à laquelle l'écran est connecté, D est le numéro de l'écran (0 pour le premier écran ...) et S est le numéro de création de surface (0 pour la première surface créée ...).

Les adaptateurs de contexte

Sans entrer dans les détails, regardons les fonctions du *IamDisplayAdapter* et du *IamSurfaceAdapter*. Ces deux adaptateurs de contexte cherchent l'ensemble des contexteurs décrivant les écrans (*DisplayContextor*) et respectivement l'ensemble des contexteurs décrivant les surfaces (*SurfacesContextor*). Alors que le *IamDisplayAdapter* cherche un unique *DisplayContextor* et cela seulement sur la machine sur laquelle il est exécuté (niveau de recherche micro), le *IamSurfaceAdapter* cherche tous les *SurfacesContextor* se trouvant sur le réseau local (niveau de recherche local).

2.5. LA DETECTION DU COUPLAGE

Dans I-AM, le couplage correspond à un lien entre deux surfaces. Pour être avertie des différents couplages, I-AM dispose d'un *IamLinksAdapter* qui lui donne la liste des liens entre les différentes surfaces.

Un lien de couplage correspond à un couple de quadruplets. Chacun des quadruplets contient les informations suivantes :

- L'identifiant de la surface mise en jeu.
- L'identifiant de la face de la surface. Cet identifiant peut prendre l'une des quatre valeurs suivantes : Nord, Sud, Est, Ouest.
- L'ordonnée du point de liaison dans le repère de la surface.
- L'abscisse du point de liaison dans le repère de la surface.

De manière à alimenter l'IamLinksAdapter de la plate-forme I-AM, nous avons réalisé deux contexteurs de couplage : le *SoftLinksContextor* et le *HardLinksContextor*. Ces deux contexteurs envoient à leurs clients (dans notre cas I-AM) le même type d'information. C'est-à-dire la liste des liens entre la surface où s'exécute le contexteur et les surfaces voisines.

Le SoftLink-Contextor

Le *SoftLinksContextor* est un contexteur élémentaire qui gère de manière logicielle la liste des liens entre les différentes surfaces. À l'origine, ce contexteur nous a permis de débuter les tests sur l'infrastructure I-AM sans attendre la fin du développement du capteur de proximité (celui-ci avait pris un peu de retard).

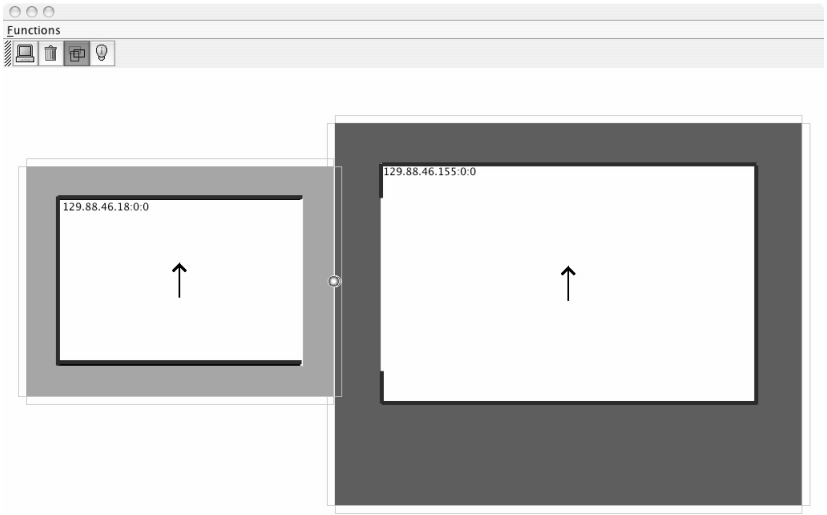


Figure 12
Interface graphique du simulateur de couplage.

Pour alimenter le *SoftLinksContextor*, nous avons réalisé un simulateur de couplage (représenté par la figure 12). Celui-ci utilise les informations publiées par le *SurfacesContextor* pour détecter les différentes surfaces présentes. Chaque surface détectée est représentée par un rectangle qui peut être déplacé dans l'IHM du simulateur. Lorsque le simulateur de couplage détecte la proximité de deux rectangles, il génère les informations de liaison et les transmet au *SoftLinksContextor*. Ce dernier n'a plus qu'à publier ces informations à ses clients.

Récemment, Barralon a utilisé ce même *SoftLinksContextor* pour diffuser des informations de couplage générées par la technique d'interaction Click-and-Couple [Barralon 05].

Le HardLinks-Contextor

Le *HardLinksContextor* est un contexteur non élémentaire. Son rôle est double. Il doit :

- agréger les informations de couplage provenant de différents contexteurs comme par exemple des *ProximityContextor* et
- élever au bon niveau d'abstraction les informations qu'il reçoit. Par exemple, pour les *ProximityContextor*, il transforme l'identifiant utilisé dans le *ProximityContextor* par l'identifiant de surface correspondant.

Ensuite, il peut transmettre ces informations à ses différents clients. Dans le cas d'I-AM, c'est le *IamLinksAdapter* qui prend en charge les données de ce contexteur. Nous rappelons que les données du *HardLinksContextor* et du *SoftLinksContextor* ont le même format.

Contrairement au système PointRight [Johanson 02] où les ressources sont décrites statiquement, I-AM dispose, grâce à l'infrastructure de

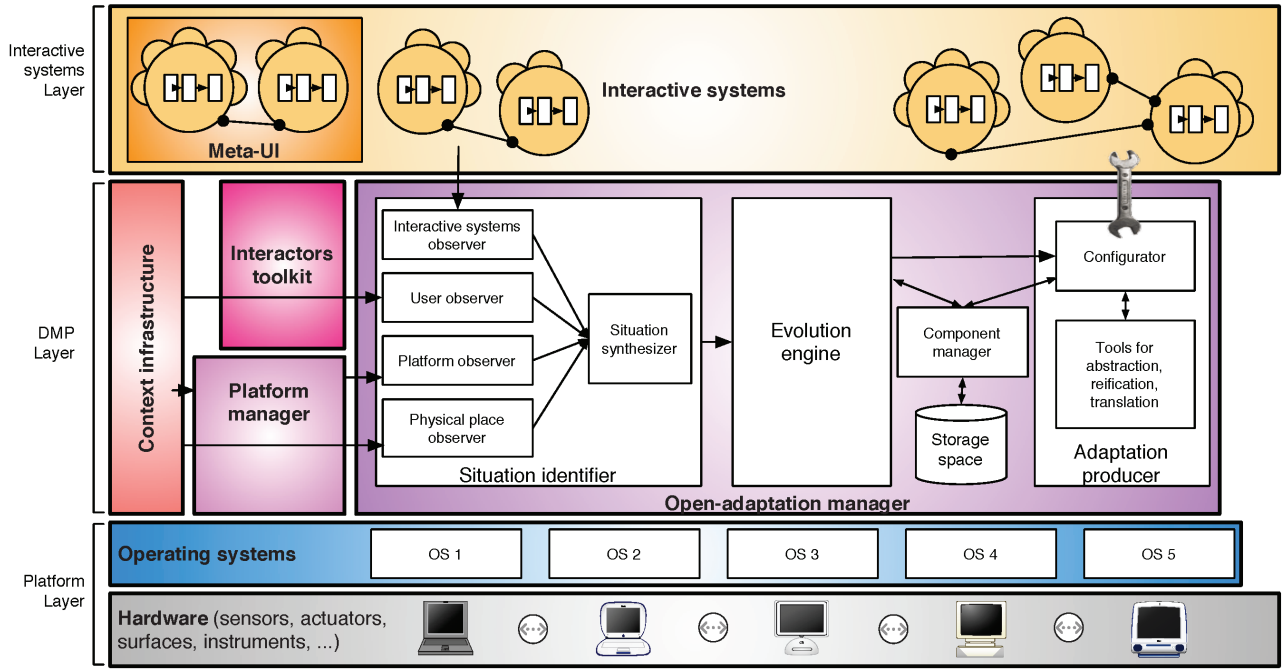


Figure 13
Architecture du projet Cameleon-RT.

contexteurs, d'un mécanisme performant pour découvrir dynamiquement les surfaces, leurs caractéristiques et leur couplage.

3. *Ethylene*

Ethylene s'inscrit dans le problème général des IHM plastiques en informatique ambiante.Il s'agit d'un intergiciel permettant la migration, la distribution et le remodelage d'IHM [Balme 04].

3.1. DESCRIPTION

La figure 13 représente la décomposition fonctionnelle d'Ethylene en trois couches:

- Au plus bas niveau d'abstraction, on retrouve la plate-forme composée de la partie matérielle (ordinateurs, capteurs, ...) ainsi que les systèmes d'exploitation fonctionnant sur ces différentes machines.
- Au plus haut, les systèmes interactifs : les applications classiques mais aussi des applications plastiques [Thevenin 99] composées éventuellement de comètes. Les comètes sont des composants auto-descriptifs dont les metadescriptions permettent l'adaptation dynamique au contexte d'interaction [Calvary 04b]. Un *Meta-UI*, dont un exemplaire préliminaire est présenté dans [Barralon 05], est aux

espaces interactifs ce qu'est le desktop aujourd'hui : il permet à l'utilisateur de contrôler, configurer, comprendre l'état de son espace interactif, notamment pendant les processus de migration, distribution et remodelage.
- Le centre, (Distribution-Migration-Plasticité) désigne Ethylene proprement dit.

La couche DMP comprend :
- Le *plate-forme manager* et *l'Interactors Toolkit* correspondant à la couverture fonctionnelle d'I-AM décrit ci-dessus.
- *L'Open-adaptation manager* est chargé du processus d'adaptation lorsque les composants des systèmes interactifs (les comètes) ne sont pas capables de s'adapter à la nouvelle situation. La première partie, la reconnaissance, est assurée par le *Situation Identifier*. La deuxième partie, le calcul de la réaction appropriée est réalisé par *l'Evolution engine* alors que les autres composants (*Storage Space*, *Component Manager* et *Adaptation producer*) servent à la mise en œuvre de la réaction précédemment calculée.
- Enfin, la *context Infrastructure* prend en charge la capture et la transformation des observables du contexte.

3.2. LE RÔLE DES CONTEXTEURS

Dans Ethylene les contexteurs trouvent leur place dans le *Context Infrastructure*. Ils y assurent la capture des observables du contexte via les contexteurs élémentaires et la transformation de ces données au bon niveau d'abstraction à l'aide des contexteurs non élémentaires. On notera que les capteurs utilisés par les contexteurs ne font pas partie de la couche DMP mais de la couche Plate-forme.

En plus de leurs présences au sein de la *Context Infrastructure*, les contexteurs, sous la forme des adaptateurs de contextes, se retrouvent dans les différents *observers* qui composent le *Situation Identifier*.

Enfin, il est important de noter que le *Situation Identifier* est une première implémentation de la couche identification présentée dans le chapitre III section 4.1 (“La couche Identification”).

3.3. PERSPECTIVES

La réalisation de cette infrastructure va permettre de :
- Poursuivre l'évaluation du modèle des contexteurs. Sont-ils adaptés aux services qu'ils sont censés rendre ?
- Evaluer la facilité d'intégration des contexteurs au sein d'applications tierces par des développeurs autres que les auteurs du modèle des contexteurs
- Valider le modèle conceptuel concernant la couche identification via le développement du Situation Identifier.

- Réaliser un grand nombre de nouveaux contexteurs et de là, enrichir le site[1] du projet contexteur de nouvelles instances directement utilisables.

1. Le site est actuellement en cours d'élaboration et devrait être prochainement accessible via l'adresse http://iihm.imag.fr/rey/Contexteur/index.html

Conclusion

I propose the idea of Spiral Contexts, a dynamic approach that consists of starting out with a small, restricted context, and gradually expanding it (in time and space), interleaved with user interaction.

Henry Lieberman, Les "Spirales de Contextes" dans l'Interaction Homme-Machine, IHM 2002, Poitier, France.

En conclusion, nous rappelons les éléments clefs de la contribution qui conduisent à leur tour à de nouvelles pistes de recherche.

1. Synthèse de la contribution

Alors que les concepts d'ordinateur évanescent 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 présentent des avantages pour les activités humaines, cette étude a contribué à clarifier la situation en trois points complémentaires : définition, modèles et mise en œuvre :

1.1. DÉFINITION DE LA NOTION DE CONTEXTE.

Un contexte n'existe pas en tant que tel mais se définit pour l'activité d'utilisateurs. Cette activité a lieu dans un réseau de contextes défini par un ensemble de rôles, relations et entités pertinents pour l'activité observée. Chacun des contextes du réseau est composé d'un ensemble de situations définies en fonction des associations rôles / entités et des associations relations / entités.

Cette définition du contexte vaut :

- selon plusieurs points de vue qui appellent la distinction entre le contexte brut, le contexte système, le contexte utilisateur et le contexte net (intersection des contextes système et utilisateur) ;
- et selon les deux étapes essentielles du processus de développement d'un système (l'étape de conception/analyse et l'étape d'exécution), qui amènent à distinguer les contextes système captable et capté, les contextes utilisateur utilisable et utilisé et les contextes net exploitable et exploité.

1.2. MODÉLE ARCHITECTURE CONCEPTUELLE

Dans le chapitre II, nous positionnons les architectures de capture du contexte comme des intergiciels dont nous détaillons les différentes couches et services dans la Pyramide du contexte. Du plus bas niveau d'abstraction au plus haut nous trouvons les couches suivantes :

- La couche de capture qui mesure les observables du contexte.
- La couche transformation qui fusion les observables capturé et les élève au bon niveau d'abstraction.
- La couche identification qui reconnait le contexte et la situation courant dans un réseau de contextes.

- La couche adaptation qui fait le lien entre les applications et l'infrastructure du contexte.

En plus de cela, nous proposons un ensemble de critères permettant l'évaluation des différentes infrastructures de capture du contexte.

1.3. LE CONTEXTEUR ET LES RÉPÉTEURS

Nous proposons un modèle conceptuel visant à la mise en œuvre de systèmes sensibles au contexte. Un contexteur est une abstraction logicielle qui fournit la valeur d'un observable relevant du contexte système capté. Inspiré des travaux de Salber, le contexteur présente comme originalité la notion de méta-données qui sont étroitement liées aux données d'entrée et de sortie. Comme chez Salber, le contexteur distingue les données de contrôle des données proprement dites. Les principaux avantages du modèle des contexteurs sont :

- leur modèle égal à égal,
- 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 plus de cela, l'infrastructure des contexteurs est soutenue par un réseau de répéteurs qui ont pour objectif le routage à grande échelle des messages de recherche. Cette deuxième architecture vient renforcer celle des contexteurs au niveau de son principal point faible : l'inondation des réseaux lors des phases de recherches.

Ces deux infrastructures ont donné lieu à une mise en œuvre en Java avec des communications en XML de manière à garantir une bonne portabilité des composants réalisés.

Enfin, nous avons validé le modèle conceptuel par la mise en œuvre de contexteurs dans deux exemples d'applications sensibles au contexte et un projet en cours de réalisation;

- l'observatoire d'activité,
- l'infrastructure I-AM,
- le projet Ethylene.

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. Elle appelle de nombreuses questions.

2. *Perspectives*

Ce travail est imparfait. Comme énoncé lors de l'introduction, cette étude se borne à offrir un espace de conception et un support à la mise en œuvre de systèmes interactifs en informatique ambiante. Bien qu'importants, les points suivants n'ont été pas traités dans le temps imparti à la thèse : l'éthique, l'évaluation des effets de l'adaptation et les erreurs d'évaluation de la situation par le système.

En plus des trois points énoncés lors de l'introduction, nous constatons que le domaine de l'Intelligence Artificielle a été laissé de coté durant cette étude et qu'il est important de finaliser certains des éléments de cette étude telle que la couche identification de la Pyramide du contexte.

L'ensemble de ces points nous permet de définir des perspectives à cette étude. Celles-ci peuvent s'organiser le long de la chaîne de conception logicielle en partant de l'étude des concepts puis par celle du modèle logiciel enfin de terminer par la réalisation de prototypes.

2.1. AU NIVEAU DES CONCEPTS

Il convient de :

- Comparer l'approche adoptée avec les modèles fondés sur les agents cognitifs de manière à enrichir la grille d'évaluation présentée au chapitre II. Cela permettra d'améliorer l'étude des systèmes existants et de mieux comprendre leurs qualités et leurs défauts de manière à concevoir un système plus adapté aux besoins des applications.
- Spécifier un langage permettant la description des contextes au niveau de la couche identification. Ce langage devra permettre d'exprimer les besoins des applications (via la couche d'adaptation de la Pyramide du contexte) de manière à ce qu'un réseau de contextes et de situations puisse être calculé au niveau de la couche identification.
- Etudier l'impact de la capture d'information sur la vie privée. Les résultats de cette analyse devraient permettre de définir les requis sur les infrastructures de capture du contexte du point de vue de la sécurité des informations collectés, du respect de la vie privé des utilisateurs et de la confiance de ces derniers dans les systèmes interactifs sensibles au contexte.

2.2. AU NIVEAU DE LA CONCEPTION LOGICIELLE

Il faut :

- Etudier les outils de spécification et alimenter notamment le processus de développement envisagé pour la plasticité des IHM. Cette étude se poursuit dans le cadre du projet Cameleon via l'architecture Cameleon-RT.
- Etudier et spécifier des protocoles de communication et d'interaction permettant de garantir une certaine vie privée des utilisateurs. Ces

protocoles devront permettre de mettre en œuvre les concepts identifiés précédemment.

2.3. AU NIVEAU DU "RUN TIME"

Il convient de :

- Evaluer, de manière plus complète, les contexteurs (consommation de ressources CPU, robustesse aux connexions / déconnexions ...) ainsi que l'infrastructure des répéteurs. Ces évaluations permettront de valider l'infrastructure des contexteurs, de mieux en connaitre les points faibles et d'orienter les développements futurs. Ce point est actuellement en cours de réalisation.
- Fusionner l'infrastructure des répéteurs avec les contexteurs et évaluer cette nouvelle infrastructure unifiée. Depuis peu de temps, le domaine du P2P prend un essor considérable et voit l'apparition de nouveaux protocoles tel que les protocoles de découverte de type Kademlia, zéro-conf et UPnP. Ces nouveautés permettent envisager la fusion de répéteurs dans l'infrastructure des contexteurs. Cette fusion devrait améliorer l'indépendance des contexteurs vis-à-vis d'infrastructures tierces et faciliter l'administration de l'ensemble (une seule infrastructure au lieu de deux).
- Proposer une réalisation de la couche identification. De manière à concrétiser et à évaluer les concepts présents dans la couche identification, il est important d'en faire une réalisation. Partie prenante de l'architecture Caméléon-RT du projet européen Caméléon, une première mise en œuvre de la couche identification devrait voir le jour prochainement.
- Fournir des outils d'administration des contexteurs. Bien que les contexteurs une fois configurés soient autonomes, il serait intéressant de bénéficier d'outils d'administrations permettant la surveillance des contexteurs présents sur le réseau. De tels outils pourraient permettre une gestion les contexteurs de manière plus générale et par conséquent de prévoir et se prémunir contre d'éventuelles pannes.

2.4. A LONG TERME

Nous prévoyons l'intégration transparente et systématique des contexteurs dans les environnements informatiques (mobiles, diffus, ambiants ou non). Pour cela, il est nécessaire d'intégrer les résultats de cette thèse aux niveaux des systèmes d'exploitation et des infrastructures (sous forme de services ?). En particulier, nous envisageons deux axes de recherche :

- Améliorer les capacités des contexteurs en enrichissant leurs possibilités de traitement et d'adaptation. Cette amélioration pourra tirer partie des travaux réalisés 1) dans le domaine de l'Intelligence Artificielle pour ce qui est de l'exploitation des données et les méta-données et 2) dans le domaine Système - Réseau avec le concept de code mobile et les avancées réalisées sur les réseaux ad-hoc et de type égal à égal (P2P).

- Optimiser le coté autonome et dynamique des contexteurs, tant au niveau de leur déploiement, de leur découvertes par les systèmes interactifs et de leurs compositions.

Pour atteindre cet objectif à long terme, il est nécessaire de travailler sur l'ensemble des étapes du cycle de vie du développement logiciel (étude des concepts, conception logicielle, réalisation et tests). Chacune des perspectives énoncées précédemment soutient une étape du cycle de vie de manière à atteindre cet (ultime ?) objectif.

Bibliographie

[Abowd 98] Gregory D. Abowd, Anind K. Dey, Robert J. Orr, and Jason Brotherton. Context-awareness in Wearable and Ubiquitous Computing Virtual Reality, Vol. 3 (1998), pp. 200-211.

[Anderson 92] Anderson, J.R. (1992). Automaticity and the ACT* theory. American Journal of Psychology, 105, 2, 165-180.

[Arch 92] UIMS Tool Devlopers' Workshop. A metamodel for runtime architecture of an interactive system. SIGCHI Bulletin, pp32 à 37, 1992.

[Balme 04] Balme, L. Coutaz, J., Demeure, A., Calvary, G. CAMELEON-RT: a Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. Second European Symposium on Ambient Intelligence, EUSAI 04, LNCS 3295, Markopoulos et al., 2004, pp. 291-302.

[Bardram 97] J. E. Bardram Plans as Situated Action: An Activity Theory Approach to Workflow Systems Proceedings of ECSCW'97 Conference, Lancaster UK, September 1997.

[Barralon 04] Barralon N., Lachenal C., Coutaz J. Couplage de ressources d'interaction. IHM'04 ACM Publication pp.13-20.

[Barralon 05] N. Barralon, V. Nguyen, G. Rey, Techniques pour le couplage de bureaux: illustration avec Ambient-Desktop. Mobilité et Ubiquité (Ubimob05), ACM Publication, 2005, à paraître.

[Base64] RFC 3548 - The Base16, Base32, and Base64 Data Encodings. http://www.faqs.org/rfcs/rfc3548.html

[Baümer 97] Bäumer, D. Riehle, D. Siberski, W et Wulf, M. The Role Object Pattern. PLoP 1997. Septembre 1997. Allerton Park Monticello, Illinois, USA.

[Bérard 99] Vision par Ordinateur pour l'Interaction Homme-Machine Fortement Couplée. Thèse de doctorat Informatique préparée au Laboratoire de Communication Langagière et Interaction Personne-Système (IMAG), Université Joseph Fourier. November 1999. 200 pages.

[Beutel 03] J. Beutel, O. Kasten and M. Ringwald: BTnodes - A Distributed Platform for Sensor Nodes. ACM SenSys 2003.

[Beyer 98] H. Beyer, K. Holtzblatt. Contextual Design. Morgan Kaufman Publ. San Francisco, 1998.

[Brézillon 02] Brézillon P. (2002) Expliciter le contexte dans les objets communicants. In: C. Kintzig, G. Poulain, G. Privat, P.-N. Favennec (Eds.): Les Objets Communicants. Hermes Science Editions, Lavoisier, Chapter 21, pp. 295-303.

[Brown 96] P.J. Brown, The Stick-e Document: a framework for creating Context-aware applications. Electronic Publishing '96 (1996) 259-272

[Brown 97] P.J. Brown, J.D. Bovey, X. Chen. Context-Aware applications : From the Laboratory to the Marketplace. IEEE Personal Communications, 4(5) (1997) 58-64

[Calvary 01a] G. Calvary, J. Coutaz, D. Thevenin. A Unifying Reference Framework for the Development of Plastic User Interfaces, EHCI'2001.

[Calvary 01b] G. Calvary, J. Coutaz, D. Thevenin. Supporting Context Changes for Plastic User Interfaces. in Proc. HCI-IHM 2001, A. Blandford, J. Vanderdonckt, P. Gray Eds.,BCS conference series, Springer Publ.pp. 349-363 .

[Calvary 04a] Calvary, G., Coutaz, J., Dâassi, O., Balme, L., Demeure, A. (2004) Towards a new generation of widgets for supporting software plasticity: the « comet », in procs of EHCI-DSVIS'2004, The 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with the 11th International Workshop on Design, Specification and Verification of Interactive Systems, July 11-13,2004.

[Calvary 04b] G. Calvary, A. Demeure, J. Coutaz, O. Daassi. Adaptation des interfaces homme-machine à leur contexte d'usage Plasticité des IHM. La présentation d'information sur mesure, Numero Special de RIA; Paris, C. et Colineau, N. (editeurs invites). Vol 18 (4) 2004. Date de parution: Septembre 2004, pp.577-606

[Card 83] Card, S.Moran, T.Newell,A. The Psychology of Human-Computer Interaction, Lawrence Erlbaum Associates,1983.

[Chaari 05] Chaari T., Laforest F., Celentano Service-Oriented Context-Aware Application Design. First International Workshop on Managing Context Information in Mobile and Pervasive Environments (MCMP), May 9 2005, Ayia Napa, CYPRUS

[Cheverst 01] K. Cheverst, N. Davies, K. Mitchell, C. Efstratiou. Using Context as a Crystal Ball : Rewards and Pitfalls. Personal and Ubiquitous Computing, 5(1), Springer Verlag Publ., 2001, pp.8-11.

[Coutaz 98] J. Coutaz, F. Bérard, E. Carraux, J. Crowley. Early Experience with the Mediaspace CoMedi. Proceedings of Engineering Human Computer Interaction, EHCI'98, Héraklion, Kluver Academics Publ. Sept. 1998. pp. 57-72.

[Coutaz 02a] J. Coutaz, G.Rey. Foundation for a Theory of Contextors. In Proc. CADUI02, ACM Publication, 2002, pp. 283-302.

[Coutaz 02b] J. Coutaz, A. Dearle, S. Dupuy-Chessa, G. Kirby, C. Lachenal, R. Morrison, G. Rey, E. Zirintsis. Gloss ontology. 2002 (GLOSS - D9.2)

[Coutaz 03] J.Coutaz, C.Lachenal, N.Barralon, G.Rey. Final reference framework for interaction surfaces. 2003 (GLOSS - D19).

[Coutaz 05] Joëlle Coutaz, James L. Crowley, Simon Dobson, David Garlan. The disappearing computer: Context is key. March 2005 Communications of the ACM, Volume 48 Issue 3, Pages: 49 - 53.

[Crowley 00] J. Crowley, J. Coutaz, F. Bérard. Things that See. Communication of the ACM, Vol 43 (3). March 2000. pp. 54-64.

[Crowley 02] J. L. Crowley, J. Coutaz, G. Rey, P. Reignier. Perceptual Components for Context Aware Computing. In Proc. Ubicomp (Ubiquitous Computing) 2002, pp 117-134.

[Cox 99] Ingemar J. Cox, Matthew L. Miller, and Andrew L. McKellips. Watermarking as communications with side information. Proceedings of the IEEE, July 1999. Special Issue,"Identification and protection of multimedia information".

[Dey 99] An Architecture To Support Context-Aware Applications. Anind K. Dey, Daniel Salber, Masayasu Futakawa and Gregory D. Abowd. GVU Technical Report GIT-GVU-99-23. June 1999.

[Dey 01] Dey A.K., Salber D., Abowd G.D., A Conceptual Frame-work 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.

[Dourish 01] P. Dourish. What we talk about when we talk about context. Submitted to HCI Journal for publication in special issue on context-aware computing, 2001.18p

[DTD] http://fr.wikipedia.org/wiki/DTD

[Dürr 03] Dürr F. et al.: On a Location Model for Fine-Grained Geocast. Ubicomp pp 18-35 (2003).

[eDonkey] http://www.edonkey2000.com/

[Encyclopédie Larousse] http://www.encyclopedie-larousse.fr/

[Espinoza 01] Fredrik Espinoza, Per Persson, Anna Sandin, Hanna Nyström, Elenor Cacciatore, Markus Bylund. GeoNotes: Social and Navigational Aspects of Location-Based Information Systems. Ubicomp 2001: Ubiquitous Computing: Third International Conference Atlanta, Georgia, USA, September 30 - October 2, 2001, Proceedings Editors: G.D. Abowd, B. Brumitt, S. Shafer (Eds.): Chapter: p. 2.

[Etzioni 99] Etzioni, A., The Limits of Privacy. New York: Basic Books, 1999.

[Fisher 01] Gerhard Fischer, « Articulating the Task at Hand Making Information Relevant to it » in the Human-Computer Interaction (HCI) Journal, Vol. 16, 2001.

[Florin 97] G Florin, S Natkin, La sécurité. Cours du CNAM, laboratoir CEDRIC.

[Flury 03] Flury, T. Privat, G. An infrastructure template for scalable locationbased services. France Télécom R&D, DIH/OCF. SOC Grenoble, France. 2003.

[Gandon 04] Fabien L. Gandon, Norman M. Sadeh - Premières Journées Francophones: Mobilité et Ubiquité, Mardi 1 juin - Jeudi 3 juin 2004, Nice, Sophia-Antipolis, Essi (Ecole Supérieure en Sciences Informatiques) p123-130.

[Garfinkel 01] Garfinkel, S., Database Nation: The Death of Privacy in the 21st Century: O'Reilly & Associates, 2001.

[Garlan 93] D. Garlan et Shaw. An introduction to software architecture, in Advances in Software Engineering and Knowledge Engigneering. World Scientific Publishing Company, 1993.

[Gibbons 03] Gibbons P.B., Karp B., Ke Y., Nath S., Seshan S.. IrisNet: An Architecture for a Worldwide Sensor Web. IEEE Pervasive Computing Mobile and Ubiquitous Computing, Special Issue Sensor & Actuatirs Networks, Vol 2(4), pp.22-33, (Oct-Dec. 2003).

[Glassey 03] Glassey R., Stevenson G., Richmond M., Wang F., Nixon P., Terzis S., and Ferguson I. Towards a middleware for generalised context management. in 1st MPAC03, (June 2003).

[Gram 96] C. Gram, G. Cockton, Design Principles for Interactive Software Chapiter 3 : internal Properties: The Software developer's Perspective. pp .53-89

[GrandDictionnaire] Office québécois de la langue française. http://www.granddictionnaire.com/

[Hachette Multimédia] http://www.encyclopedie-hachette.com/W3E/

[Halverson 94] 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).

[Heer 03] J Heer, A Newberger, C Beckmann, JI Hong, Liquid: Context-Aware Distributed Queries. In Proceedings Ubicomp 2003, pp140 - 148

[Hess 03] Christopher K. Hess and Roy H. Campbell, A Context-Aware Data Management System for Ubiquitous Computing Applications in International Conference of Distributed Computing Systems (ICDCS 2003), Providence, Rhode Island, (May 19-22, 2003).

[HID] HID: Human Interface Device. http://www.usb.org/developers/hidpage/

[Hinckley 00] K Hinckley, J Pierce, M Sinclair, E Horvitz. Sensing Techniques for Mobile Interaction Ken Hinckley, Jeff Pierce, Mike Sinclair, Eric Horvitz Symposium on User Interface Software and Technology, CHI Letters 2 (2) , ACM UIST 2000 pp. 91 - 100.

[Hinckley 03] Hinckley K.: Synchronous gestures for multiple persons and computers. ACM UIST 2003: pp 149-158.

[Hong 01] Jason I. Hong and James A. Landay, An Infrastructure Approach to Context-Aware Computing. In Human-Computer Interaction ,Vol. 16, (2001).

[IrDA] IrDA: InfraRed Device Association. http://www.irda.org/

[Jiang 02] Changhao Jiang, Peter Steenkiste. A Hybrid Location Model with Computable Location Identifier for Ubiquitous Computing. Ubicomp 2002.

[Johanson 02] B. Johanson, G. Hutchins, T. Winograd, et M. Stone. PointRight: experience with flexible input redirection in interactive workspaces. Proceedings of the 15th annual ACM symposium on User interface software and technology UIST 2002. Paris, France. Pages 227-234.

[Julien 04] Julien, Christine, Supporting Context-Aware Application Development in Ad Hoc Mobile Networks.August, 2004, Saint Louis, Missouri.

[Kiczales 92] Kiczales G., Towards a New Model of Abstraction in Software Engineering, Proc. IMSA'92. Worshop on Reflection and Metalevel Architectures, Tokyo, (1992).

[Kirby 02] 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.

[Kolski 97] Kolski Interfaces Homme-Machine : applications aux systèmes industriels complexes par Christophe Kolski. Édition Hermes. 1997

[Kurmas 00] Zachary Kurmas and Ann L. Chervenak, “Evaluating Backup Algorithms,” IEEE Symposium on Mass Storage Systems, 2000.

[Lachenal 04] C. Lachenal, Modèle et infrastructure logicielle pour l’interaction multi-instrument multisurface. Thèse pour le titre de docteur de l’université Joseph Fourier (Grenoble I). 17 décembre 2004. 200 pages.

[Lahlou 03] Lahlou, S. Jegou, F. European Disappearing Computer Privay Design Guidelines V1.0. Ambient Agoras Report D15.4, Disappearing Computer Initiative, oct., 2003

[Lahlou 05] Saadi Lahlou, Marc Langheinrich, Carsten Roecker. Privacy and Trust Issues with Invisible Computers. Appeared in: Communications of the ACM, Volume 48, number 3 (March 2005) pp. 59–60

[Lyytinen 02] Lyytinen, K. and Yoo, Y. Issues and Challenges in Ubiquitous. Computing. Communications of the ACM, 45(12), Pages 62-65.

[Mackay 96] Mackay, W. (March 1996). Réalité Augmentée : le meilleur des deux mondes. La Recherche, Special issue on L'ordinateur au doigt et à l'Ïil. Vol. 284.

[Marmasse 04] Natalia Marmasse, Chris Schmandt and David Spectre. WatchMe: Communication and Awareness Between Members of a Closely-Knit Group. Ubicomp04. p 214-231.

[Maymounkov 02] P. Maymounkov and D. Mazieres. Kademlia: A peer to peer information system based on the xor metric. In Proceedings of IPTPS02, Cambridge, USA, March 2002.

[McCall 77] McCall, J.; Richards, P.; Walters, G., "Factors in Software Quality," three volumes, NTIS AD-A049-014, AD-A049-015, AD-A049-055, November 1977.

[Munro 01] A. Munro, P.Welen A Wilson. Interaction Archetypes. Delivrable D4, Global Smart Space Project 2001 IST Basic Research Project, FET Disappearing Computer.

[Nabaztag] Nabaztag, le lapin interactif. Violet : The smart object compagny. http://www.violet.net/index.jsp.

[Nigay 94] L. Nigay. Conception et modélisation logicielles des systèmes interactifs : Application aux interfaces multimodales. Thèse de l'Université Joseph Fourier, janvier 1994, Grenoble, 315 pages.

[Pascoe 98] J. Pascoe, Adding Generic Contextual Capabilities to Wearable Computers. 2nd International Symposium on Wearable Computers (1998) pages 92-99.

[Reignier 05] Patrick Reignier, James L. Crowley, Alban Caparossi, Dominique Vaufreydaz, Oliver Brdiczka, Nicolas Gourier, Sebastien Pesnel. Context aware video acquisition. D 4.3 Projet FAME (IST-2000-28323) 5 mai 2005.

[RESIDE-HIS] Projet IMAG RESIDE-HIS (2000-2002). http://www-clips.imag.fr/geod/projets/HIS/ .

[Rey 01] G. Rey. 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, June, 2001, 84 pages.

[Rey 02] G. Rey, J. Coutaz. Le Contexteur : une Abstraction Logicielle pour la Réalisation de Systèmes Interactifs Sensibles au Contexte. In Proc. IHM2002,pp 105-112.

[Rey 04] G.Rey, J. Coutaz. Le Contexteur : Capture et distribution dynamique d'information contextuelle. Mobilité et Ubiquité (Ubimob04), ACM Publication, 2004, pp 131 à 138.

[Ryan 97] N. Ryan, J. Pascoe, D. Morse, Enhanced Reality Fieldwork : the Context-Aware Archeological Assistant. V. Gaffney, M. van Leusen, S. Exxon Computer Applications in Archeology (1997).

[Salber 95] D. Salber. De l'interaction individuelle aux systèmes multi-utilisateurs. L'exemple de la Communication Homme-Homme-Médiatisée. Thèse de doctorat Informatique préparée au Laboratoire de Génie Informatique (IMAG), Université Joseph Fourier. September, 8th 1995. 303 pages.

[Salber 99] D. Salber, A.K. Dey, G. D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. Proceedings CHI'99.

[Salber 01] P. Gray, D. Salber, Modelling and Usins Sensed Context Information in the design of Interactive applications EHCI 2001 (May 2001) 92-111

[Scénario Gloss] http://iihm.imag.fr/rey/GLOSS/

[Schilit 94a] B.N. Schilit, M. Theimer, Disseminating Active Map Information to Mobile Hosts, IEEE Network, (1994) 22 – 32

[Schilit 94b] B.N. Schilit, N.I. Adams and R. Want. Context-Aware Computing Applications. Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'94), IEEE Press, pp 85-90

[Schillit 95] Schilit B.: A Context-Aware System Architecture for Mobile Distributed Computing. PhD Thesis, Columbia University, (May 1995).

[Schmidt 99] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, Walter Van de Velde. Advanced Interaction in Context. 1rst International Symposium on Handheld and Ubiquitous Computing (HUC99), Karlsruhe, Germany, 1999 & Lecture notes in computer science; Vol 1707, ISBN 3-540-66550-1; Springer, 1999, pp 89-101.

[Schmidt 00] Albrecht Schmidt. Implicit Human Computer Interaction Through Context. Personal Technologies Volume 4(2&3), June 2000. pp191-199.

[Schmidt 02] Albrecht Schmidt, Ubiquitous Computing - Computing in Context. A thesis submitted to Lancaster University for the degree of Ph.D. in Computer Science, November, 2002.

[Schuman 87] L. A. Schuman, Plans and situated actions. The problem of human-machine communication. Cambridge: Cambridge University Press. (1987).

[Seeheim 85] PFAFF, G. E., Ed. User Interface Management Systems. Springer-Verlag, Berlin, 1985.

[Shaw 96] Mary Shaw and David Garlan Software architecture book, perspectives on an emerging discipline. Prentice Hall, 1996, ISBN 0-13-182957-2.

[SOAP] http://www.w3.org/TR/soap/

[Steenkiste 03] Judd, G, Steenkiste: Providing Contextual Information to Pervasive Computing Applications P.IEEE International Conference on Pervasive Computing, March 23-25, (2003).

[Stoica 01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of the ACM SIGCOMM ’01 Conference, San Diego, California, August 2001.

[Suman 03] Suman Nath, Amol Deshpande, Phillip B. Gibbons, Srinivasan Seshan, Mining a World of Smart Sensors. Student poster at Mobicom 2002, Extended abstract appears in Mobile Computing and Communications Review (MC2R,) Volume 7, Issue 1 (January 2003).

[Tandler 01] Tandler, P., Prante, T., Müller-Tomfelde, C., Streitz, N., Steinmetz, R. ConnecTables: Dynamic Coupling of Displays for the Flexible Creation of Shared Workspaces. In Pro-ceedings of the ACM UIST'01 Symposium on User Interface Software and Technology, Orlando, Florida, 2001, pp.11-20.

[Thevenin 99] D. Thevenin, J. Coutaz. Plasticity of User Interfaces: Framework and Research Agenda. In Proceedings of INTERACT'99, 1999, pp. 110-117.

[UDP Multicast] RFC 1112 - Host extensions for IP multicasting. http://www.faqs.org/rfcs/rfc1112.html

[Wang 04] Wenjie Wang, Cheng Jin, Sugih Jamin. Network Overlay Construction under Limited End-to-End Addressability. EECS, Computer Science Technical Report 2004.

[Ward 97] Ward, A. Jones, A. Hopper, A New Location Technique for the Active Office. IEEE personnal Communications (1997) 42-47

[Webencyclo] http://www.webencyclo.com

[Weisser 91] Weisser, M,. "The computer for the twenty-first century. Sci. Am, Sept. 1991), 94-104.

[Weisser 93] M. Weiser, Some Computer Science Problems in Ubiquitous Computing, Communications of the ACM, July 1993. (reprinted as "Ubiquitous Computing". Nikkei Electronics; December 6, 1993; pp. 137-143.)

[Winograd 01] Terry winograd, Architectures for context. In Human-Computer Interaction ,Vol. 16, (2001). pp 402 - 419.

[XML] http://www.w3c.org/XML/

[XML Schema] http://www.w3.org/TR/xmlschema-0/

[Yan 00] H.Yan, T. Selker. Context – aware office Assistant. Proceedings of the 2000 International Conference on Intelligent User Interfaces (IUI'2000) ,H Lieberman (eds), ACM Press

[ZéroCong] http://www.zeroconf.org/

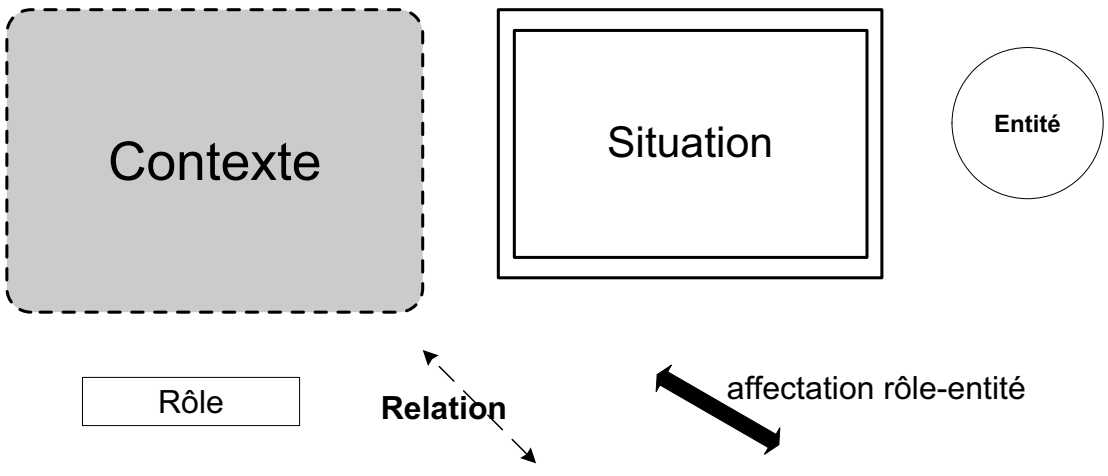
Annexe A *Notation graphique du contexte*

De manière à simplifier la modélisation des réseaux de contextes, une notation graphique a été introduite. Reprenons en détail chacun des éléments de cette notation. La figure 1 illustre leur rendu visuel.

1. Un contexte

Un contexte est représenté par un rectangle aux bords arrondis (en gris sur la figure 1). Les contextes sont tous étiquetés par un label permettant de

Figure 1
Ensemble des éléments composant la notation graphique des réseaux de contextes.



les identifier. Ce label peut se situer soit dans le rectangle lui même, soit à l'extérieur du rectangle. Par commodité une notation de type C_x (avec x un numéro) sera utilisée pour nommer (de manière unique) les contextes d'un même réseau de contextes. Cependant il est tout à fait possible de donner des noms explicites aux différents contextes pour faciliter la compréhension des schémas.

2. *Une situation*

Une situation est représentée par un double rectangle auquel est accolé un label représentant le nom de la situation. Comme pour les contextes, le label peut se situer soit à l'intérieur, soit à l'extérieur du double rectangle. Par commodité une notation de type S_x (avec x un numéro) sera utilisée pour nommer (de manière unique) les situations d'un même contexte. Cependant il est tout à fait possible de donner des noms explicites aux différentes situations pour faciliter la compréhension des schémas.

3. *Une entité*

Une entité est représentée par un cercle (en blanc sur la figure 1) contenant un label. Le label du cercle correspond au nom usuel de l'entité. Par commodité une notation de type e_x (avec x un numéro) sera utilisée pour nommer (de manière unique) les entités d'un même contexte ou d'une même situation. Cependant il est tout à fait possible de donner des noms explicites aux différentes entités pour faciliter la compréhension des schémas.

4. *Un rôle et son affectation*

Un rôle est représenté par un rectangle simple contenant le nom du rôle. L'association d'un rôle avec une entité est représentée par une double flèche noire allant du cercle modélisant l'entité au rectangle représentant le rôle joué par cette entité. Une entité peut jouer plusieurs rôles et plusieurs entités peuvent jouer le même rôle. Il est possible de dédoubler

(ou non) les entités et/ou les rôles de manière à rendre le schéma de la modélisation plus lisible.

5. *Une relation*

Une relation est représentée par une flèche en pointillés à laquelle est ajouté un label décrivant la relation. Si la relation est symétrique, il est possible d'utiliser une double flèche en pointillés plutôt que deux flèches simples de manière à améliorer la lisibilité du schéma.

Annexe B *Définition du contexte*

Le texte ci-dessous reprend la définition du contexte présentée dans le chapitre I de ce document. Cette dernière a été épurée de tout exemple de manière à offrir une vision plus synthétique de la notion de contexte.

1. Réseau de contextes et contexte

Un réseau de contextes *Rc*, est défini sur trois ensembles (*Roles*, *Relations*, *Entites*) et un prédicat (*joueRole*) :

- *Roles* est l'ensemble des rôles considérés par le concepteur du système interactif. $Roles = \{r_1, r_2, ... , r_n\}$ où r_i est un rôle,
- *Relations* correspond à l'ensemble des relations entre les entités considérées par le concepteur du système interactif. $Relations = \{rel_1, rel_2, ... , rel_n\}$ où rel_i est une relation,
- *Entites* désigne l'ensemble des entités considérées par le concepteur du système interactif. $Entites = \{e_1, e_2, ... , e_n\}$ où e_i est une entité.
- *joueRole(e,r)* est un prédicat qui se vérifie si et seulement si l'entité *e* joue le rôle *r* avec $e \in Entites$ et $r \in Roles$.

Chaque noeud du réseau *Rc* correspond à un contexte C_i.

Un contexte C_i est défini par le couple (R_i, Rel_i) où :

- R_i est l'ensemble des rôles joués par les entités, avec $R_i \subseteq Roles$. De plus, quel que soit le $r \in R_i$, il existe une entité $e \in Entites$ tel que *e* joue le rôle *r* (propriété 1).

$$\forall (r \in R_i), \exists (e \in Entites) \ / \ joueRole\ (e,r) \qquad (1)$$

- Rel_i est l'ensemble des relations (entre entités), avec $Rel_i \subseteq Relations$. De plus, quel que soit la relation rel (d'arité j) $\in Rel_i$, il existe j entités e_1 à $e_j \in Entites$ tel que les entités e_1 à e_j entretiennent la relation rel (propriété 2).

$$\begin{aligned} &\forall((rel \in Rel_i) \wedge (arity(rel) = j)), \\ &\exists(e_1 \ldots e_j \in Entites) \,/\, rel(e_1 \ldots e_j) \end{aligned} \tag{2}$$

De ces définitions et propriétés, on déduit qu'il y a changement de contexte lorsque l'une des conditions suivantes devient vraie :

- l'ensemble R_i des rôles remplis change : apparition d'un nouveau rôle et/ou disparition d'un rôle,
- l'ensemble Rel_i des relations entretenues change : apparition d'une nouvelle relations et/ou disparition d'une relation.

Ce qui s'écrit formellement :

$$\begin{aligned} &\forall((C_1 = (R_1, Rel_1)) \in Rc), \forall((C_2 = (R_2, Rel_2)) \in Rc)/ \\ &(C_1 = C_2) \Leftrightarrow ((R_1 = R_2) \wedge (Rel_1 = Rel_2)) \end{aligned} \tag{3}$$

Le contexte C_i, nœud du réseau de contexte Rc, respecte les conditions suivantes :

$$\begin{aligned} &\forall(C_i \in Rc), \exists R_i, Rel_i/ \\ &(C = (R_i, Rel_i)) \Leftrightarrow (R_i \in \wp(Roles)) \wedge (Rel_i \in \wp(Relations)) \end{aligned} \tag{4}$$

où $\wp(Roles)$ et $\wp(Relations)$ sont, respectivement, l'ensemble des parties de $Roles$ et l'ensemble des parties de $Relations$.

La cardinalité de Rc est le produit des cardinalités de $\wp(Roles)$ et de $\wp(Relations)$. Soit n la cardinalité de $Roles$ et m la cardinalité de $Relations$, les formules suivantes détaillent le calcule de la cardinalité de Rc :

$$Card(\wp(Roles)) = \sum_{k=0}^{n} C_n^k = \sum_{k=0}^{n} \frac{n(n-1)\ldots(n-k+1)}{k!} = 2^n \tag{5}$$

$$Card(\wp(Relations)) = \sum_{k=0}^{m} C_m^k = 2^m \tag{6}$$

$$\begin{aligned} Card(Rc) &= Card(\wp(Roles)) \times Card(\wp(Relations)) \\ Card(Rc) &= 2^n \times 2^m = 2^{n+m} \end{aligned} \tag{7}$$

2. *Contexte et Situation*

Un contexte $C = (R,Rel)$ est un réseau de situations tel que toutes les situations de C partagent les mêmes ensembles de rôles R et de relations Rel.

Au sein d'un contexte $C = (R,Rel)$, une situation S est définie sur trois ensemble (Ent, $AssoRoEnt$, $AssoRelEnt$) et un prédicat $estPresent$:

- Ent est l'ensemble des entités présentes dans S, avec $Ent \subseteq Entites$,

$$\forall(e \in Ent) \,/\, ((e \in Entite) \wedge estPresent(e)) \qquad (8)$$

où le prédicat $estPresent(e)$ est vrai si et seulement si l'entité e est présente dans S.

- $AssoRoEnt$ est l'ensemble des associations entre les rôles de R et les entités de Ent. Rappelons que chaque rôle r_i est joué par au moins une entité et que chaque entité joue zéro, un ou plusieurs rôles,

$$\forall(a \in AssoRoEnt), \exists(e \in Ent \wedge r \in R) \,/\, a = (e,r) \wedge joueRôle\ (e,r) \qquad (9)$$

- $AssoRelEnt$ est l'ensemble des associations entre les relations de Rel et les entités de Ent.

$$\forall(b \in AssoRelEnt), \exists(e_1 \ldots e_j \in Ent \wedge rel \in Rel \wedge arity(rel) = j\ ) \,/\, b = (rel, e_1, \ldots, e_j) \wedge rel(e_1, \ldots, e_j) \qquad (10)$$

Au sein d'un contexte $C = (R,Re)$, il y a changement de situation si l'une au moins des conditions suivantes est remplie :

- l'ensemble Ent des entités change : apparition et/ou disparition d'une entité,
- l'ensemble $AssoRoEnt$ des associations entre les rôles et les entités change : apparition et/ou disparition d'une nouvelle association rôle-entité,
- l'ensemble $AssoRelEnt$ des associations entre les relations et les entités change : apparition et/ou disparition d'une nouvelle association relation-entités

De manière plus formelle, une situation S se définit comme suit :

Soit $C = (R,Re)$, $R \subseteq Roles$, $Rel \subseteq Relations$ alors :

$$\forall(S \in C = (R, Rel)), \exists Ent, AssoRoEnt, AssoRelEnt \,/\, (S = Ent, AssoRoEnt, AssoRelEnt) \Leftrightarrow (Ent \in \wp(Entites)) \wedge (AssoRoEnt = f(Ent, R)) \wedge (AssoRelEnt = g(Ent, Rel)) \qquad (11)$$

où $\wp(Entites)$ est l'ensemble des parties de $Entites$, f est la fonction d'association entre une entité e et un rôle r telle que «e joue r» et g la

fonction d'association entre une ou plusieurs entités $e_1 \ldots e_j$ et une relation *rel* telle que $rel(e_1, ..., e_j)$ est vrai.

De plus, la propriété suivante est vérifiée quelle que soit la situation S :

$$\begin{gathered}\forall((S_1 = Ent_1, AssoRoEnt_1, AssoRelEnt_1) \in C), \\ \forall((S_2 = Ent_2, AssoRoEnt_2, AssoRelEnt_2) \in C) / \\ (S_1 = S_2) \Leftrightarrow (Ent_1 = Ent_2) \wedge (AssoRoEnt_1 = AssoRoEnt_2) \\ \wedge (AssoRelEnt_1 = AssoRelEnt_2)\end{gathered} \quad (12)$$